

## ON BOUNDED DATABASE SCHEMES AND BOUNDED HORN-CLAUSE PROGRAMS\*

YEHOShUA SAGIV†

**Abstract.** A lossless database scheme with full implicational dependencies is considered. The following condition is necessary in order that restricted projections of the representative instance could be expressed in relational algebra (i.e., in first-order logic): The dependencies of the database scheme are equivalent to a single join dependency and some equality-generating dependencies. An important special case is when the database scheme has only full tuple-generating dependencies. In this case, restricted projections of the representative instance can be expressed in relational algebra if and only if the following condition is true: The dependencies of the database scheme are equivalent to a single join dependency. Testing this condition is decidable. This result also applies to the class of Horn-clause programs consisting only of typed rules with one predicate symbol  $R$  (and without function symbols). A program consisting of rules of this form is equivalent to a nonrecursive program if and only if it is equivalent to a project-join mapping. Finally, it is also shown that a tableau mapping is idempotent if and only if it is a project-join mapping.

**Key words.** database, dependency, Horn-clause rule, losslessness, query, relational algebra, representative instance, tableau

AMS(MOS) subject classifications. 68P, 68Q

**1. Introduction.** The decomposition of a relation produces several new relations in some normal form and, consequently, certain update and storage anomalies are removed (cf. [Ma], [U1]). The original relation, however, remains a correct representation of the various relationships among its attributes, and frequently it is easier to formulate a query with respect to the original relation rather than the new relations. More specifically, many queries that can be formulated over the original relation as a projection and a selection, also require joins and possibly unions when formulated over the new relations, because all the correct access paths among the new relations have to be included in the answer. Surprisingly, in some simple cases it is (not just more difficult but actually) impossible to express projections of the original relation in relational algebra (i.e., first-order logic) if we insist that the new relations be the only operands [MUV]. When making statements like the previous one, we have, of course, to describe the connection between the new relations that are actually stored in the database and the original relation that does not exist in the database. We adopt the *representative instance* model [Ho], [Me], [Sa1], [Sa3], [Ya] that has been widely accepted as the correct way of producing the original relation from the new ones. The representative instance contains all the data that is obtained by losslessly joining (cf. [U1]) some of the new relations; and it has null values in order to account for partial information, i.e., tuples that cannot be extended, using lossless joins, to complete tuples over all the attributes, because of missing information.

If we choose to allow users to formulate queries over the representative instance, we are faced with the problem of optimizing these queries. Queries usually involve only a few attributes of the original relation and, therefore, it is desirable to have algebraic (i.e., relational algebra) expressions, with the new relations as their operands,

---

\* Received by the editors December 3, 1985; accepted for publication (in revised form) January 23, 1987. Part of this work was done while the author visited Stanford University, Stanford, California 94305-2085. This work was supported by a grant from AT&T Foundation, a grant from the IBM Corporation and National Science Foundation grant IST-84-12791. At Hebrew University, this work was supported in part by a grant from the Charles H. Revson Foundation.

† Department of Computer Science, Givat Ram 91904, Hebrew University, Jerusalem, Israel.

that compute restricted projections<sup>1</sup> of the representative instance. Hopefully, these expressions could be optimized by known techniques, and evaluated in considerably less time than required to compute the whole representative instance.

Maier et al. [MUV] showed (using the Compactness Theorem) that if there is a first-order formula that expresses the restricted projection onto some  $X$ , then there is also a union of tableaux that expresses it. Since the result of [MUV] uses the Compactness Theorem, it is valid in the unrestricted case, i.e., when databases are either finite or infinite. In this paper we consider lossless database schemes with *full implicational dependencies* [BV2], [Fa], [YP]. We give a necessary condition for the existence of a union of tableaux expressing the restricted projection onto the set of all the attributes. The condition states that the dependencies of the database scheme must be equivalent to a single join dependency and some equality-generating dependencies, and testing whether it is satisfied is decidable. Moreover, if the dependencies of the database scheme are only full tuple-generating dependencies (and no equality-generating dependencies), then equivalence to a single join dependency is both necessary and sufficient for the existence of unions of tableaux that compute restricted projections of the representative onto any set of attributes. Our results hold for finite databases. For unrestricted databases, “a union of tableaux” can be replaced with “an algebraic expression” (i.e., “a first-order formula”) in the above results, by combining them with those of [MUV].

Our results also apply to the following fragment of Horn-clause programs. The only rules allowed are typed rules with only one predicate symbol  $R$  and neither function symbols nor equality (“typed” means that no variable appears in more than one column of  $R$ , but of course a variable may appear in several occurrences of  $R$  as long as it is always in the same column). A program consisting of a set of rules of this form is equivalent to a nonrecursive program if and only if it is equivalent to a project-join mapping. Testing this condition is decidable. Cosmadakis and Kanellakis [CK] have recently investigated this class of Horn-clause programs.

## 2. Preliminaries.

**2.1. Schemes, states and instances.** Relations are tables of information in which the rows are *records* or *tuples* of data, and the columns are labeled by *attributes*. A *relation scheme* is a set of attributes, and it describes the fixed structure of a table. A *relation  $r$  over a relation scheme  $R$*  is a finite set of tuples, where each tuple is conveniently regarded as a mapping from the attributes of  $R$  to their domains. The value of a tuple  $\mu$  for an attribute  $A$  is denoted  $\mu(A)$ . We assume that each attribute has an infinite domain of *constants* (e.g., integers).

A *database scheme* consists of relation schemes and *dependencies*, and is denoted by  $([R_1, \dots, R_n], D)$ , where each  $R_i$  is a relation scheme, and  $D$  is a set of dependencies that will be defined later. The set of all the attributes, i.e.,  $\cup_{i=1}^n R_i$ , is denoted by  $U$ . A *state* of the database scheme  $([R_1, \dots, R_n], D)$  is a collection of relations  $r_1, \dots, r_n$  over  $R_1, \dots, R_n$ , respectively.

The *projection* of a relation  $r$  onto a set of attributes  $X$ , written  $\pi_X(r)$ , is obtained by removing columns not in  $X$  and eliminating duplicate tuples. A (universal) *instance  $I$*  is a relation over  $U$ . An instance  $I$  *generates* the state  $\pi_{R_1}(I), \dots, \pi_{R_n}(I)$ , which is denoted by  $state(I)$ .

---

<sup>1</sup> A restricted projection is a projection followed by elimination of all tuples that have nulls in some columns. Essentially, restricted projections correspond to the most basic queries that can be posed over the representative instance.



**2.2. Tableaux and tableau mappings.** Tableaux are formulas that express queries over a given database scheme  $([R_1, \dots, R_n], D)$ . A *tableau*  $[ASU1], [ASU2]$  is a table, similar to a relation, with columns that correspond to the attributes of  $U$  and rows that are filled with *variables*. We denote a tableau either by a single letter (e.g.,  $T$ ) or by explicitly specifying its rows, i.e.,  $(w_1, \dots, w_n)/s$ , where  $w_1, \dots, w_n$  are the rows of the tableau and  $s$  is a special row called the *summary*. The summary  $s$  may have blanks in some columns; the rows  $w_i$  have variables in all the columns. Each variable appearing in the summary must also appear in some  $w_i$ , and no variable appears in more than one column. The variables appearing in the summary are called *distinguished variables*, and other variables are called *nondistinguished variables*. Each row has a *tag*, which is one of the relation schemes  $R_i$ . A row with a tag  $R_i$  has a unique nondistinguished variable (i.e., a variable that appears nowhere else) in each column whose attribute is not in  $R_i$ . The phrase “the rows of the tableau  $(w_1, \dots, w_n)/s$ ” refers only to the  $w_i$ , i.e., the summary is excluded.

*Example 2.1.* In examples we usually represent a tableau as a table in which the summary appears at the top and is underlined. Distinguished variables are represented by  $a$ 's with subscripts, and nondistinguished variables are represented by  $b$ 's with subscripts (and sometimes also by other letters). In this example we explicitly show the attribute of each column, but we will not do so in future examples. Suppose that the set<sup>2</sup> of all the attributes is  $ABCD$ . The following is a tableau whose rows are tagged with the relation schemes  $AB, ABC, ABD$ .

	A	B	C	D
	$a_1$	$a_2$	$a_3$	$a_4$
$AB$	$a_1$	$a_2$	$b_1$	$b_2$
$ABC$	$b_5$	$a_2$	$a_3$	$b_3$
$ABD$	$b_5$	$a_2$	$b_4$	$a_4$

We say that  $h$  is a *symbol mapping* of a tableau  $T$  if  $h$  maps each variable of  $T$  to a constant. The result of applying  $h$  to  $u$ , where  $u$  is either a row or the summary of  $T$ , is a tuple defined naturally as follows. The tuple  $h(u)$  maps attribute  $A$  to  $h(u(A))$ , where  $u(A)$  is the variable appearing in column  $A$  of  $u$ . If  $u$  has a tag  $R_i$ , only the projection of  $h(u)$  onto  $R_i$  is of interest, and by a slight abuse of notation, we will denote that projection by  $h(u)$ . The symbol mapping  $h$  is a *valuation* of  $T$  into a database  $r_1, \dots, r_n$  if  $h$  maps every row of  $T$  with tag  $R_i$  to a tuple of  $r_i$ .

A tableau  $T$  defines a mapping from states (of the given database scheme) to relations. Tableau  $T$  maps a state  $r_1, \dots, r_n$  to the following relation, denoted  $T(r_1, \dots, r_n)$ ,

$$\{h(s) \mid s \text{ is the summary of } T \text{ and } h \text{ is a valuation of } T \text{ into } r_1, \dots, r_n\}.$$

Note that  $T(r_1, \dots, r_n)$  is a relation over the set of attributes in which the summary of  $T$  has nonblank symbols.

We also consider tableaux that map instances to relations. A tableau defines a mapping of instances if all its rows have the tag  $U$ , and traditionally, we refer to a

<sup>2</sup> We use the common notation in which the letters  $A, B, C, \dots$  represent single attributes, the letters  $\dots, X, Y, Z$  represent sets of attributes, and a set or union of sets is written as a string of attributes.

tableau of this kind as an *untagged* tableau (and in examples we write it without tags). Unless explicitly said otherwise, tagged and untagged tableaux considered in this paper are *full* in the sense that their summaries have variables in all the columns, i.e., the value of a tableau is always an instance. It is easy to see that a full untagged tableau mapping  $T$  is *monotonic*, i.e., for all instances  $I$ ,  $I \subseteq T(I)$ .

A *union of tableaux* [SY] is an expression of the form  $\bigcup_{i=1}^m T_i$ , where  $T_1, \dots, T_m$  are tableaux. If the  $T_i$  are tagged, the value of  $\bigcup_{i=1}^m T_i$ , given a state  $r_1, \dots, r_n$ , is  $\bigcup_{i=1}^m T_i(r_1, \dots, r_n)$ . If the  $T_i$  are untagged, the value for an instance  $I$  is  $\bigcup_{i=1}^m T_i(I)$ . The mappings defined by tableaux and union of tableaux can always be expressed in relational algebra [Co].

**2.3. Dependencies.** Untagged tableaux also define *full implicational dependencies*<sup>3</sup> [BV2], [Fa], [YP]. These dependencies are either *tuple-generating dependencies* (tgds) or *equality-generating dependencies* (egds). A tgd  $d$  is written as an untagged tableau  $(w_1, \dots, w_n)/s$ , and it is *satisfied* by an instance  $I$  if for all valuations  $h$  of  $(w_1, \dots, w_n)/s$  into  $I$ , the tuple  $h(s)$  is in  $I$ . An egd is also written as an untagged tableau, but its summary is of the form  $a = b$ , where  $a$  and  $b$  are variables (both  $a$  and  $b$  must appear in the same column of the tableau). An egd  $(w_1, \dots, w_n)/a = b$  is *satisfied* by an instance  $I$  if for all valuations  $h$  of  $(w_1, \dots, w_n)/a = b$  into  $I$ , the equality  $h(a) = h(b)$  holds. As a technicality, we consider one of  $a$  and  $b$ , say  $a$ , as the only distinguished variable of the egd.

A tgd in which no nondistinguished variable occurs more than once is called a *join dependency*. A database scheme  $([R_1, \dots, R_n], D)$  has an associated join dependency, denoted by  $\bowtie R_i$ , such that for each  $R_i$ , the join dependency  $\bowtie R_i$  has a row with distinguished variables in the columns of  $R_i$  and distinct nondistinguished variables in the rest of the columns. When a tableau for a join dependency is viewed as an untagged tableau mapping, it is usually called a *project-join* mapping (because this tableau mapping is the same as joining some projections of an instance).

**2.4. Tableaux as mappings, dependencies and instances.** Given an untagged tableau  $T$ , we call it either a mapping or a tgd, depending on how we view it. We use the term “tableau” as a generic name for either mappings or tgds (but not egds). In some proofs in this paper we alternate between viewing a given tableau  $T$  as a mapping and as a tgd. Similarly, a tagged tableau may also be considered as an untagged mapping or a tgd simply by ignoring the tags.

A tableau (or more precisely, the rows of a tableau) may be converted into an instance by mapping each variable to a distinct constant. Since the one-to-one mapping that converts a tableau to an instance is only a formality, we shall refer to the tableau itself as an instance instead of using a mapping. When viewing a tableau as an instance, we always consider only the rows without the summary.

**2.5. The chase.** We permit the use of *nulls* in instances. A null represents an unknown value and is denoted by  $\delta_i$ . Two nulls are equal only if they have the same subscript, and a null is never equal to a constant. If we assume that an instance  $I$  should satisfy a set of dependencies  $D$ , then we may infer that some nulls can be replaced with constants or with other nulls, and tuples currently not in  $I$  must belong to it. The process of making these inferences is called the *chase* [ABU], [MMS], and it is defined in terms of rules that are associated with the dependencies of  $D$ .

<sup>3</sup> We do not consider embedded dependencies in this paper.

The rule for a  $\text{tgd } (w_1, \dots, w_n)/s$  states that if there is a valuation  $h$  of the  $w_i$  into  $I$ , such that  $h(s)$  is not a tuple of  $I$ , then  $h(s)$  is added to  $I$ . The rule for an  $\text{egd } (w_1, \dots, w_n)/a=b$  states that if there is a valuation  $h$  of the  $w_i$  into  $I$ , such that  $h(a) \neq h(b)$  and at least one of  $h(a)$  and  $h(b)$  is a null, then  $h(a)$  and  $h(b)$  are equated as follows. We arbitrarily choose a null among  $h(a)$  and  $h(b)$ , say  $h(b)$ , and replace all occurrences of  $h(b)$  in  $I$  with  $h(a)$ . The *chase* of  $I$  with respect to  $D$ , written  $\text{CHASE}_D(I)$ , is obtained by repeatedly applying the rules for the dependencies in  $D$  to the instance  $I$  until no rule can be applied anymore. The chase process terminates (since all dependencies are full), and  $\text{CHASE}_D(I)$  is uniquely defined up to renaming of nulls, provided that it satisfies (the dependencies of)  $D$  [MMS]. Clearly,  $\text{CHASE}_D(I)$  satisfies all the  $\text{tgds}$  of  $D$ , but not necessarily all the  $\text{egds}$  of  $D$ .

Let  $([R_1, \dots, R_n], D)$  be a database scheme. Given a state  $r_1, \dots, r_n$ , we create a special instance, called the *representative instance* [Ho], [Me], [Sa1], [Sa3], [Ya], in the following way. First, we augment each  $r_i$  with columns for the attributes in  $R - R_i$ , and fill each entry of these columns with a unique null (i.e., it appears nowhere else). The union of all the augmented relations is an instance denoted by  $A(r_1, \dots, r_n)$ . The representative instance of  $r_1, \dots, r_n$ , denoted  $RI(r_1, \dots, r_n)$ , is  $\text{CHASE}_D(A(r_1, \dots, r_n))$ .

A *consistent state* of the database scheme  $([R_1, \dots, R_n], D)$  is a state  $r_1, \dots, r_n$  such that  $RI(r_1, \dots, r_n)$  satisfies  $D$ . A *consistent instance* of  $D$  is an instance  $I$  without nulls, such that  $\text{CHASE}_D(I)$  satisfies  $D$ . The set of all consistent instances of  $D$  is denoted  $\text{CON}(D)$ . Clearly, an instance that satisfies  $D$  is also a consistent instance, but the converse is not necessarily true. Another important observation is the fact that if  $I$  is a consistent instance, then  $\text{state}(I)$  is a consistent state.

The chase can also be applied to the rows of a tableau  $T$ . When equating variables, distinguished variables are treated as constants and nondistinguished variables as nulls. Since each column of  $T$  has at most one distinguished variable, the set of rows of  $\text{CHASE}_D(T)$  always satisfies  $D$ . Note that  $\text{CHASE}_D(T)$  is a tableau with the same summary as  $T$  and with the rows that were generated from those of  $T$  by the chase ( $\text{CHASE}_D(T)$  includes the original rows of  $T$  but possibly with some nondistinguished variables replaced by other variables).

**2.6. Implications of dependencies and lossless database schemes.** A set of dependencies  $D$  implies a dependency  $d$ , written  $D \models d$ , if whenever an instance  $I$  satisfies  $D$ , then  $I$  also satisfies  $d$ . We can use the chase to test implications of dependencies [BV2], [MMS]. The set  $D$  implies  $d$  if and only if (the set of rows of)  $\text{CHASE}_D(d)$  satisfies  $d$ , or equivalently,

- (1)  $\text{CHASE}_D(d)$  contains a row which is equal to  $s$ , when  $d$  is a  $\text{tgd}$  with a summary  $s$ , and
- (2)  $a$  and  $b$  are equated during the chase of  $d$ , when  $d$  is an  $\text{egd}$  with a summary  $a = b$ .

A set of dependencies  $D_1$  implies another set  $D_2$ , written  $D_1 \models D_2$ , if  $D_1$  implies every dependency of  $D_2$ .  $D_1$  and  $D_2$  are *equivalent*, written  $D_1 \models \Leftrightarrow D_2$ , if  $D_1 \models D_2$  and  $D_2 \models D_1$ .

The following is an important observation. Two sets of dependencies  $D_1$  and  $D_2$  are equivalent if and only if for all instances  $I$ ,  $\text{CHASE}_{D_1}(I) = \text{CHASE}_{D_2}(I)$ . We will further discuss this observation in § 2.8.

A database scheme  $([R_1, \dots, R_n], D)$  is *lossless* if  $D \models \bowtie R_i$ . A database scheme has to be lossless in order to reconstruct tuples over  $U$  whose projections are stored in the database. Recall that the representative instance of a database state represents

all the data that either exist in the database state or can be inferred by correctly connecting tuples from various relations. If the database scheme is not lossless, then the representative instance can never have tuples with constants in all the columns of  $U$ , i.e., tuples without nulls [MUV]. In this paper we consider only lossless database schemes.

**2.7. Containment of tableau expressions.** We define tableau containment only for untagged tableau mappings. It is similarly defined for tagged tableaux [ASU1], but we do not use it in this paper. A tableau  $T_1$  *contains* a tableau  $T_2$  *over* a set of instances  $C$ , written  $T_2 \subseteq_C T_1$ , if for all instances  $I \in C$ ,  $T_2(I) \subseteq T_1(I)$ . Tableaux  $T_1$  and  $T_2$  are *equivalent over*  $C$ , written  $T_1 \equiv_C T_2$ , if for all instances  $I \in C$ ,  $T_2(I) = T_1(I)$ . If  $C$  is omitted, then  $C$  is assumed to be the set of all instances. Containment and equivalence are similarly defined for unions of tableaux.

A *homomorphism*  $h$  of  $T_1$  into  $T_2$  is a mapping of the variables of  $T_1$  into the variables of  $T_2$  that preserves summary and rows, that is

- (1) if  $w$  is a row of  $T_1$ , then  $h(w)$  is a row of  $T_2$ , and
- (2) for each column  $A$ , the distinguished variable of  $T_1$  in  $A$  is mapped to the distinguished variable of  $T_2$  in  $A$ .

If the rows of  $T_2$  are viewed as an instance, then a homomorphism is a valuation of  $T_1$  into the rows of  $T_2$  with the additional restriction that distinguished variables are mapped to distinguished variables. The existence of a homomorphism from  $T_1$  into  $T_2$  is a sufficient condition for  $T_2 \subseteq_C T_1$ , and it is also a necessary condition if the rows of  $T_2$  form an instance of  $C$  [ASU1]. The mapping of the rows of  $T_1$  into the rows of  $T_2$  induced by a homomorphism  $h$  is called a *containment mapping*. Essentially, a containment mapping  $\psi: T_1 \rightarrow T_2$  satisfies the following two conditions:

- (1) If rows  $w_1$  and  $w_2$  are equal in some column  $A$ , then so are  $\psi(w_1)$  and  $\psi(w_2)$ , and
- (2) If row  $w_1$  has a distinguished variable in some column  $A$ , then so does  $\psi(w_1)$ .

If there is a containment mapping from a tableau  $T$  into itself whose image does not include all the rows of  $T$ , then rows not in the image of the mapping can be eliminated and the result is a tableau equivalent to  $T$  (over all instances) [ASU2], [CM]. A tableau  $T$  is *minimal* if no row can be eliminated by some containment mapping. Efficient algorithms for minimizing tableaux are given in [ASU2], [Sa2].

A union of tableaux  $\bigcup_{i=1}^n T_i$  is contained in another union of tableaux  $\bigcup_{i=1}^m V_i$  if and only if each tableau  $T_i$  is contained in some  $V_j$  [SY].

**2.8. Bounded tgds and Horn-clause rules.** The chase can be viewed as an iterative process. For some sets of dependencies this iterative process is *bounded*, i.e., a fixed number of iterations is sufficient regardless of the input. To simplify the formal definitions, we assume that  $D$  has only tgds; the definition of boundedness when there are both tgds and egds can be found in [MUV]. So, suppose that  $D$  is a set of tgds and  $I$  is an instance, and consider the computation of  $CHASE_D(I)$ . The first iteration of the chase consists of repeatedly applying the dependencies of  $D$  to the original tuples of  $I$ , but not to the newly added tuples. Generally, in each iteration the dependencies are applied repeatedly to the rows generated in all previous iterations. Note that the following is true for any set of untagged tableaux  $\{T_1, \dots, T_m\}$  and any instance  $I$  (recall that  $\bigcup_{i=1}^m T_i(I)$  is the result of applying the mapping  $\bigcup_{i=1}^m T_i$  to  $I$ ): The instance  $\bigcup_{i=1}^m T_i(I)$  is equal to the result of the first iteration of chasing  $I$  with the tgds  $T_1, \dots, T_m$ .

The chase process with respect to a set of dependencies  $D$  is *bounded* if there is a fixed integer  $k$ , such that for all instances  $I$ , the computation of  $CHASE_D(I)$  requires

$k$  or fewer iterations. We will say that  $D$  is a *bounded* set of dependencies if the chase with respect to  $D$  is bounded.

The problem of determining boundedness is an important one, since the chase can be executed more efficiently when it is bounded. In fact, the chase can be expressed in relational algebra when it is bounded, but not in general [MUV]. We will elaborate on this point in § 4.

The boundedness problem also occurs in *deductive databases* (cf. [GM]). Deductive databases use *Horn-clause rules* in order to express queries, and tgds are essentially Horn-clause rules. A tgd  $(w_1, \dots, w_n)/s$  is just the following Horn-clause formula.

$$\forall \bar{x}(R(w_1) \wedge \dots \wedge R(w_n) \rightarrow R(s)).$$

Here  $R$  is relation scheme<sup>4</sup> consisting of all the attributes, i.e.,  $R = U$ , and  $\bar{x}$  is a list of all the variables in the tgd. In the notation of deductive databases (or logic programming), the above formula is written as the following rule.

$$R(s) :- R(w_1), \dots, R(w_n).$$

In general, Horn-clause rules may have many different predicates and need not be typed (they may also have function symbols). We will consider, however, only rules that correspond to tgds.

*Example 2.2.* Consider the following untagged tableau:

$a_1$	$a_2$	$a_3$	$a_4$
$a_1$	$a_2$	$b_1$	$b_2$
$e$	$a_2$	$a_3$	$b_3$
$e$	$a_2$	$b_4$	$a_4$

The tgd expressed by this tableau can also be written as the following Horn-clause rule.

$$R(a_1, a_2, a_3, a_4) :- R(a_1, a_2, b_1, b_2), R(e, a_2, a_3, b_3), R(e, a_2, b_4, a_4).$$

A set  $P$  of Horn-clause rules is viewed as a recursive program whose value for an instance  $I$  is just  $CHASE_P(I)$ , where the rules of  $P$  are considered as tgds when the chase is applied. Thus, viewing tableaux as tgds is essentially the same as viewing them as Horn-clause rules, whereas viewing tableaux as tgds is different from viewing them as mappings.

In this paper we will characterize when a set of tgds (and hence the corresponding set of Horn-clause rules), is bounded. This result is important not only for relational databases, but also in the context of developing efficient methods for evaluating Horn-clause rules. When a set of Horn-clause rules is bounded, it can be replaced with a relational expression and, thus, the task of optimizing the evaluation of the rules becomes considerably easier. Characterizations of boundedness for other fragments of Horn-clause rules have recently been described in [Io], [Na].

<sup>4</sup> Technically a relation scheme is a set of attributes, but we use it also as a relation name, or in logic terminology, as a predicate name.

In summary, a set  $S$  of untagged tableaux can be viewed either as

- (1) a set of Horn-clause rules,
- (2) a set of tgds, or
- (3) a mapping defined by the union of its tableaux.

Let  $S_1$  and  $S_2$  be two sets of untagged tableaux. The following are two important observations regarding the equivalence of  $S_1$  and  $S_2$ . First, the two sets are equivalent as sets of Horn-clause rules (i.e., they produce the same result for all instances) if and only if they are equivalent as sets of tgds. Second, if the two sets are equivalent as unions of tableaux, then they are also equivalent as sets of tgds (or Horn-clause rules), but the converse is not necessarily true. Equivalences among Horn-clause rules in more general cases are discussed in [Sa5].

**2.9. Composition of untagged tableaux.** Let  $T$  and  $V$  be untagged tableaux corresponding to either mappings or tgds. We will shortly describe how to construct a tableau  $V \circ T$ , called the *composition* of  $T$  and  $V$ , that has the following properties. If  $T$  and  $V$  are considered as mappings, then for all instances  $I$ ,  $V \circ T(I) = V(T(I))$ . If  $T$  and  $V$  are considered as dependencies, then  $V \circ T$  is equivalent to  $\{T, V\}$ , i.e.,  $V \circ T \models T, V$ .

First, we define the result of *substituting* a row  $w$  for the summary  $s$  of a tableau  $T$ , written  $T/w$ , as the tableau obtained in the following way. For each column  $A$ , we replace all occurrences of the distinguished variable  $s(A)$  with  $w(A)$ . Note that  $w$  becomes the summary of  $T/w$ . Next, we will describe how to construct  $V \circ T$ . Let  $w_1, \dots, w_n$  be the rows of  $V$ . We create  $n$  copies of  $T$ , denoted  $T_1, \dots, T_n$ , and in each copy we use variables that do not appear in any other copy. The tableau  $V \circ T$  has the summary of  $V$  and all the rows of the tableaux  $T_1/w_1, \dots, T_n/w_n$ . It is easy to prove that  $V \circ T$  has the above properties [BV1], [FMUY].

We will now consider the composition of a tableau  $T$  with itself, denoted  $T^2$ . At first, note that this composition can be viewed as taking a Horn-clause rule and replacing each predicate in the right-hand side with its definition. Next, we will show explicitly how  $T^2$  is constructed and develop a notation for its rows that will be used in later proofs. Let  $T_1, \dots, T_n$  be the copies of  $T$  as described above, and suppose that in all of them the rows  $w_1, \dots, w_n$  of  $T$  appear in the same order. We denote the rows of  $T_i/w_i$  as  $w_1^i, \dots, w_n^i$ . Tableau  $T^2$  consists of all the rows  $w_j^i (1 \leq i, j \leq n)$  and it has the same summary as  $T$ . Note that row  $w_j^i$  has in column  $A$  either

- (1) the same variable as row  $w_i$  (of  $T$ ) if row  $w_j$  has a distinguished variable in column  $A$ , or
- (2) a nondistinguished variable that appears only in  $w_1^i, \dots, w_n^i$  if row  $w_j$  has a nondistinguished variable in column  $A$ .

Thus, the variables of  $T^2$  are those of  $T$  and  $T_i$ .

*Example 2.3.* Let  $T$  be the following tableau, which was also considered in Example 2.2.

$a_1$	$a_2$	$a_3$	$a_4$
$a_1$	$a_2$	$b_1$	$b_2$
$e$	$a_2$	$a_3$	$b_3$
$e$	$a_2$	$b_4$	$a_4$

The rows of  $T$  are denoted, from top to bottom,  $w_1$ ,  $w_2$  and  $w_3$ . Note that  $e$  is the only repeated (i.e., appearing in more than one row) nondistinguished variable of  $T$ .

In order to construct  $T^2$ , we first create three copies of  $T$  with disjoint sets of variables.

$$T_1 = \begin{array}{|c|c|c|c|} \hline a_1^1 & a_2^1 & a_3^1 & a_4^1 \\ \hline a_1^1 & a_2^1 & b_1^1 & b_2^1 \\ e^1 & a_2^1 & a_3^1 & b_3^1 \\ e^1 & a_2^1 & b_4^1 & a_4^1 \\ \hline \end{array}, \quad T_2 = \begin{array}{|c|c|c|c|} \hline a_1^2 & a_2^2 & a_3^2 & a_4^2 \\ \hline a_1^2 & a_2^2 & b_1^2 & b_2^2 \\ e^2 & a_2^2 & a_3^2 & b_3^2 \\ e^2 & a_2^2 & b_4^2 & a_4^2 \\ \hline \end{array},$$

$$T_3 = \begin{array}{|c|c|c|c|} \hline a_1^3 & a_2^3 & a_3^3 & a_4^3 \\ \hline a_1^3 & a_2^3 & b_1^3 & b_2^3 \\ e^3 & a_2^3 & a_3^3 & b_3^3 \\ e^3 & a_2^3 & b_4^3 & a_4^3 \\ \hline \end{array}.$$

In each  $T_i$  we substitute  $w_i$  for the summary to obtain  $T_i/w_i$ . The resulting tableaux are as follows.

$$T_1/w_1 = \begin{array}{|c|c|c|c|} \hline a_1 & a_2 & b_1 & b_2 \\ \hline a_1 & a_2 & b_1^1 & b_2^1 \\ e^1 & a_2 & b_1 & b_3^1 \\ e^1 & a_2 & b_4^1 & b_2 \\ \hline \end{array}, \quad T_2/w_2 = \begin{array}{|c|c|c|c|} \hline e & a_2 & a_3 & b_3 \\ \hline e & a_2 & b_1^2 & b_2^2 \\ e^2 & a_2 & a_3 & b_3^2 \\ e^2 & a_2 & b_4^2 & b_3 \\ \hline \end{array},$$

$$T_3/w_3 = \begin{array}{|c|c|c|c|} \hline e & a_2 & b_4 & a_4 \\ \hline e & a_2 & b_1^3 & b_2^3 \\ e^3 & a_2 & b_4 & b_3^3 \\ e^3 & a_2 & b_4^3 & a_4 \\ \hline \end{array}.$$

$T^2$  has the nine rows of  $T_1/w_1$ ,  $T_2/w_2$  and  $T_3/w_3$ ; the summary is  $(a_1, a_2, a_3, a_4)$ . However,  $T^2$  can be minimized [ASU2], [CM], and the result is

$$\begin{array}{|c|c|c|c|} \hline a_1 & a_2 & a_3 & a_4 \\ \hline a_1 & a_2 & b_1^1 & b_2^1 \\ e^2 & a_2 & a_3 & b_3^2 \\ e^3 & a_2 & b_4^3 & a_4 \\ \hline \end{array}.$$

Note that  $T^2$  does not have repeated nondistinguished variables, namely, it is a project-join mapping (i.e., a join dependency). It is easy to show that if we compose  $T^2$  with itself, the result (after it is minimized) is the same as  $T^2$  (up to renaming of variables).

As another example, let  $V$  be the tableau

$a_1$	$a_2$
$a_1$	$b_1$
$b_2$	$b_1$
$b_2$	$a_2$

The following three tableaux are obtained after substituting each row of  $V$  for the summary of some copy of  $V$ :

$a_1$	$b_1$	$b_2$	$b_1$	$b_2$	$a_2$
$a_1$	$b_1^1$	$b_2$	$b_1^2$	$b_2$	$b_1^3$
$b_2^1$	$b_1^1$	$b_2^2$	$b_1^2$	$b_2^3$	$b_1^3$
$b_2^1$	$b_1$	$b_2^2$	$b_1$	$b_2^3$	$a_2$

$V^2$  has the nine rows of the above three tableaux, and its summary is  $(a_1, a_2)$ . No row of  $V^2$  can be removed by minimization. Essentially, if we continue to compose  $V$  with itself, we get a growing chain of the form

$$((a_1, b_1), (b_2, b_1), (b_2, b_3), \dots, (b_n, b_{n-1}), (b_n, a_2))/(a_1, a_2).$$

Note that each nondistinguished variable appears exactly twice. No two tableaux in the sequence  $V, V^2, V^3, \dots$  are equivalent as mappings, but they are all equivalent as tgds. As noted earlier, the composition of a tableau with itself always produces an equivalent tgd.

Recall that a tableau mapping is monotonic and, therefore,  $T \subseteq T^2$  (as the above example shows,  $T$  and  $T^2$  are not necessarily equivalent). Next, we will define two specific containment mappings from  $T^2$  to  $T$  that will be used later. The first, denoted  $\bar{\theta}$ , is defined for  $i, j$  by  $\bar{\theta}(w_j^i) = w_i$ . The second, denoted  $\underline{\theta}$ , is defined by  $\underline{\theta}(w_j^i) = w_j$ .

**PROPOSITION 2.1.** *Both  $\bar{\theta}$  and  $\underline{\theta}$  are containment mappings.*

*Proof.* The proof follows from observations (1) and (2) that were stated prior to Example 2.3. Suppose that  $w_j^i$  has a distinguished variable in column  $A$ , then so do  $w_i$  and  $w_j$ . Therefore, both  $\bar{\theta}$  and  $\underline{\theta}$  map distinguished variables to distinguished variables. Now suppose that  $w_j^i$  and  $w_k^m$  have the same variable  $b$  in column  $A$ . We claim that  $\bar{\theta}$  and  $\underline{\theta}$  map  $w_j^i$  and  $w_k^m$  to rows that have the same variable in column  $A$ .

**CASE 1.**  $b$  appears in  $T$ . In this case, both  $w_i$  and  $w_m$  must have  $b$  in column  $A$ , and both  $w_j$  and  $w_k$  must have a distinguished variable in column  $A$ . Thus, the claim is true.

**CASE 2.**  $b$  does not appear in  $T$ . In this case, by observation (2),  $i = m$ , and  $w_j$  and  $w_k$  have the same variable in column  $A$ . Thus,  $\bar{\theta}$  maps  $w_j^i$  and  $w_k^m$  to the same row, and  $\underline{\theta}$  maps them to rows that have the same variable in column  $A$ . Hence, the claim is true.  $\square$

### 3. Idempotent tableaux.

**3.1. Idempotency over all instances.** Let  $T$  be an untagged tableau mapping.  $T$  is *idempotent* if  $T \equiv T^2$ . In this section we will characterize idempotent tableaux, i.e., we will show that a tableau is idempotent if and only if it is a project-join mapping (i.e., a join dependency). Later we will use this result to show that a set  $D$  of tgds is bounded



if and only if it is equivalent to a join dependency. Note that a join dependency is not only bounded, but is also idempotent (i.e., its bound is 1).

**THEOREM 3.1.** *Let  $T$  be a minimal tableau. There is a homomorphism from  $T$  into  $T^2$  if and only if  $T$  is a project-join mapping (i.e., a join dependency).*

*Proof.* Let  $T$  be a minimal tableau which is not a project-join mapping. We will show that there is no homomorphism from  $T$  into  $T^2$ . Suppose that the claim is false and, so, let  $\psi$  be a containment mapping from the rows of  $T$  to the rows of  $T^2$ . Since  $T$  is not a project-join mapping, it has (by definition) a repeated nondistinguished variable  $b$ , i.e.,  $b$  appears in more than one row. Since  $\psi$  is a containment mapping from  $T$  into  $T^2$ , we can compose either  $\bar{\theta}$  or  $\theta$  with  $\psi$  in order to get a containment mapping from  $T$  to itself. The composed mappings are denoted  $\bar{\theta} \circ \psi$  and  $\theta \circ \psi$ . We will consider three cases depending upon the type of the variable to which  $\psi$  maps  $b$ , and contradict the minimality of  $T$  in each case.

**CASE 1.**  $\psi$  maps  $b$  to a distinguished variable. Thus,  $\bar{\theta} \circ \psi$  also maps  $b$  to a distinguished variable. If we compose  $\bar{\theta} \circ \psi$  with itself  $n$  times (where  $n$  is the number of rows of  $T$ ), we will get a containment mapping from  $T$  to itself whose image does not include any row having  $b$  (since a distinguished variable cannot be mapped to  $b$ ), and that contradicts the minimality of  $T$ .

**CASE 2.**  $\psi$  maps  $b$  to a nondistinguished variable that does not appear in  $T$ . In this case, the containment mapping  $\bar{\theta} \circ \psi$  maps all the rows that have  $b$  to a single row of  $T$ , in contradiction to the minimality of  $T$ .

**CASE 3.**  $\psi$  maps  $b$  to a nondistinguished variable of  $T$ . Suppose that  $b$  appears in column  $A$  of some row  $w_k$  of  $T$ , and  $\psi(w_k) = w_j^i$ . Since  $w_j^i$  has a variable of  $T$  in column  $A$ , it follows that  $w_j$  has a distinguished variable in column  $A$ . Therefore,  $\theta \circ \psi$  maps  $b$  to a distinguished variable and, consequently, the minimality of  $T$  is contradicted as in Case 1.

Since the minimality of  $T$  is contradicted in all these cases, it follows that there is no containment mapping from  $T$  to  $T^2$  if  $T$  has a repeated nondistinguished variable. Conversely, if  $T$  has no repeated nondistinguished variables (i.e., it is a project-join mapping), then  $T \equiv T^2$  [BMSU], and therefore there is a homomorphism from  $T$  into  $T^2$  [ASU1], [CM].  $\square$

It follows immediately from Theorem 3.1 that a minimal tableau  $T$  is idempotent if and only if it is a project-join mapping. In fact, it is easy to extend this result to idempotency over any set of instances  $C$  provided that the rows of  $T^2$  form an instance of  $C$ . This is shown in the next corollary.

**COROLLARY 3.1.** *Let  $T$  be a minimal untagged tableau.*

(1)  $T \equiv T^2$  if and only if  $T$  is a project-join mapping (i.e., a join dependency).

(2) Let  $C$  be a set of instances, and suppose that the rows of  $T^2$  form an instance of  $C$ . Then  $T \equiv_C T^2$  if and only if  $T$  is a project-join mapping.

*Proof.* At first, note that (1) is a special case of (2) (simply choose  $C$  to be the set of all instances). Thus, it remains to prove (2). If  $T$  is a project-join mapping, then there is a homomorphism from  $T$  into  $T^2$  (by Theorem 3.1) and hence  $T \equiv_C T^2$  (for any  $C$ ). If  $T \equiv_C T^2$  and the rows of  $T^2$  form an instance of  $C$ , then we can show as in [ASU1] that there is a homomorphism from  $T$  into  $T^2$ , and hence by Theorem 3.1,  $T$  is a project-join mapping.  $\square$

**3.2. Idempotency over consistent instances.** When a database scheme has egds as well as tgds, the only instances of interest are the consistent ones. Consequently, we would like to characterize the tableaux that are idempotent over  $CON(D)$ , where  $D$  is a set of tgds and egds. It may seem at first that Corollary 3.1 characterizes idempotency

over any set of instances  $C$ . However, the characterization of Corollary 3.1 applies to a tableau  $V$  only when the rows of  $V^2$  form an instance of  $C$ . This does not present a problem when  $D$  has only tgds, since  $CON(D)$  includes all instances when there are no egds. However, when there are egds, the rows of  $V^2$  do not necessarily form a consistent instance<sup>5</sup> of  $D$ . In order to overcome this difficulty, we will introduce a tableau  $\bar{V}$  (to be defined shortly), such that

- (1)  $V \equiv_{CON(D)} \bar{V}$ , and
- (2) the rows of  $\bar{V}^2$  form a consistent instance of  $D$ .

In fact, we need an additional assumption that states that  $V$  is a lossless tableau, i.e.,  $D \models V$ . This assumption is a reasonable one, since only lossless tableaux correspond to semantically correct queries (we will further discuss this point in § 4). The above properties of  $V$  and  $\bar{V}$  imply the following. By (2), the characterization of Corollary 3.1 applies to  $\bar{V}$ . Moreover, as we shall prove shortly,  $V$  is idempotent over  $CON(D)$  if and only if it is equivalent over  $CON(D)$  to  $\bar{V}$ , which must also be idempotent. Therefore,  $V$  is idempotent over  $CON(D)$  if and only if  $V$  is equivalent over  $CON(D)$  to a project-join mapping. Before we formally prove all these facts in the next theorem, we will define  $\bar{V}$ .

Tableau  $\bar{V}$  is the *equality chase* of  $V$  with respect to  $D$ , written  $ECHASE_D(V)$ . Tableau  $ECHASE_D(V)$  is obtained from  $V$  by equating variables of  $V$  whenever they are equated in  $CHASE_D(V)$ . In other words,  $ECHASE_D(V)$  is the result of computing  $CHASE_D(V)$  and then deleting the rows that were added during the chase. At first, we will show that  $V$  and  $ECHASE_D(V)$  are equivalent over  $CON(D)$ .

**PROPOSITION 3.1.** *The set of rows of  $ECHASE_D(V)$  is a consistent instance of  $D$ , and  $ECHASE_D(V) \equiv_{CON(D)} V$ .*

*Proof.* The set of rows of  $ECHASE_D(V)$  is a consistent instance of  $D$ , since (by definition) no variables are equated when it is chased with  $D$ .

We will now prove the equivalence of  $V$  and  $ECHASE_D(V)$  over  $CON(D)$ . At first, we will show that if  $I$  is a consistent instance of  $D$ , then every tuple of  $V(I)$  is also in  $ECHASE_D(V)(I)$ . So, suppose that  $h$  is a valuation of  $V$  into  $I$ , and let  $w$  and  $u$  be rows of  $V$ . If the variables in column  $A$  of these rows become equal in  $ECHASE_D(V)$ , then  $h(w)$  and  $h(u)$  must be equal on  $A$ , because  $I$  is a consistent instance of  $D$  (see [ASU1], [MMS] for a more detailed proof of this fact). Therefore,  $h$  is also a valuation of  $ECHASE_D(V)$  into  $I$  and, so, every tuple in  $V(I)$  is also in  $ECHASE_D(V)(I)$ . Thus, we have shown that for all  $I \in CON(D)$ ,  $V(I) \subseteq ECHASE_D(V)(I)$ . Containment in the other direction follows from the fact that there is a homomorphism of  $V$  into  $ECHASE_D(V)$ , since equalities among rows of  $V$  are preserved in  $ECHASE_D(V)$ . Thus,  $ECHASE_D(V) \equiv_{CON(D)} V$ .  $\square$

The next theorem shows that a lossless tableau  $V$  is idempotent over  $CON(D)$  if and only if  $ECHASE_D(V)$  is idempotent over  $CON(D)$ . It is also shown that the characterization of Corollary 3.1 applies to  $ECHASE_D(V)$ . Therefore, by the equivalence of  $V$  and  $ECHASE_D(V)$  over  $CON(D)$ , tableau  $V$  is idempotent over  $CON(D)$  if and only if it is equivalent to a project-join mapping over  $CON(D)$ .

**THEOREM 3.2.** *Let  $D$  be a set of egds and tgds, and  $V$  be an untagged tableau such that  $D \models V$ . We denote  $ECHASE_D(V)$  by  $\bar{V}$ . The following are equivalent:*

- (1)  $V \equiv_{CON(D)} V^2$ .
- (2)  $\bar{V} \equiv_{CON(D)} \bar{V}^2$ .
- (3) *When  $\bar{V}$  is minimized, the result is a project-join mapping.*

<sup>5</sup> Note that the rows of a tableau  $T$  form a consistent instance of  $D$  if no variables are equated during  $CHASE_D(T)$ .

*Proof.* Suppose that  $\bar{V}$  is minimal. At first, we will show that  $V \equiv_{CON(D)} V^2$  if and only if  $\bar{V} \equiv_{CON(D)} \bar{V}^2$ . Let  $I \in CON(D)$ . By Proposition 3.1,

$$(A) \quad \bar{V}(I) = V(I).$$

Since  $D \models V$ , it follows (by [MMS]) that  $CHASE_D(V(I)) = CHASE_D(I)$  and, therefore,  $V(I)$  is also a consistent instance of  $D$  (since no constants are equated during the chase of  $I$ ). Thus, Proposition 3.1 implies

$$(B) \quad \bar{V}^2(I) = V^2(I).$$

By (A) and (B),  $V \equiv_{CON(D)} V^2$  if and only if  $\bar{V} \equiv_{CON(D)} \bar{V}^2$ .

Next, we will show that  $\bar{V}^2 \in CON(D)$  (i.e., the rows of  $\bar{V}^2$  form a consistent instance of  $D$ ). By Corollary 3.1, that will prove that (2) and (3) are equivalent. Let  $(w_1, \dots, w_n)$  be the rows of  $\bar{V}$ , and  $(w_j^i)$  be the rows<sup>6</sup> of  $\bar{V}^2$ . Clearly,  $\bar{V} \in CON(D)$ . Suppose that  $\bar{V}^2 \notin CON(D)$ , that is, when we compute  $CHASE_D(\bar{V}^2)$ , two variables of  $\bar{V}^2$ , say  $a$  and  $b$ , are equated. Thus,  $D \models (w_j^i)/a = b$ . We will show that (the rows of)  $CHASE_D(\bar{V})$  violates  $(w_j^i)/a = b$ , which contradicts the fact that  $D \models (w_j^i)/a = b$  (because  $CHASE_D(\bar{V})$  satisfies  $D$ ) and, hence, completes the proof.

CASE 1. Both  $a$  and  $b$  are variables of  $\bar{V}^2$  that appear also in  $\bar{V}$ . Consider  $\bar{\theta}$  as a valuation of  $\bar{V}^2$  into  $\bar{V}$ . Since  $\bar{V} = ECHASE_D(V)$ , no variables are equated during the chase of  $\bar{V}$ , and so  $\bar{\theta}$  is also a valuation of  $\bar{V}^2$  into  $CHASE_D(\bar{V})$  that maps  $a$  and  $b$  to distinct variables of  $CHASE_D(\bar{V})$  (since in this case each one of  $a$  and  $b$  is mapped to itself). Therefore,  $\bar{\theta}$  is a valuation showing that  $CHASE_D(\bar{V})$  violates  $(w_j^i)/a = b$ .

CASE 2. At most one of  $a$  and  $b$  appears also in  $\bar{V}$ . If there is an  $i$ , such that  $a$  and  $b$  appear in rows  $w_j^i$  and  $w_k^i$ , respectively, then  $\bar{\theta}$  maps  $a$  and  $b$  to distinct variables. If there is no  $i$  satisfying the above condition, then there are two subcases to be considered.

SUBCASE 1. Exactly one of  $a$  and  $b$ , say  $a$ , appears also in  $\bar{V}$ . In this case,  $\bar{\theta}$  maps  $a$  to a distinguished variable and  $b$  to a nondistinguished variable.

SUBCASE 2. Neither  $a$  nor  $b$  is in  $\bar{V}$ . By the assumption leading to the two subcases, there are  $w_j^i$  and  $w_k^m$  that contain  $a$  and  $b$ , respectively, and  $i \neq m$ . We define a valuation  $h$  by mapping all the rows  $w_q^p$  ( $p \neq i$  and  $1 \leq q \leq n$ ) using  $\bar{\theta}$ , and mapping the rows  $w_1^i, \dots, w_n^i$  to the row of  $CHASE_D(\bar{V})$  consisting of all the distinguished variables (since  $D \models V$ , this row exists). Since  $h$  maps  $a$  and  $b$  to a distinguished variable and a nondistinguished variable, respectively, the desired violation is shown. It remains to be proved that  $h$  is indeed a valuation. Suppose that rows  $w_q^p$  ( $p \neq i$ ) and  $w_k^i$  have the same variable  $v$  in some column  $A$ . Since  $v$  appears in both  $w_q^p$  and  $w_k^i$  and  $p \neq i$ , it follows that  $w_q^p$  has a distinguished variable in column  $A$ . Thus, both  $w_q^p$  ( $p \neq i$ ) and  $w_k^i$  are mapped to rows having the same variable in column  $A$ . Clearly,  $h$  preserves equalities among any pair of rows  $w_q^p$  and  $w_k^m$  whenever both  $p$  and  $m$  are either equal or not equal to  $i$ , since  $\bar{\theta}$  is a containment mapping and rows  $w_1^i, \dots, w_n^i$  are mapped to a single row. Hence, we have shown that  $h$  is indeed a valuation.  $\square$

#### 4. Computing restricted projections of representative instances.

**4.1. Bounded database schemes.** There are two modes of retrieving information from a database. Either the information is found in a single relation, or it has to be obtained by joining tuples of various relations. The first mode is conceptually easy, whereas the second requires either the user or the system to determine which joins are correct. Correctness of joins is based on the semantics of the database scheme, which

<sup>6</sup> We write  $(w_j^i)$  instead of  $(w_1^i, \dots, w_n^i)$ .

is described by the dependencies. Since dependencies are formal statements in first-order logic, they can be used to infer formally all the tuples that are obtained by correct joins. In fact, this is exactly what happens during the chase that generates the representative instance  $RI(r_1, \dots, r_n)$ . The chase starts with  $A(r_1, \dots, r_n)$ , which consists of all the tuples of  $r_1, \dots, r_n$  (after they have been augmented with nulls), and generates new tuples from existing ones (either by adding tuples or equating symbols). The generation of new tuples amounts to correctly (or losslessly) joining existing tuples according to the dependencies of the database scheme. Therefore, the representative instance  $RI(r_1, \dots, r_n)$  contains all the data that is either in the relations  $r_1, \dots, r_n$  or can be inferred by correct joins. Clearly, the conclusion that the representative instance gives the complete picture of the data is based only on the premise that the dependencies fully describe the semantics of the database scheme.

Of course, the dependencies are stated for a universal relation scheme consisting of all the attributes. This is appropriate when the relation schemes are the result of decomposing one universal relation scheme  $R$  (which consists of all the attributes). If this is not the case, then our results apply to each group of relation schemes obtained by decomposing a bigger relation scheme. Since decomposition is a common tool in designing database schemes, our results are relevant in practice. To simplify the discussion, we will continue to consider a database scheme  $([R_1, \dots, R_n], D)$  in which all the relation schemes  $R_1, \dots, R_n$  have been obtained by decomposing one universal relation scheme  $R$ .

So far, we have concluded that the representative instance contains all the data that can be inferred from the database. Consequently, queries should be evaluated according to the representative instance and, so, they may as well be formulated in terms of the universal relation scheme  $R$  rather than  $R_1, \dots, R_n$ . Before going on with the discussion of query evaluation, we will further explain why formulation in terms of  $R$  is natural and easy. It is well known that decomposition is intended to remove certain operational (i.e., update and storage) anomalies, and it does so while preserving the semantics of the database scheme (cf. [U1]). Thus,  $R_1, \dots, R_n$  and  $R$  have the same semantics and so, in principle, any query can be formulated either in terms of  $R_1, \dots, R_n$  or in terms of  $R$ . There are, however, substantial differences between the two formulations. First, formulating a query in terms of  $R_1, \dots, R_n$  is generally more cumbersome, since the query should specify how to join the  $R_i$  (unless it refers to only one of the  $R_i$ ). In comparison, many queries that refer to attributes of several  $R_i$  can be formulated in terms of  $R$  using only projection and selection. Second, sometimes it is just impossible to formulate a query in terms of  $R_1, \dots, R_n$  in relational algebra (i.e., in first-order logic), whereas the same query can be formulated in terms of  $R$  using projection alone. The reason for that is the fact that the chase process is recursive,<sup>7</sup> and recursion cannot be expressed in relational algebra. In this paper we essentially investigate when a project query on  $R$  can also be expressed in terms of  $R_1, \dots, R_n$  in relational algebra.

We will now explain how to evaluate a query according to the representative instance. If a query refers to a set of attributes  $X$ , then it should be evaluated, in principle, according to the projection of  $RI(r_1, \dots, r_n)$  onto  $X$ . However, since  $\pi_X(RI(r_1, \dots, r_n))$  may still have nulls that were introduced during the computation

---

<sup>7</sup> A common misconception is the belief that a fixed number of joins describes all the lossless joins among relation schemes  $R_1, \dots, R_n$ . This is not true even if there are only functional dependencies [MUV]. In general, the chase is the only method for joining tuples losslessly in all possible ways, and that requires a number of iterations that depend on the actual relations  $r_1, \dots, r_n$ .

of the representative instance, it is probably not the most suitable choice. A more natural choice is the *restricted projection* of  $RI(r_1, \dots, r_n)$  onto  $X$ , denoted  $\pi \downarrow_X (RI(r_1, \dots, r_n))$ , which is defined to be the projection of  $RI(r_1, \dots, r_n)$  onto  $X$  followed by the elimination of all the tuples with nulls in some columns.

Since a query that refers to  $X$  is evaluated according to  $\pi \downarrow_X (RI(r_1, \dots, r_n))$ , a desirable property of a database scheme is the ability to compute efficiently restricted projections of the representative instance. Of course, we can always perform the chase on  $A(r_1, \dots, r_n)$ , which is the initial instance created by augmenting the database relations with nulls, but that involves all the data in the database and, consequently, is time consuming. An alternative approach is to look for an expression  $E_X$  in relational algebra, such that for all consistent states  $r_1, \dots, r_n$ ,

$$E_X(r_1, \dots, r_n) = \pi \downarrow_X (RI(r_1, \dots, r_n)).$$

Note that the operands of  $E_X$  are only the relations  $r_1, \dots, r_n$ , and not the representative instance itself; the operators of  $E_X$  are the ordinary operators of relational algebra. If  $E_X$  satisfies the above condition, then we say that  $E_X$  *computes*  $\pi \downarrow_X (RI(r_1, \dots, r_n))$ . Hopefully, if  $E_X$  exists and can be constructed, then it might be optimized by known techniques and evaluated in considerably less time than required to perform the chase on  $A(r_1, \dots, r_n)$ .

Maier et al. [MUV] have shown that for unrestricted databases if there is a relational algebra expression that computes  $\pi \downarrow_X (RI(r_1, \dots, r_n))$ , then there is also a (finite) union of (tagged) tableaux  $\cup_{i=1}^m T_i$  that computes  $\pi \downarrow_X (RI(r_1, \dots, r_n))$ . Not surprisingly, the tableaux in this union (when considered as tgds) are all lossless, i.e.,  $D \models T_1, \dots, T_m$ . Maier et al. [MUV] have also shown that for finite databases the following two conditions are equivalent:

- (1) For all  $X$ , there is a union of tableaux that computes  $\pi \downarrow_X (RI(r_1, \dots, r_n))$ .
- (2) The database scheme  $([R_1, \dots, R_n], D)$  is bounded.

Boundedness of the database scheme is similar, but not identical, to boundedness of  $D$ , which was defined in § 2.8 (boundedness of the database scheme depends on both  $D$  and  $R_1, \dots, R_n$ , and is formally defined in [MUV]). However, when  $D$  has only full tgds, our results imply that the two are the same, i.e., (1) is equivalent to the following:

- (2') The set of dependencies  $D$  is bounded.

In the next section, we will show that a set of tgds  $D$  is bounded if and only if it is equivalent to a join dependency. We will also show that this condition is decidable. In § 6, we will consider database schemes with both tgds and egds, and give a necessary condition for the existence of a union of tableaux that computes  $\pi \downarrow_U (RI(r_1, \dots, r_n))$ . This condition states that the set of dependencies is equivalent to a single join dependency and some egds; and the condition is decidable. Thus, a database scheme cannot be bounded unless the effect of its tgds is equivalent to that of a single join dependency (i.e., the tgds are equivalent to a join dependency over all consistent instances). In obtaining these results, we assume that the database scheme is lossless, i.e.,  $D \models R_i$ . This assumption is reasonable, since a database scheme has to be lossless in order to reconstruct tuples over all the attributes whose projections are stored in the database (in other words, unless the database scheme is lossless,  $\pi \downarrow_U (RI(r_1, \dots, r_n))$  is always empty [MUV]).

**4.2. Idempotency of tableaux computing restricted projections.** In this section, we consider a lossless database scheme  $([R_1, \dots, R_n], D)$  with a set  $D$  of full tgds and egds. We will show that if  $\pi \downarrow_U (RI(r_1, \dots, r_n))$  is computed by a union of tagged tableaux  $\cup_{i=1}^m T_i$ , then  $\cup_{i=1}^m T_i$  is idempotent over  $CON(D)$ , which is the set of all

consistent instances of  $D$ . In fact,  $\bigcup_{i=1}^m T_i$  is equivalent to a single idempotent tableau over  $CON(D)$ .

In the following proofs, we will use  $T$  to denote the composition of the  $T_i$ , i.e., we start by composing  $T_1$  and  $T_2$ , then composing the result with  $T_3$ , and so on, until we get  $T$ . Note that when we compose the  $T_i$ , we ignore the tags and  $T$  is an untagged tableau. The following lemma of [MUV] states that any union of tableaux that computes a restricted projection consists of lossless tableaux.

LEMMA 4.1. [MUV]. *Let  $([R_1, \dots, R_n], D)$  be a database scheme with tgds and egds. Suppose that  $\bigcup_{i=1}^m T_i$  computes  $\pi \downarrow_X (RI(r_1, \dots, r_n))$ . When the  $T_i$  are considered as dependencies,<sup>8</sup>  $D \models T_1, \dots, T_m$ .*

The next lemma describes fundamental properties of any union of tableaux  $\bigcup_{i=1}^m T_i$  and the composition  $T$  of its tableaux.

LEMMA 4.2. [BV1], [FMUY]. *Let  $\bigcup_{i=1}^m T_i$  be any union of tableaux, and  $T$  be the result of composing  $T_1, \dots, T_m$ .*

$$(1) \quad T \models T_1, \dots, T_m, \text{ and}$$

$$(2) \quad \bigcup_{i=1}^m T_i \subseteq T.$$

Note that in (1), all tableaux are considered as dependencies, and in (2), as untagged tableau mappings.

We will now prove the main lemma of this section.

LEMMA 4.3. *Let  $([R_1, \dots, R_n], D)$  be a database scheme with full tgds and egds. If  $\bigcup_{i=1}^m T_i$  computes  $\pi \downarrow_U (RI(r_1, \dots, r_n))$ , then  $T \equiv_{CON(D)} T^2$ , where  $T$  is the composition of the  $T_i$ .*

*Proof.* Let  $\rho = (r_1, \dots, r_n)$  be any consistent state of  $D$ . By our assumption

$$(1) \quad \pi \downarrow_U (RI(\rho)) = \bigcup_{i=1}^m T_i(\rho).$$

By the definition of the application of tableaux to instances and states

$$(2) \quad \bigcup_{i=1}^m T_i(\rho) \subseteq \pi \downarrow_U \left( \bigcup_{i=1}^m T_i(A(\rho)) \right)$$

where the  $T_i$  are considered with tags on the left-hand side, and without tags on the right-hand side. Since  $\bigcup_{i=1}^m T_i \subseteq T$  and  $T$  (as a dependency) is implied by  $D$  (by Lemmas 4.1 and 4.2), we get

$$(3) \quad \pi \downarrow_U \left( \bigcup_{i=1}^m T_i(A(\rho)) \right) \subseteq \pi \downarrow_U (T(A(\rho))) \subseteq \pi \downarrow_U (RI(\rho)).$$

It follows from equations (1)-(3)

$$(4) \quad \pi \downarrow_U (RI(\rho)) = \pi \downarrow_U (T(A(\rho))).$$

For each row  $w$  of  $T$  there is an  $R_i$ , such that  $w$  has distinguished or repeated nondistinguished variables only in the columns of  $R_i$  (since  $T$  is the composition of tagged tableaux). Consequently, for all universal instances  $I \in CON(D)$ ,

$$(5) \quad T(I) = \pi \downarrow_U (T(A(\pi_{R_1}(I), \dots, \pi_{R_n}(I)))).$$

It follows from (4) and (5) that for all instances  $I \in CON(D)$ ,

$$(6) \quad T(I) = \pi \downarrow_U (RI(\pi_{R_1}(I), \dots, \pi_{R_n}(I))).$$

<sup>8</sup> Note that when  $X \neq U$ , the  $T_i$  are not full tableaux. Although we do not consider embedded dependencies in this paper, the meaning of the lemma should be clear. We are going to use the lemma only when  $X = U$ .

By Lemmas 4.1 and 4.2,  $D \models T$  and, so, for all instances of  $I \in \text{CON}(D)$ , we have  $T(I) = T(T(I))$  (otherwise, (6) cannot hold).  $\square$

**COROLLARY 4.1.** *Let  $\bigcup_{i=1}^m T_i$  and  $T$  be as in Lemma 4.3.*

(1)  $\bigcup_{i=1}^m T_i \equiv_{\text{CON}(D)} T$ .

(2) *If  $\text{CON}(D)$  is the set of all instances, then there exists a  $T_i$  such that  $T_i \equiv T$ .*

*Proof.* Similarly to (6), we can also show that  $\bigcup_{i=1}^m T_i(I) = \pi \downarrow_U (RI(\text{state}(I)))$  for all instances  $I \in \text{CON}(D)$  and, thus, (1) is true. Condition (2) follows from (1) by the results of [SY].  $\square$

**5. Database schemes with full tuple-generating dependencies.** In this section, we consider a lossless database scheme  $([R_1, \dots, R_n], D)$  where  $D$  consists of only full tgds. We will show that there is a union of tableaux  $\bigcup_{i=1}^m T_i$  that computes  $\pi \downarrow_U (RI(r_1, \dots, r_n))$  if and only if  $D$  is equivalent to a join dependency. There are two interesting corollaries of this result. First, if there is a union of tableaux that computes  $\pi \downarrow_U (RI(r_1, \dots, r_n))$ , then for all  $X \subseteq U$ , there is a union of tableaux that computes  $\pi \downarrow_X (RI(r_1, \dots, r_n))$ . Second,  $D$  is bounded if and only if it is equivalent to a join dependency.

Note that since  $D$  has only tgds,  $\text{CON}(D)$  is the set of all instances (without nulls). In the following proofs, we will replace  $D$  with a single equivalent tgd  $d$  by composing the tableaux of the tgds of  $D$  (by Lemma 4.2, this composition results in a tgd equivalent to  $D$ ). The first theorem in this section shows that when  $D$  has only tgds, and there is a union of tableaux  $\bigcup_{i=1}^m T_i$  that computes  $\pi \downarrow_U (RI(r_1, \dots, r_n))$ , then  $T_1, \dots, T_m$  are not only implied by  $D$  (as shown in Lemma 4.1), but actually are equivalent to  $D$ .

**THEOREM 5.1.** *Suppose that  $([R_1, \dots, R_n], D)$  is a lossless database scheme, where  $D$  consists of only full tgds. If  $\bigcup_{i=1}^m T_i$  is a union of tableaux that computes  $\pi \downarrow_U (RI(r_1, \dots, r_n))$ , then  $T_1, \dots, T_m \models D$ .*

*Proof.* Since the database scheme is lossless,  $\pi \downarrow_U (RI(\text{state}(I))) = \text{CHASE}_{\{d\}}(I)$  for all instances  $I$  (without nulls). In particular, we can view the rows of  $d$  as an instance, and since  $\text{CHASE}_{\{d\}}(d)$  contains a row with only distinguished variables, so does  $\bigcup_{i=1}^m T_i(\text{state}(d))$  ( $\text{state}(d)$  is the state generated by the set of rows of  $d$ ). Thus,  $T_1, \dots, T_m \models d$ . The other direction follows from Lemma 4.1.  $\square$

We will now prove the main theorem of this section.

**THEOREM 5.2.** *Let  $([R_1, \dots, R_n], D)$  be a lossless database scheme, where  $D$  is a set of full tgds. The following are equivalent:*

- (1) *For all  $X \subseteq U$ , there is a union of tableaux that computes  $\pi \downarrow_X (RI(r_1, \dots, r_n))$ .*
- (2) *There is a union of tableaux that computes  $\pi \downarrow_U (RI(r_1, \dots, r_n))$ .*
- (3)  *$D$  is equivalent to a single join dependency.*

*Moreover, the above conditions are decidable; and when they hold there is an algorithm that constructs the join dependency to which  $D$  is equivalent, and an algorithm that constructs for all  $X \subseteq U$ , a union of tableaux that computes  $\pi \downarrow_X (RI(r_1, \dots, r_n))$ .*

*Proof.* Clearly, (1) implies (2).

Now suppose that (2) is true, and we will show that (3) is true as well. So, let  $\bigcup_{i=1}^m T_i$  be a union of tableaux that computes  $\pi \downarrow_U (RI(r_1, \dots, r_n))$ , and let  $T$  be obtained by taking the composition of the  $T_i$  and then minimizing it. By Lemma 4.3 and Corollary 3.1,  $T$  is a join dependency, and by Theorem 5.1 and Lemma 4.2,  $T$  is equivalent to  $D$ . Thus, (2) implies (3).

The proof that (3) implies (1) follows from [Sa4]. In particular, when  $D$  consists of a single join dependency, then for every  $X \subseteq U$ , we can construct in exponential time (in the size of the database scheme alone) a union of tableaux that computes

$\pi \downarrow_X (RI(r_1, \dots, r_n))$  [Sa4]. If  $D$  consists of the join dependency  $\bowtie R_i$ , then we can construct in polynomial time an expression that computes  $\pi \downarrow_X (RI(r_1, \dots, r_n))$  [Sa1], [Sa4]. The only operators in this expression are project, join, and union and, so, it can be converted into a union of tableaux in exponential time [SY], but there is really no need to do that in order to evaluate it efficiently.

We have shown that the three conditions are equivalent, and expressions can be constructed when  $D$  consists of a single join dependency. Now we will show how to test whether  $D$  is equivalent to a join dependency, and how to find this join dependency when it exists. Recall that  $d$  is obtained from  $D$  by composing the tgds of  $D$  in some arbitrary order. Let  $J(d)$  be the join dependency obtained from  $d$  by replacing all occurrences of repeated nondistinguished variables with new distinct nondistinguished variables. We claim that if  $j$  is a join dependency such that  $d \models j$ , then  $d \models J(d)$ . First, we show that  $j \models J(d)$ . Since  $j$  is a join dependency, a single application of the chase rule for  $j$  shows that  $j \models d$  (because  $j$  is idempotent). Therefore, there is a valuation  $h$  of  $j$  into  $d$  that maps each distinguished variable of  $j$  to a distinguished variable of  $d$ , since  $CHASE_{\{j\}}(d)$  contains the row consisting of all the distinguished variables (because  $j \models d$ ). Clearly,  $h$  is also a valuation of  $j$  into  $J(d)$  that maps distinguished variables to distinguished variables and, hence, shows that  $j \models J(d)$ .

By mapping each row of  $J(d)$  to its corresponding row in  $d$ , we get a valuation that maps distinguished variables to distinguished variables and, thus,  $J(d) \models d$ . By our assumption,  $d \models j$ . Thus, we have shown that  $j \models J(d) \models d \models j$  and, so,  $d$  and  $J(d)$  are equivalent, as claimed. Consequently, in order to test whether  $D$  is equivalent to a join dependency, we construct  $d$  and then  $J(d)$ , and test whether  $d \models J(d)$  (note that  $J(d) \models d$  is always true).  $\square$

The following example shows that even if a tgd is equivalent to a join dependency, it is not necessarily a join dependency.

*Example 5.1.* Let  $d$  be the following tgd (which is the same as tableau  $T$  of Example 2.3). Note that the tableau of  $d$  is minimal.

$a_1$	$a_2$	$a_3$	$a_4$
$a_1$	$a_2$	$b_1$	$b_2$
$e$	$a_2$	$a_3$	$b_3$
$e$	$a_2$	$b_4$	$a_4$

The tgd  $d$  is equivalent to  $J(d)$ , which is the following join dependency:

$a_1$	$a_2$	$a_3$	$a_4$
$a_1$	$a_2$	$b_1$	$b_2$
$e^2$	$a_2$	$a_3$	$b_3$
$e^3$	$a_2$	$b_4$	$a_4$

Recall that in Example 2.3, tableau  $J(d)$  was obtained after minimizing the composition of  $d$  with itself. Thus,  $d$  and  $J(d)$  are indeed equivalent.

We can also show that  $d \models J(d)$  by considering the computation of  $CHASE_{\{d\}}(J(d))$ . The computation starts with the rows of  $J(d)$ . At first, we can map the first row of  $d$  to the first row of  $J(d)$ , and the last two rows of  $d$  to the second row of  $J(d)$ , and that application generates the row

$$(A) \quad (a_1, a_2, a_3, b_3).$$



Next, we can map the first row of  $d$  to the first row of  $J(d)$ , and the last two rows of  $d$  to the third row of  $J(d)$ , and obtain the row

$$(B) \quad (a_1, a_2, b_4, a_4).$$

Now the first row of  $d$  can be mapped to the first row of  $J(d)$ , the second row to (A), and the third to (B), and the result is

$$(a_1, a_2, a_3, a_4)$$

which shows that  $d \equiv J(d)$ . Obviously,  $J(d) \equiv d$  and, thus, the two are equivalent.

As discussed in § 2.8, the following corollary is important in deductive databases, where tgds are viewed as Horn-clause rules:

**COROLLARY 5.1.** *Suppose that  $D$  is a set of full tgds. The following are equivalent and decidable.*

- (1)  $D$  is bounded, i.e., there is a  $k$ , such that for all instances  $I$ , the computation of  $CHASE_D(I)$  takes  $k$  or fewer iterations.
- (2) There is a union of tableaux  $E$  such that for all instances  $I$ ,  $E(I) = CHASE_D(I)$ .
- (3)  $D$  is equivalent to a single join dependency.

*Proof.* In order to prove the theorem, we consider the special database scheme  $([U], D)$ , i.e., this database scheme has a single relation scheme consisting of all the attributes. A database state for this database scheme is any instance  $I$ . The representative instance of  $I$  is just  $CHASE_D(I)$ , and that is also the restricted projection of the representative instance onto  $U$ .

Boundedness of  $D$  coincides with boundedness of the database scheme  $([U], D)$ , as defined in [MUV]. Thus, by the results of [MUV], condition (1) of Theorem 5.2 is equivalent to (1) of this corollary. Therefore, by Theorem 5.2, (1), (2) and (3) are equivalent and decidable.  $\square$

**6. Database schemes with full tgds and egds.** In this section we consider a lossless database scheme  $([R_1, \dots, R_n], D)$ , where  $D$  consists of full tgds and egds. We will show that if there is a union of tagged tableaux  $\bigcup_{i=1}^m T_i$  that computes  $\pi \downarrow_U (RI(r_1, \dots, r_n))$ , then  $D$  is equivalent to a join dependency and some egds (i.e.,  $D$  is equivalent to a join dependency over all consistent instances). We will also show that this necessary condition is decidable.

Let  $E$  be the set of all the egds in  $D$ . All the tgds in  $D$  can be replaced with a single equivalent tgd  $d$ , as done in the previous section. Thus,  $D$  is equivalent to  $\{d\} \cup E$ . The next theorem shows that  $\{d\} \cup E$  can be replaced with an equivalent set in which the only tgd is  $ECHASE_D(d)$ .

**THEOREM 6.1.** *Let  $B$  be a set of dependencies consisting of a tgd  $t$  and a set of egds  $F$ . There is a set of egds  $G$  (that can be constructed effectively) such that*

$$ECHASE_B(t), F, G \equiv t, F.$$

*Proof.* Let  $(w_1, \dots, w_n)$  be the rows of  $t$ .  $G$  consists of all egds  $(w_1, \dots, w_n)/a = b$ , where  $a$  and  $b$  are equated in  $CHASE_B(t)$ .  $\square$

Since the rows of  $ECHASE_D(d)$  form a consistent instance of  $D$ , we can show that  $D$  is equivalent to some egds and the tgds  $T_1, \dots, T_m$ .

**THEOREM 6.2.** *Suppose that  $([R_1, \dots, R_n], D)$  is a lossless database scheme, and there is a union of tagged tableaux  $\bigcup_{i=1}^m T_i$  that computes  $\pi \downarrow_U (RI(r_1, \dots, r_n))$ . Let  $E$  be the set of all egds in  $D$ , and  $d$  be the composition of all the tgds in  $D$ . The following are equivalent:*

- (1) There is a set of egds  $G_1$ , such that  $ECHASE_D(d), E, G_1 \equiv d, E$ .
- (2)  $T_1, \dots, T_m \equiv ECHASE_D(d)$ .
- (3)  $T_1, \dots, T_m, E, G_1 \equiv D$ .

*Proof.* Part (1) follows from Theorem 6.1. Part (2) follows from the fact that the rows of  $ECHASE_D(d)$  form a consistent instance, using the same argument as in the proof of Theorem 5.1. Part (3) follows from parts (1) and (2), and Lemma 4.1.  $\square$

We will now prove the main theorem of this section.

**THEOREM 6.3.** *Consider a lossless database scheme  $([R_1, \dots, R_n], D)$  with egds and full tgds. Suppose that  $\bigcup_{i=1}^m T_i$  is a union of tableaux that computes  $\pi \downarrow_U (RI(r_1, \dots, r_n))$ . Then  $D$  is equivalent to a set consisting of a single join dependency and some equality-generating dependencies. Moreover, this condition is decidable.*

*Proof.* Let  $T$  be the pairwise composition of the  $T_i$ , and recall that  $CON(D)$  is the set of all consistent instances of  $D$ . By Lemma 4.3,  $T \equiv_{CON(D)} T^2$  and, so, by Theorem 3.2,  $ECHASE_D(T)$  (after it is minimized) is a join dependency. By part (1) of Lemma 4.2 and part (3) of Theorem 6.2, there is a set of egds  $G_1$ , such that  $T, E, G_1 \models D$ . Therefore, by Theorem 6.1, there is a set of egds  $G_2$ , such that  $ECHASE_D(T), E, G_1, G_2 \models D$ .

We have shown that  $D$  is equivalent to a single join dependency and some egds when  $\pi \downarrow_U (RI(r_1, \dots, r_n))$  is computed by a union of tableaux. Next, we will show how to test whether  $D$  satisfies this necessary condition, i.e., whether  $D$  is equivalent to a single join dependency and some egds. Consider  $D = d \cup E$  and  $j \cup K$ , where  $j$  is a join dependency,  $d$  is a tgd, and  $E$  and  $K$  are sets of egds. By Theorem 6.1,  $d \cup E$  is equivalent to  $ECHASE_D(d) \cup H$ , where  $H$  is a set of some egds. We claim that if  $ECHASE_D(d), H \models j, K$ , then  $ECHASE_D(d), H \models J(ECHASE_D(d)), H$ , where  $J(ECHASE_D(d))$  is obtained from  $ECHASE_D(d)$  by replacing all occurrences of repeated nondistinguished variables with new distinct nondistinguished variables.

First, we show that  $j, K \models J(ECHASE_D(d)), H$ . Since no variables are equated when  $ECHASE_D(d)$  is chased with  $D$  and we have assumed that  $j, K \models ECHASE_D(d), H$ , it follows that  $j \models ECHASE_D(d)$ . Since  $j$  is a join dependency, a single application of the rule for  $j$  shows that  $j \models ECHASE_D(d)$  (because of the idempotency of  $j$ ). Therefore, there is a valuation  $h$  of  $j$  into  $ECHASE_D(d)$  that maps each distinguished variable of  $j$  to a distinguished variable of  $ECHASE_D(d)$ . Clearly,  $h$  is also a valuation of  $j$  into  $J(ECHASE_D(d))$  that maps distinguished variables to distinguished variables and, hence, shows that  $j \models J(ECHASE_D(d))$ . By our assumption,  $j, K \models H$  and, therefore,  $j, K \models J(ECHASE_D(d)), H$ .

By mapping each row of  $J(ECHASE_D(d))$  to its corresponding row in  $ECHASE_D(d)$ , we get a valuation that maps distinguished variables to distinguished variables and, therefore

$$J(ECHASE_D(d)) \models ECHASE_D(d)$$

and consequently

$$J(ECHASE_D(d)), H \models ECHASE_D(d), H.$$

By our assumption,

$$ECHASE_D(d), H \models j, K.$$

Thus, we have shown that

$$j, K \models J(ECHASE_D(d)), H \models ECHASE_D(d), H \models j, K$$

and, so,  $ECHASE_D(d) \cup H$  and  $J(ECHASE_D(d)) \cup H$  are equivalent, as claimed. Thus, in order to check whether  $D$  is equivalent to a join dependency and some egds, we construct  $J(ECHASE_D(d))$  and test whether it is implied by  $D$ .  $\square$

**7. Conclusion.** We have considered lossless database schemes with full implicational dependencies. If there is a union of tableaux for the restricted projection of the representative instance onto the set of all the attributes, then the dependencies of the database scheme are equivalent, over the set of all consistent instances, to a single join dependency. If there are no equality-generating dependencies, then the condition is also sufficient, and a restricted projection onto any set of attributes can be expressed by a union of tableaux.

The only known case in which there are equality-generating dependencies and expressions for restricted projections is in that of *independent* database schemes [GY], [Sa3] with functional dependencies and a single join dependency. In this case, a union of tableaux for any restricted projection (and even for tableau queries) can be constructed in the worst case in exponential time (only in the size of the database scheme), and in some cases (e.g., when there are only functional dependencies) in polynomial time [Sa4]. This result is a generalization of [AC], [IIK] who have considered independent database schemes with only functional dependencies.

Our results also characterize when a set of tgds (which can also be viewed as Horn-clause rules) is bounded. A set of tgds is bounded if and only if it is equivalent to a single join dependency. As shown in § 5, this condition is decidable.

**Acknowledgment.** The author thanks Moshe Vardi for helpful comments.

#### REFERENCES

- [ABU] A. V. AHO, C. BEERI AND J. D. ULLMAN, *The theory of joins in relational databases*, ACM Trans. Database Systems, 4 (1979), pp. 297-314.
- [ASU1] A. V. AHO, Y. SAGIV AND J. D. ULLMAN, *Equivalences among relational expressions*, this Journal, 8 (1979), pp. 218-246.
- [ASU2] ———, *Efficient optimization of a class of relational expressions*, ACM Trans. Database Systems, 4 (1979), pp. 435-454.
- [Arm] W. W. ARMSTRONG, *Dependency structures of database relationships*, Proc. IFIP 74, North-Holland, Amsterdam, 1974, pp. 580-583.
- [AC] P. ATZENI AND E. P. F. CHAN, *Efficient query answering in the representative instance approach*, Proc. Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Portland, OR, March 1985, pp. 181-188.
- [BB] C. BEERI AND P. A. BERNSTEIN, *Computational problems related to the design of normal form relational schemes*, ACM Trans. Database Systems, 4 (1979), pp. 30-59.
- [BMSU] C. BEERI, A. O. MENDELZON, Y. SAGIV AND J. D. ULLMAN, *Equivalence of relational database schemes*, this Journal, 10 (1981), pp. 352-370.
- [BV1] C. BEERI AND M. Y. VARDI, *Formal systems for tuple and equality generating dependencies*, this Journal, 13 (1984), pp. 76-98.
- [BV2] ———, *A proof procedure for data dependencies*, J. Assoc. Comput. Mach., 31 (1984), pp. 718-741.
- [CM] A. K. CHANDRA AND P. M. MERLIN, *Optimal implementation of conjunctive queries in relational data bases*, Proc. Ninth Annual ACM Symposium on Theory of Computing, May 1976, pp. 77-90.
- [Co] E. F. CODD, *A relational model for large shared data banks*, Comm. ACM, 13 (1970), pp. 377-387.
- [CK] S. S. COSMADAKIS AND P. C. KANELLAKIS, *Parallel evaluation of recursive rule queries*, Proc. Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Boston, MA, March 1986, pp. 280-292.
- [Fa] R. FAGIN, *Horn clauses and database dependencies*, J. Assoc. Comput. Mach., 29 (1982), pp. 952-983.
- [FMUY] R. FAGIN, D. MAIER, J. D. ULLMAN AND M. YANNAKAKIS, *Tools for template dependencies*, this Journal, 12 (1983), pp. 36-59.
- [GM] H. GALLAIRE AND J. MINKER, eds., *Logic and Databases*, Plenum, New York, 1978.
- [GY] M. GRAHAM AND M. YANNAKAKIS, *Independent database schemas*, J. Comput. System Sci., 28 (1984), pp. 121-141.

- [Ho] P. HONEYMAN, *Testing satisfaction of functional dependencies*, J. Assoc. Comput. Mach., 29 (1982), pp. 668–677.
- [Io] Y. E. IOANNIDIS, *A time bound on the materialization of some recursively defined views*, Proc. Internat. Conf. on Very Large Data Bases, Stockholm, Sweden, 1985.
- [IIK] M. ITO, M. IWASAKI AND T. KASAMI, *Some results on the representative instance in relational databases*, this Journal, 14 (1985), pp. 334–354.
- [Ma] D. MAIER, *The Theory of Relational Databases*, Computer Science Press, Rockville, MD, 1983.
- [MMS] D. MAIER, A. O. MENDELZON AND Y. SAGIV, *Testing implications of data dependencies*, ACM Trans. Database Systems, 4 (1979), pp. 455–469.
- [MUV] D. MAIER, J. D. ULLMAN AND M. Y. VARDI, *On the foundations of the universal relation model*, ACM Trans. Database Systems, 9 (1984), pp. 283–308.
- [Me] A. O. MENDELZON, *Database states and their tableaux*, ACM Trans. Database Systems, 9 (1984), pp. 264–282.
- [Na] J. NAUGHTON, *Data independent recursion in deductive databases*, Proc. Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Boston, MA, March 1986, pp. 267–279.
- [Sa1] Y. SAGIV, *Can we use the universal instance assumption without using nulls?*, Proc. ACM-SIGMOD Internat. Conf. on Management of Data, Ann Arbor, MI, April 1981, pp. 108–120.
- [Sa2] ———, *Quadratic algorithms for minimizing joins in restricted relational expressions*, this Journal, 12 (1983), pp. 316–328.
- [Sa3] ———, *A characterization of globally consistent databases and their correct access paths*, ACM Trans. Database Systems, 8 (1983), pp. 266–286.
- [Sa4] ———, *Evaluation of queries in independent database schemes*, manuscript submitted.
- [Sa5] ———, *Optimizing datalog programs*, Proc. Workshop on Foundations of Deductive Databases and Logic Programming, Washington, D.C., August 1986.
- [SY] Y. SAGIV AND M. YANNAKAKIS, *Equivalences among relational expressions with the union and difference operators*, J. Assoc. Comput. Mach., 27 (1980), pp. 633–655.
- [U1] J. D. ULLMAN, *Principles of Database Systems*, second ed., Computer Science Press, Rockville, MD, 1982.
- [Ya] M. YANNAKAKIS, *Algorithms for acyclic database schemes*, Proc. Seventh Internat. Conf. on Very Large Data Bases, September 1981, pp. 82–94.
- [YP] M. YANNAKAKIS AND C. H. PAPADIMITRIOU, *Algebraic dependencies*, J. Comput. Systems Sci., 25 (1982), pp. 2–41.

## ANALYSIS OF A HYBRID ALGORITHM FOR PACKING UNEQUAL BINS\*

D. K. FRIESEN† AND F. S. KUHL†

**Abstract.** We consider a bin-packing problem where the sizes of the bins are allowed to vary and where the goal is to maximize the number of pieces packed. This problem is NP-hard. We examine a new efficient approximation algorithm which is a hybrid of two algorithms reported earlier, First-Fit Decreasing (FFD) and Best-Two Fit (B2F). The hybrid is iterative: it attempts to pack smaller and smaller suffixes of its list of pieces until it succeeds in packing an entire suffix. During each attempt, the hybrid operates by partitioning the current list of pieces and packing one part by B2F and afterward packing the other part of its list by FFD. We prove that no instance of the problem exists where an optimal algorithm can pack more than  $\frac{4}{3}$  the number of pieces the hybrid can pack.

We also give a sequence of examples for which the ratio of the number of pieces packed optimally and by our algorithm increases asymptotically to  $\frac{4}{3}$ .

**Key words.** heuristic, approximation algorithm, bin packing, worst-case performance

**1. Introduction.** A one-dimensional bin-packing problem is, in general, a combinatorial optimization problem involving the partition of a set of objects into subsets. Specifically we are given: a set of *bins*  $\mathbf{B} = \{B_1, B_2, \dots, B_M\}$  and a list of *pieces*  $L = \{p_1, p_2, \dots, p_N\}$ . For each bin  $B_i$  and piece  $p_j$  we are also given a finite, positive size, denoted  $s(B_i)$  or  $s(p_j)$ . Two problems of interest are:

(BP) Given  $\mathbf{B}$  and  $L$  as above. Question: Does there exist an assignment of each  $p \in L$  to a bin of  $\mathbf{B}$  such that for each bin  $B'$  the set of pieces assigned to  $B'$  is  $P'$   $\sum_{p \in P'} s(p) \leq s(B')$ ?

(BPN) Given  $\mathbf{B}$  and  $L$  as above, and positive integer  $J$ . Question: Does there exist a subset  $L^*$  of  $L$  such that  $|L^*| \geq J$ , each  $p \in L^*$  is assigned to some bin of  $\mathbf{B}$ , and, if the set of pieces assigned to bin  $B'$  is  $P'$ , then  $\sum_{p \in P'} s(p) \leq s(B')$ ?

In problem BP we seek to pack the entire list  $L$  of pieces into the given bins. In BPN, we seek to pack a subset of  $L$  of specified size. As an optimization problem, BPN asks: What is the largest subset of  $L$  that can be packed into  $\mathbf{B}$ ? If all the bins of  $\mathbf{B}$  have the same size, BP may be stated as an optimization problem: What is the smallest number of bins needed to pack all of  $L$ ?

Bin-packing problems arise in a variety of storage-allocation and scheduling problems. Given a set of varying-length strings to be stored in fixed-length blocks, if the goal is to minimize the number of blocks needed to store all the strings, we have BP [7]. If the goal is to store a subset of the strings of greatest cardinality in a fixed number of blocks, we have BPN. The nonpreemptive scheduling of a system of independent tasks can also be viewed as a bin-packing problem, where the tasks are viewed as pieces and the bins as processors. If the processors are identical and the goal is to minimize the number needed to achieve a given makespan, we have BP. If the goal is to schedule the largest subset of tasks on a given number of processors, we have BPN. The goal of minimizing the makespan has been considered as a variant of BP [5].

\* Received by the editors September 15, 1986; accepted for publication (in revised form) March 10, 1987.

† The MITRE Corporation, Civil Systems Division, McLean, Virginia 22102-3481.

Bin packing is known to be NP-hard [6], so it is unlikely that any efficient algorithm exists for the solution of the general problem. Various approximation algorithms have been reported. Some of the first efforts at establishing guarantees of performance for approximation algorithms involved bin packing [6], and the present work follows a similar course.

For a given instance of BPN and approximation algorithm  $A$ , we let  $n_O$  be the number of pieces an optimal algorithm packs, and  $n_A$  be the number of pieces  $A$  packs. Define  $R_A = n_O/n_A$ . Evidently  $R_A \geq 1$  for all instances. As a measure of performance of  $A$ , we choose the “worst-case performance,” i.e., the supremum of  $R_A$  over all instances of  $BPN$ .

The algorithm analyzed here is a hybrid of First-Fit Decreasing (FFD) and Best-Two Fit (B2F). The latter was defined and its worst-case performance on BP was analyzed in [3]. We call our new algorithm a hybrid because a given input is partitioned and different parts are given to the FFD and B2F algorithms. This is in contrast to a compound algorithm, where both B2F and FFD are run on a given input and the better packing selected [4]. For the hybrid algorithm  $H$  to be presented here, we prove an asymptotically tight bound of  $\frac{4}{3}$  on its worst-case performance. This bound is lower than the lowest published bound for an approximation algorithm applied to BPN with bins of varying size.

In [2], BPN is considered with bins of equal size. The authors give bounds for two algorithms. The Smallest-Piece-First (SPF) approximation algorithm has  $n_{OPT}/n_{SPF} \leq 2 - 1/m$ , where  $m$  is the number of bins, and  $n_{OPT}$  and  $n_{SPF}$  are the numbers of pieces packed optimally and by SPF, respectively. And this bound is tight, in the sense that [2] exhibits examples which achieve this bound. They also give a tight bound for First-Fit Increasing (FFI) of  $\frac{4}{3}$ .

The Iterated FFD (FFD\*) is analyzed in [1]. The authors outline a proof that  $n_{OPT} \leq \frac{7}{6}n_{FFD^*} + 3$ . (The original proof in [9] takes over 100 pages.) They give examples for even numbers of bins that achieve  $n_{OPT} = \frac{8}{7}n_{FFD^*}$ . If the limit exists of the ratio  $n_{OPT}/n_{FFD^*}$  as the number of pieces packed increases, it therefore lies in  $[\frac{8}{7}, \frac{7}{6}]$ .

Langston [8] repeats the analysis of [1] and [2] with varying bin sizes. SPF has a tight bound of 2, regardless of the order the bins are sorted in. FFI also has a tight bound of 2. The iterated FFD, FFD\*, from [8], does best with the bins sorted in increasing size. It is shown that, if  $n_F$  is the number of pieces packed by FFD\* with bins sorted by increasing size,  $n_{OPT} \leq \frac{3}{2}n_F + 1$ , and gives an example where  $n_{OPT} = \frac{11}{8}n_F$ . Thus the limit of  $n_{OPT}/n_F$  lies in  $[\frac{11}{8}, \frac{3}{2}]$ .

## 2. Main result.

**2.1. The hybrid algorithm.** Our algorithm is defined by procedures HSTAR, H, B2F and FFD, given below. The input to HSTAR is a set of bins  $\{B_1, \dots, B_M\}$ , each bin  $B_i$  having its positive size  $s(B_i)$ . We assume the bins are sorted so that  $s(B_1) \leq s(B_2) \leq \dots \leq s(B_M)$ . We also give HSTAR a set of pieces  $\{p_1, \dots, p_N\}$  and, for each piece  $p_i$ , its positive size  $s(p_i)$ . We assume the pieces are sorted so that  $s(p_1) \geq s(p_2) \geq \dots \geq s(p_N)$ . Procedures H, B2F and FFD each return a Boolean value which indicates whether or not they succeeded in packing all the pieces passed to them. The only purpose of H is to partition its input into two lists. The larger pieces, defined as those  $p_i$  such that  $s(p_i) > s(p_f)/2$  ( $f$  is defined in the algorithm), are packed by B2F; the remainder (whose relative size may vary greatly) are packed by FFD. HSTAR is an iterative algorithm which will take successively smaller suffixes of  $\{p_1, \dots, p_N\}$  until it succeeds in packing an entire suffix.

```

proc HSTAR;
  f:= 1;
  while (H( $p_f, \dots, p_N$ ) = false) f:= f + 1;
end HSTAR;

proc H( $p_f, \dots, p_N$ );
  mark all bins as empty;
  let k be the largest i such that  $s(p_i) > s(p_f)/2$ ;
  if ( $k > f$ )
    then b2fok := B2F( $p_f, \dots, p_k$ );
    else b2fok := true;
  end if;
  if ( $k < N$ )
    then ffdok := FFD( $p_{k+1}, \dots, p_N$ );
    else ffdok := true;
  end if;
  return (b2fok and ffdok);
end H;

proc B2F( $p_f, \dots, p_k$ );
  mark  $p_f, \dots, p_k$  as available (unpacked);
  for j:= 1 to M do
    /* pack bin  $B_j$  */
    for i:= f to k do
      if ( $p_i$  is available and fits in unused space) then
        /* pack  $p_i$  */
        mark  $p_i$  unavailable;
        decrease space in  $B_j$  by  $s(p_i)$ ;
      end if;
    end for;
    /* can we replace top piece with two? */
    if (some piece has been packed in  $B_j$ , and the
        two smallest available pieces will fit in  $B_j$ 
        after removing the top piece) then
      let  $p_t$  be the top piece currently packed in  $B_j$ ;
      let  $p_a$  be the smallest available piece;
      let empty be the space in  $B_j$  left after removing  $p_t$ ;
      let  $p_b$  be the largest available piece such that
         $empty \geq s(p_a) + s(p_b)$ ;
      let  $p_c$  be the largest available piece such that
         $empty \geq s(p_c) + s(p_b)$ ;
      /* then  $p_b$  and  $p_c$  replace  $p_t$  */
      mark  $p_t$  available and  $p_b$  and  $p_c$  unavailable;
      adjust total size of pieces packed in  $B_j$ ;
    end if;
  end for;
  if (no  $p_i$  in  $p_f, \dots, p_k$  is available)
    then return (true);
    else return (false);
  end if;
end B2F;

```

```

proc FFD( $p_{k+1}, \dots, p_N$ );
  for  $i := k + 1$  to  $N$  do
    /* pack piece  $p_i$  */
     $j := 1$ ;
    while ( $j \leq M$  and  $p_i$  will not fit in  $B_j$ )  $j := j + 1$ ;
    if ( $j < M$  or  $p_i$  fits in  $B_M$ )
      then assign  $p_i$  to  $B_j$ ;
      else return (false);
    end if;
  end for;
  return (true);
end FFD;

```

The chief reason for considering this hybrid algorithm, instead of simply an iterative version of B2F, is that the analysis of the hybrid appears to be much easier than analysis of B2F operating on pieces of unrestricted size. The worst-case example for our hybrid, presented below, involves only the operation of B2F, so the worst-case bound for B2F\* on varying bins cannot be better than  $\frac{4}{3}$ .

Expected performance is another matter. We performed computational experiments on HSTAR, B2F\* and FFD\* to estimate their expected behavior. For each instance, bin sizes were made to increase linearly and piece sizes were generated pseudorandomly from a uniform distribution and then sorted. For each instance we computed  $R_A = n_O/n_A$ , estimating  $n_O$  by assuming that the optimal algorithm could pack perfectly.  $R_A$  for each algorithm and all instances averaged within a few percent of 1, supporting the notion that worst-case behavior is rare. In general, B2F\* and HSTAR performed significantly better than FFD\*. For some instances, B2F\* outperformed HSTAR; for others, the reverse was true.

Several remarks on the design of H are worthwhile. Firstly, because every piece  $p_i$  that B2F considers has  $s(p_i) > s(p_f)/2$ , the total size of any two pieces B2F considers is strictly larger than that of any one piece. This affects the structure of B2F packings in ways discussed below. Secondly, we note that the method of replacement in B2F is used in the analysis. Finally, we note that the consecutive search employed by HSTAR on its list avoids the possibility of anomalies. But our main theorem as given would still apply if HSTAR used a nonconsecutive, e.g., binary, search.

Let us consider the time complexity of HSTAR. FFD runs in time  $O(MN)$ , and B2F in  $O(5MN) = O(MN)$ . H adds nothing to the order of the time complexity, so HSTAR as given is  $O(MN^2)$ . One must add time for the sorting of bins and pieces that we assume when HSTAR is invoked.

We denote the number of pieces HSTAR succeeds in packing from  $L$  into  $\mathbf{B}$  by  $n_{H^*}(L, \mathbf{B})$ . We denote the optimal algorithm by OPT and the number of pieces it packs of  $L$  into  $\mathbf{B}$  by  $n_O(L, \mathbf{B})$ . From the definition of the algorithm, HSTAR does not succeed until it packs all the pieces from some  $p_i$  to  $p_N$ . In view of the ordering of  $L$ , it is evident that when HSTAR succeeds it packs a suffix  $\{p_i, p_{i+1}, \dots, p_N\}$  of  $L$ , for some  $i$ . The measure of worst-case performance we use is

$$R_{H^*} = \sup_{\text{all}(L, \mathbf{B})} n_O(L, \mathbf{B}) / n_{H^*}(L, \mathbf{B}).$$

We shall show that  $R_{H^*} \leq \frac{4}{3}$ .

Before that, we present a family of instances of  $L$  and  $\mathbf{B}$  such that  $n_O(L, \mathbf{B}) / n_{H^*}(L, \mathbf{B}) \rightarrow \frac{4}{3}$ . For a positive integer  $k$ , an instance of the family is as follows.  $\mathbf{B}$  contains 1 bin of size  $2 - 2e$ , 2 bins of size  $2 - e$ , 4 bins of size  $2 - e/2, \dots$ , and  $2^{k-1}$



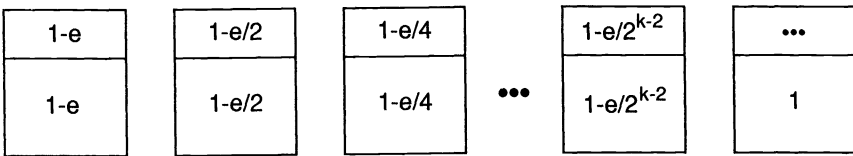
bins of size  $2 - e/2^{k-2}$ , for a total of  $2^k - 1$  bins.  $L$  contains  $2^k - 1$  pieces of size 1, 2 pieces of size  $1 - e$ , 4 pieces of size  $1 - e/2, \dots$ , and  $2^{k-1}$  pieces of size  $1 - e/2^{k-2}$ , for a total of  $2(2^k - 1) - 1$  pieces. The packing by OPT and HSTAR of such an instance is given in Fig. 1.

Define  $n_H(L, \mathbf{B})$  to be the number of pieces H (not HSTAR) will pack out of  $L$  into  $\mathbf{B}$ , if the entire list  $L$  is presented to H. Suppose an  $L$  and  $\mathbf{B}$  such that OPT can pack all of  $L$  into  $\mathbf{B}$ . If H always succeeds in packing suffixes of  $L$  that have  $\leq \frac{3}{4}$  the pieces of  $L$ , then HSTAR will always halt having packed a suffix containing  $\geq \frac{3}{4}$  the pieces of  $L$ , and  $R_{H^*} \leq \frac{4}{3}$ . Our main result may be stated as follows.

**THEOREM 2.1.1.** *Given any  $L$  and  $\mathbf{B}$  for which  $n_O(L, \mathbf{B}) = N$  (i.e., OPT packs all pieces of  $L$ ), H will succeed in packing into  $\mathbf{B}$  every suffix of  $L \{p_i, p_{i+1}, \dots, p_N\}$  for which  $n_O(L, \mathbf{B}) \leq \frac{4}{3}n_H(L, \mathbf{B})$ , i.e., for which  $N - i + 1 \leq 3N/4$ .*

The proof of this theorem will occupy §§ 2.2 and 2.3, and is organized as follows. In § 2.2 we assume the existence of an  $L$  and  $\mathbf{B}$  that constitute a counterexample which is minimal in a sense to be defined below, and which fails in the FFD portion of H. We derive a contradiction showing that in a minimal counterexample only the B2F portion of H is actually used. In § 2.3, we examine the behavior of B2F on pieces meeting the restriction we arrive at in § 2.2, and we show that under that restriction the packing of B2F never violates the  $\frac{4}{3}$  bound of Theorem 2.1.1. We conclude that no minimal counterexample, and hence no counterexample whatever, exists to Theorem 2.1.1.

B2F:



OPT:

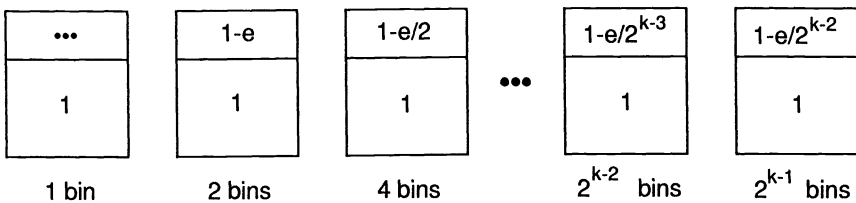


FIG. 1

**2.2. The minimal counterexample.** A counterexample to Theorem 2.1.1 is some  $L = \{p_1, p_2, \dots, p_N\}$ , suffix  $L_f = \{p_f, p_{f+1}, \dots, p_N\}$  and  $\mathbf{B} = \{B_1, B_2, \dots, B_M\}$  such that  $n_O(L, \mathbf{B}) = N$ ,  $N - f + 1 \leq 3N/4$ , and H will fail to pack into  $\mathbf{B}$  one or more pieces out of  $L_f$ . If such a counterexample exists, we construct from it a counterexample that is minimal in the number of pieces retained from  $L$ . Before doing so, we must discuss properties common to all H packings.

We extend the definition of the size of a piece or bin to sets. If  $P$  is a set of pieces, then  $s(P) = \sum_{p \in P} s(p)$ , and similarly for sets of bins.

Let us classify the pieces packed by B2F. Suppose that when B2F packs bin  $B_i$  it has available the ordered list of pieces  $L' = \{q_1, q_2, q_3, \dots\}$ , a subset of  $L_f$ . If B2F is

able to pack the first  $k$  pieces of  $L'$ , say  $q_1, q_2, \dots, q_k$ , in  $B_i$ , then we say pieces  $q_1, q_2, \dots, q_k$  have *type X* or, are *X-pieces*, more specifically,  *$X_k$ -pieces*. Any piece  $q_{k+j}$ ,  $j \geq 2$ , packed after omitting an available piece, has *type Y* (or *type  $Y_k$* ). If B2F replaces the top piece in  $B_i$  with two smaller pieces, the replacements have *type F* (or *type  $F_k$* ). We can also similarly classify small pieces as types *X* or *Y* with respect to the way FFD packs them. (Type *F* and *Y* pieces are sometimes called *fallback* pieces.)

We say that a bin (rather than a piece)  $B_i$  has *type  $X_k$*  (or, is an  $X_k$  bin) if B2F packs  $k$  *X* pieces in  $B_i$  and no other pieces, and B2F does not replace the top *X* piece with two *F* pieces. Bin  $B_i$  has *type  $F_k$*  if B2F packs  $k$  *X* pieces and then replaces the top piece with two *F* pieces. Bin  $B_i$  has *type  $Y_k$*  if B2F packs  $k$  *X* pieces and a *Y* piece.

LEMMA 2.2.1. *Given a list of pieces  $L$  and set of bins  $\mathbf{B}$  and the H packing of  $L$  into  $\mathbf{B}$ . Let  $B'_i$  be the set of pieces H packs into bin  $B_i$ . Suppose we form  $L'$  and set  $\mathbf{B}'$  of bins by either of the following methods.*

- (A) *Form  $L'$  by deleting from  $L$  all pieces in  $B'_i$ . Form  $\mathbf{B}'$  by deleting  $B_i$  from  $\mathbf{B}$ .*
- (B) *Let  $p \in L$  be a piece not packed into any bin of  $\mathbf{B}$ . Form  $L'$  by deleting  $p$  from  $L$ . Let  $\mathbf{B}' = \mathbf{B}$ .*

*Then in either case, any bin in the new packing (of  $L'$  into  $\mathbf{B}'$ ) will be packed with the same set of pieces it was packed with in the old (of  $L$  into  $\mathbf{B}$ ).*

*Proof.* We begin with the first construction.

Consider first bin  $B_j$ ,  $j < i$ . If all (large) pieces originally packed by B2F in  $B_j$  had type *X*, then no  $p \in B'_i$  was examined in packing  $B_j$  and the new B2F packing of  $B_j$  will be the same. If B2F packed  $B_j$  with one or more *Y*-pieces and some  $p \in B'_i$  was examined during the packing of  $B_j$ ,  $p$  evidently would not fit in  $B_j$  and its absence from  $L'$  affects nothing. These remarks apply also to small pieces packed in  $B_j$ .

Now suppose that in the original packing of  $B_j$  some  $p \in B'_i$  had been the top piece and was replaced by B2F with two *F* pieces. We consider the moment in the packing of  $L'$  (the new packing) when B2F would pack  $p$  in  $B_j$  if it were available. Let the two *F* pieces in the original packing be  $f_1$  and  $f_2$ , and assume  $s(f_1) \geq s(f_2)$ . If in  $L'$  there is a piece  $p'$ ,  $s(p) \geq s(p') > s(f_1)$  and no other piece of  $L'$  will fit in the space with  $p'$ , then  $p'$  will be packed by B2F as type *X* (or *Y*) and will then be replaced by  $f_1$  and  $f_2$ . Now suppose that  $p'$  is packed in the space and that  $L'$  contains piece  $y$ ,  $s(p') \geq s(y) > s(f_1)$ , which fits with  $p'$ . Then  $p'$  would have fit with the smallest piece in  $L$  and would have been used in the original replacement instead of  $f_1$ . Thus no such  $y$  exists, and  $f_1$  and  $f_2$  will always replace the top piece. Therefore the set of pieces packed from  $L'$  into  $B_j$  will be the same as the set packed from  $L$ .

We note that with the foregoing, the set of pieces available from  $L'$  to pack  $B_{i+1}$  for both B2F and FFD will be the same as was available from  $L$ . Thus the packing from  $L'$  of any  $B_j$ ,  $j > i$ , will not differ from the original.

Now let us consider the second construction. Suppose  $p \in L$  is not packed by H at all. If we form  $L'$  by deleting  $p$  from  $L$  then H will pack into each  $B_j$  of  $\mathbf{B}$  the same set of pieces it packed from  $L$ . To see this, we consider the packing of  $L$  into  $B_j$ . If  $p$  was never packed into  $B_j$ , its absence affects nothing. If  $p$  was packed into  $B_j$  as the top piece and then replaced, the absence of  $p$  does not affect the replacement, as we showed above.  $\square$

Given a counterexample, we now form a minimal counterexample. We form list  $L'$  by deleting from  $L$  all but the largest of the pieces H failed to pack out of  $L_f$ . We form  $\mathbf{B}'$  by deleting from  $\mathbf{B}$  any bin  $B_i$  in which neither OPT nor H packs any piece of  $L$ . Now OPT can pack  $L'$  into  $\mathbf{B}'$  and, by Lemma 2.2.1, H cannot pack  $L' \cap L_f$  into  $\mathbf{B}'$ , since one unpacked piece remains. The one piece in  $L'$  which H fails to pack when it starts at  $p_f$  we shall call  $p_l$ . If  $s(p_l) \leq s(p_f)/2$ , the construction of the FFD part of

H ensures that  $p_l$  is the last piece H examines. If  $p_l$  is the last piece that H examines, define  $L_1 = L' \cap \{p_1, p_2, \dots, p_l\}$  and  $L_2 = L' \cap \{p_f, p_{f+1}, \dots, p_l\}$ , and we have  $n_O(L_1, \mathbf{B}') \cong \frac{4}{3}[n_H(L_2, \mathbf{B}') + 1]$ . (Note that we had  $n_O(L, \mathbf{B}) \cong \frac{4}{3}[n_H(L_f, \mathbf{B}) + 1]$  and that in forming  $L_1$  and  $L_2$  we have deleted an equal number of pieces from each.) If  $p_l$  is not the last piece H examines (which could happen if  $s(p_l) > s(p_f)/2$  and B2F fails to pack  $p_l$ ), we then define  $L_1 = L' \cap \{p_1, \dots, p_N\}$  and  $L_2 = L' \cap \{p_f, \dots, p_N\}$ . Now  $\mathbf{B}'$ ,  $L_1$  and  $L_2$  provide us with our minimal counterexample.

We wish to compare the H packing of  $L_2$  to any optimal packing of  $L_1$ . The goal of this section is to show that if  $p_l$  is the last piece H examines, then  $s(p_l) > s(p_f)/2$  and thus that FFD will never operate on a minimal counterexample. For the rest of this section  $n_O$  shall mean  $n_O(L_1, \mathbf{B}')$  and  $n_H$  shall mean  $n_H(L_2, \mathbf{B}')$ , unless otherwise qualified.

We normalize the sizes of pieces and bins so that  $s(p_f) = 1$ , and we assume that  $s(p_1) = s(p_2) = \dots = s(p_{f-1}) = 1$ . (If there is an optimal packing of  $L_1$  including  $p_1, p_2, \dots, p_{f-1}$  and they are larger than 1, the same packing is valid if the sizes of  $p_1, \dots, p_{f-1}$  are reduced to 1.) The pieces  $p_1, p_2, \dots, p_f$  are the largest in  $L_1$ . H packs at least one piece ( $p_f$ ) of size 1, and the pieces, aside from  $p_l$ , packed by OPT but not by H, ( $p_1, \dots, p_{f-1}$ ), have size 1. The bins are ordered in increasing size, and there is at least one bin as large as 1, since H can pack additional pieces of size 1. The B2F part of H will attempt to pack all the pieces of  $L_2$  with size  $> \frac{1}{2}$ , while the FFD part will attempt to pack all the pieces with size  $\leq \frac{1}{2}$ . We say  $p_i$  is *large* when  $s(p_i) > \frac{1}{2}$ ; if  $p_i$  is *small* we mean  $s(p_i) \leq \frac{1}{2}$ .

We now give as lemmas several elementary properties of a minimal counterexample. Note that Lemmas 2.2.2-2.2.5 do not depend on  $p_l$  being the last piece in  $L_2$ .

LEMMA 2.2.2. *The H packing of a minimal counterexample has no empty bins.*

*Proof.* If H packs no pieces in  $B_i$ , then  $s(B_i) < 1$ , since every piece  $p$  available to pack  $B_i$  has  $s(p) \leq 1$  and none would fit. Suppose OPT packs  $B_i$  with pieces  $q_1, q_2, \dots$ . Each  $q_i$  must be packed in some  $B_k$ ,  $k < i$ , or else the  $q_i$  would have been available to be packed in  $B_i$ . No  $q_i$  can be packed by B2F as an  $F$ -piece because there can be no  $F$  bins of size  $< 1$ . Since each  $q_i$  is either a small piece or is packed by B2F as type  $X$  or  $Y$ , we can delete  $q_i$  from  $L_1$  and  $L_2$  and form a smaller counterexample, contradicting minimality.  $\square$

LEMMA 2.2.3. *The optimal packing of a minimal counterexample has no empty bins.*

*Proof.* If  $B_i$  is empty in the optimal packing, form  $\mathbf{B}^*$  by deleting  $B_i$  from  $\mathbf{B}'$  and form  $L'_1$  and  $L'_2$  by deleting the pieces H packs into  $B_i$  from  $L_1$  and  $L_2$ , respectively. By Lemma 2.2.1,  $L'_1$ ,  $L'_2$  and  $\mathbf{B}^*$  are a smaller counterexample, thus contradicting minimality.  $\square$

LEMMA 2.2.4. *If in the optimal packing of a minimal counterexample,  $B_i$  contains only one piece  $p_j$ , then  $p_j$  is packed by H (if H packs  $p_j$  at all) in  $B_k$ ,  $k > i$ , and, for every piece  $p$  packed by H in  $B_i$ ,  $s(p) < s(p_j)$ .*

*Proof.* Suppose  $p_j$  is packed in  $B_k$ ,  $k \leq i$ . We then contradict minimality by forming a new counterexample as follows. Delete the pieces H packs in  $B_k$  from  $L_1$  and  $L_2$ . In the optimal packing, move whatever remains of the former contents of  $B_k$  to the now empty  $B_i$  (note that  $s(B_k) \leq s(B_i)$ ). Delete  $B_k$  from  $\mathbf{B}'$ .

Now we show that the H packing of  $B_i$  contains no pieces as large as  $p_j$ . If  $p_j$  is packed by H in  $B_k$ ,  $k > i$ , and H packs a piece  $p_m$  in  $B_i$  with  $s(p_m) \geq s(p_j)$ , we contradict minimality as follows. Delete any pieces H packs in  $B_i$  (notably  $p_m$ ) from  $L_1$  and  $L_2$ . In the optimal packing, move  $p_j$  (if it remains) to the position formerly occupied by  $p_m$ . Delete  $B_i$  from  $\mathbf{B}'$ . Thus if OPT packs  $p_j$  by itself, H cannot pack  $p_j$  in  $B_i$ , nor can H pack any  $p_m$  in  $B_i$  with  $s(p_m) \geq s(p_j)$ .  $\square$

To establish  $s(p_i) > \frac{1}{2}$ , we need another fact which we give as a lemma.

LEMMA 2.2.5. *If OPT packs but one piece  $p_j$  of a minimal counterexample in bin  $B_i$ , then  $p_j$  is a large piece, and B2F packs  $B_i$  as an  $F_1$  bin.*

*Proof.* Suppose  $p_j$  is a small piece. If H packed  $p_j$ , then, by Lemma 2.2.4, H packed  $p_j$  in  $B_k$ ,  $k > i$ , and H packed no large pieces in  $B_i$ . But  $p_j$  was available to FFD to pack  $B_i$  and  $p_j$  would have fit. If FFD packed  $p_j$  later, it packed some  $p_m$  in  $B_i$  with  $s(p_m) \cong s(p_j)$ , in violation of Lemma 2.2.4. If H did not pack  $p_j$ , then it packed some  $p_m$  with  $s(p_m) \cong s(p_j)$  in  $B_i$ , violating Lemma 2.2.4. Therefore  $p_j$  is not a small piece.

Now suppose  $p_j$  is a large piece. By Lemma 2.2.4, H must pack  $p_j$  (or  $p_f$  if  $j < f$ ); recall that then  $s(p_f) = s(p_j) = 1$  in  $B_k$ ,  $k > i$ . Now B2F had room in  $B_i$  to pack  $p_j$  ( $p_f$ ) and  $p_j$  ( $p_f$ ) was available to it. If B2F did not pack  $B_i$  as an  $F_1$  bin, it packed some piece  $p_m$  of type  $X$  or  $Y$ ,  $s(p_m) \cong s(p_j)$ , in  $B_i$  in violation of Lemma 2.2.4. Thus B2F packed  $B_i$  as an  $F_1$ .  $\square$

Recall that  $M$  is the number of bins in a minimal counterexample. Let  $F$  be the set of bins packed by OPT with only one piece. (By Lemma 2.2.5, B2F packs every bin of  $F$  as type  $F_1$ .) Let  $f = |F|$ . Recall that  $L_1$  and  $L_2$  are the sets of pieces packed by OPT and H, respectively. Let  $L_{OF}$  and  $L_{HF}$  be the sets of pieces packed by OPT and H, respectively, in bins of  $F$ . Let  $L_{OR}$  and  $L_{HR}$  be the sets of pieces packed by OPT and H in bins not in  $F$ .

LEMMA 2.2.6. *If  $p_i$  is the last piece in  $L_2$ ,  $s(p_i) > \frac{1}{2}$ .*

*Proof.* Because we consider a counterexample,  $n_O > \frac{4}{3}$  or  $n_O - n_H > n_O/4$ . But  $n_O \cong 2M - f$  because, by Lemma 2.2.5, OPT packs every bin outside  $F$  with at least 2 pieces, and every bin in  $F$  with 1 piece. Therefore  $n_O - n_H > \frac{1}{4}(2M - f)$ . But OPT packs at most 1 piece that H does not which has size  $< 1$ ; all other additional pieces have size 1 by assumption. Thus

$$(1) \quad s(L_1) - s(L_2) > \frac{1}{4}(2M - f).$$

Now  $s(L_1) = s(L_{OF}) + s(L_{OR})$  and  $s(L_2) = s(L_{HF}) + s(L_{HR})$  and

$$(2) \quad s(L_{OF}) - s(L_{HF}) + s(L_{OR}) - s(L_{HR}) \cong \frac{1}{4}(2M - f).$$

But  $s(L_{OF}) - s(L_{HF}) < 0$ . To see this, let  $B_i$  be any bin of  $F$ . Recall that B2F packs  $B_i$  as a type  $F_1$  (with two large pieces), so that the total size of pieces packed in  $B_i$  by H is  $> 1$ . By definition of  $F$ , OPT packs but one piece in  $B_i$ , with size  $\leq 1$ . Summing over all bins  $B_i$ ,  $s(L_{OF}) - s(L_{HF}) < 0$ . Substituting in (2) yields

$$(3) \quad s(L_{OR}) - s(L_{HR}) > \frac{1}{4}(2M - f).$$

There are  $M - f$  bins outside  $F$ , so among those bins the average empty space in the H packing must be

$$> \frac{1}{4} \left[ \frac{2M - f}{M - f} \right] > \frac{1}{4} \left[ \frac{2M - 2f}{M - f} \right] = \frac{1}{2}.$$

There must be some bin  $B_i$  outside  $F$  whose H packing has empty space  $> \frac{1}{2}$ . Since  $p_i$  would not fit in  $B_i$ ,  $s(p_i) > \frac{1}{2}$ .  $\square$

**2.3. B2F operating on large pieces.** We consider in this section the behavior only of the B2F part of H on a minimal counterexample.

We assume we are given a minimal counterexample consisting of bins  $\mathbf{B} = \{B_1, \dots, B_M\}$ , sorted so that  $s(B_1) \cong s(B_2) \cong \dots \cong s(B_M)$ , and pieces  $L = \{p_1, \dots, p_N\}$  sorted so that  $s(p_1) \cong s(p_2) \cong \dots \cong s(p_N)$ . Further, OPT packs all of  $L$  into  $\mathbf{B}$ , so that  $n_O(L, \mathbf{B}) = N$ , and there is a suffix of  $L$ ,  $L_1 = \{p_f, p_{f+1}, \dots, p_N\}$  such that B2F packs

all but one piece of  $L_1$  into  $\mathbf{B}$ . Therefore  $n_H(L_1, \mathbf{B}) = N - f$ , and because we consider a counterexample,  $3N/4 \geq N - f + 1$ . In the previous section we assumed that the one piece  $H$  failed to pack was the last piece of its list; here we make no such assumption, as it is possible for B2F to fail to pack a piece from the middle.

As before, we normalize the size of pieces and bins so that  $s(p_f) = 1$ . By assumption,  $s(p_N) > \frac{1}{2}$ . We also assume that  $s(B_1) > 1$ : each bin  $B_i$  with  $s(B_i) \leq 1$  must be packed by OPT with 2 pieces. Since all pieces have size  $> \frac{1}{2}$ , there can be no  $B_i$  with  $s(B_i) \leq 1$ .

Throughout this section we shall for notational convenience let  $n_O = n_O(L, \mathbf{B})$  and  $n_H = n_H(L_1, \mathbf{B})$  (it being understood that B2F is the only part of  $H$  actually executed).

We retain the previous definitions of bin and piece types. Note that B2F cannot, with the size pieces we consider here, omit a piece and then pack two more pieces further down the list: the two pieces would be larger than the omitted piece. Similarly it is not possible that B2F would pack the first  $k$  pieces of  $L_1$ , pack a noncontiguous ( $Y$ ) piece  $p_j$  and then replace  $p_j$  with two smaller pieces. Because we assume that  $s(B_1) > 1$ ,  $B_1$  is packed as type  $X_i, F_i$ , or  $Y_i$ ,  $i \geq 1$ . Because  $B_1$  is the smallest bin and  $p_f$  will necessarily fit in  $B_1$ , B2F is always able to pack at least the first piece available to it when it begins to pack each bin.

After the first appearance of a bin of type  $X_k, F_k$  or  $Y_k$ , there can be no bins of type  $X_{k-1}, F_{k-1}$ , or  $Y_{k-1}$ . To see this, suppose bin  $B_i$  has type  $X_k, F_k$  or  $Y_k$ . There was room to pack the first  $k$  pieces available to B2F at the time. When B2F packs  $B_j$ ,  $j > i$ ,  $s(B_j) \geq s(B_i)$ , and the first  $k$  pieces available to pack  $B_j$  are no larger than those that were available to pack  $B_i$ . So  $B_j$  will have type  $X_r, F_r$  or  $Y_r$  with  $r \geq k$ .

Let  $B_i$  be a bin of a B2F packing and let  $P$  be the set of pieces B2F packs in  $B_i$ . If  $s(P) < \frac{3}{4}s(B_i)$ , we say  $B_i$  is *bad*.

In the following definitions,  $P$  and  $Q$  are regions and  $i$  is a positive number. Define:

$x_{PQ}^i$  to be the number of type  $X$  pieces packed by B2F in region  $P$  but packed by OPT in  $Q$  in a bin with  $i$  pieces;

$f_{PQ}^i$  the number of type  $Y$  or  $F$  pieces packed by B2F in  $P$  but by OPT in  $Q$  in a bin with  $i$  pieces;

$f_{PQ}$  the same as  $f_{PQ}^i$  but without regard to how many pieces are in the optimal packing:  $f_{PQ} = \sum_i f_{PQ}^i$ ;

$q_{PQ}^i$  the number of pieces of any type packed by B2F in  $P$  but by OPT in  $Q$  in a bin with  $i$  pieces;

$M_P$  the number of bins of region  $P$ ;

$n_{OP}$  the number of pieces packed by OPT in region  $P$ ;

$n_{HP}$  the number of pieces packed by B2F in region  $P$ .

The symbol  $x_{(r < D)D}^i$  denotes the number of  $X$ -pieces packed by B2F in any region before  $D$  (in this case  $X, F$  or  $Y$ ), but packed by OPT in  $D$ . Similar symbols are defined analogously. When we consider the number of pieces packed in a particular bin  $B$ ,  $n_{HB}$  will be the number of pieces  $H$  packs in  $B$ ;  $n_{OB}$  the number of pieces OPT packs in  $B$ ; and for a region  $P$ ,  $x_{PB}^i, f_{PB}^i, x_{PB}$ , and  $f_{BP}$  are defined analogously.

LEMMA 2.3.1. *Let  $X$  be the region through the last  $X_1$  bin. Then*

$$n_{OX} - n_{HX} \leq \frac{1}{3}n_H - \frac{2}{3}f_{(X, r > X)}.$$

*Proof.* Let  $B_X$  be the last bin of region  $X$ . Since no second piece would fit in  $B_X$ ,  $s(B_X) < 2$ . Thus, because each piece has size  $> \frac{1}{2}$ , OPT can pack no more than three pieces in  $B_X$ . Further, if OPT packs three pieces in any bin  $B_i$  of region  $X$ , all must be  $F$  pieces packed by  $H$  in a bin before  $B_X$ . To see this, suppose OPT packed 3 pieces  $a, b$  and  $c$  in bin  $B_i$ . Let  $p_X$  be the piece B2F packed in  $B_X$ . Then  $s(p_X) + s(c) < s(a) + s(b) + s(c) \leq s(B_i) \leq s(B_X)$ , so  $c$  would have fit with  $p_X$  in  $B_X$ . But  $p_X$  was

packed alone, so  $c$  must not have been available when B2F packed  $B_X$ . If  $c$  were type  $X_1$  it would be larger than  $p_X$  and any other remaining pieces, so another piece would have fit with  $p_X$ . Thus  $c$  was packed as type  $F$  before  $B_X$ . This argument applies to  $a$  and  $b$  as well. Thus

$$(1.1) \quad n_O - 2M_X \leq \frac{1}{3}f_{XX}^3$$

and

$$(1.2) \quad 2M_X - n_H \leq x_X$$

because any piece packed by H in a bin by itself must be a single  $X_1$  piece. Adding (1.1) and (1.2) yields

$$(1.3) \quad n_O - n_H \leq \frac{1}{3}f_{XX}^3 + x_X.$$

Suppose OPT packs two pieces in a bin of region  $X$ . If both were available when H packed  $B_X$ , or if one were available and the other were at least as large as the last  $X_1$  piece packed, then  $B_X$  would have been type  $Y_1$  or  $F_1$ . Hence at least one of those pieces must be a type  $F$  piece packed by H before  $B_X$ . Thus

$$(1.4) \quad n_O - M_X \leq \frac{2}{3}f_{XX}^3 + f_{XX}^2.$$

Also, because any bin packed by H with a fallback piece is a type  $Y_1$  or  $F_1$  bin and thus has two pieces,

$$(1.5) \quad n_H - M_X \geq \frac{1}{2}f_X.$$

Subtracting (1.5) from (1.4) yields

$$(1.6) \quad n_O - n_H \leq \frac{2}{3}f_{XX}^3 + f_{XX}^2 - \frac{1}{2}f_X.$$

Multiplying (1.3) by  $\frac{1}{3}$  and (1.6) by  $\frac{2}{3}$  and adding yields

$$\begin{aligned} n_O - n_H &\leq \left(\frac{1}{9} + \frac{4}{9}\right)f_{XX}^3 + \frac{2}{3}f_{XX}^2 + \frac{1}{3}x_X - \frac{1}{3}f_X \\ &\leq \frac{2}{3}f_{XX}^3 + \frac{1}{3}x_X - \frac{1}{3}f_X. \end{aligned}$$

Now  $f_X = f_{XX} + f_{(X,r>X)}$ , so

$$n_O - n_H \leq \frac{1}{3}f_{XX} + \frac{1}{3}x_X - \frac{1}{3}f_{(X,r>X)}.$$

Moreover,  $n_H = f_X + x_X$  so

$$n_O - n_H \leq \frac{1}{3}n_H - \frac{2}{3}f_{(X,r>X)},$$

as the lemma requires.  $\square$

LEMMA 2.3.2. *Let region  $F$  be the region beyond  $X$  through the last  $F_1$  bin.*

$$n_{OF} - n_{HF} \leq \frac{1}{3}n_{HF} - \frac{2}{3}f_{(F,r \neq F)} + \frac{2}{3}f_{XF}.$$

*Proof.* Let  $B_F$  be the last bin in region  $F$ . Since no piece available when H packed  $B_F$  would fit with the last piece H packed in  $B_F$  and then replaced (or  $B_F$  would have been type  $Y_1$ ), the same arguments used in the proof of Lemma 2.3.1 imply that if OPT packs three pieces in a bin of  $F$ , none were available when H packed  $B_F$ . Those pieces must be  $f_X$  or  $f_F$  pieces. Hence

$$(2.1) \quad n_{OF} - n_{HF} = n_{OF} - 2M_F \leq \frac{1}{3}[f_{FF}^3 + f_{XF}^3].$$

Also, if OPT packs two pieces in a bin of  $F$ , at least one is a fallback piece from  $F$  or  $X$ . Hence

$$(2.2) \quad n_{OF} - M_F \leq \frac{2}{3}(f_{FF}^3 + f_{XF}^3) + f_{FF}^2 + f_{XF}^2.$$

And

$$(2.3) \quad n_{HF} - M_F \geq \frac{1}{2}f_F.$$

Combining (2.2) and (2.3) yields

$$(2.4) \quad n_{OF} - n_{HF} \leq \frac{2}{3}f_{FF}^3 + \frac{2}{3}f_{XF}^3 + f_{FF}^2 + f_{XF}^2 - \frac{1}{2}f_F.$$

Multiplying (2.1) by  $\frac{1}{3}$ , (2.4) by  $\frac{2}{3}$ , and adding gives

$$\begin{aligned} n_{OF} - n_{HF} &\leq \frac{5}{9}f_{FF}^3 + \frac{5}{9}f_{XF}^3 + \frac{2}{3}f_{FF}^2 + \frac{2}{3}f_{XF}^2 - \frac{1}{3}f_F \\ &\leq \frac{2}{3}f_{FF}^3 + \frac{2}{3}f_{XF}^3 - \frac{1}{3}[f_{FF} + f_{(F,r \neq F)}] \\ &\leq \frac{1}{3}f_{FF} + \frac{2}{3}f_{XF} - \frac{1}{3}f_{(F,r \neq F)} \\ &\leq \frac{1}{3}n_{HF} + \frac{2}{3}f_{XF} - \frac{2}{3}f_{(F,r \neq F)} \end{aligned}$$

as desired.  $\square$

LEMMA 2.3.3. *If there are no bad  $X_2$  bins, then  $n_O \leq \frac{4}{3}n_H$ .*

*Proof.* Let  $A$  be the union of regions  $X$  and  $F$ . Lemmas 2.3.1 and 2.3.2 imply that

$$(3.1) \quad n_{OA} - n_{HA} \leq \frac{1}{3}n_{HA} - \frac{2}{3}f_{(A,r > A)}.$$

Let  $R$  be the union of the remaining regions; let  $q$  be the piece not packed by H; let  $R^*$  be the set of pieces packed by OPT in  $R$ ; and let  $R'$  be the set of pieces packed by H in  $R$ . Let  $S$  be the set of pieces of  $R^*$  packed by H as type  $Y$  or  $F$  pieces in  $A$  (thus  $|S| = f_{AR}$ ). If  $p \in R^* \setminus (S \cup R' \cup \{q\})$ ,  $p$  is packed by OPT in  $R$  but by B2F in neither  $R$  nor  $A$ , since  $p \neq q$ ,  $p$  is an  $X_1$  or  $s(p) = 1$ , and  $s(p) \geq s(p')$  for all  $p' \in R'$ . Since each bin of  $R$  is at least  $\frac{3}{4}$  full,

$$(3.2) \quad s(R^*) \leq \frac{4}{3}s(R').$$

Let  $Q = R^* \cap \{q\}$  (thus  $Q = \emptyset$  if  $q$  was not packed by OPT in  $R$ ). Then

$$s(R^*) = s(R^* \cap R') + s(Q) + s(S) + s(R^* \setminus (R' \cup S \cup Q))$$

and  $s(R') = s(R' \cap R^*) + s(R' \setminus R^*)$ .

Substituting into (3.2) and combining terms yields

$$(3.3) \quad s(Q) + s(S) + s(R^* \setminus (R' \cup S \cup Q)) \leq \frac{1}{3}s(R' \cap R^*) + \frac{4}{3}s(R' \setminus R^*).$$

If we let  $m = \max_{p \in R'} s(p)$ , then  $m \leq 1$ ; and for  $p \in S \cup Q$ ,  $s(p) > m/2$ . Further, for  $p \in R^* \setminus (R' \cup S \cup Q)$ ,  $s(p) \geq m$ . Substituting these into (3.3) yields

$$(3.4) \quad \begin{aligned} &s(Q) + \left(\frac{m}{2}\right)|S| + m(|R^*| - |S| - |R^* \cap R'|) - s(Q) \\ &< \left(\frac{m}{3}\right)(|R' \cap R^*|) + \left(\frac{4}{3}m\right)(|R' \setminus R^*|). \end{aligned}$$

Combining like terms and dividing by  $m$  yields

$$\frac{s(Q)}{m} - |Q| - \frac{1}{2}|S| + |R^*| < \frac{4}{3}|R'|$$

or

$$|R^*| < \frac{1}{2}|S| + \frac{4}{3}|R'| + |Q| - \frac{s(Q)}{m}.$$

If  $Q = \emptyset$ ,

$$n_{OR} < \frac{1}{2}f_{AR} + \frac{4}{3}n_{HR} < \frac{1}{2}f_{AR} + \frac{4}{3}n_{HR} + \frac{1}{2}.$$

If  $Q \neq \emptyset$ ,  $s(q) > m/2$  and  $m \leq 1$ , so

$$(3.5) \quad n_{OR} < \frac{1}{2}f_{AR} + \frac{4}{3}n_{HR} + \frac{1}{2}.$$

Combining (3.5) and (3.1),

$$(3.6) \quad n_O < \frac{4}{3}n_H - \frac{1}{6}f_{AR} + \frac{1}{2}.$$

Since  $n_O$  is an integer, if  $f_{AR} \geq 1$ , we would have  $n_O \leq \frac{4}{3}n_H$ . Thus  $f_{AR} = 0$ , and all pieces packed by OPT in  $R$  are also packed by H in  $R$ , if H packs them at all. Pieces packed by H in  $R$ , but not packed by OPT in  $R$  must be replaced by larger pieces by OPT, so region  $R$  is a counterexample; by minimality there is no region  $A$ . From the argument used to show (3.6) we see that if we consider  $R$ ,  $R^*$ , and  $R'$  as a counterexample,

$$s(q) + s(R') + |R^*| - |R'| - 1 \leq \sum_{B \in \mathbf{B}} s(B) = \frac{4}{3}s(R') - \frac{4}{3} \left[ s(R') - \frac{3}{4} \sum_{B \in \mathbf{B}} s(B) \right].$$

Therefore

$$(3.7) \quad \begin{aligned} |R^*| &\leq \frac{1}{3}s(R') - \frac{4}{3} \left[ s(R') - \frac{3}{4} \sum_{B \in \mathbf{B}} s(B) \right] + |R'| + 1 - s(q) \\ &= \frac{4}{3}|R'| - \frac{1}{3} \sum_{p \in R'} [1 - s(p)] - \frac{4}{3} \left[ s(R') - \frac{3}{4} \sum_{B \in \mathbf{B}} s(B) \right] + 1 - s(q). \end{aligned}$$

If all terms on the right of (3.7) except  $\frac{4}{3}|R'|$  sum to less than  $\frac{1}{3}$ , then since  $|R^*|$  is an integer,  $|R^*| \leq \frac{4}{3}|R'|$  as desired. Therefore we shall show that for any bin  $B$ ,

$$-\frac{1}{3} \sum_{p \in B} [1 - s(p)] - \frac{4}{3} \left[ \sum_{p \in B} s(p) - \frac{3}{4}s(B) \right] + 1 - s(q) < \frac{1}{3}.$$

If this were not so,

$$\frac{1}{3} \sum_{p \in B} [1 - s(p)] + \frac{4}{3} \left[ \sum_{p \in B} s(p) - \frac{3}{4}s(B) \right] - 1 + s(q) \leq -\frac{1}{3}$$

and

$$\frac{1}{3}|B| + \sum_{p \in B} s(p) + s(q) - s(B) \leq \frac{2}{3}.$$

But  $|B| \geq 2$  (since region  $R$  is beyond  $F$ ) and  $\sum_{p \in B} s(p) + s(q) > s(B)$  and we have a contradiction.  $\square$

There must therefore be a bad  $X_2$  bin if we have a counterexample.

LEMMA 2.3.4. *If  $B$  is the last  $X_2$  bin and  $q_1$  and  $q_2$  are any two pieces available but not used when H packed  $B$ , then*

$$s(q_1) + s(q_2) > \frac{5}{6} \sum_{p \in B} s(p).$$

*Proof.* Let  $x_1$  and  $x_2$  be two  $X$  pieces packed in  $B$ . Since  $q_1$  and  $q_2$  did not replace  $x_2$ ,

$$s(x_1) + s(q_1) + s(q_2) > s(B) > \frac{4}{3}[s(x_1) + s(x_2)]$$



and

$$\begin{aligned} s(q_1) + s(q_2) &> \frac{4}{3}[s(x_1) + s(x_2)] - s(x_1) \\ &> \frac{4}{3}[s(x_1) + s(x_2)] - \frac{1}{2}[s(x_1) + s(x_2)] \end{aligned}$$

and the lemma follows.  $\square$

Recall that region  $A$  contains all the  $F_1$  and  $X_1$  bins. We define region  $S$  to include all  $Y_1$  bins where the fallback piece is smaller than the last piece  $q_s$  packed beyond the last  $X_2$  bin. All other  $Y_1$  bins not in  $A$  constitute region  $Y$ , except those bins where the fallback piece is as large as the last  $X_2$  piece. These bins, and all bins through the last  $X_2$  bin, constitute region  $D$ . All remaining bins constitute region  $Q$ . Note that all bins of  $S$  precede all bins of  $Y$ . Either region may be empty. Note further that pieces smaller than  $q_s$  are fallback pieces if they are packed in  $Q$ .

LEMMA 2.3.5.

$$n_{OS} - n_{HS} \leq \frac{1}{3}n_{HS} + \frac{2}{3}f_{AS} + \frac{2}{3}f_{DS} + \frac{5}{9}f_{QS} + \frac{2}{9}x_{QS} - \frac{2}{3}f_{(S,r \neq S)}.$$

*Proof.* Any bin  $B$  of  $S$  has  $n_{HB} = 2$ . If a bin of  $S$  contains more than 2 pieces in the OPT packing, it must contain 3, and each piece must be already packed by H ( $f_A$  or  $f_S$ ) or no larger than the piece used by H ( $f_D$  or  $f_Q$ ). Hence

$$\begin{aligned} n_{OS} - n_{HS} &\leq \frac{1}{3}f_{AS} + \frac{1}{3}f_{SS} + \frac{1}{3}f_{DS} + \frac{1}{3}f_{QS} \\ &\leq \frac{2}{3}f_{AS} + \frac{2}{3}f_{DS} + \frac{5}{9}f_{QS} + \frac{1}{3}n_{HS} - n_{HS} - \frac{1}{3}x_S - \frac{1}{3}f_{(S,r \neq S)} \end{aligned}$$

and, since  $x_S \geq f_{(S,r \neq S)}$ , the lemma follows.  $\square$

LEMMA 2.3.6.

$$n_{OY} - n_{HY} \leq \frac{2}{3}f_{AY} + \frac{2}{3}f_{DY} + \frac{2}{3}f_{SY} + \frac{5}{9}f_{QY} + \frac{2}{9}x_{QY} + \frac{1}{3}n_{HY} - \frac{1}{3}f_{(Y,r \neq Y)}.$$

*Proof.* As in Lemma 2.3.5, we need consider only bins packed by OPT with 3 pieces, and again those pieces were already packed by H ( $f_A, f_S, f_Y$ ) or are no larger than the fallback piece. But we must include  $X_Q$  pieces. Thus

$$\begin{aligned} n_{OY} - n_{HY} &\leq \frac{1}{3}f_{AY} + \frac{1}{3}f_{SY} + \frac{1}{3}f_{YY} + \frac{1}{3}f_{DY} + \frac{1}{3}f_{QY} + \frac{1}{3}x_{QY} \\ &\leq \frac{1}{3}f_{YY} + \frac{1}{3}f_{(r \neq Y, Y)} + \frac{1}{3}x_{QY} + \frac{1}{3}n_{HY} - \frac{2}{3}f_Y. \end{aligned}$$

Now  $f_Y = M_Y \geq \frac{1}{3}x_{QY}$ , so  $\frac{1}{3}f_Y \geq \frac{1}{9}x_{QY}$  and

$$n_{OY} - n_{HY} \leq \frac{1}{3}f_{YY} + \frac{1}{3}f_{(r \neq Y, Y)} + \frac{2}{9}x_{QY} + \frac{1}{3}n_{HY} - \frac{1}{3}f_{(Y,r \neq Y)}$$

and the lemma follows.  $\square$

LEMMA 2.3.7.

$$n_{OD} - n_{HD} \leq \frac{1}{3}n_{HD} + \frac{2}{3}f_{AD} + \frac{2}{3}f_{SD} + \frac{2}{9}x_{QD} + \frac{2}{9}f_{QD} + \frac{2}{9}f_{YD} - \frac{2}{3}f_{(D,r \neq D)}.$$

*Proof.* If  $B$  is a bin of  $D$  in which OPT packed 3 pieces, either one of the pieces was not available when H packed  $B$  and is smaller than any piece that was available ( $f_A, f_S, f_D$ ), or OPT packed at least 3 pieces smaller than any  $X_2$  piece ( $f_Q, x_Q, f_Y$ ). (Otherwise the two smaller pieces would be able to replace the smallest  $X_2$  piece in the last  $X_2$  bin.) Thus, abusing the notation by making it refer to numbers of pieces in bin  $B$ ,

$$(7.1) \quad n_{OB} - 2 \leq f_{AB} + f_{SB} + f_{DB} + \frac{1}{3}f_{QB} + \frac{1}{3}f_{YB} + \frac{1}{3}x_{QB}.$$

If OPT packs  $B$  with more than 3 pieces, we claim that

$$(7.2) \quad n_{OB} - 2 \leq \frac{3}{4}f_{AB} + \frac{3}{4}f_{DB} + \frac{3}{4}f_{SB} + \frac{1}{4}f_{QB} + \frac{1}{4}x_{QB} + \frac{1}{4}f_{YB}.$$

To see this, let  $f = f_{AB} + f_{DB} + f_{SB}$  and  $x = f_{QB} + x_{QB} + f_{YB}$ , and let  $m$  be the average size of the  $X$  pieces in the last  $X_2$  bin. Then, since  $s(B) < 3m$ ,

$$\frac{1}{2}mf + \frac{5}{6}mx + m(n_{OB} - x - f) < 2m + s(q),$$

where  $q$  is the piece not packed by H. Then

$$(7.3) \quad n_{OB} - 2 < \frac{1}{2}f + \frac{1}{6}x + \frac{1}{m}s(q).$$

If the claim is false,  $\frac{3}{4}f + \frac{1}{4}x < \frac{1}{2}f + \frac{1}{6}x + (1/m)s(q)$ , and, since  $q$  is smaller than any  $X_2$ ,  $\frac{1}{4}f + \frac{1}{12}x < (1/m)s(q) \leq 1$ . Thus  $f \leq 3$ .

If  $f = 3$ , then  $x \leq 2$  and  $n_{OB} - 2 < \frac{1}{2} \cdot 3 + \frac{1}{6} \cdot 2 + (1/m)s(q) < 3$ . But then  $n_{OB} - 2 \leq \frac{3}{4}f + \frac{1}{4}x$ .

If  $f = 2$ , then  $x \leq 5$  and from (7.3),  $n_{OB} - 2 \leq \frac{1}{2} \cdot 1 + \frac{1}{6} \cdot 5 + (1/m)s(q) < 3$  and  $n_{OB} - 2 \leq \frac{3}{4}f + \frac{1}{4}x$  unless  $x \leq 1$ . If  $x \leq 1$ ,  $n_{OB} - 2 \leq \frac{1}{2} \cdot 2 + \frac{1}{6} \cdot 1 + (1/m)s(q) < 2$ .

If  $f = 1$ , then  $\frac{1}{2} \cdot 1 + \frac{5}{6}x < 3$ , so  $x \leq 2$  and  $n_{OB} - 2 \leq \frac{1}{2} \cdot 1 + \frac{1}{6} \cdot 2 + (1/m)s(q) < 2$ .

If  $f = 0$ , then  $x \leq 3$  and  $n_{OB} - 2 \leq \frac{1}{6} \cdot 3 + (1/m)s(q) < 2$ . Thus the claim is true.

Combining (7.1) and (7.2), using  $f^1$  to refer to bins where  $n_{OB} - 2 = 1$ , and  $f^2, x^2$  to refer to bins where  $n_{OB} - 2 \geq 2$ ,

$$(7.4) \quad n_{OD} - 2M_D = \sum_{B \in D} n_{OB} - 2 \leq \frac{3}{4}f_{AD}^2 + \frac{3}{4}f_{DD}^2 + \frac{3}{4}f_{SD}^2 + \frac{1}{4}f_{QD}^2 + \frac{1}{4}x_{QD}^2 + \frac{1}{4}f_{YD}^2 + f_{AD}^1 + f_{DD}^1 + f_{SD}^1 + \frac{1}{3}f_{QD}^1 + \frac{1}{3}x_{QD}^1 + \frac{1}{3}f_{YD}^1.$$

Also  $n_{HD} - 2M_D \geq \frac{1}{2}f_D$ , so

$$(7.5) \quad n_{OD} - n_{HD} \leq \frac{3}{4}f^2 + \frac{1}{4}x_{QD}^2 + f^1 + \frac{1}{3}x_{QD}^1 - \frac{1}{2}f_D.$$

We now obtain another inequality by computing  $n_{OD} - 3M_D$ . For any bin  $B$  packed by OPT with more than 3 pieces,  $n_{OB} - 3 \leq \frac{2}{3}[n_{OB} - 2]$  (since  $n_{OB} \leq 5$ ), so  $n_{OD} - 3M_D \leq \frac{1}{2}f_D^2 + \frac{1}{6}x_D^2$ . Also  $3M_D - n_{HD} \leq \frac{1}{2}[x_D - \frac{1}{2}f_D]$ . Combining these yields

$$(7.6) \quad n_{OD} - n_{HD} \leq \frac{1}{2}f_{DD}^2 + \frac{1}{2}f_{AD}^2 + \frac{1}{2}f_{SD}^2 + \frac{1}{6}x_{DD}^2 + \frac{1}{2}x_D - \frac{1}{4}f_D.$$

Multiplying (7.5) by  $\frac{2}{3}$  and (7.6) by  $\frac{1}{3}$  and adding yields:

$$\begin{aligned} n_{OD} - n_{HD} &\leq \frac{2}{3}f_{DD} + \frac{2}{3}f_{AD} + \frac{2}{3}f_{SD} + \frac{2}{9}x_{DD} + \frac{2}{9}x_{QD} + \frac{1}{6}x_D - \frac{5}{12}f_D \\ &\leq \frac{2}{3}[f_{AD} + f_{SD}] + \frac{2}{9}[x_{QD} + f_{QD} + f_{YD}] + \frac{1}{3}n_{HD} - \frac{2}{3}f_{(D,r \neq D)} \end{aligned}$$

as desired in the lemma.  $\square$

LEMMA 2.3.8 (Summary Lemma). *Let  $I$  be the region consisting of all bins through the last  $X_2$  bin, and let  $Q$  be all remaining bins.*

$$n_{OI} - n_{HI} \leq \frac{1}{3}n_{HI} + \frac{2}{9}x_{QI} - \frac{2}{3}f_{IQ} + \frac{5}{9}f_{QI} - \frac{1}{3}f_{YQ},$$

where  $f_{IQ}$  does not contain the  $f_{YQ}$  contribution.

*Proof.* The lemma follows from adding the inequalities for the subregions A, S, Y, and D. The  $\frac{5}{9}f_{QI}$  term contains only contributions from the Y region. The term  $f_{YQ}$  is nonzero only if a piece larger than  $q_s$  is packed by H in a  $Y_1$  bin but is packed by OPT in  $Q$ . By Lemma 2.3.6, such pieces can be subtracted with a coefficient no larger than  $\frac{1}{3}$ .  $\square$

We examine the remaining bins in the following lemmas. Because of a slight problem with small fallback pieces in the  $F_2$  region, we give separate proofs for the  $F_2$  and  $X_3$  regions. All other  $F_n$  and  $X_n$  bins are treated together.

LEMMA 2.3.9. *In region  $R = F_2 \cup Y_2$ ,*

$$n_{OR} - n_{HR} \leq \frac{1}{3}n_{HR} + \frac{2}{3}f_{IR} + \frac{5}{9}f_{QR} + \frac{2}{9}x_{QR} + \frac{2}{9}f_{YR} - \frac{2}{3}f_{RQ_s} - \frac{2}{9}x_{RQ} - \frac{5}{9}f_{RQI}$$

where region  $Q$  contains all bins after region  $R$ ;  $f_{RQ_s}$  and  $f_{RQ_l}$  refer, respectively, to fallback  $F_2$  pieces which are smaller than  $q_s$  and at least as large as  $q_s$ .

*Proof.* We prove the lemma by examining bins individually and summing over all bins in the region. If we can show, for each bin  $B$  in region  $R$ , that

$$(9.1) \quad n_{OB} - n_{HB} \leq \frac{2}{3}f_{IB} + \frac{5}{9}f_{QB} + \frac{2}{9}x_{QB} + \frac{2}{9}f_{YB} + \frac{5}{9}f_{RB} + \frac{2}{9}x_{RB} - \frac{1}{3}f_{Bs} - \frac{2}{9}f_{BI} + \frac{1}{9}x_B$$

then the lemma will follow from summing this formula over all bins in  $R$ .

Suppose that  $B$  is a bin of  $R$  for which (9.1) fails. Throughout  $R$  we know that  $n_{HB}$  is 3, that  $x_B$  is 1 or 2, and that  $f_{Bs} + f_{BI}$  is 1 or 2. Since any piece that OPT packs in  $B$  is at least half the size of any piece of type  $X$  that H packs in  $B$ , and all other pieces (besides  $f_I$ ) are at least  $\frac{5}{6}$  the size, we have easily that

$$\frac{1}{2}f_{IB} + \frac{5}{6}(f_{QB} + f_{RB} + f_{YB} + x_{QB} + x_{RB}) + n_{OB} - (f_{IB} + f_{QB} + f_{RB} + f_{YB} + x_{QB} + x_{RB}) < 3.$$

This gives us the following, letting  $f$  stand for  $f_{IB}$ , and  $q$  for all the other types of pieces listed above:

$$(9.2) \quad n_{OB} - 3 < \frac{1}{2}f + \frac{1}{6}q.$$

If (9.1) fails, it is easy to see that  $n_{OB}$  must be at least 3 and that no bin in  $R$  can contain more than 5 pieces. We have 3 cases to consider.

*Case 1.*  $n_{OB} = 5$ . In this case, we have from (9.2) that  $f_{IB} \geq 4$ , and (9.1) follows, since  $2 \leq \frac{2}{3} \cdot 4 - \frac{1}{3} \cdot 2 + \frac{1}{9} \cdot 1$ .

*Case 2.*  $n_{OB} = 4$ . In this case, (9.2) implies  $f \geq 2$ . If  $f \geq 3$ , (9.1) follows as in Case 1. If  $f = 2$ , then (9.2) implies  $q \geq 1$ . Then  $1 \leq \frac{2}{3} \cdot 2 + \frac{2}{9} \cdot 1 - \frac{1}{3} \cdot 2 + \frac{1}{9} = 1$ , and (9.1) follows.

*Case 3.*  $n_{OB} = 3$ . In this case, we know from (9.2) that  $f$  or  $q$  must be at least 1. If  $f_{IB} \geq 1$  or  $f_{QB} \geq 1$ , then (9.1) follows. If  $B$  is type  $Y_2$  and  $f_{IB} = f_{QB} = 0$ , then (9.1) still follows if  $f_{IB} = f_{QB} = 0$ . If  $B$  is type  $F_2$ , then  $B$  cannot contain 2 pieces as large as the  $X$  piece H packs in  $B$  and also a piece as large as any available when H packed  $B$ . Thus  $f_{YB} + x_{QB} = 2$ . In that case, at least one of the fallback pieces packed by H must be at least as large as  $q_s$  because of the way H replaces pieces, and  $f_{BI} \geq 1$ . Substituting into (9.1) yields the lemma.  $\square$

LEMMA 2.3.10. *If region  $R$  is the region  $X_3$ ,*

$$n_{OR} - n_{HR} \leq \frac{1}{3}n_{HR} + \frac{2}{3}f_{IR} + \frac{5}{9}f_{IR} + \frac{5}{9}f_{QR} + \frac{2}{9}x_{QR} - \frac{2}{9}x_{RQ} - \frac{5}{9}f_{RQ},$$

where  $f_{IR}$  is the number of pieces at least as large as  $q_s$  packed by H and OPT in  $R$ .

*Proof.* As for the previous lemma, this proof is carried out by showing that, for each bin  $B$  in region  $R$ ,

$$(10.1) \quad n_{OB} - n_{HB} \leq \frac{2}{3}f_{IB} + \frac{2}{3}f_{sB} + \frac{5}{9}f_{IB} + \frac{2}{9}x_{QB} + \frac{1}{9}x_B - \frac{2}{9}f_B.$$

This inequality certainly holds if  $n_{OB} \leq 3$ , since  $n_{HB} + \frac{1}{9}x_B - \frac{2}{9}f_B > 3$ . Suppose  $4 \leq n_{OB} \leq 7$ . We know that

$$(10.2) \quad n_{OB} - 4 < \frac{1}{2}f + \frac{1}{6}q$$

where  $f = f_{IB}$  and  $q = f_{sB} + f_{IB} + x_{QB} + f_{QB}$ .

*Case 1.*  $n_{OB} = 7$ . In this case, (10.2) implies  $f_{IB} \geq 6$ , and (10.1) follows easily.

*Case 2.*  $n_{OB} = 6$ . In this case,  $f_{IB} \geq 4$ , and (10.1) follows.

*Case 3.*  $n_{OB} = 5$ . Now  $f_{IB} \geq 1$ . If  $f_{IB} \geq 3$ , (10.1) follows easily. Suppose  $f_{IB} = 2$ . Then  $q \geq 1$ , so if  $B$  is a fallback bin,  $5 - 4 \leq \frac{4}{3} + \frac{2}{9} + \frac{2}{9} - \frac{4}{9}$  and (10.1) holds again. If  $B$  is not a fallback bin, then a similar computation shows that (10.1) holds if  $q \geq 3$ , or if  $f_{IB}$  or  $f_{sB}$  is nonzero. If there are two  $X_Q$  pieces packed by OPT in  $B$ , they would replace the last  $X$  piece H packed, and if there is only one  $X_Q$  piece, it would fit with

the 3 pieces H packed in  $B$ . (If the pieces are larger than  $q_s$  but already packed, then other pieces could be used.)

Suppose  $f_{IB} = 1$ . If  $B$  is a fallback bin, then  $n_{OB} - n_{HB} = 1 \leq \frac{2}{3} \cdot 1 + \frac{2}{9} \cdot 4 + \frac{1}{9} \cdot 2 - \frac{2}{9} \cdot 2$ . If  $B$  is not a fallback bin, then (10.1) holds if  $f_{IB} + f_{sB} + f_{QB} \geq 1$ , since  $2 \leq \frac{2}{3} + \frac{5}{9} + 3 \cdot \frac{2}{9} + 3 \cdot \frac{1}{9}$  and  $q = 4$ . If  $x_{QB} = 4$ , however, any  $x_{QB}$  piece would fit in  $B$  as a fallback piece, contradicting the fact that  $B$  has type  $X_3$ . This follows since any  $f_{IB}$  piece and any three  $x_{QB}$  pieces are at least as large as the three pieces H packed in  $B$ .

*Case 4.*  $n_{OB} = 4$ . In this case, all we know from (10.2) is that  $f + q \geq 1$ . If  $B$  is a fallback bin then (10.1) holds. So suppose that  $B$  is an  $X_3$  bin. If  $f_{IB}$  or  $f_{sB}$  are at least 1, then again (10.1) holds. If  $f_{IB}$ ,  $x_{QB}$  or  $f_{QB}$  are 1, then  $q_s$  would have fit, making  $B$  type  $Y_3$ . Similarly, if  $f_{IB} + x_{QB} = 2$ ,  $B$  would be type  $F_3$ . But if  $f_{IB} - x_{QB} \geq 3$ , then  $1 \leq \frac{2}{9} \cdot 3 + \frac{1}{9} \cdot 3$  and (10.1) holds.  $\square$

LEMMA 2.3.11. *If  $R = F_n$  or  $Y_n$ ,  $n \geq 3$ , or  $X_n$ ,  $n \geq 4$ , then*

$$n_{OR} - n_{HR} \leq \frac{1}{3}n_{HR} + \frac{2}{3}f_{IR} + \frac{5}{9}f_{QR} + \frac{2}{9}x_{Q'R} - \frac{2}{9}x_{RQ} - \frac{5}{9}f_{RQ},$$

where region  $Q$  contains all bins after  $I$ , and  $Q'$  contains all bins after  $R$ .

*Proof.* We begin by proving a general inequality that applies in each of the cases. Let  $B$  be any bin of  $R$ . Then any pieces from region  $I$  must have size at least half that of the  $X$  pieces that H packs in  $B$ . Any other pieces that are smaller than the  $X$  pieces must be at least  $\frac{5}{6}$  their size. Consequently,

$$\frac{1}{2}f_{IB} + \frac{5}{6}(f_{QB} + x_{Q'B}) + n_{OB} - f_{IB} - f_{QB} - x_{Q'B} < n + 1$$

and we can simplify this to

$$n_{OB} - n < \frac{1}{2}f_{IB} + \frac{1}{6}(f_{QB} + x_{Q'B}) + 1.$$

Since  $n_{OB} - n$  is an integer,

$$(11.1) \quad n_{OB} - n \leq \frac{1}{2}f_{IB} + \frac{1}{6}(f_{QB} + x_{Q'B}) + \frac{5}{6}.$$

Now we assume that  $B$  is a bin of  $R$  in which OPT packs at least  $n + 2$  pieces. We claim that if  $R$  is a region  $F_n$  or  $X_n$ , then

$$n_{OB} - n \leq \frac{3}{4}f_{IB} + \frac{1}{2}f_{QB} + \frac{1}{4}x_{Q'B}$$

while, if  $R$  is  $Y_n$ , then

$$n_{OB} - n \leq \frac{3}{4}f_{IB} + \frac{1}{2}f_{QB} + \frac{1}{4}x_{Q'B} + \frac{1}{4}.$$

Note that the only difference between the claim for  $Y_n$  and the others is the  $\frac{1}{4}$  on the end. We shall proceed with the proof for the case when  $R$  is  $X_n$  or  $F_n$  and point out where the  $\frac{1}{4}$  is needed in three places to fill in the details for the  $Y_n$  case.

If the claim fails for  $R = X_n$  or  $F_n$  then from (11.1) we have by simplifying

$$\frac{1}{4}f_{IB} + \frac{1}{3}f_{QB} + \frac{1}{12}x_{Q'B} < \frac{5}{6},$$

and consequently  $f_{IB} \leq 3$  and  $f_{IB} + f_{QB} + x_{Q'B} \leq 9$ .

If  $f_{IB} = 3$ , then  $f_{QB}$  and  $x_{Q'B}$  must be 0. But then  $n_{OB} - n$  must be at most  $\frac{1}{2} \cdot 3 + \frac{5}{6}$  if the claim fails. Since it must be an integer, it can only be 2; and then  $2 \leq \frac{3}{4} \cdot 3$ , contradicting the supposition that the claim failed.

If  $f_{IB} = 2$ , then  $f_{QB} = 0$ , and  $x_{Q'B} \leq 3$ . Again we have  $n_{OB} - n \leq 2$  from (11.1) and the claim holds unless  $x_{Q'B} \leq 1$ . In this case, however, the  $x_{Q'B}$  piece would fit with the  $n$  pieces packed by H in the last bin of region  $R$ , contradicting the fact that  $R$  is  $X_n$  or  $F_n$ . If  $R$  is region  $Y_n$ , then the additional  $\frac{1}{4}$  added to the claim is sufficient if  $x_{Q'B} = 1$ . Note that  $x_{Q'B}$  cannot be 0 by (11.1).

If  $f_{IB} = 1$ , then  $f_{QB} + x_{Q'B} \leq 6$ , if the claim is false. From (11.1) we again obtain  $n_{OB} - n = 2$  and the claim holds unless  $f_{QB} \leq 2$  and  $f_{QB} + x_{Q'B} \leq 4$ . But also from (11.1),

$$2 \leq \frac{1}{2} + \frac{1}{6}(f_{QB} + x_{Q'B}) + \frac{5}{6}$$

implies that  $f_{QB} + x_{Q'B} = 4$  and, if  $f_{QB} \geq 1$ , the claim holds. Thus if the claim is false,  $f_{QB} = 0$  and  $x_{Q'B} = 4$ . As above we now find that any of the  $x_{Q'B}$  pieces would fit in the last bin of  $R$ , contradicting the fact that the bin is of  $F$  or  $X$  type. And again the  $\frac{1}{4}$  in the claim when  $R = Y_n$  is sufficient for this case as well.

Finally, if  $f_{IB} = 0$ , then we can again show that  $n_{OB} - n = 2$ , and that  $f_{QB} = 0$  and  $x_{Q'B} \leq 7$ . If  $x_{Q'B} < 7$  the claim holds, so it must be exactly 7. Again we find that any of the smaller pieces would fit in the last bin of  $R$ . And again in the  $Y_n$  case, the additional  $\frac{1}{4}$  is sufficient to cause the claim to hold.

Suppose now that  $n_{OB} = n + 1$ , and that  $R$  is of type  $X_n$  or  $F_n$ . If OPT packs  $B$  with no pieces smaller than those available to pack the last bin  $B_R$  of  $R$ , then there must be at least two pieces smaller than the pieces of type  $X$  in  $B_R$ , if  $R$  is an  $F_n$ , and 3 such pieces if  $R$  is an  $X_n$ . Thus if  $R = F_n$ , (using  $f^1$  and  $x^1$  to denote pieces packed by OPT in a bin with  $n + 1$  pieces)

$$(11.2) \quad n_{OB} - n \leq f_{IB}^1 + f_{QB}^1 + \frac{1}{2}x_{Q'B}^1,$$

and if  $R$  is  $X_n$ ,

$$(11.3) \quad n_{OB} - n \leq f_{IB}^1 + f_{QB}^1 + \frac{1}{3}x_{Q'B}^1.$$

For  $R = F_n$  we have from (11.2) and the claim (using  $f^2$  and  $x^2$  to denote pieces packed by OPT in a bin of  $n + 2$  pieces):

$$n_{OB} - n \leq \frac{3}{4}f_{IB}^2 + f_{IB}^1 + f_{QB}^1 + \frac{1}{2}f_{QB}^2 + \frac{1}{4}x_{Q'B}^2 + \frac{1}{2}x_{Q'B}^1,$$

and summing over all bins of  $R$ , using  $n_{HR} \geq n|R| + \frac{1}{2}f_R$ ,

$$(11.4) \quad n_{OR} - n_{HR} \leq \frac{3}{4}f_{IR}^2 + f_{IR}^1 + f_{QR}^1 + \frac{1}{2}f_{QR}^2 + \frac{1}{4}x_{Q'R}^2 + \frac{1}{2}x_{Q'R}^1 - \frac{1}{2}f_R.$$

Thus, for  $R = F_n$ ,

$$(11.5) \quad n_{OR} - n_{HR} \leq \frac{3}{4}f_{IR}^2 + f_{IR}^1 + \frac{1}{2}f_{QR}^2 + f_{QR}^1 + \frac{1}{4}x_{QR}^2 + \frac{1}{2}x_{QR}^1 - \frac{1}{2}f_R$$

For  $R = Y_n$  we can obtain the same result since  $n_{HR} = n|R| + f_R$ . Also, from (11.1),

$$n_{OB} - (n + 1) \leq \frac{1}{2}f_{IB}^2 + \frac{1}{6}f_{QB}^2 + \frac{1}{6}x_{Q'B}^2$$

and, summing over all bins of  $R$ ,

$$(11.6) \quad n_{OR} - n_{HR} \leq \frac{1}{2}f_{IR}^2 + \frac{1}{6}f_{QR}^2 + \frac{1}{6}x_{QR}^2.$$

Multiplying (11.5) by  $\frac{5}{9}$  and (11.6) by  $\frac{4}{9}$ ,

$$\begin{aligned} n_{OR} - n_{HR} &\leq \frac{2}{3}f_{IR} + \frac{5}{9}f_{QR} + \frac{2}{9}x_{QR} - \frac{1}{2}f_R \\ &\leq \frac{1}{3}n_{HR} + \frac{2}{3}f_{IR} + \frac{5}{9}f_{QR} + \frac{2}{9}x_{QR} - \frac{1}{3}x_{QR} - \frac{5}{9}f_R \end{aligned}$$

and the lemma holds when  $R = F_n$  or  $Y_n$ .

Similarly when  $R$  is an  $X_n$  region, we get from (11.3) and the claim, and from summing over all bins of  $R$ ,

$$(11.7) \quad n_{OR} - n_{HR} \leq \frac{3}{4}f_{IR}^2 + f_{IR}^1 + f_{QR}^1 + \frac{1}{2}f_{QR}^2 + \frac{1}{4}x_{Q'R}^2 + \frac{1}{3}x_{Q'R}^1 - \frac{1}{2}f_R.$$

For any bin  $B$  of  $R$ ,  $n_{HB} \cong (1/n)x_B - (n-1)f_B/2n$ . Thus

$$n_{OB} - n_{HB} \cong \frac{1}{2}f_{IR}^2 + \frac{1}{6}(f_{QB}^2 + x_{QB}^2) + \frac{1}{n}x_B - \frac{n-1}{2n}f_B$$

and

$$(11.8) \quad n_{OR} - n_{HR} \cong \frac{1}{2}f_{IR}^2 + \frac{1}{6}(f_{QR}^2 + x_{QR}^2) + \frac{1}{n}x_R - \frac{n-1}{2n}f_R.$$

Multiplying (11.7) by  $\frac{5}{9}$  and (11.8) by  $\frac{4}{9}$  yields

$$n_{OR} - n_{HR} \cong \frac{2}{3}f_{IR} + \frac{5}{9}f_{QR} + \frac{2}{9}x_{QR} + \frac{4}{9}x_R.$$

Since  $n \geq 4$ , the lemma follows in this case as well.  $\square$

Adding Lemmas 2.3.8 through 2.3.11 yields the final result.

#### REFERENCES

- [1] E. G. COFFMAN, JR. AND J. Y.-T. LEUNG, *Combinatorial analysis of an efficient algorithm for processor and storage allocation*, this Journal, 8 (1979), pp. 202-217.
- [2] E. G. COFFMAN, JR., J. Y.-T. LEUNG AND D. W. TING, *Bin-packing: maximizing the number of pieces packed*, Acta Inform., 9 (1978), pp. 263-271.
- [3] D. K. FRIESEN, *Analysis of a new bin packing algorithm and its application to scheduling*, Tech. Rept. TAMUDCS-81-011R, Dept. of Computer Science, Texas A&M Univ., College Station, TX, 1981.
- [4] D. K. FRIESEN AND M. A. LANGSTON, *Analysis of a compound bin packing algorithm*, Tech. Rept. TAMUDCS-81-010R, Dept. of Computer Science, Texas A&M Univ., College Station, TX, 1981.
- [5] ———, *Bounds for MULTIFIT scheduling on uniform processors*, this Journal, 12 (1983), pp. 60-70.
- [6] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability*, W. H. Freeman, San Francisco, 1979.
- [7] D. S. JOHNSON, A. DEMERS, J. D. ULLMAN, M. R. GAREY AND R. L. GRAHAM, *Worst-case performance bounds for simple one-dimensional packing algorithms*, this Journal 3 (1974), pp. 299-325.
- [8] M. A. LANGSTON, *Performance bounds of heuristics for a computer resource allocation problem*, SIAM J. Algebraic Discrete Methods, 5 (1984), pp. 154-161.
- [9] J. Y.-T. LEUNG, *Efficient algorithms for storage allocation problems*, Ph.D. dissertation, Computer Science Dept., Pennsylvania State Univ., State College, PA, 1977.

## AN OPTIMAL ALGORITHM FOR FINDING A MAXIMUM INDEPENDENT SET OF A CIRCULAR-ARC GRAPH\*

SUMIO MASUDA† AND KAZUO NAKAJIMA‡

**Abstract.** A new algorithm is presented for finding a maximum independent set of a circular-arc graph. When the graph is given in the form of a family of  $n$  arcs, our algorithm requires only  $O(n \cdot \log n)$  time and  $O(n)$  space. Furthermore, if the endpoints of the arcs are already sorted, it runs in  $O(n)$  time. This algorithm is time- and space-optimal to within a constant factor.

**Key words.** optimal algorithm, circular-arc graph, maximum independent set

**AMS(MOS) subject classifications.** 68C25, 68E10

**1. Introduction.** Let  $G = (V, E)$  be a graph. Two distinct vertices  $u$  and  $v$  in  $V$  are said to be *independent* from each other if  $(u, v) \notin E$ ; otherwise they are said to be *adjacent* to each other. A subset  $X$  of  $V$  is called an *independent set* of  $G$  if any two vertices in  $X$  are independent. A *maximum independent set* of  $G$  is an independent set whose cardinality is the largest among all independent sets of  $G$ .

Consider a finite family  $S$  of nonempty sets. A graph  $G = (V, E)$  is called an *intersection graph* for  $S$  if there is a one-to-one correspondence between  $S$  and  $V$  such that two sets in  $S$  have a nonempty intersection if and only if the corresponding vertices in  $V$  are adjacent to each other. If  $S$  is a family of intervals on the real line, then  $G$  is called an *interval graph*. When  $S$  is a family of arcs on a circle,  $G$  is called a *circular-arc graph* for  $S$ .

Interval graphs have been used in many practical applications [6], [12], [14], [15], and, as such, a wide variety of algorithms have been developed [2], [5], [7], [9]–[11]. Furthermore, as a generalization of interval graphs, circular-arc graphs have received considerable attention in recent years. Tucker [13] gave an  $O(n^3)$  time algorithm for recognizing circular-arc graphs, where  $n$  is the number of the vertices in a given graph. Garey, Johnson, Miller and Papadimitriou [3] showed that the vertex coloring problem is NP-complete for circular-arc graphs. Gavril [4] developed polynomial time algorithms for finding a maximum clique, a maximum independent set, and a minimum covering by disjoint cliques of a circular-arc graph. When the graph is given in the form of a family of  $n$  arcs, the algorithms produce solutions in  $O(n^{3.5})$ ,  $O(n^4)$ , and  $O(n^5)$  time, respectively. Later, Gupta, Lee and Leung [7] gave  $O(n^2)$  time implementations of the last two algorithms of Gavril's. Leung [10] developed an efficient algorithm for generating all maximal independent sets of a circular-arc graph. Recently, Hsu [8] presented an algorithm for finding a maximum weight clique for the case when each vertex is assigned a real number as its weight. Its time complexity is  $O(n \cdot m)$ , where  $m$  is the number of edges of the graph.

---

\* Received by the editors November 24, 1986; accepted for publication (in revised form) March 10, 1987. This work was supported in part by National Science Foundation grants MIP-84-51510 and CDR-85-00108, and in part by a grant from AT&T Information Systems.

† Electrical Engineering Department and Systems Research Center, University of Maryland, College Park, Maryland 20742. This author is on leave from the Department of Information and Computer Sciences, Osaka University, Toyonaka, Osaka 560, Japan.

‡ Electrical Engineering Department, Institute for Advanced Computer Studies and Systems Research Center, University of Maryland, College Park, Maryland 20742.

In this paper, we present a new algorithm for finding a maximum independent set of a circular-arc graph. We show an  $O(n \cdot \log n)$  time and  $O(n)$  space implementation of the algorithm when the graph is given in the form of a family of  $n$  arcs on a circle. If the endpoints of the arcs are already sorted, the algorithm is shown to run in  $O(n)$  time.

It should be noted that Gupta et al. [7] have proven that it requires  $\Omega(n \cdot \log n)$  time in the worst case to find a maximum independent set of an interval graph with  $n$  vertices. Since every interval graph is a circular-arc graph, our algorithm is both time- and space-optimal to within a constant factor.

**2. Definitions and notation.** Let  $S = \{a_1, a_2, \dots, a_n\}$  be a family of arcs on a circle  $C$ . Each endpoint of the arcs is assigned a positive integer, called a *coordinate*. The endpoints are located on the circumference of  $C$  in ascending order of the values of the coordinates in the clockwise direction. Without loss of generality, we can assume that (i) all endpoints of the arcs in  $S$  are distinct, and (ii) no single arc in  $S$  covers the entire circle  $C$  by itself.

For simplicity, we call the endpoint with coordinate  $j$  as point  $j$ . Suppose that an arc begins at point  $j$  and ends at point  $k$  in the clockwise direction. Then, we denote such an arc by  $(j, k)$ , and call points  $j$  and  $k$  as the *head* and the *tail*, respectively, of the arc  $(j, k)$ . For  $i = 1, 2, \dots, n$ , let  $h_i$  and  $t_i$  denote the coordinates of the head and tail, respectively, of arc  $a_i$ , that is,  $a_i = (h_i, t_i)$ . We show an example of a family of arcs in Fig. 1, where  $a_1 = (1, 7)$ ,  $a_2 = (3, 5)$ ,  $a_3 = (6, 9)$ ,  $a_4 = (8, 12)$ ,  $a_5 = (10, 13)$ ,  $a_6 = (11, 15)$ ,  $a_7 = (14, 4)$  and  $a_8 = (16, 2)$ .

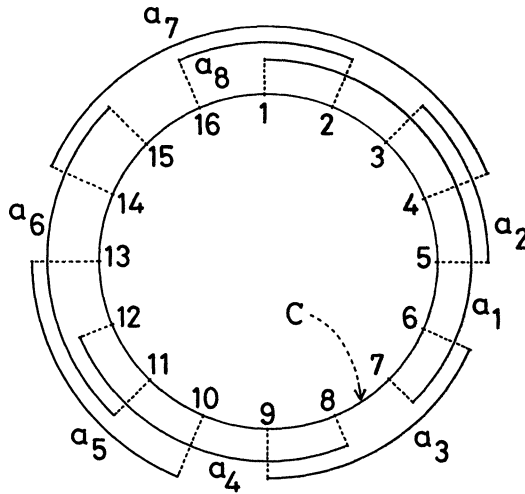


FIG. 1. A family of arcs on a circle  $C$ .

For an arc  $a_i \in S$  and an endpoint  $j$  of another arc in  $S$ , we say that  $a_i$  *contains* point  $j$  if one of the following three conditions holds (see Fig. 2):

- (i)  $1 \leq h_i < j < t_i \leq 2n$ .
- (ii)  $1 \leq t_i < h_i < j \leq 2n$ .
- (iii)  $1 \leq j < t_i < h_i \leq 2n$ .

For two distinct arcs  $a_i$  and  $a_j$  in  $S$ , we say that they *intersect* with each other if one of them contains at least one of the endpoints of the other arc; otherwise  $a_i$  and  $a_j$  are said to be *independent* from each other. If  $a_i$  contains both endpoints of  $a_j$ , we



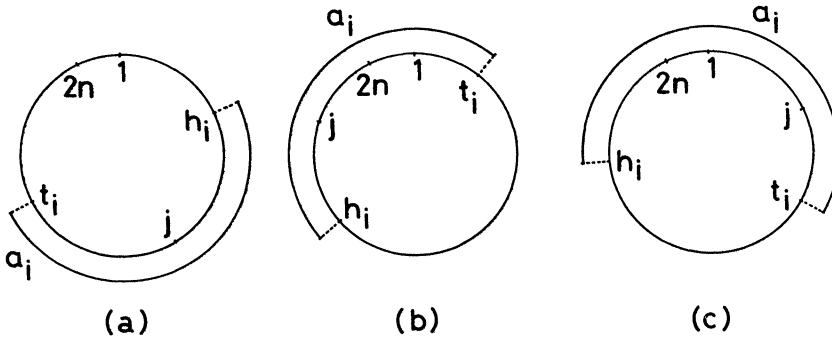


FIG. 2. Cases in which arc  $a_i$  contains point  $j$ . (a)  $1 \leq h_i < j < t_i \leq 2n$ . (b)  $1 \leq t_i < h_i < j \leq 2n$ . (c)  $1 \leq j < t_i < h_i \leq 2n$ .

say that  $a_i$  contains  $a_j$ . The circular-arc graph for  $S$ , denoted by  $G_S$ , is defined as follows:

$$G_S \triangleq (V_S, E_S), \text{ where}$$

$$V_S \triangleq \{v_1, v_2, \dots, v_n\} \text{ and}$$

$$E_S \triangleq \{(v_i, v_j) \mid a_i \text{ and } a_j \text{ intersect with each other}\}.$$

For example, Fig. 3 depicts the circular-arc graph for the family of arcs given in Fig. 1.

A subfamily  $S'$  of  $S$  is called an *independent arc family* (abbreviated to an IAF) if any two arcs in  $S'$  are independent from each other. A *maximum independent arc family* (abbreviated to an MIAF) of  $S$  is an IAF whose cardinality is the largest among all IAF's of  $S$ . For example, the family of arcs shown in Fig. 1 has two MIAF's,  $\{a_2, a_3, a_5, a_8\}$  and  $\{a_2, a_3, a_6, a_8\}$ . Clearly, the MIAF's of  $S$  and the maximum independent sets of  $G_S$  are in one-to-one correspondence. In the following section, we will present an algorithm for finding an MIAF of a family of arcs.

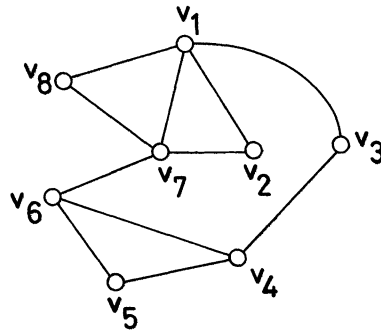


FIG. 3. The circular-arc graph for the family of arcs in Fig. 1.

**3. Outline of the algorithm.** Let  $S = \{a_1, a_2, \dots, a_n\}$  be a family of  $n$  arcs on a circle  $C$ .  $S$  is said to be *canonical* if (i)  $h_i$ 's and  $t_i$ 's for  $i = 1, 2, \dots, n$  are all distinct integers between 1 and  $2n$ , and (ii) point 1 is the head of arc  $a_1$ . For instance, the family of arcs shown in Fig. 1 is canonical, but the one given in Fig. 4 is not. It should be noted, however, that these two families of arcs correspond to the same circular-arc graph, which is shown in Fig. 3. When  $S$  is not canonical, by using a regular sorting

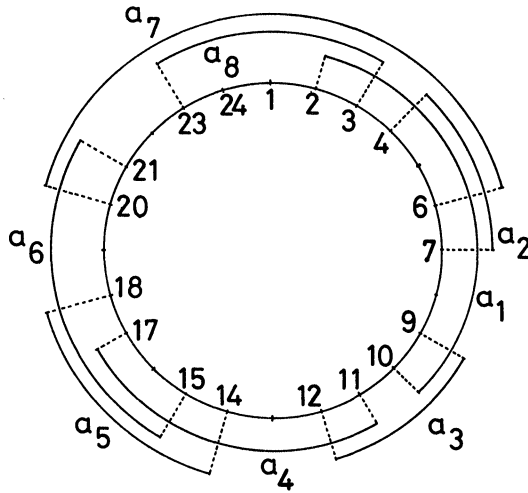


FIG. 4. A noncanonical family of arcs.

algorithm [1], one can construct, in  $O(n \cdot \log n)$  time, a family of arcs  $S'$  such that  $G_S = G_{S'}$ . Throughout this paper, we assume that the family  $S$  is canonical.

For a subfamily  $S'$  of  $S$ , let  $\alpha(S')$  denote the cardinality of a maximum independent set of  $G_{S'}$ , or equivalently that of an MIAF of  $S'$ . We start with the following theorem.

**THEOREM 1.** *Suppose that an arc  $a_i \in S$  contains another arc  $a_j \in S$ . Then, any MIAF of  $S - \{a_i\}$  is an MIAF of  $S$ .*

*Proof.* It is clear that  $\alpha(S) \geq \alpha(S - \{a_i\})$ . Let  $X$  be an MIAF of  $S$ . If  $a_i \notin X$ , then  $X$  is an IAF of  $S - \{a_i\}$ . On the other hand, if  $a_i \in X$ , then  $(X - \{a_i\}) \cup \{a_j\}$  is an IAF of  $S - \{a_i\}$ . These imply that  $\alpha(S) \leq \alpha(S - \{a_i\})$ . Thus,  $\alpha(S) = \alpha(S - \{a_i\})$ , and hence any MIAF of  $S - \{a_i\}$  is an MIAF of  $S$ .  $\square$

An arc  $a_i = (h_i, t_i) \in S$  is called a *forward arc* if  $h_i < t_i$ ; otherwise  $a_i$  is called a *backward arc*. For example, there are two backward arcs,  $a_7$  and  $a_8$ , in the family of arcs of Fig. 1. In our algorithm, we first remove all forward arcs which contain other forward arcs. Let  $S_F$  denote the resultant family of the forward arcs, and let  $S_B$  denote the family of all backward arcs in  $S$ . Then, we have the following lemma and theorem.

**LEMMA 1.**  $S_F \neq \emptyset$ .

*Proof.* Since  $S$  is canonical, it has at least one forward arc, that is,  $a_1$ . If  $a_1$  is the only forward arc in  $S$ ,  $S_F = \{a_1\} \neq \emptyset$ . On the other hand, if  $S$  has more than one forward arc, there exists at least one forward arc which does not contain any other forward arc. Thus,  $S_F \neq \emptyset$  in either case.  $\square$

**THEOREM 2.**  $1 \leq \alpha(S_F) \leq \alpha(S) \leq \alpha(S_F) + 1$ .

*Proof.* Since  $S_F \neq \emptyset$  from Lemma 1 and  $S_F \subseteq S$ ,  $1 \leq \alpha(S_F) \leq \alpha(S)$ . From Theorem 1,  $\alpha(S) = \alpha(S_F \cup S_B)$ , and hence  $\alpha(S) \leq \alpha(S_F) + \alpha(S_B)$ . Furthermore, it is clear that  $\alpha(S_B) = 1$  if  $S_B \neq \emptyset$ , and that  $\alpha(S_B) = 0$  if  $S_B = \emptyset$ . Therefore, the theorem holds.  $\square$

Our algorithm tests whether there exist an MIAF,  $X$  of  $S_F$  and an arc  $a_j \in S_B$  such that  $X \cup \{a_j\}$  is an IAF. From Theorem 2, if such an MIAF,  $X$  and an arc  $a_j$  exist, then  $X \cup \{a_j\}$  is an MIAF of  $S$ ; otherwise any MIAF of  $S_F$  is an MIAF of  $S$ . In general, the number of MIAF's of  $S_F$  may be an exponential function of  $|S_F|$ . However, our algorithm efficiently performs this test by exploiting a property of an MIAF of  $S$  which will be described later in Theorem 3.

Let  $X = \{a_{i_1}, a_{i_2}, \dots, a_{i_k}\}$  with  $h_{i_1} < t_{i_1} < h_{i_2} < t_{i_2} < \dots < h_{i_k} < t_{i_k}$  be an IAF of  $S_F$ . Then we call  $h_{i_1}$  (resp.,  $t_{i_k}$ ) the *starting coordinate* (resp., the *ending coordinate*) of  $X$

and denote it by  $sc(X)$  (resp.,  $ec(X)$ ). Let  $X_1$  and  $X_2$  be two IAF's of  $S_F$ . We say that  $X_1$  dominates  $X_2$  if one of the following conditions holds:

- (i)  $sc(X_2) < sc(X_1)$  and  $ec(X_1) \leq ec(X_2)$ .
- (ii)  $sc(X_2) \leq sc(X_1)$  and  $ec(X_1) < ec(X_2)$ .

An MIAF,  $X$  of  $S_F$  is called a *dominant maximum independent arc family* (abbreviated to a DMIAF) of  $S_F$  if no other MIAF of  $S_F$  dominates  $X$ .

LEMMA 2. *Let  $X$  be an MIAF of  $S_F \cup S_B$ . If there exists an MIAF,  $X_1$  of  $S_F$  which dominates  $X \cap S_F$ , then  $X_1 \cup (X \cap S_B)$  is an MIAF of  $S$ .*

*Proof.* Suppose that  $X \cap S_B = \emptyset$ . Then  $\alpha(S_F) = \alpha(S_F \cup S_B) = \alpha(S)$ . Since  $|X_1| = \alpha(S_F)$ ,  $X_1 \cup (X \cap S_B) = X_1$  is an MIAF of  $S$ . On the other hand, suppose that  $X \cap S_B \neq \emptyset$ . Since  $\alpha(S_B) \leq 1$ ,  $|X \cap S_B| = 1$ . Let  $a_i$  be the unique element in  $X \cap S_B$ . Then, it is clear that  $t_i < sc(X - \{a_i\})$  and  $ec(X - \{a_i\}) < h_i$ . Since  $X_1$  dominates  $X \cap S_F = X - \{a_i\}$ ,  $sc(X \cap S_F) \leq sc(X_1)$  and  $ec(X_1) \leq ec(X \cap S_F)$ . Therefore, we have  $t_i < sc(X_1)$  and  $ec(X_1) < h_i$  (see Fig. 5). This implies that  $X_1 \cup \{a_i\}$  is an IAF of  $S$ . Since  $|X_1 \cup \{a_i\}| = \alpha(S_F) + 1$ ,  $X_1 \cup (X \cap S_B) = X_1 \cup \{a_i\}$  is an MIAF of  $S$  from Theorem 2.  $\square$

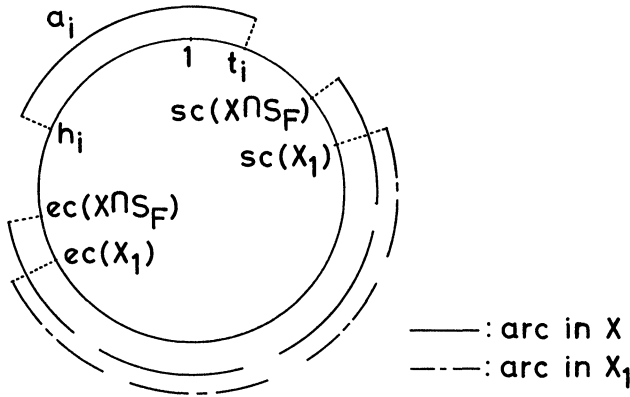


FIG. 5. An illustration for the proof of Lemma 2.

LEMMA 3. *Let  $X$  be an MIAF of  $S_F \cup S_B$ . Then, there exists a DMIAF,  $X_1$  of  $S_F$  such that  $X_1 \cup (X \cap S_B)$  is an MIAF of  $S$ .*

*Proof.* If  $X \cap S_F$  is a DMIAF of  $S_F$ , then the theorem trivially holds. Otherwise, there exists a DMIAF of  $S_F$ , say  $X_2$ , which dominates  $X \cap S_F$ . From Lemma 2,  $X_2 \cup (X \cap S_B)$  is an MIAF of  $S$ .  $\square$

Lemma 3 implies that it suffices to consider DMIAF's of  $S_F$  in order to test whether there exist an MIAF,  $X$  of  $S_F$  and an arc  $a_j \in S_B$  such that  $X \cup \{a_j\}$  is an IAF of  $S$ . In the following, we will show that the space which must be searched to find a desired MIAF of  $S_F$  can be reduced further.

The next lemma is obvious from the definition of a DMIAF.

LEMMA 4. *Let  $X_1$  and  $X_2$  be two DMIAF's of  $S_F$ . Then,  $sc(X_1) = sc(X_2)$  if and only if  $ec(X_1) = ec(X_2)$ . Furthermore,  $sc(X_1) < sc(X_2)$  if and only if  $ec(X_1) < ec(X_2)$ .  $\square$*

Let  $D$  be the set of all DMIAF's of  $S_F$ . A subset of  $D$ ,  $R = \{X_1, X_2, \dots, X_k\}$  is called an *essential DMIAF set* for  $S_F$  if it satisfies the following two conditions:

- (i)  $sc(X_i) \neq sc(X_j)$  for any  $1 \leq i \neq j \leq k$ .
- (ii) For any DMIAF,  $X$  of  $S_F$ , there exists an integer  $j$  such that  $1 \leq j \leq k$  and  $sc(X_j) = sc(X)$ .

Let us consider the family  $S$  of arcs shown in Fig. 6. In this particular case,  $S_F$  consists of eight arcs  $a_1, a_2, \dots, a_8$ , and has ten MIAF's:  $\{a_1, a_4, a_7\}$ ,  $\{a_1, a_4, a_8\}$ ,  $\{a_1, a_5, a_8\}$ ,

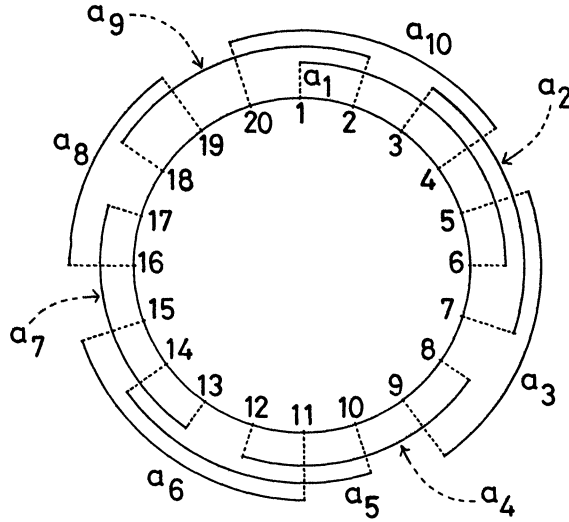


FIG. 6. A canonical family of arcs  $S$ .  $\{\{a_2, a_4, a_7\}, \{a_3, a_5, a_8\}\}$  and  $\{\{a_2, a_4, a_7\}, \{a_3, a_6, a_8\}\}$  are essential DMIAF sets for  $S_F$ .

$\{a_1, a_6, a_8\}$ ,  $\{a_2, a_4, a_7\}$ ,  $\{a_2, a_4, a_8\}$ ,  $\{a_2, a_5, a_8\}$ ,  $\{a_2, a_6, a_8\}$ ,  $\{a_3, a_5, a_8\}$  and  $\{a_3, a_6, a_8\}$ . Among them,  $\{a_2, a_4, a_7\}$ ,  $\{a_3, a_5, a_8\}$  and  $\{a_3, a_6, a_8\}$  are DMIAF's. Since the last two have the same starting coordinate, each of the sets  $\{\{a_2, a_4, a_7\}, \{a_3, a_5, a_8\}\}$  and  $\{\{a_2, a_4, a_7\}, \{a_3, a_6, a_8\}\}$  is an essential DMIAF set for  $S_F$ .

From Lemma 3 and the first half of Lemma 4, we have the following theorem.

**THEOREM 3.** *Let  $X$  be an MIAF of  $S_F \cup S_B$ , and let  $R_F$  be an essential DMIAF set for  $S_F$ . Then, there exists a DMIAF,  $X_1 \in R_F$  such that  $X_1 \cup (X \cap S_B)$  is an MIAF of  $S$ .  $\square$*

We now show the framework of our algorithm. Its correctness follows directly from Theorems 2 and 3.

#### ALGORITHM FIND-MIAF.

**Input:** A canonical family of arcs  $S = \{a_1, a_2, \dots, a_n\}$ .

**Output:** An MIAF of  $S$ .

**Method:**

1. Determine  $S_F$  and  $S_B$ .
2. Find an essential DMIAF set for  $S_F$ ,  $R_F = \{X_1, X_2, \dots, X_k\}$ .
3. If there exist a DMIAF  $X_i \in R_F$  and an arc  $a_j \in S_B$  such that  $t_j < sc(X_i)$  and  $ec(X_i) < h_j$ , then generate  $X_i \cup \{a_j\}$ ; otherwise generate an arbitrary MIAF of  $S_F$ .  $\square$

In the next section, we give a linear time implementation of this algorithm.

#### 4. Efficient implementation of the algorithm.

**4.1. Determination of  $S_F$  and  $S_B$ .** Suppose that a canonical family of arcs  $S = \{a_1, a_2, \dots, a_n\}$  is given as an input to Algorithm FIND-MIAF. Let  $S'_F$  denote the family of all forward arcs in  $S$ . Recall that  $S_F$  is the family of all forward arcs in  $S$  which do not contain any other forward arc in  $S$ . Note that  $S = S'_F \cup S_B$  and  $S'_F \cap S_B = \emptyset$ . We can partition  $S$  into  $S'_F$  and  $S_B$  in  $O(n)$  time. In order to extract  $S_F$  from  $S'_F$ , we

first create an empty queue  $Q$  and initialize  $S_F$  to be empty. Then we visit the endpoints of the arcs in  $S'_F$  one by one in ascending order of their coordinates. If we find the head of some arc  $a_i$ , we insert integer  $i$  into  $Q$ . If we find the tail of some arc  $a_i$ , we check whether integer  $i$  exists in  $Q$  or not. If it does not, we do nothing; otherwise we delete from  $Q$  the integer  $i$  and all integers which are placed before  $i$ , and then we add arc  $a_i$  to  $S_F$ .

Suppose that an integer  $j$  is deleted when the tail of  $a_i (i \neq j)$  is visited. Since the tail of  $a_j$  has not been visited,  $t_i < t_j$ . Furthermore,  $h_j < h_i$  since the integers are inserted into  $Q$  in ascending order of the coordinates of the heads of the corresponding arcs. These imply that  $a_j$  contains  $a_i$ . Thus,  $a_j \notin S_F$ . On the other hand, arc  $a_i$  does not contain any other forward arc; otherwise the tail of some arc  $a_k$  such that  $h_k > h_i$  would have been visited earlier than that of  $a_i$  and integer  $i$  would have already been deleted. Therefore,  $a_i \in S_F$ .

Since  $S$  is canonical, the coordinates of the endpoints of the arcs in  $S'_F$  are distinct integers between 1 and  $2n$ . Therefore, this procedure determines  $S_F$  in  $O(n)$  time and with  $O(n)$  space. Thus, the next theorem follows.

**THEOREM 4.** *Determination of  $S_F$  and  $S_B$  can be done in  $O(n)$  time and with  $O(n)$  space.  $\square$*

**4.2. Finding an essential DMIAF set for  $S_F$ .** Suppose that  $S_F$  has been obtained. Without loss of generality, we assume that  $S_F = \{a_1, a_2, \dots, a_{|S_F|}\}$  with  $h_1 < h_2 < \dots < h_{|S_F|}$ . (The renumbering of the subscripts of the arcs can be performed in  $O(n)$  time by a bucket sort [1], if necessary.) The following lemma provides an important property of an essential DMIAF set for  $S_F$ .

**LEMMA 5.** *Let  $R_F = \{X_1, X_2, \dots, X_k\}$  be an essential DMIAF set for  $S_F$ . Then,  $X_i \cap X_j = \emptyset$  for  $1 \leq i \neq j \leq k$ .*

*Proof.* Let  $i$  and  $j$  be integers such that  $1 \leq i \neq j \leq k$ . By definition,  $X_i$  and  $X_j$  are DMIAF's of  $S_F$ . Without any loss of generality, we can assume that  $sc(X_i) < sc(X_j)$  and  $ec(X_i) < ec(X_j)$ .

Assume that  $X_i \cap X_j \neq \emptyset$ , and let  $a_k$  be an element in  $X_i \cap X_j$ . Let  $X_i^-$  and  $X_i^+$  denote  $\{a_q \in X_i | t_q < h_k\}$  and  $\{a_q \in X_i | h_q > t_k\}$ , respectively. Similarly, let  $X_j^-$  and  $X_j^+$  denote  $\{a_q \in X_j | t_q < h_k\}$  and  $\{a_q \in X_j | h_q > t_k\}$ , respectively. Then, clearly  $X_j^- \cup \{a_k\} \cup X_i^+$  is an IAF of  $S_F$ . This implies that  $|X_j^-| \geq |X_i^+|$  since  $X_j = X_j^- \cup \{a_k\} \cup X_j^+$  is an MIAF of  $S_F$ . Similarly, since  $X_i^- \cup \{a_k\} \cup X_j^+$  is an IAF of  $S_F$  and  $X_i$  is an MIAF of  $S_F$ , we have  $|X_j^+| \geq |X_i^-|$ . Therefore,  $|X_j^+| = |X_i^-|$ . This implies that  $X_j^- \cup \{a_k\} \cup X_i^+$  is an MIAF of  $S_F$  since it is an IAF of  $S_F$  and  $|X_j^- \cup \{a_k\} \cup X_i^+| = |X_j|$ . It is clear that  $sc(X_i) < sc(X_j) = sc(X_j^- \cup \{a_k\} \cup X_i^+)$  and  $ec(X_i) = ec(X_j^- \cup \{a_k\} \cup X_i^+) < ec(X_j)$ . Therefore,  $X_j^- \cup \{a_k\} \cup X_i^+$  dominates both  $X_i$  and  $X_j$ . This contradicts the facts that  $X_i$  and  $X_j$  are DMIAF's of  $S_F$ . Consequently,  $X_i \cap X_j = \emptyset$ .  $\square$

**COROLLARY 1.** *Let  $R_F = \{X_1, X_2, \dots, X_k\}$  be an essential DMIAF set for  $S_F$ . Then  $|X_1| + |X_2| + \dots + |X_k| \leq |S_F|$ .*

*Proof.* It is clear from Lemma 5.  $\square$

Let  $Z$  be defined as  $\{a_i \in S_F | h_1 \leq h_i < t_1\}$ . Then, the following lemma is obtained.

**LEMMA 6.** *For any MIAF,  $X$  of  $S_F$ ,  $|X \cap Z| = 1$ .*

*Proof.* Any two arcs in  $Z$  intersect with each other, and hence  $|X \cap Z| \leq 1$ . Furthermore,  $|X \cap Z| \neq 0$ ; otherwise,  $X \cup \{a_1\}$  would be an IAF of  $S_F$ . Therefore,  $|X \cap Z| = 1$ .  $\square$

For an IAF of  $S_F$ ,  $X = \{a_{i_1}, a_{i_2}, \dots, a_{i_j}\}$  with  $h_{i_1} < h_{i_2} < \dots < h_{i_j}$ ,  $a_{i_1}$  is called the *starting arc* of  $X$ . For each arc  $a_i \in S_F$ , an IAF containing  $a_i$  as its starting arc is called a *largest IAF* for  $a_i$  if it contains the maximum number of arcs. Then we define  $YS_i$

as the set of all largest IAF's for  $a_i$ . For example, consider the family of arcs of Fig. 6 again. Then,  $YS_1 = \{\{a_1, a_4, a_7\}, \{a_1, a_4, a_8\}, \{a_1, a_5, a_8\}, \{a_1, a_6, a_8\}\}$  and  $YS_3 = \{\{a_3, a_5, a_8\}, \{a_3, a_6, a_8\}\}$ .

For  $i = 1, 2, \dots, |S_F|$ , let  $Y_i$  be an IAF in  $YS_i$  whose ending coordinate is the minimum among all IAF's in  $YS_i$ . Suppose that  $Z = \{a_1, a_2, \dots, a_m\}$ . By assumption,  $h_1 < h_2 < \dots < h_m$ . We show below several theorems and lemmas which play important roles in finding an essential DMIAF set for  $S_F$ .

**LEMMA 7.** *Let  $X$  be a DMIAF of  $S_F$  and let  $a_i$  be its starting arc. Then,  $1 \leq i \leq m$  and  $Y_i$  is a DMIAF of  $S_F$ .*

*Proof.* From Lemma 6,  $a_i \in Z$ , that is,  $1 \leq i \leq m$ . It is clear that  $X$  is a largest IAF for  $a_i$ . Therefore,  $|Y_i| = |X|$ , and hence  $Y_i$  is an MIAF of  $S_F$ . Furthermore, since  $Y_i$  has the minimum ending coordinate among all IAF's in  $YS_i$  and  $X$  is not dominated by  $Y_i$ ,  $ec(Y_i) = ec(X)$ . These imply that  $Y_i$  is a DMIAF of  $S_F$ .  $\square$

**THEOREM 5.** *Let  $R$  be the set of all DMIAF's in  $\{Y_1, Y_2, \dots, Y_m\}$ . Then,  $R$  is an essential DMIAF set for  $S_F$ .*

*Proof.* Let  $X$  be any DMIAF of  $S_F$ . Then, from Lemma 7, a DMIAF of  $S_F$ ,  $Y_i$  exists in  $R$  such that  $1 \leq i \leq m$  and  $sc(X) = sc(Y_i)$ . Therefore, there exists a subset of  $R$  which is an essential DMIAF set for  $S_F$ . Since  $h_1 < h_2 < \dots < h_m$ ,  $sc(Y_1) < sc(Y_2) < \dots < sc(Y_m)$ . This implies that no two DMIAF's in  $R$  have the same starting coordinate. Thus,  $R$  itself is an essential DMIAF set for  $S_F$ .  $\square$

**LEMMA 8.** *Suppose that  $Y_i$  is not an MIAF of  $S_F$  for some integer  $i$  such that  $1 \leq i < m$ . Then,  $Y_j$  is not an MIAF of  $S_F$  for  $j = i+1, i+2, \dots, m$ .*

*Proof.* Assume that  $Y_k$  is an MIAF of  $S_F$  for some  $k$  such that  $i+1 \leq k \leq m$ . Then  $|Y_k| > |Y_i| \geq 1$ , and hence  $|Y_k| \geq 2$ . Since  $h_i < h_k$  and  $a_i$  does not contain  $a_k$ ,  $t_i < t_k$ . Furthermore, it is clear that  $t_k < sc(Y_k - \{a_k\})$ . Therefore,  $\{a_i\} \cup (Y_k - \{a_k\})$  is an MIAF of  $S_F$ . Since its starting arc is  $a_i$ , every IAF in  $YS_i$  is an MIAF of  $S_F$ . This contradicts the hypothesis that  $Y_i$  is not an MIAF of  $S_F$ . Therefore, there does not exist such an integer  $k$  that  $i+1 \leq k \leq m$  and that  $Y_k$  is an MIAF of  $S_F$ .  $\square$

**COROLLARY 2.**  $Y_1$  is an MIAF of  $S_F$ .

*Proof.* It is clear from Lemmas 7 and 8.  $\square$

**LEMMA 9.** *Suppose that  $Y_i$  is an MIAF of  $S_F$  for some integer  $i$  such that  $1 \leq i \leq m$ . Then, if  $Y_i$  is not a DMIAF of  $S_F$ , there exists an integer  $j$  such that  $i < j \leq m$  and that  $Y_j$  is a DMIAF of  $S_F$  which dominates  $Y_i$ .*

*Proof.* If  $Y_i$  is not a DMIAF of  $S_F$ , then there exists a DMIAF of  $S_F$ , say  $X$ , which dominates  $Y_i$ . By definition,  $Y_i$  has the smallest ending coordinate among all largest IAF's for  $a_i$ , and hence  $a_i \notin X$ . This implies that  $h_i = sc(Y_i) < sc(X)$  and  $ec(X) \leq ec(Y_i)$ . Let  $a_j$  be the starting arc of  $X$ . Then, from Lemma 7,  $j \leq m$  and  $Y_j$  is a DMIAF of  $S_F$ . Since  $sc(X) = h_j$ ,  $h_i < h_j$ , and hence  $i < j \leq m$ . Furthermore, since  $sc(Y_j) = sc(X)$ ,  $ec(Y_j) = ec(X)$  from Lemma 4. Therefore,  $Y_j$  dominates  $Y_i$ .  $\square$

**COROLLARY 3.** *If  $Y_m$  is an MIAF of  $S_F$ , then it is a DMIAF of  $S_F$ .*

*Proof.* It is clear from Lemma 9.  $\square$

**COROLLARY 4.** *Suppose that  $Y_i$  is an MIAF of  $S_F$  for some integer  $i$  such that  $1 \leq i < m$ . Then, if  $Y_{i+1}$  is not an MIAF of  $S_F$ ,  $Y_i$  is a DMIAF of  $S_F$ .*

*Proof.* It is clear from Lemmas 8 and 9.  $\square$

**LEMMA 10.** *Suppose that both  $Y_i$  and  $Y_{i+1}$  are MIAF's of  $S_F$  for some integer  $i$  such that  $1 \leq i < m$ . Then,  $Y_i$  is a DMIAF of  $S_F$  if and only if  $ec(Y_i) < ec(Y_{i+1})$ .*

*Proof.* Clearly  $h_i = sc(Y_i) < sc(Y_{i+1}) = h_{i+1}$ . So, if  $ec(Y_i) \geq ec(Y_{i+1})$ , then  $Y_{i+1}$  dominates  $Y_i$ . Thus, if  $Y_i$  is a DMIAF of  $S_F$ , then  $ec(Y_i) < ec(Y_{i+1})$ .

Suppose that  $Y_i$  is not a DMIAF of  $S_F$ . From Lemma 9, there exists an integer  $j$  such that  $i+1 \leq j \leq m$  and that  $Y_j$  is a DMIAF of  $S_F$  which dominates  $Y_i$ . Since  $a_i$

does not contain  $a_j$  and  $Y_j$  dominates  $Y_i$ ,  $|Y_i| \geq 2$ , and hence  $|Y_j| \geq 2$ . Furthermore, if  $j \neq i+1$ ,  $t_{i+1} < t_j$ , and hence  $t_{i+1} < sc(Y_j - \{a_j\})$ . This inequality also holds when  $j = i+1$ . Therefore,  $ec(Y_{i+1}) \leq ec(Y_j - \{a_j\})$ ; otherwise  $\{a_{i+1}\} \cup (Y_j - \{a_j\})$  would be selected as  $Y_{i+1}$ . Since  $ec(Y_j - \{a_j\}) = ec(Y_j) \leq ec(Y_i)$ , we have  $ec(Y_{i+1}) \leq ec(Y_i)$ . Thus, if  $ec(Y_i) < ec(Y_{i+1})$ , then  $Y_i$  is a DMIAF of  $S_F$ .  $\square$

**THEOREM 6.** *Suppose that  $Y_i$  is an MIAF of  $S_F$  for some integer  $i$  such that  $1 \leq i \leq m$ . Then,  $Y_i$  is a DMIAF of  $S_F$  if and only if one of the following three conditions holds:*

- (i)  $i < m$  and  $Y_{i+1}$  is not an MIAF of  $S_F$ .
- (ii)  $i < m$  and  $ec(Y_i) < ec(Y_{i+1})$ .
- (iii)  $i = m$ .

*Proof.* It is clear from Lemma 10 and Corollaries 3 and 4.  $\square$

We now present a procedure to find an essential DMIAF set for  $S_F$ . Its correctness can be easily proven by using Theorems 5 and 6, Lemma 8 and Corollary 2.

**PROCEDURE FIND-EDS.**

1.  $Z \leftarrow \{a_i \in S_F \mid h_1 \leq h_i < t_1\}$ . Suppose that  $Z = \{a_1, a_2, \dots, a_m\}$ .
2. For  $i = 1, 2, \dots, m$ , find  $ec(Y_i)$  and  $|Y_i|$ , where  $Y_i$  is defined as before.
3.  $R \leftarrow \emptyset$ .  $i \leftarrow 1$ .
4. While  $|Y_i| = |Y_1|$  and  $i < m$ , execute the following instructions 1 and 2.
  - (1) If  $|Y_{i+1}| < |Y_1|$  or  $ec(Y_i) < ec(Y_{i+1})$ , then determine  $Y_i$  and  $R \leftarrow R \cup \{Y_i\}$ .
  - (2)  $i \leftarrow i+1$ .
5. If  $i = m$  and  $|Y_m| = |Y_1|$ , then determine  $Y_m$  and  $R \leftarrow R \cup \{Y_m\}$ .
6. Generate  $R$ .  $\square$

As an example, consider the family of arcs of Fig. 6. Since  $t_1 = 6$ ,  $Z$  is determined as  $\{a_1, a_2, a_3\}$  at Step 1. Then, Step 2 finds  $ec(Y_1) = 17$ ,  $|Y_1| = 3$ ;  $ec(Y_2) = 17$ ,  $|Y_2| = 3$ ; and  $ec(Y_3) = 19$ ,  $|Y_3| = 3$ . At Step 4,  $Y_1$  is not added to  $R$  since  $|Y_2| = |Y_1| = 3$  and  $ec(Y_1) = ec(Y_2)$ . On the other hand,  $Y_2$  is added to  $R$  since  $ec(Y_2) < ec(Y_3)$ . Similarly,  $Y_3$  is added to  $R$  at Step 5. While  $Y_2$  is uniquely determined as  $\{a_2, a_4, a_7\}$ ,  $Y_3$  may be chosen from two candidates,  $\{a_3, a_5, a_8\}$  and  $\{a_3, a_6, a_8\}$ . Therefore, the resultant essential DMIAF set is either  $\{\{a_2, a_4, a_7\}, \{a_3, a_5, a_8\}\}$  or  $\{\{a_2, a_4, a_7\}, \{a_3, a_6, a_8\}\}$ .

In what follows, we describe an efficient implementation of Procedure FIND-EDS.

For each arc  $a_i \in S_F$ , let  $\text{NEXT}(a_i)$  be defined as an arc  $a_k$  such that  $h_k = \text{Min}\{h_j \mid a_j \in S_F \text{ and } h_j > t_i\}$  if  $\{a_j \in S_F \mid h_j > t_i\} \neq \emptyset$ , and otherwise defined as "null." For example, for the family of arcs of Fig. 6,  $\text{NEXT}(a_1) = a_4$ ,  $\text{NEXT}(a_2) = a_4$ ,  $\text{NEXT}(a_3) = a_5$ ,  $\text{NEXT}(a_4) = a_7$ ,  $\text{NEXT}(a_5) = a_8$ ,  $\text{NEXT}(a_6) = a_8$ ,  $\text{NEXT}(a_7) =$  "null," and  $\text{NEXT}(a_8) =$  "null."

For each arc  $a_i \in S_F$ , let  $N_i$  be defined as  $\{a_{i_1} = a_i, a_{i_2}, \dots, a_{i_k}\}$  such that  $\text{NEXT}(a_{i_j}) = a_{i_{j+1}}$  for  $j = 1, 2, \dots, k-1$  and  $\text{NEXT}(a_{i_k}) =$  "null." Then we have the following lemma.

**LEMMA 11.** *For each arc  $a_i \in Z$ ,  $N_i$  is an IAF of  $S_F$ . Furthermore,  $|Y_i| = |N_i|$  and  $ec(Y_i) = ec(N_i)$ .*

*Proof.* It is clear from the definitions that  $N_i$  is an IAF of  $S_F$ . Suppose that  $N_i = \{a_{i_1} = a_i, a_{i_2}, \dots, a_{i_k}\}$  with  $h_{i_1} < h_{i_2} < \dots < h_{i_k}$  and  $Y_i = \{a_{i_1} = a_i, a_{i_2}, \dots, a_{i_j}\}$  with  $h_{i_1} < h_{i_2} < \dots < h_{i_j}$ . By definition,  $Y_i$  is a largest IAF for  $a_i$ , and hence  $j \geq k$ . Since no arc in  $S_F$  contains any other arc in  $S_F$ , we can show that  $h_{i_p} \leq h_{i'_p}$  and  $t_{i_p} \leq t_{i'_p}$  for  $p = 1, 2, \dots, k$  by an easy induction proof on the value of  $p$ . Thus,  $t_{i_k} \leq t_{i'_k}$ , and hence  $ec(N_i) \leq ec(Y_i)$ . Since  $\text{NEXT}(a_{i_k}) =$  "null,"  $\text{NEXT}(a_{i'_k}) =$  "null." This implies that  $k = j$ , that is,  $|N_i| = |Y_i|$ . This further implies from the definition of  $Y_i$  that  $ec(N_i) \leq ec(Y_i)$ . Therefore,  $ec(N_i) = ec(Y_i)$ .  $\square$

For each arc  $a_i \in S_F$ ,  $\text{NEXT}(a_i)$  can be determined as follows. We first create an empty set  $P$ , and then start visiting the endpoints of the arcs of  $S_F$  in ascending order of their coordinates. Suppose we find the head of some arc  $a_j$ . If  $P$  is empty, we do nothing. On the other hand, if  $P$  is not empty, we set  $\text{NEXT}(a_k)$  to  $a_j$  for each element  $a_k$  in  $P$  and then delete all such elements from  $P$ . If we find the tail of some arc  $a_j$  which is not the last endpoint, we add  $a_j$  to  $P$ . If the tail is the last endpoint, we set  $\text{NEXT}(a_k)$  to “null” for each element  $a_k$  in  $P \cup \{a_j\}$ . Since the coordinates of the endpoints of the arcs in  $S_F$  are distinct integers between 1 and  $2n$ , this procedure requires  $O(n)$  time and space.

We now define a digraph  $H_F$  as follows:

$$H_F = (W_F, A_F), \quad \text{where}$$

$$W_F = \{w_i | a_i \in S_F\} \quad \text{and}$$

$$A_F = \{(w_i \rightarrow w_j) | \text{NEXT}(a_i) = a_j\}.$$

As an example, Fig. 7 illustrates the graph  $H_F$  which corresponds to the family of arcs of Fig. 6.

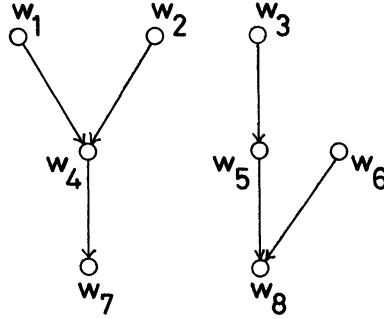


FIG. 7. The graph  $H_F$  for the family of arcs of Fig. 6.

Since the outdegree of each vertex in  $H_F$  is at most one,  $|A_F| < |S_F|$ . Thus, we have the following lemma.

LEMMA 12. *The construction of  $H_F$  with the aforementioned computation of  $\text{NEXT}(\cdot)$  requires  $O(n)$  time and space.  $\square$*

The next lemma is obvious from the definition of  $H_F$ .

LEMMA 13.  *$H_F$  has the following properties.*

- (i) *For each vertex  $w_i$  with outdegree 0,  $|N_i| = 1$  and  $ec(N_i) = t_i$ .*
- (ii) *For each edge  $(w_i \rightarrow w_j)$  in  $A_F$ ,  $|N_i| = |N_j| + 1$  and  $ec(N_i) = ec(N_j)$ .*
- (iii) *For each arc  $a_i \in Z$ , the maximal directed path in  $H_F$  starting from  $w_i$  corresponds to  $N_i$ .  $\square$*

According to Lemma 13 (i) and (ii), one can easily determine  $|N_i|$  and  $ec(N_i)$  for all arcs  $a_i \in S_F$  in  $O(|S_F|)$  time. From Lemma 11, for each arc  $a_i \in Z$ ,  $|Y_i| = |N_i|$  and  $ec(Y_i) = ec(N_i)$ . Therefore, the following lemma is obtained from Lemma 12.

LEMMA 14. *Step 2 of Procedure FIND-EDS requires  $O(n)$  time and space.  $\square$*

Steps 4 and 5 of Procedure FIND-EDS can be performed based on Lemma 11 and Lemma 13 (iii). Each time we find an integer  $i$  for which the conditions of Step 4 (i) or Step 5 are satisfied, we can determine  $Y_i$  by finding a maximal directed path in  $H_F$  starting from  $w_i$ . Therefore, Steps 4 and 5 can be executed in  $O(|Z| + \sum_{Y_i \in R} |Y_i|)$  time, where  $R$  is the essential DMIAF set which is obtained at the termination of the



procedure. Since  $\sum_{Y_i \in R} |Y_i| \leq |S_F|$  from Corollary 1, the following theorem is obtained from Lemma 14.

**THEOREM 7.** *An essential DMIAF set for  $S_F$  can be formed in  $O(n)$  time and with  $O(n)$  space.  $\square$*

**4.3. Determination of an MIAF of  $S$ .** Suppose that an essential DMIAF set for  $S_F$ ,  $R_F = \{X_1, X_2, \dots, X_k\}$  has been obtained. For each arc  $a_j \in S_B$ , let  $\text{NEXT}(a_j)$  be defined as  $X_i$  such that  $sc(X_i) = \text{Min}\{sc(X_p) | X_p \in R_F \text{ and } sc(X_p) > t_j\}$  if  $\{X_p \in R_F | sc(X_p) > t_j\} \neq \emptyset$ , and otherwise defined as “null.” Then, the following theorem holds.

**THEOREM 8.** *Suppose that there exist an arc  $a_j \in S_B$  and a DMIAF,  $X_i \in R_F$  such that  $t_j < sc(X_i)$  and  $ec(X_i) < h_j$ . Then,  $t_j < sc(\text{NEXT}(a_j))$  and  $ec(\text{NEXT}(a_j)) < h_j$ .*

*Proof.* Assume that  $X_i \neq \text{NEXT}(a_j)$ . From the definition of  $\text{NEXT}(a_j)$ ,  $t_j < sc(\text{NEXT}(a_j)) < sc(X_i)$ . Therefore,  $ec(\text{NEXT}(a_j)) < ec(X_i)$  from Lemma 4, and hence  $ec(\text{NEXT}(a_j)) < h_j$ .  $\square$

By a procedure similar to the one for computing  $\text{NEXT}(\cdot)$  for the arcs in  $S_F$ , one can determine  $\text{NEXT}(a_j)$  for all arcs  $a_j \in S_B$  in  $O(n)$  time. In this case, after the creation of an empty set  $P$ , we visit the tails of the arcs in  $S_B$  and the heads of the starting arcs of the DMIAF’s in  $R_F$  in ascending order of their coordinates until the last head is visited. The other part of the procedure is almost the same as that of the previous one.

After finding  $\text{NEXT}(\cdot)$  for all arcs in  $S_B$ , according to Theorem 8, we can easily find, in  $O(|S_B|)$  time, an arc  $a_j \in S_B$  and a DMIAF,  $X_i \in R_F$  such that  $t_j < sc(X_i)$  and  $ec(X_i) < h_j$ , if they exist. If such an arc and a DMIAF do not exist, any DMIAF in  $R_F$  is an MIAF of  $S$ . Thus, we have the following theorem.

**THEOREM 9.** *Suppose that an essential DMIAF set for  $S_F$  has been obtained. Then, an MIAF of  $S$  can be obtained in  $O(n)$  time and with  $O(n)$  space.  $\square$*

**4.4. Time and space complexities of the algorithm.** We now show the following two theorems.

**THEOREM 10.** *The time and space complexities of Algorithm FIND-MIAF each are  $O(n)$ .*

*Proof.* It is clear from Theorems 4, 7 and 9.  $\square$

**THEOREM 11.** *Given a family  $S$  of  $n$  arcs on a circle, a maximum independent set of its corresponding circular-arc graph  $G_S$  can be found in  $O(n \cdot \log n)$  time and with  $O(n)$  space. These complexities are optimal to within a constant factor.*

*Proof.* As mentioned before, one can construct a canonical family of arcs  $S'$  such that  $G_S = G_{S'}$  in  $O(n \cdot \log n)$  time and with  $O(n)$  space. The application of Algorithm FIND-MIAF to the resultant family of arcs  $S'$  requires  $O(n)$  time and space due to Theorem 10. Therefore, we can find a maximum independent set of  $G_S$  in  $O(n \cdot \log n)$  time and with  $O(n)$  space.

Every interval graph is a circular-arc graph. Furthermore, it is known that it requires  $\Omega(n \cdot \log n)$  time in the worst case to find a maximum independent set of an interval graph when the graph is given in the form of a family of  $n$  intervals [7]. Therefore, our algorithm is time- and space-optimal to within a constant factor.  $\square$

**5. Conclusion.** In this paper, we have presented an optimal algorithm for finding a maximum independent set of a circular-arc graph. When the graph is given in the form of a family of  $n$  arcs on a circle, our algorithm runs in  $O(n \cdot \log n)$  time and with  $O(n)$  space. Moreover, it requires only  $O(n)$  time if the endpoints of the arcs are already sorted, in other words, if the order of their appearances on the circle is known.

It does not seem that our algorithm can be extended to the problem of finding a maximum weight independent set when each vertex is assigned a weight. It is interesting to develop an optimal algorithm for such a problem.

## REFERENCES

- [1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [2] K. S. BOOTH AND G. S. LUEKER, *Testing for consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms*, J. Comput. System Sci., 13 (1976), pp. 335-379.
- [3] M. R. GAREY, D. S. JOHNSON, G. L. MILLER AND C. H. PAPADIMITRIOU, *The complexity of coloring circular arcs and chords*, SIAM J. Algebraic Discrete Methods, 1 (1980), pp. 216-227.
- [4] F. GAVRIL, *Algorithms on circular-arc graphs*, Networks, 4 (1974), pp. 357-369.
- [5] M. C. GOLUBIC, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
- [6] U. I. GUPTA, D. T. LEE AND J. Y.-T. LEUNG, *An optimal solution for the channel assignment problem*, IEEE Trans. Comput., C-28 (1979), pp. 807-810.
- [7] ———, *Efficient algorithms for interval graphs and circular-arc graphs*, Networks, 12 (1982), pp. 459-467.
- [8] W.-L. HSU, *Maximum weight clique algorithms for circular-arc graphs and circle graphs*, this Journal, 14 (1985), pp. 224-231.
- [9] M. KEIL, *Finding Hamiltonian circuits in interval graphs*, Inform. Process. Lett., 20 (1985), pp. 201-206.
- [10] J. Y.-T. LEUNG, *Fast algorithms for generating all maximal independent sets of interval, circular-arc and chordal graphs*, J. Algorithms, 5 (1984), pp. 22-35.
- [11] G. S. LUEKER AND K. S. BOOTH, *A linear time algorithm for deciding interval graph isomorphism*, J. Assoc. Comput. Mach., 26 (1979), pp. 183-195.
- [12] T. OHTSUKI, H. MORI, E. S. KUH, T. KASHIWABARA AND T. FUJISAWA, *One dimensional logic gate assignment and interval graphs*, IEEE Trans. Circuits and Systems, CAS-26 (1979), pp. 675-684.
- [13] A. TUCKER, *An efficient test for circular-arc graphs*, this Journal, 9 (1980), pp. 1-24.
- [14] O. WING, S. HUANG AND R. WANG, *Gate matrix layout*, IEEE Trans. Computer-Aided Design, CAD-4 (1985), pp. 220-231.
- [15] O. J. YU AND O. WING, *Interval-graph-based PLA folding*, Proc. 1985 IEEE International Symposium on Circuits and Systems, Kyoto, Japan, 1985, pp. 1463-1466.

## ON THE COMPLEXITY OF COVERING VERTICES BY FACES IN A PLANAR GRAPH\*

DANIEL BIENSTOCK† AND CLYDE L. MONMA†

**Abstract.** The pair  $(G, D)$  consisting of a planar graph  $G = (V, E)$  with  $n$  vertices together with a subset of  $d$  special vertices  $D \subseteq V$  is called  $k$ -planar if there is an embedding of  $G$  in the plane so that at most  $k$  faces of  $G$  are required to cover all of the vertices in  $D$ . Checking 1-planarity can be done in linear-time since it reduces to a problem of checking planarity of a related graph. We present an algorithm which given a graph  $G$  and a value  $k$  either determines that  $G$  is not  $k$ -planar or generates an appropriate embedding and associated minimum cover in  $O(c^k n)$  time, where  $c$  is a constant. Hence, the algorithm runs in linear time for any fixed  $k$ . The fact that the time required by the algorithm grows exponentially in  $k$  is to be expected since we also show that for arbitrary  $k$ , the associated decision problem is strongly NP-complete, even when the planar graph has essentially a unique planar embedding,  $d = \theta(n)$ , and all facial cycles have bounded length. These results provide a polynomial-time recognition algorithm for special cases of Steiner tree problems in graphs which are solvable in polynomial time.

**Key words.** complexity, planar graphs, Steiner trees

**AMS(MOS) subject classifications.** 05, 68

**1. Introduction.** Recently, there has been a great deal of interest in solving the Steiner tree problem in graphs. This problem is NP-complete even for planar grid graphs [GJ1]. (See [GJ2] for an excellent introduction to the area of computational complexity.) So recent work has centered on efficiently-solvable special cases and heuristic methods; see [Wi] for a survey of work on this problem.

Throughout this paper we deal with undirected graphs of the form  $G = (V, E)$ , where  $V$  is a set of  $n$  vertices and  $E$  is a set of edges connecting pairs of vertices. A graph is called *planar* if it can be embedded in the plane. A graph  $G = (V, E)$  together with  $d$  special vertices  $D \subseteq V$  is called  *$k$ -planar* if there is a planar embedding of  $G$  so that at most  $k$  faces of  $G$  are required to cover all of the vertices in  $D$ . Clearly, a planar graph is the same as an  $n$ -planar graph. The *planarity number* of  $G$  is the minimum  $k$  such that  $G$  is  $k$ -planar.

A recent paper by [EMV] presents an algorithm which solves the Steiner problem in an arbitrary graph; their algorithm runs in polynomial time for  $k$ -planar graphs, for any fixed  $k$ , with  $D$  being the vertices required to be in the Steiner tree. It is easy to see that checking 1-planarity of  $G = (V, E)$  with special vertices  $D \subseteq V$  is equivalent to testing the planarity of the associated graph  $G^* = (V^*, E^*)$ , where  $V^* = V \cup \{\sigma\}$  and  $E^* = E \cup \{(\sigma, v) : v \in D\}$ , and so can be done in linear time [HT2]. They leave as an open question the complexity of testing  $k$ -planarity for fixed  $k \geq 2$ .

In § 2, we present an algorithm which checks to see if a given  $(G, D)$  pair is  $k$ -planar given a *fixed* embedding of  $G$  and if so, determines the planarity number of  $G$  in  $O(c^k n)$  time, when  $c$  is a constant. This is used in § 3 to generate an appropriate embedding of  $G$  and a cover of  $D$  by  $k$  or fewer faces, if possible, in  $O(c^k n)$  time. Hence, the algorithm runs in linear time for any fixed  $k$ . The fact that the time required grows exponentially in  $k$  is to be expected as we show in § 4 that for arbitrary  $k$ , the associated decision problem is strongly NP-complete, even when the planar graph has essentially a unique planar embedding,  $d = \theta(n)$ , and all facial cycles have bounded length.

\* Received by the editors September 22, 1986; accepted for publication (in revised form) April 16, 1987.

† Bell Communications Research, Morristown, New Jersey 07960.

In § 5, we present an optimization algorithm which finds the minimum number of faces required to cover all special vertices of a planar graph with a fixed embedding in  $2^{O(\sqrt{d})}$  time. This exact algorithm is used to obtain a polynomial-time approximation algorithm which is asymptotically optimal (i.e., the relative error converges to zero), for the class of graphs we showed to be NP-complete in § 4.

**2. Testing  $k$ -planarity for a fixed embedding.** Consider a fixed embedding of the graph  $G = (V, E)$ . In this section we describe an algorithm that tests whether  $k$  faces are sufficient to cover all special vertices of  $D$  in this particular embedding and, if so, whether it determines the planarity number. This algorithm requires  $O(c^k n)$  time for some constant  $c$ , and is used as a subroutine in our  $k$ -planarity testing algorithm for a variable embedding in the next section. We note that if  $G$  is three-vertex connected, then  $G$  has essentially a unique embedding and so the results of this section apply.

Throughout this section, we assume that the embedding of  $G$  is fixed. We transform the problem of covering  $D$  with faces into one of covering certain special faces as follows. We transform each vertex of  $D$  into a polygon, that is, if  $v \in D$  has edges  $e_1, e_2, \dots, e_m$  incident on it, we replace  $v$  by a polygon with vertices  $v_1, \dots, v_m$  and edges  $(v_1, v_2), \dots, (v_m, v_1)$ ; such that for  $1 \leq i \leq m$ ,  $e_i$  becomes incident to  $v_i$  (if the degree of  $v$  is 2, the polygon is a face of length two). We will refer to the new graph by  $\hat{G}$ , and the set of faces enclosed by the new polygons will be called  $\hat{D}$ .

Let  $G' = (V', E')$  denote the dual graph of  $\hat{G}$ . The vertices of  $G'$  will be called points. The set of points of  $G'$  corresponding to  $\hat{D}$  will be called  $D'$ . Now our problem becomes that of testing whether  $G'$  contains a set  $X$  of points such that

- (i)  $X \cap D' = \emptyset$ ,
- (ii)  $X$  dominates  $D'$ ,
- (iii)  $|X| \leq k$ .

If such a set  $X$  exists, the corresponding set of faces  $\hat{G}$  will be called a face cover.

Let  $S$  denote the subgraph of  $G'$  induced by all points in  $D'$  and all points adjacent to some point in  $D'$ . Now if  $S$  has more than  $k$  connected components, then certainly no set  $X$  satisfying (i)–(iii) exists. Hence assume otherwise. We have the following result.

**LEMMA 1.** *If a set  $X$  satisfying (i)–(iii) exists, the diameter of every connected component of  $S$  is at most  $O(k)$ .*

*Proof.* Aiming for a contradiction, let  $C$  be a connected component of  $S$  with diameter larger than  $8k + 6$ , and let  $p = f_1, f_2, \dots, f_j$  be such a diameter. There are two cases.

(i)  $|p \cap D'| \geq 3(k + 1)$ . By construction, no point of  $D'$  is adjacent to another point of  $D'$ . Let  $p \cap D' = \{s_0, s_1, \dots, s_t\}$  with labeling to reflect the ordering of these points in  $p$ , and set  $Z = \{s_0, s_3, s_6, \dots, s_{3i}, s_{3(i+1)}, \dots\}$ . Then at least  $|Z|$  points are needed to dominate  $Z$ . But this is a contradiction since  $|Z| \geq 3(k + 1)/3 = k + 1$ .

(ii)  $|p \cap D'| < 3(k + 1)$ . By construction, every point of  $S - D'$  is adjacent to at least one point of  $D'$ . Let  $p - D' = \{g_0, \dots, g_r\}$  with labeling to reflect the ordering of these points in  $p$ , and set  $Y = \{g_0, g_5, g_{10}, \dots, g_{5i}, g_{5(i+1)}, \dots\}$ . For each  $i$ , set  $d_i$  to be an arbitrary point of  $D'$  adjacent to  $g_{5i}$ . Clearly, if  $i \neq j$ , then  $d_i \neq d_j$ . Further, a different point of  $S - D'$  is required to dominate each point  $d_i$ . But the number of such points is at least  $\frac{1}{5}(8k + 7 - 3k - 2) = k + 1$ , again a contradiction.  $\square$

Our algorithms will exploit this bounded dual diameter structure. The computations take place in the primal graph. Now, if a graph has dual diameter  $t$ , then every face is within distance  $t$  of an arbitrary face. Algorithm XTND given below will, when input an embedded planar graph  $H$  with  $n$  vertices, distinguished subset of faces  $E$ ,

and a constant  $L$ , test in linear time whether every face of  $H$  is within distance  $L$  of the outer face. If so, XTND will compute a minimum face cover of  $E$  in time bounded by  $2^{O(L)}n$ . In order to motivate XTND, we will describe it in three steps. First we consider a special type of planar graph called a Halin graph. Next, we analyze the structure of bounded diameter planar graphs. Finally, we describe XTND in the general case.

**2.1. Halin graphs.** An embedded planar graph  $H$  is called a *Halin graph* if its dual has a dominating set of cardinality one. Assume that the corresponding face of  $H$  is the outer face. Without loss of generality, the outer facial cycle  $C$  of  $H$  is simple. For if  $v$  is a multiple vertex of  $C$ , let  $e_1 = (u_1, v)$  and  $e_2 = (v, u_2)$  be two consecutive edges of  $C$ . Then we subdivide  $e_1$  and  $e_2$  by adding vertices  $w_1$  and  $w_2$ , and add the edge  $(w_1, w_2)$  embedded in the outer face. Clearly the new graph is still Halin. This type of operation is called a *patching*. Patchings can also be used to ensure that all vertices in  $C$  have degree two or three. Similarly, if an interior vertex of  $H$  has degree one, we can shrink the corresponding edge. The final graph  $H'$  we obtain will be the union of a cycle  $C$  and forest  $T$  embedded inside  $C$  with all leaves in  $C$ . Moreover, from a face cover problem in  $H$  we obtain an equivalent problem in  $H'$ . Now we need some definitions adapted from [CNP].

(1) A *level 1 fan* of  $H'$  is a maximal set of paths  $p_1, p_2, \dots, p_n$  with a common endpoint  $u \notin C$ , which are otherwise disjoint, with opposite endpoint in  $C$ , and all interior vertices of degree 2 in  $H'$ . The vertex  $u$  will be called the *center* of the fan.

(2) To define *level  $t$  fans*, for  $t > 1$ , we proceed as follows. As above, let  $p_1, \dots, p_m$  be a maximal set of paths with common endpoint  $u \notin C$ , otherwise disjoint, and with degree two interior vertices. Suppose that for  $1 \leq i \leq m$ , the endpoint of  $p_i$  different from  $u$  is the center of a level  $j_i$  fan, with  $j_i \leq t - 1$ , and that for some  $i$ ,  $j_i = t - 1$ . Then the collection of paths and fans is called a *level  $t$  fan*, and  $u$  its center. The fan whose center is the endpoint of  $p_i$  is called the *descendant fan* of  $p_i$ .

(3) If  $H'$  is the union of a level 1 fan and  $C$ , we say  $H'$  is a *wheel*.

**THEOREM 2** (Adapted from [CNP]). *The Halin graph  $H'$  has at least one level 1 fan, and this fan can be constructed in linear time.*  $\square$

This theorem was used in [CNP] to construct a polynomial-time dynamic programming algorithm for the traveling salesman problem in Halin graphs. We will make a similar use here towards the face cover problem.

The intuition behind the approach is the following: we can describe the properties of a fan using a bounded size list. This works because a level 1 fan has only two faces that share edges with other fans. When constructing the list for a level  $t$  fan, we only need to look separately at the lists for the descendant fans. Thus if  $F$  is a level  $t$  fan with  $m$  descendant fans, the list for  $F$  can be constructed in  $O(m)$  time. Altogether this translates into a linear time algorithm for solving the minimum face cover in  $H'$ . Details are provided next.

If the forest  $T$  located inside  $C$  is not a tree, we reduce this to the single tree case as follows. Let  $e_1, e_2$  be consecutive edges incident on vertices of  $C$ , that belong to different trees. Then subdivide  $e_i$  by introducing a new vertex  $w_i$ , for  $i = 1, 2$  and add the edge  $(w_1, w_2)$ . This new edge, together with the position of  $e_i$  between  $w_i$  and  $C$ ,  $i = 1, 2$ , and a segment of  $C$ , bounds a “new” face of  $H'$ . We make this face forbidden (that is, we cannot use it towards a cover).

Clearly this procedure does not change the problem and repeatedly applying it will merge the forest into a single tree. Now if the outer face of  $H'$  is in  $E$ , then any internal face of  $H'$  will cover it. On the other hand, if the outer face of  $H'$  is not in

$E$ , then it will cover every other face of  $H'$ , and this is obviously optimal. Hence we may assume, without loss of generality, that the outer face is forbidden.

Suppose  $f$  is a fan of  $H'$  with center  $u$  and paths  $p_1, \dots, p_m$  in clockwise order. For  $1 \leq i \leq m$  let  $\hat{p}_i$  be the *continuation* of  $p_i$ ; that is, a path of the form  $p_i, p'_i$  ending in  $C$ , such that  $p'_i$  is obtained by following the most counterclockwise path out of  $p_i$ , and for  $i > 1$ ,  $p'_i$  is obtained by following the most clockwise path out of  $p_i$ . The endpoints of  $\hat{p}_1$  and  $\hat{p}_m$  in  $C$  are called the *extremes* of  $f$ . For  $2 \leq i \leq m$ , denote by  $f_i$  the region bounded by  $\hat{p}_1, \hat{p}_i$  and the corresponding section of  $C$ ; and set  $E_i$  to be the subset of faces of  $E$  contained in  $f_i$ . For  $i = 1$ ,  $f_1$  consists of  $p_1$  and its descendant fan  $f$ , while  $E_1$  is the subset of  $E$  contained in  $f$ . Finally, define  $l(f, i, a_1, a_i)$  to be the minimum number of internal faces needed to cover  $E_i$ , with the constraints that:

(1) If  $a_1 = 0$ , at least one face of  $E_i$  that has an edge on  $\hat{p}_1$  has not been covered.  
 (2) If  $a_1 = 1$ , (1) does not apply and at least one face with an edge of  $\hat{p}_1$  is used in the cover.

(3) If  $a_1 = 2$ , neither (1) nor (2) apply.

(4) Similar considerations apply for  $a_i = 0, 1$  or  $2$ .

Algorithm XTND takes as input a graph  $H'$  and keeps track of an auxiliary graph  $A$ . At the start,  $A = H'$ . In general, the vertices of  $A$  in its outer facial cycle will correspond to fans of  $H'$ ; these vertices will be labeled by the corresponding fans. The output of Algorithm XTND is  $M$ , the minimum number of faces of  $H'$  needed to cover  $E$ . It is well known that repeatedly shrinking fans in a Halin graph eventually leads to a wheel.

ALGORITHM XTND (HALIN CASE).

- (1) Find a level 1 fan  $g$  in  $A$ . Let  $f$  be the corresponding fan in  $H'$ , with center  $u$ , defining paths  $p_1, \dots, p_m$  and continuations  $\hat{p}_1, \dots, \hat{p}_m$ . For  $1 \leq i \leq m$ , let  $h_i$  be the descendant fan of  $p_i$ , with  $s_i$  defining paths (if  $h_i$  is a vertex, set  $s_i = 0$ ).
  - (a) Set  $l(f, 1, x, y) = l(h_1, s_1, x, y)$  for all  $x, y$ .
  - (b) For  $i > 1$ , suppose first that the face between  $p_{i-1}$  and  $p_i$  is not in  $E$ , and it is not forbidden. Then, for example,

$$l(f, i, 2, 2) = \min_{x,y} \{ \min \{ l(f, i-1, 2, x) + 1 \\ + l(h_i, s_i, y, 2) \}, \min_{x>0, y>0} \{ l(f, i-1, 2, x) + l(h_i, s_i, y, 2) \} \}.$$

Similar formulas are used to compute all other parameters  $l(f, i, \cdot, \cdot)$ , and also when the face between  $p_{i-1}$  and  $p_i$  is forbidden, or if it is in  $E$ .

- (2) If  $g$  is a wheel in  $A$ , then let  $x$  be the face of  $H'$  between  $\hat{p}_m$  and  $\hat{p}_1$ .

$$(a) \text{ If } x \notin D, \text{ then } M = \min \left\{ \min_{x,y} \{ l(f, m, x, y) + 1 \}, \min_{x>0, y>0} \{ l(f, m, x, y) \} \right\}.$$

$$(b) \text{ If } x \in E, \text{ then } M = \min \left\{ \min_y \{ l(f, m, 1, y) \}, \min_x \{ l(f, m, x, 1) \} \right\}.$$

(c) Stop and output the value of  $M$ .

- (3) Otherwise, shrink  $g$  into a single vertex in the outer face of  $A$ , and go to (1).

Algorithm XTND clearly works correctly, and since the workload in finding and shrinking fans in  $A$  is linear in the size of each fan, the total complexity is linear.

Notice that there is a last center  $r$  of a fan found in graph  $A$ . In fact, it is not difficult to see that this vertex may be prescribed before running XTND, by choosing fans with center  $u \neq r$  whenever possible.

There are two observations concerning the Halin case that will be very useful later. First, let  $f$  be a fan of  $H'$  with center  $u$ , and consider a set of contiguous faces of  $f$ ; that is, a set  $S$  of faces of  $f$  incident to  $u$  that appear consecutively as we travel around  $f$ . Then if we want to force all faces in  $S$  to take the same *value* (that is, all chosen or rejected), Algorithm XTND given above can still be used, almost verbatim, to compute a minimum face cover of  $E$ . Similar considerations apply if all faces of  $S$  are in  $E$  and covering any of them is interpreted as covering *all* of them. In both cases we will refer to the set  $S$  as a *split face*.

Second, suppose  $e_1$  and  $e_2$  are edges incident to  $C$  that appear consecutively as we travel around it. Then either at some point of the algorithm  $e_1$  and  $e_2$  will be part of consecutive paths in a fan, or they will be part of the first and last paths in the very last fan (a wheel) considered by the algorithm. In any case, let  $u$  be the center of the fan, and let  $x$  be the face of  $H'$  bounded by  $C$  and the paths containing  $e_1$  and  $e_2$ . Then we can split  $x$  by adding arbitrary edges incident to  $u$ . By making the set of additional faces a split face, we obtain an equivalent problem. This new problem is easily solved by using the same sequence of fans as before. The vertex  $u$  will be called the ancestor of  $e_1$  and  $e_2$ . This concludes the analysis of the Halin case.

**2.2. Structure of bounded dual diameter graphs.** We next investigate the structure of planar graphs of dual diameter bounded by a certain constant, as it pertains to our problem. Intuitively, our approach is as follows (this description is slightly incorrect as we describe later). Given a plane graph of dual diameter at most  $L$ , by consecutively “peeling” away layers of faces at a given distance from the outer face we will reach a “central” graph after at most  $L$  layers. This central graph must be Halin; we use a version of the algorithm in § 2.1 where we now consider fans that are extended with gridlike graphs with at most  $L$  rows, to solve the minimum face cover.

This description is incorrect in that there may be more than one “central” graph; as we peel layers the graph may decompose arbitrarily. We deal with this difficulty by using a special partial order on the components that we encounter recursively, and proceed essentially as outlined in the previous paragraph. [Ba] introduces a class of planar graphs called  $k$ -outerplanar. If a graph is  $k$ -outerplanar its dual diameter is at least  $k$ ; both concepts are somewhat related (in turn, the radius  $r$  of the graph [RS] satisfies  $k - 1 \leq r \leq k$  and these two parameters are closely related).<sup>1</sup> [Ba] describes an algorithm for decomposing  $k$ -outerplanar graphs. Our procedure UNWRAP for peeling a bounded dual diameter graph is reminiscent of the one in [Ba], with some important differences which are necessary to make our face cover algorithm work.

Let  $H$  be an  $n$ -vertex plane graph of dual diameter  $L$ , with outer facial cycle  $C$ . Procedure UNWRAP proceeds as follows. First, any vertex of degree two and its two incident edges can be replaced by a single edge. Also, the patch operation allows us to assume that  $C$  contains no cutvertices. Further, if  $e$  is an edge incident to a vertex of  $C$ , we can assume that both endpoints have degree three (using patchings or expanding the endpoints into polygons, which will later be used as forbidden faces). Next, we delete  $C$ , together with all edges incident to it. Then  $H$  will be split into several connected components, each of which is a union of trees and maximal two-connected graphs, joined in a treelike manner. If two of the two-connected graphs share one vertex (a cutvertex) we can use the patch operation to obtain a larger graph. Assume we carry this out as many times as necessary. The final two-connected graphs

<sup>1</sup> In polynomial time, one can minimize over all embeddings the radius, the maximum dual distance to the outer face, and the outerplanarity. However, minimizing the dual diameter is NP-hard [BM].

will be called the *height-1 islands* (see Fig. 1). Clearly, if  $X$  is such an island, the distance from an internal face of  $X$  to the outer face of  $X$  is at most  $L-1$ . Finally, it is not difficult to see that a vertex in the outer face of  $X$  may be assumed to be adjacent to at most one vertex not in  $X$ . Now we can recursively use UNWRAP with each height-1 island to obtain height-2 islands. The procedure will terminate with at most  $L-1$  recursive calls. The top islands found (all of height at most  $L-1$ ) will be Halin graphs. The set of islands constitutes a partial order, which is constructed by UNWRAP in time  $O(n)$ . Having peeled away all of the layers we put them back together while preserving the modifications that were introduced (i.e., all of the subdivisions and new edges). It is in this graph  $H^*$ , rather than  $H$ , that we apply XTND, after a few more modifications described in the next section.

**2.3. General case of XTND.** We need one more piece of notation. The edges joining the outer face of a height- $i$  island  $I$  to the outer face of the height- $(i-1)$  island enclosing  $I$  are called the *links* of  $I$ . Notice that if  $u$  is an endpoint of a link, then  $u$  has degree at most four, by the construction used. The edges of  $H^*$  that are not links are called layer edges. Let  $R$  be the maximum dual distance to the outerface.

We first consider the simplest possible case, which we call the *concentric case*. This arises when in every call to UNWRAP we discover precisely one island (and no trees). That is, there is exactly one height- $i$  island for each  $1 \leq i \leq R-1$  (see Fig. 2(a)). Let  $K$  denote the height- $(R-1)$  island.

We modify  $H^*$  as follows, if necessary. Let  $C^*$  be the outer facial cycle of  $H^*$ . Then by subdividing layer edges and introducing some new link edges and edges inside  $K$ , we can assume that every link edge is contained in a (unique) path from  $C^*$  to the interior of  $K$ , of length  $R$ , and similarly, every vertex in the outer facial cycle of  $K$  is contained in such a path. Notice that the new edges will split faces, but all members of a split face are “consecutive”. For convenience, we still refer to the graph by  $H^*$ .  $H^*$  has  $O(Rn)$  vertices. See Fig. 2(b), 2(c).

The final problem we obtain will have split faces, but an extremely simple structure. This structure allows us to essentially use the same Algorithm XTND given before, with the fan structure of  $K$  driving the computation. The main difference lies in that, for every fan  $f$  of  $K$ , we simultaneously consider the entire “grid” of faces stretching

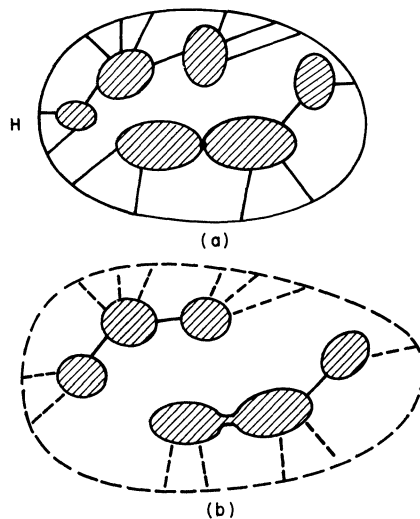


FIG. 1



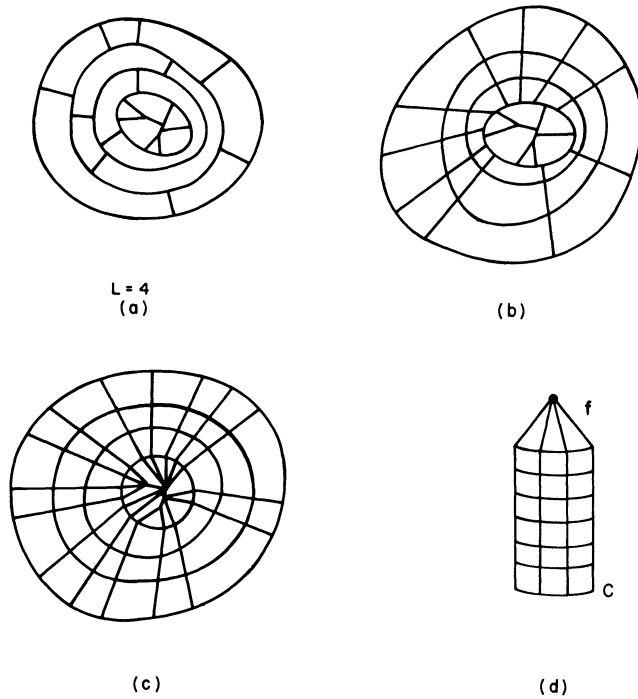


FIG. 2

from  $f$  to  $C$  (see Fig. 2(d)). Such a grid  $g$  will have arbitrarily many faces; however, the “left” and “right” paths of it are incident to at most  $O(L)$  faces. The dynamic programming recursion will consider all possibilities for each such face (whether chosen or rejected for the cover, and in or not in the set  $E$ ) in addition to the usual recursion for the fan  $f$ . Clearly there will be at most  $2^{O(L)}$  possible states. Further, if a face  $f_1$ , adjacent to the left path in  $g$ , and a face  $f_2$ , incident to the right path in  $g$ , actually correspond to the same split face, then we need only consider states of the dynamic program where  $f_1$  and  $f_2$  take the same value. It is easy to see that the overall procedure takes time at most  $2^{O(L)}n$ . This ends the description of the concentric case.

The general case is only slightly more complex, but we need to develop a bit more machinery. We essentially construct a partial order on the various islands, and solve a sequence of problems moving upwards in the partial order. The last (i.e., topmost) problem to be solved will be of the concentric type described above.

As before, let  $H$  denote the overall graph, with outer facial cycle  $C$ . Choose an arbitrary height- $(L-1)$  island  $K$ , with outer facial cycle  $C'$ . Let  $e_1, e_2$  be two consecutive links joining  $C'$  to the height- $(L-2)$  island enclosing  $K$ , then as in the concentric case, by splitting faces we can assume that for  $i=1, 2$ ,  $e_i$  is contained in a length  $R-1$  simple path from  $C'$  to  $C$  (notice that the length restriction says that  $p_i$  does not unnecessarily cut through islands). Then the wedge of  $e_1, e_2$  is the subgraph of  $H$  bounded by  $p_1, p_2$  and the corresponding segments of  $C', C$ . The wedges of  $K$  are constructed for all consecutive links (see Fig. 3(a), 3(b)).

Proceeding recursively, let  $W$  be a wedge of some island; and  $J$  be a highest island enclosed by  $W$ , say  $J$  of height  $i$ . The boundary of  $W$  will be made up of two paths  $p_1, p_2$ , a segment of  $C$  and a segment  $x$  of at most two layer edges. Notice that at most one link edge  $e$  joins  $J$  to  $x$ . By face splitting we can guarantee that  $e$  exists.

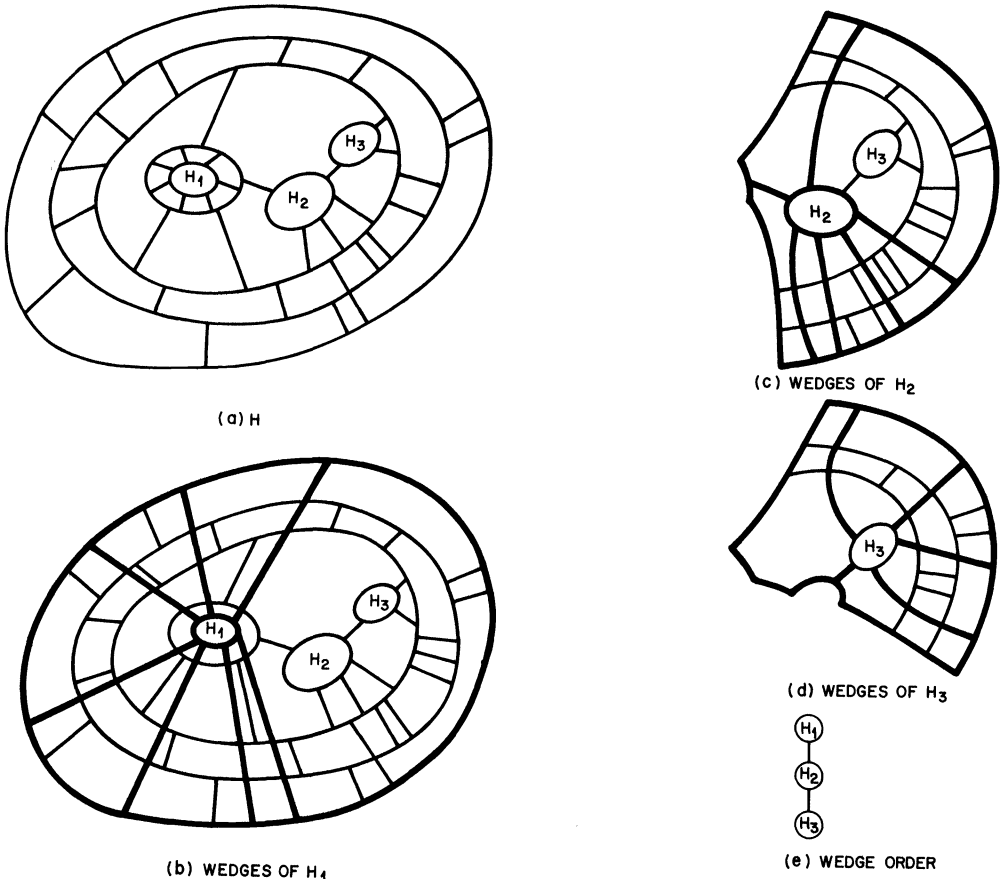


FIG. 3

Now we construct the wedges of  $J$  by including the link edges incident to vertices of the outer face of  $J$  in paths of length  $i$  to  $C$ , contained in  $W$ . The only exception is the edge  $e$  which will define two special wedges called the boundary wedges. This procedure is repeated recursively until we have computed wedges for all islands; trees are handled in a similar way (see Figs. 3(c) and 3(d)). Notice that this procedure constructs a partial order on a subset of islands and trees, with  $K$  at the top. We call this order the *wedge order* of  $H$  (see Fig. 3(e)).

One fact is worth pointing out: through additional face splitting if necessary, for every wedge  $W$  of a height- $i$  island, the number of faces in the two boundary wedges enclosed by  $W$  (if any) is altogether  $2i$ .

Let  $H^*$  denote the graph resulting from  $H$  after applying all the face splittings. Then  $H^*$  contains  $O(Ln)$  vertices, since each set of face splitting is caused by some edge or island of  $H$ .

Our Algorithm XTND will operate on  $H^*$  by moving up the wedge order. As shown previously,  $E$  denotes the set of faces to be covered. Let  $I$  be an island at the bottom of the order.  $I$  is contained inside some wedge  $W$  that belongs to the father of  $I$  in the wedge order. Let  $I'$  denote the union of  $I$  and all its wedges except for the two boundary ones.  $I'$  has the structure of a Halin graph with a grid glued to part of its outer face. Then we compute the minimum number of internal faces of  $I'$  needed to cover all faces of  $E \cap I'$ ; subject to each possible set of *constraints* corresponding

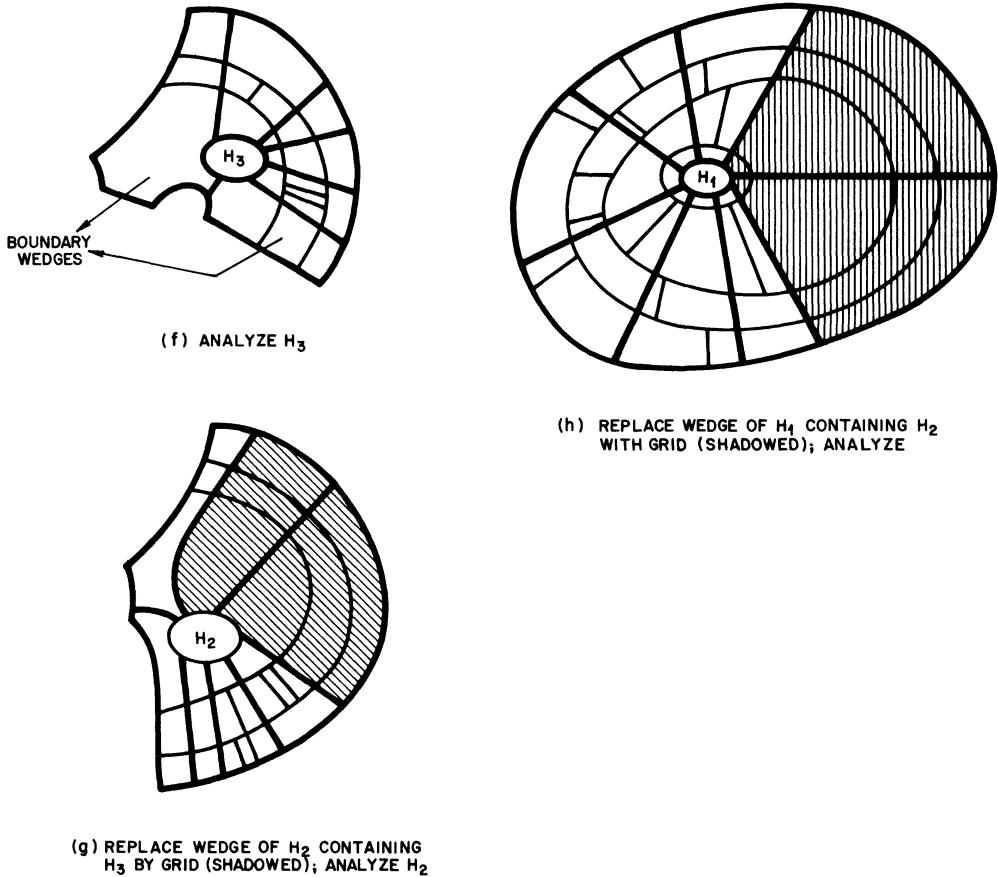


FIG. 3. (Continued.)

to the patterns of the faces in the boundary wedges of  $I$  (i.e., whether chosen, left uncovered or rejected; taking split faces into account, there are  $2^{O(L)}$  such patterns). Each of the computations is carried out much as in the case for Halin graphs.

Having carried out these computations, we replace  $W$  with a grid of width two, remove  $I$  from the wedge order and proceed. For the general step, we again pick an island from the bottom of the order and proceed as above. The only difference is that we may encounter wedges of the island containing width two grids that correspond to previously analyzed islands. But we have computed all relevant information for such grids, and it is easy to work this into the dynamic programming recursion. Details are left to the reader.

To analyze the complexity of the algorithm, notice that the total amount of work done on each wedge  $W$  is at most  $2^{O(L)}|W|$ . Hence, the complexity of the overall algorithm is at most  $2^{O(L)}n$ .

**3. Testing  $k$ -planarity for a variable embedding.** In this section, we return to the original problem of testing whether an  $n$ -vertex planar graph  $G = (V, E)$  containing a distinguished subset of vertices  $D$ , admits an embedding in which  $D$  can be covered with  $k$  or fewer faces.

If  $G$  is three-connected, then we use Algorithm XTND of § 2, after expanding  $D$  into polygons. We show later how to reduce the one-connected case to that of two-connectivity. In what follows, we will therefore assume  $G$  is two-connected.

The basic approach consists of decomposing  $G$  into (roughly) its triconnected components [HT1]. These components are then inductively assembled into  $G$ , using dynamic programming. Next we will give some definitions.

(1) A *block* of  $G$  is a connected subgraph  $H$  of  $G$  with two distinguished vertices  $u_1$  and  $u_2$  with the property that either  $G = H$ , or all paths from  $H - \{u_1, u_2\}$  to  $G - H$  must pass through  $u_1$  and  $u_2$ . The vertices  $u_1$  and  $u_2$  will be called the *extremes* of  $H$ . Notice that since  $G$  is two-connected, then in any embedding of  $G$ ,  $u_1$  and  $u_2$  will be in a common face.

(2) Let  $H$  be a block of  $G$  with extremes  $u_1$  and  $u_2$ . Let  $xyzw$  be a binary vector of length four. Define  $F(H, xyzw)$  to be the minimum number of internal faces of  $H$  needed to cover all vertices of  $D$  in  $H$ , taken over all embeddings of  $H$  with  $u_1$  and  $u_2$  in the outer face, with the following constraints:

- (i) If  $x = y = 1$  then there exist vertices  $v_1$  and  $v_2$  in  $D \cap H$  with  $v_1 \neq v_2$  and  $v_i \neq u_j$  for all  $i, j$ , that are *not* covered by the chosen internal faces of  $H$ , and such that  $v_1$  is in one path from  $u_1$  to  $u_2$  in the outer face of  $H$ , and  $v_2$  in the other (in an optimal embedding that attains  $F(H, 11zw)$ ).
- (ii) If  $x + y = 1$ , exactly one of the paths from  $u_1$  to  $u_2$  in the outer face of  $H$  contains an uncovered vertex  $v \in D \cap H$ , with  $v \neq u_1, u_2$ .
- (iii) If  $x = y = 0$ , then all vertices of  $D \cap H$ , with the possible exception of  $u_1$  or  $u_2$ , are covered.
- (iv) If  $z = 0$  (resp., 1), then  $u_1 \in D$  is (resp., is not) covered.
- (v) If  $w = 0$  (resp., 1), then  $u_2 \in D$  is (resp., is not) covered.

(If,  $u_1 \notin D$ , then we always set  $z = 0$ ; similarly for  $u_2 \notin D$ . An embedding that attains  $F(H, xyzw)$  will be denoted by  $E(H, xyzw)$ .)

Notice that if in an optimal embedding of  $G$ , we use  $E(H, xyzw)$  to embed  $H$ , where  $x + y \geq 1$  and  $z + w \geq 1$ , then any face of  $G - H$  used to cover the uncovered vertices of  $H$  different from  $u_1$  and  $u_2$  will also cover  $u_1$  and  $u_2$ . Hence, for  $x + y \geq 1$ , we reset  $F(H, xy00) = \min_{z,w} F(H, xyzw)$ . Therefore, the only 4-vectors we need to keep track of are (0000), (1000), (1100), (0010), (0001) and (0011).

Every planar graph admits a recursive decomposition into blocks. This decomposition can be represented by a rooted tree, where each vertex corresponding to some block of  $G$ , with the root representing  $G$ , and the leaves (essentially) correspond to the triconnected components of  $G$ . For each block appearing in the tree, one of three canonical ways of decomposing it occurs: a "Series" Case, a "Parallel" Case and a "Messy" Case. Our Algorithm DYN for testing  $k$ -planarity proceeds upwards from the leaves in the decomposition tree by computing the  $F$  parameters. Once a block  $B$  has been analyzed, it is replaced in its parent by a small (bounded size) gadget (and we keep track of the  $F$  parameters of  $B$  to compute those of its parent).

In order to simplify the description, we will present together Algorithm DYN and the recursive decomposition structure. Further, the algorithm will be described recursively (i.e., proceeding from the root down). The complexity is analyzed later.

Now suppose  $\{u_1^*, u_2^*\}$  is an arbitrary cutset of  $G$ . Then in any embedding of  $G$ ,  $u_1^*$  and  $u_2^*$  will be in at least one common face, without loss of generality, the outer face of  $G$ . Then testing  $k$ -planarity of  $G$  is achieved by regarding  $G$  as a block with extremes  $u_1^*$  and  $u_2^*$ , and computing

$$\min \{F(G, 0000), 1 + \min \{F(G, xyzw): x + y + z + w \geq 1\}\}.$$

ALGORITHM DYN.

Input: a block  $H$  with extremes  $u_1, u_2$

Output: all quantities  $F(H, xyzw)$

There are three cases.

(1) *Series Case.*  $H$  has a cutvertex  $v$ . Then  $v \neq u_i, i = 1, 2$ , and we write  $H = H_1 \cup H_2$ , where  $u_i \in H_i, i = 1, 2$ , and  $H_1 \cap H_2 = v$  (see Fig. 4(a)). It is easy to compute the  $F$  parameters for  $H$  in constant time from those of  $H_1, H_2$ . Refer to Appendix A(1) for details.

(2) *Parallel Case.* Fix  $z$  and  $w$ .  $H$  is two-connected and  $\{u_1, u_2\}$  is a cutset of  $H$ . Write  $H = \cup_{j=1}^m H_j$ , where  $H_j \cap H_k = \{u_1, u_2\}$  for each  $j \neq k$ , and each  $H_j$  is a block with extremes  $\{u_1, u_2\}$  (see Fig. 4(b)).

Intuitively, we proceed as follows. Suppose we select two blocks to act as “leftmost” and “rightmost.” In Appendix A(2) we show that it is possible to select either a “best” embedding for each of the remaining “internal” blocks, or else a simple tie situation may arise. In either case we are allowed to essentially ignore the detailed structure of the internal blocks, and there is an efficient (linear time) procedure to pair them up and obtain an optimal embedding (permutation and rotations of the internal blocks). Further, we are always able to select a “best” leftmost and rightmost block. The overall procedure runs in time linear in  $m$ . Details are provided in Appendix A(2).

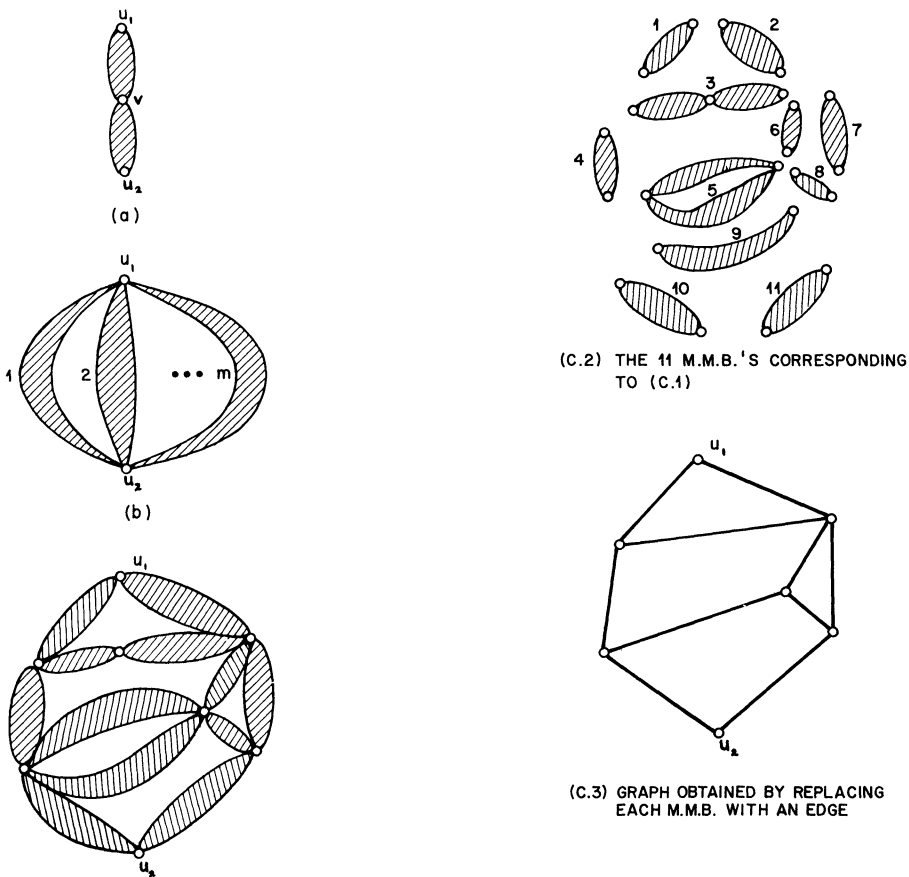


FIG. 4

(3) *Messy Case.* Cases (1) and (2) do not apply (see Fig. 4(c.1)). Then either (i) there is a cutset  $\{v, w\} (\neq \{u_1, u_2\})$  of  $H$ , or (ii)  $H$  is three connected. In the latter case, we can apply the results of § 2 (by using forbidden faces, if necessary). So assume (i) occurs.

Now for every cutset  $\{v, w\}$  of  $H$  we can define the main block of  $\{v, w\}$  as the union of all connected components of  $H - \{v, w\}$  not containing  $u_1$  or  $u_2$ , together with  $\{v, w\}$ , if any such components exist. Now if for all cutsets  $\{v, w\}$  the main block is empty, it is easy to see that  $H$  has a unique embedding with  $\{u_1, u_2\}$  restricted to lie in the outer face, and therefore we are essentially in alternative (ii). Otherwise, the main blocks can be ordered by inclusion; call those at the top the maximal main blocks (see Fig. 4(c.2)), or m.m.b.'s for short. If we replace every m.m.b. by an edge, the resulting graph  $H'$  has a unique embedding with  $\{u_1, u_2\}$  restricted to lie in the outer face (Fig. 4(c.3)).

Our strategy is to analyze each m.m.b.  $W$  recursively first. Next we replace  $W$  by an appropriate small gadget, attached to the rest of  $H$  by the extremes of  $W$ . We want the gadget to "retain" all the relevant properties of  $W$ , while having essentially a unique embedding (it will turn out that either we can find a "best" embedding for  $W$ , or there are a few ties; the gadgets we use will take care of either case). In this manner, we will reduce the problem on  $H$  to one where the embedding has been prescribed, and we can use the results of § 2. In summary, the approach is as follows:

- (a) Analyze every m.m.b.
- (b) For every m.m.b.  $W$ , compute its best embedding(s) and replace  $W$  with a small gadget. We will keep track of a *weight* corresponding to  $W$  that describes the cost of "internal" faces of  $W$ .
- (c) Compute each parameter  $F$  for the resulting graph  $\hat{H}$ , and to each, add the sum of the weights of the m.m.b.'s to obtain the  $F$  parameters of  $H$ .

Details are provided in Appendix A(3). The overall complexity of this case will be, at most,  $2^{O(k)}|\hat{H}|$ .

This concludes the description of Algorithm DYN. An efficient implementation of DYN would first compute the block decomposition. This is accomplished by using the algorithm in [HT1] to decompose a graph into its "split components". This algorithm is easily modified to output the decomposition tree in linear time, with each block labeled series, parallel or messy (in the messy case the decomposition is into m.m.b.'s). If  $v$  is a vertex of the tree that represents a block  $W$ , at  $v$  we store the graph  $W'$  resulting from  $W$  by replacing each of its children with an edge. The overall space required is linear. Finally, we "force" the original extremes  $u_1^*$ ,  $u_2^*$  chosen for  $G$  to lie in the outer face by adding at the start the edge  $(u_1^*, u_2^*)$  if necessary.

Having computed the decomposition, we proceed upwards with DYN. If  $W'$  is the graph stored at some vertex of the tree, the work involved in analyzing  $W'$  (when we reach it) is  $O(|W'|)$  in the series and parallel cases, and at most  $2^{O(k)}|W'|$  in the messy case. Thus we conclude as follows.

**LEMMA 3.** *The complexity of testing whether  $G$  is  $k$ -planar with DYN is at most  $O(c^k n)$ , where  $c$  is independent of  $n$ .  $\square$*

Next we pass to the one-connected case.

A one-connected graph  $G$  is the union of *bricks*; that is, edges or 2-connected subgraphs that intersect at cutvertices only.  $G$  will have a "treelike" structure; namely, if a path leaves a brick  $B$  through a cutvertex  $v$ , the only way the path can return to  $B$  is by crossing  $v$  again. Now suppose  $G = (V, E)$  has connectivity one; let  $v$  be an arbitrary cutvertex, and write  $G = \bigcup_{i=0}^m G_i$ , where  $G_j \cap G_k = v$ , for all  $j \neq k$ , and each  $G_i - v$  is nonempty and connected. Let  $D_i = D \cap G_i$ .

If  $v \in D$ , then  $v$  will be covered by some face  $f$ . Now we can always embed every  $G_i$  simultaneously so that  $f$  is a face of it without requiring more than the minimum number of faces to cover all of  $D$ . Thus, if

$$\hat{k} = \text{planarity number of } G$$

and

$$\hat{k}_i = \text{planarity number of } G_i, \text{ for } i = 0, \dots, m \text{ (where we cover } D_i),$$

then

$$\hat{k} = \sum_{i=0}^m \hat{k}_i - (m-1),$$

and the problem decomposes with  $v$  in each  $D_i$ .

Now suppose  $v \notin D$ . By not including  $v$  in  $D$ , we can compute  $\hat{k}_i$  and use  $\sum_{i=0}^m \hat{k}_i$  as the number of faces to cover  $D$ . If we use  $\bar{D}_i = \{v\} \cup D_i$  instead of  $D_i$  for a given graph  $G_i$ , we might increase the planarity number of  $G_i$  by 1. However, if we add  $v$  to  $D$  and compute the planarity number of  $G$  as before, we might actually decrease the number of faces required to cover  $D$  because the graphs  $G_i$  will be able to share the additional face. Hence, if we try both possibilities (i.e., whether or not to add  $v$  to  $D$ ), and decompose into a problem for each  $G_i$ , we will certainly solve the original problem. However, this may create too much work if a brick of  $G$  contains too many cutvertices. Instead we will use an approach inspired by the next lemma.

LEMMA 4.

$$\hat{k}_{i,1} = \text{planarity number of } G_i \text{ to cover } D_i.$$

$$\hat{k}_{i,2} = \text{planarity number of } G_i \text{ to cover } \bar{D}_i.$$

Then

- (a) If, for at least one  $i \geq 1$ ,  $\hat{k}_{i,2} = \hat{k}_{i,1}$ , then it is optimal to add  $v$  to  $D$ .
- (b) If, for all  $i \geq 1$ ,  $\hat{k}_{i,2} = \hat{k}_{i,1} + 1$ , then it is optimal not to add  $v$  to  $D$ .

*Proof.* The proof is clear.  $\square$

Notice that  $G_0$  is not considered in the lemma.

Next we proceed as follows. Let  $G$  be a graph whose  $k$ -planarity we want to test; exactly one brick  $B$  of  $G$  has been painted red. Let  $v \in B$  be a cutvertex of  $G$ . If  $v \in D$ , then proceed as outlined above. If  $v \notin D$ , then (as previously) we form the graphs  $G_i$ , so that  $B$  is contained in  $G_0$ , and we paint red the brick of  $G_i$  for  $i \geq 1$ , that contains  $v$ . Next, we solve, in each  $G_i$  for  $i \geq 1$ , the two problems to cover  $D_i$ , and to cover  $\bar{D}_i$ ; we then use Lemma 4 to decide whether or not to add  $v$  to  $D$ . Notice that each graph  $G_i$  contains exactly one red brick. Clearly, repeating this procedure will eventually reduce the problem in  $G$  to at most two separate problems in each brick. Thus the complexity is again linear.

**4. NP-completeness for arbitrary  $k$ .** In this section we show that the problem of determining whether a  $(G, D)$  pair is  $k$ -planar or not is strongly NP-complete when  $k$  is variable. Hence, it is unlikely that a polynomial-time algorithm exists for recognizing a  $k$ -planar graph for arbitrary  $k$ .

In order to do this, we define the following decision problem which we call FACE COVER: "Given a planar graph  $G = (V, E)$  together with a subset of vertices  $D \subseteq V$  and an integer  $K$ , can  $G$  be embedded in the plane so that at most  $K$  faces are required to cover all of the vertices in  $D$ ?" We will show that FACE COVER is strongly NP-complete even when  $G$  is three-connected and so has essentially a unique planar

embedding, when  $d = \theta(n)$ , and when all of the faces of  $G$  have bounded length. A graph  $G$  is 3-connected if the removal of any two vertices of  $G$  leaves the rest of  $G$  connected. It is well known [Wh] that a 3-connected planar graph has an embedding which is essentially unique; i.e., all embeddings have the same facial structure and differ only in which face is the outer face.

The reduction will be from VERTEX COVER: “Given a graph  $G = (V, E)$  and integer  $L$ , is there a subset of vertices  $W \subseteq V$  with  $|W| \leq L$ , such that for every edge  $(u, v) \in E$  at least one of  $u$  or  $v$  belongs to  $W$ ?” VERTEX COVER is strongly NP-complete for cubic planar graphs [GJS].

**THEOREM 5.** FACE COVER is strongly NP-complete even when  $G$  is 3-connected,  $d = \theta(n)$ , and all faces of  $G$  have bounded length.

*Proof.* Consider an instance of VERTEX COVER given by a cubic planar graph  $G = (V, E)$  and integer  $K$ , where  $G$  has no loops or parallel edges and every edge is contained in exactly two faces. This problem is known to be strongly NP-complete [GJS].

We obtain an instance of FACE COVER by setting  $K = L$  and  $\bar{G} = (\bar{V}, \bar{E})$  being the planar dual of  $G$ , i.e., place a vertex  $v \in \bar{V}$  in every face  $f_v$  of  $G$ , and an edge  $(u, v) \in \bar{E}$  if faces  $f_u$  and  $f_v$  of  $G$  share an edge. Subdivide each edge  $(u, v) \in \bar{E}$  by adding a vertex  $x(u, v)$  and denote the new graph by  $\hat{G} = (\hat{V}, \hat{E})$ . Let  $D$  be the set of vertices  $x(u, v)$  for  $(u, v) \in \bar{E}$ . Given the particular embeddings of  $G$  and  $\hat{G}$ , there is clearly a one-to-one correspondence between vertices in  $D$  and edges of  $G$ , and similarly, between faces of  $\hat{G}$  and vertices of  $G$ . Thus, there is a one-to-one correspondence between a face cover of  $(\hat{G}, D)$  and a vertex cover of  $G$  of the same cardinality.

In general,  $\hat{G}$  might have other planar embeddings where a face cover in this new embedding of  $\hat{G}$  does not correspond to any vertex cover in  $G$ . To remedy this, we form the graph  $G^* = (V^*, E^*)$  from  $\hat{G}$  by adding, inside each face of  $\hat{G}$ , edges to form a cycle containing all vertices of  $D$  in that face, and embed the cycle inside the face. It is easy to see that the face cover problem on  $G^*$  is equivalent to the face cover problem in  $\hat{G}$ . We will show that  $G^*$  is 3-connected and so this embedding is essentially unique [Wh] which will complete the proof. To see that  $G^*$  is three-connected, notice that in  $\bar{G}$ , all facial cycles consist of precisely three edges. However, these cycles may not be simple (i.e., triangles) if  $G$  has bridges. Nevertheless, in  $G^*$  all facial cycles have length three and are simple, in other words  $G^*$  is triangulated and, thus, three-connected, since  $G^*$  is a simple graph.

Finally, note that  $|D|$  equals the number of edges of  $\bar{G}$ , which is  $\theta(n)$ .  $\square$

(Note: An alternative proof of NP-completeness for the case  $D = V$  appears in [FHS].)

**5. An exact and an approximate algorithm for fixed embeddings.** Consider the variation of FACE COVER in which we are forced to use a given embedding. That is, FIXED EMBEDDING FACE COVER (FEFC): “Given an embedded planar graph  $G = (V, E)$ , an integer  $k$ , and a subset  $D \subseteq V$ , can  $D$  be covered with at most  $k$  faces?” Clearly FEFC is NP-complete.

In this section we will present:

- (a) A set of transformations for FEFC that allow us to assume that  $d = \theta(n)$ , while not increasing the maximum facial length.
- (b) An exact algorithm for FEFC that runs in time  $2^{O(\sqrt{n})}$ .
- (c) A polynomial-time approximation algorithm for FEFC that is asymptotically optimal if  $G$  has bounded length facial cycles, as is the case in our NP-



completeness proof. (Our basic approach is similar to that of [LT] in that we use the planar separator theorem.)

The following set of transformations can be visualized as simplifying the problem. Their main objective is to allow us to assume that  $d = \theta(n)$  without increasing the facial cycle length and preserving the value of the optimal solution. These transformations are crucial for the proof of Theorem 6 presented later.

- (1) Every vertex of  $V - D$  of degree one is shrunk into its neighbor.
- (2) Every facial cycle of length two (resp. one) is shrunk into a single edge (resp. vertex).
- (3) In every face  $g$  with at most one vertex in  $D$ , all vertices in  $V - D$  are shrunk into a single vertex.
- (4) Every vertex  $v$  in  $V - D$  of degree two, adjacent to vertices  $u$  and  $w$ , is deleted, and edges  $(u, v)$  and  $(v, w)$  are replaced by the edge  $(u, w)$ .
- (5) Any two vertices of  $V - D$  that are adjacent are shrunk into a single vertex.
- (6) For every loop  $e = (v, v)$  such that (say) the subgraph contained in the interior of  $e$  has an outer facial cycle consisting of  $v$  and vertices of  $V - D$  only,  $e$  is deleted.
- (7) If a vertex  $v \in D$  is contained in a unique face of  $f$ , then  $f$  must appear in any cover; hence, we delete  $v$ , and remove from  $D$  all members in  $f$ .

This concludes the list of transformations. Clearly, Transformations (1)–(7) can be applied until no longer possible, in polynomial time, to obtain an equivalent problem.

Theorem 6 given below places a lower bound on the size of  $D$  in a loop-free graph where none of Transformations (1)–(7) can be applied. However, if we apply (1)–(7) to a graph  $G$ , the resulting graph  $G'$  may contain loops. In order to count vertices of  $D$  in  $G'$ , we modify it as follows:

- (a) Every loop  $(v, v)$  with  $v \in D$  or  $v$  adjacent to at least three vertices of  $D$  is deleted.
- (b) Otherwise, let  $e = (v, v)$  be a loop, and let  $x$  be the only neighbor of  $v$  in the interior of  $e$ , with  $x \in D$ . Now, (7) or (2) cannot be applied; hence, the interior of  $e$  contains vertices other than  $x$ . But there must be at least one such vertex  $w \in D$  such that  $(w, v)$  can be added while preserving planarity; this is true because of (a), and the fact that neither (3) nor (6) can be applied. In that case,  $(w, v)$  is added. Now  $v$  has three neighbors in  $D$ .

It is easy to verify that after applying (a) and (b) to  $G'$ , the resulting graph  $G''$  is such that none of the transformations (1)–(7) can be applied, and that  $G''$  is loop-free.

**THEOREM 6.** *Let  $G = (V, E)$  be a loop-free planar graph with a fixed embedding, where  $|V| = n$  and  $|D| = d$ , and suppose that none of the Transformations (1)–(7) can be applied. Then  $d \geq (n + 4)/3$ .*

*Proof.* Notice that every vertex of  $V - D$  has degree at least three and is only adjacent to vertices of  $D$ . Suppose that  $G$  contains a pair of parallel edges  $e = (u, v) = e'$ , with say,  $v \in V - D$ . Then the interior region bounded by  $e$  and  $e'$  must contain a vertex  $w \in D$  with  $w \neq u$  such that either  $v$  and  $w$  are adjacent, or the edge  $(v, w)$  can be added to the embedding (in which case we do so). Similar considerations apply to the exterior of  $e'$ .

Now, delete all edges with both endpoints in  $D$ , let  $C$  be an arbitrary connected component with  $\hat{n}$  vertices and  $\hat{d}$  elements of  $D$ . Clearly,  $C$  has no facial cycles of length one or two. The latter follows because if  $e = (u, v) = e'$  are a pair of parallel edges with  $u \in D$ ,  $v \in V - D$ , then there exist vertices  $w_1, w_2 \in D$  adjacent to  $v$ , and located in the interior and exterior of  $e'$ ; this is guaranteed by the previous paragraph.

Consequently, the number of  $\hat{e}$  of edges of  $C$  satisfies  $\hat{e} \geq 3\hat{n} - 6$ . On the other hand, if  $\hat{f}$  denotes the number of faces of  $C$ ,  $\hat{f} = \sum_{k \geq 3} \hat{f}_k$ , where  $\hat{f}_k$  is the number of

faces of length  $k$ . Since  $C$  is bipartite,  $\hat{f}_3 = 0$ . Thus,  $\hat{e} = \frac{1}{2} \sum_k k \hat{f}_k \geq 2 \sum_{k \geq 4} \hat{f}_k = 2\hat{f}$ . Moreover,  $\hat{e} \geq 3(\hat{n} - \hat{d})$ , since for each vertex of  $C$  not in  $D$ , we count at least three edges. Hence,

$$\hat{e} = 3\hat{n} - x \quad \text{where } 6 \leq x \leq 3\hat{d}.$$

By Euler's formula,

$$\hat{f} = 2\hat{n} - x + 2 \leq \frac{3\hat{n} - x}{2},$$

or

$$\hat{n} + 4 \leq x \leq 3\hat{d},$$

i.e.,

$$\hat{d} \geq \frac{\hat{n}}{3} + \frac{4}{3},$$

which concludes the proof.  $\square$

We note that the bound in Theorem 6 is best possible.

We now present an exact algorithm for finding the minimum number of faces required to cover all special vertices of a planar graph given a fixed embedding in  $2^{O(\sqrt{n})}$  time. As in Theorem 7, we may take  $d = \theta(n)$  by the application of appropriate transformations in the embedded graph. Let  $G = (V, E)$ ,  $D \subseteq V$ ,  $d = |D|$ , and  $n = |V|$  be the input. The algorithm proceeds as follows:

- (1) Let  $S$  be an  $O(\sqrt{n})$ -separator of  $G$ . Write  $G = G_1 \cup G_2$ , where  $G_1 \cap G_2 = G(S)$ , the subgraph of  $G$  induced by  $S$ . Let  $D_i$  be the subset of  $S$  contained in  $G_i$ . Write  $G_i = (V_i, E_i)$ , and embed  $G_i$  as it appears in  $G$ .
- (2) A face  $f$  of  $G$  that contains vertices of  $G_1 - S$  and of  $G_2 - S$  will be called a boundary face. Ideally, we would like to proceed independently with  $G_1$  and  $G_2$ . However, these two graphs interact on the boundary faces. Thus, we modify  $G_1$  (and similarly,  $G_2$ ) so that the boundary face structure is "preserved" in a "legal" way. This is attained in two steps.
  - (i) For every boundary face  $f$ , we replace each path of  $f$  that intersects  $V_1$  at its endpoints with an edge. The resulting graph contains a face  $f_1$  that corresponds to  $f$ ; we call  $f_1$  an inherited face. Select an arbitrary added edge  $(u, v)$  of  $f_1$  and subdivide it by introducing a grey vertex  $w(f)$ . Carry out this transformation for all boundary faces of  $G_1$ ; call the resulting graph  $G'_1$ .

Now we are essentially ready to proceed independently with graph  $G'_1$ .

**The** idea is to use the grey vertices to force boundary faces to be used in a **cover**: if  $f$  is such a face and we change the color of  $w(f)$  from grey to black, **this** should force us to use  $f$ . However, there is a problem.  $G_2$  may have several connected components and the removal of each component could introduce in  $G'_1$  a (possibly large) new face that does not correspond to any face of  $G$ . We call such a face a gap face. Each gap face is made up of edges added in (i). We handle this problem as follows.

- (ii) In each gap face subdivide every edge by introducing a new white vertex. Connect all such vertices consecutively as we travel around the face with red edges. Call the resulting graph  $\hat{G}_1$ . Similarly, define  $\hat{G}_2$ .

- (3) Write  $\hat{S} = S \cap D$ . Next, for every partition  $\hat{S} = \hat{S}_1 \cup \hat{S}_2$  with  $\hat{S}_1 \cap \hat{S}_2 = \emptyset$ , and every subset  $X$  of grey vertices, we color  $X$  black, the remaining grey vertices white, and solve the following two face cover problems.
- (a) In  $\hat{G}_1$ , cover  $(D_1 - S) \cup \hat{S}_1 \cup X$ , with minimum value  $k_1(\hat{S}_1, X)$ .
- (b) In  $\hat{G}_2$ , cover  $(D_2 - S) \cup \hat{S}_2 \cup X$ , with minimum value  $k_2(\hat{S}_2, X)$ ; set  $f(\hat{S}_1, \hat{S}_2, X) = k_1(\hat{S}_1, X) + k_2(\hat{S}_2, X) - |Y|$ , where  $Y$  is the set of inherited faces used in both covers. Then the minimum cover of  $D$  has value  $\min_{\hat{S}_1, \hat{S}_2, X} f(\hat{S}_1, \hat{S}_2, X)$ .

The proof of correctness of the algorithm proceeds as follows.

- (1) Consider any face cover of  $D$  in  $G$ . Then an arbitrary subset of the boundary faces will be used, and if any vertex of  $S \cap D$  is not covered by a boundary face, then it is either covered by a face of  $G_1$  that contains no vertices of  $G_2 - S$  (an internal face of  $G_1$ ), or it is covered by a face of  $G_2$  that has no vertices of  $G_1 - S$  (an internal face of  $G_2$ ).
- (2) Consider any of the problems on graph  $\hat{G}_1$ , with  $\hat{S}_1$  and  $X$  as above. Then, without loss of generality, *none* of the faces containing red edges is ever used in an optimal solution, since the black vertices that any such face may cover are also covered by an inherited or an internal face. Thus, all vertices of  $X$  are covered by inherited faces; and all vertices of  $\hat{S}_1$  are either covered by inherited faces, or by faces of  $\hat{G}_1$  that are copies of internal faces of  $G_1$  (similarly for  $\hat{G}_2$ ). Consequently, for each  $\hat{S}_1, \hat{S}_2$  and  $X$ , we can take the two optimal solutions on  $\hat{G}_1$  and  $\hat{G}_2$ , respectively, and obtain a face cover of  $D$  in  $G$  of cardinality precisely  $f(\hat{S}_1, \hat{S}_2, X)$ .
- (3) Finally, consider an arbitrary face cover  $F$  of  $D$  in  $G$ . Let  $F_i$  be the set of internal faces of  $G_i$  used in  $F$  for  $i = 1, 2$ . Let  $Z$  be the set of boundary faces used in  $F$ . Notice that  $|F| = |F_1 \cup Z| + |F_2 \cup Z| - |Z|$ . Also, set  $\hat{S}_1$  is the set of vertices of  $S \cap D$  covered by internal faces of  $G_1$ , and  $\hat{S}_2 = (S \cap D) - \hat{S}_1$ . Let  $X(Z)$  be the set of grey vertices of  $\hat{G}_1$  contained in those inherited faces corresponding to  $Z$ . Then  $k_i(\hat{S}_i, X(Z)) \leq |F_i \cup Z|$  for  $i = 1, 2$ , and thus,  $k_1(\hat{S}_1, X(Z)) + k_2(\hat{S}_2, X(Z)) \leq |F| + |Z|$  which implies that  $f(\hat{S}_1, \hat{S}_2, X(Z)) \leq |F|$ .

This concludes the proof of the correctness of the algorithm.

To derive the complexity of the algorithm, the number of edges and vertices added to each graph  $G_i$  to obtain  $\hat{G}_i$  is  $O(|S|)$ . Consequently,  $G_i$  has at most  $\frac{2}{3}n + O(\sqrt{n})$  vertices. Furthermore, the total number of triples  $(\hat{S}_1, \hat{S}_2, X)$  is at most  $2^{O(|S|)}$ . As a result, if  $T(n)$  is the worst-case complexity of the algorithm, we have  $T(n) \leq 2^{O(\sqrt{n})} T(\frac{2}{3}n + O(\sqrt{n}))$ ; from which  $T(n) \leq 2^{O(\sqrt{n})}$  is straightforward.

The approximation algorithm is somewhat similar to the exact algorithm, with the exception that we will use planar separators that produce "equal" size subgraphs. Let  $G = (V, E)$  be the input with  $D \subseteq V$ . Let  $S$  be a "50/50" separator of  $G$ . Write  $G = G_1 \cup G_2$ , where each  $G_i = (V_i, E_i)$  is defined as in the exact algorithm. Next, add edges to each  $G_i$  to obtain the inherited faces, and call the resulting graph  $\bar{G}_i$ . Notice that the faces of  $\bar{G}_i$  correspond to the internal faces of  $G_1$ , boundary faces, and also (possibly) to a third type of face that corresponds to connected components of  $G_2$ . (The vertices on these faces are all contained in  $S$ .) Proceed similarly with  $G_2$ . Set  $D_i = (D \cap V_i) - S$ . Now, suppose we solve the following problems with  $i = 1, 2$ :

- (\*) In  $\bar{G}_i$ , cover  $D_i$ ; let the optimum cover have size  $\bar{k}_i$ .

Now, by taking the union of the optimum covers we can obtain a cover of  $D$  in  $G$  by adding at most  $|S|$  faces. Hence, if we denote by  $k$  the size of an optimum cover

of  $D$  in  $G$ , we have  $k \leq \bar{k}_1 + \bar{k}_2 + |S|$ . On the other hand, given a cover  $F$  for  $G$  we can obtain a cover for  $\bar{G}_1$  and one for  $\bar{G}_2$  by restricting  $F$  appropriately. Hence,  $|F| \geq \bar{k}_1 + \bar{k}_2 - O(|S|)$ , or  $|k - \bar{k}_1 + \bar{k}_2| = O(|S|) = O(\sqrt{n})$ .

Our algorithm will not solve the problems on  $\bar{G}_i$  exactly. Rather, we keep subdividing each graph and modifying the resulting subgraphs until we obtain (on each branch of the recursion) graphs of size  $O(\log^2 n)$ . We solve these problems using the exact algorithm, and by piecing together their solutions, we will obtain a face cover of  $D$  in  $G$ . It is not difficult to verify that the problems produced at the  $i$ th recursive step have at most  $O(n2^{-i})$  vertices. Hence, the total number of recursive levels is at most  $T = \log n - 2 \log \log n + O(1)$ . Consequently, the total error is, up to a constant, at most

$$\sum_{i=0}^T 2^i \sqrt{\frac{n}{2^i}} = O(\sqrt{n}2^{T/2}) = O\left(\frac{n}{\log n}\right).$$

Hence, if  $G$  has bounded length facial cycles, and since  $d = \theta(n)$ , the relative error of our approximation algorithm is  $O(1/\log n)$ .

To estimate the complexity of this procedure, notice that the number of problems to be solved exactly is  $O(2^T) = O(n/\log n)$ , and each such problem takes time at most  $2^{O(\sqrt{\log^2 n})} = n^{O(1)}$ . The total number of vertices in the recursion tree is also  $O(2^T)$ , and we conclude that the algorithm runs in polynomial time.

**6. Concluding remarks.** We have shown that checking  $k$ -planarity of graph  $G$  with  $n$  vertices  $D \subseteq V$  can be done in linear time for any fixed  $k$ . This provides an efficient recognition algorithm for this class of graphs for which the Steiner tree problem can be solved in polynomial time [EMV]. We have also shown that if  $k$  is not fixed, the associated decision problem is NP-Complete even if  $G$  has essentially a unique embedding,  $d = \theta(n)$ , and all facial cycles have bounded length. We obtain a polynomial-time algorithm for this latter case which is asymptotically optimal.

We note that the work of Robertson and Seymour on Wagner's conjecture could be used to check  $k$ -planarity for any fixed  $k$  in  $O(n^4)$  time [Se]. However, their algorithm would *not* provide an embedding and covering as our algorithm does. It might be possible to specialize their result to our problem. We leave this as an open problem.

#### Appendix A—The three cases of DYN.

(1) *Series Case.* The formulas for this case are: Fix  $z$  and  $w$ . Then

$$F(H, 00zw) = \min \left\{ F(H_1, 00z0) + \min_t \{F(H_2, 00tw)\}, \min_t \{F(H_1, 00zt)\} \right. \\ \left. + F(H_2, 000w) \right\}.$$

Similarly,

$$F(H, 10zw) = \min \left\{ \min_{w'} \{F(H_1, x'y'zw') : x' + y' = 1\} \right. \\ \left. + \min_{z''} \{F(H_2, x''y''z''w) : x'' + y'' \leq 1\}, \right.$$

$$\begin{aligned} & \min_{w'} \{F(H_1, x'y'zw'): x' + y' \leq 1\} \\ & + \min_{z''} \{F(H_2, x''y''z''w): x'' + y'' = 1\}, \\ & \min \{F(H_1, x'y'z1): x' + y' \leq 1\} \\ & + \min \{F(H_2, x''y''1w): x'' + y'' \leq 1\} \}. \end{aligned}$$

All other parameters are computed analogously.

(2) *Parallel Case.* Fix  $z$  and  $w$ . Suppose we choose two candidates  $H_l$  and  $H_r$  as the “left” and “right” blocks in  $E(H, xyzw)$ , where, for example, we would use embeddings  $E(H_l, xb_lzw)$  and  $E(H_r, b_r yzw)$  for some  $b_l, b_r \in \{0, 1\}$ . Subject to this specific choice (i.e.,  $l, r, b_l$  and  $b_r$ ), we show how to compute the best embedding of  $H$ . Now suppose  $k \neq l, r$ . Which embedding should be used for  $H_k$ ? Now, if  $z + w \geq 1$ , we must always use  $E(H_k, x'y'00)$  for some  $x', y'$  (and also,  $b_l = b_r = 0$ ). Assume  $z = w = 0$ .

- (a) If  $F(H_k, 0100) \leq F(H_k, 0000) - 1$ , then  $E(H_k, 0100)$  is preferred over  $E(H_k, 0000)$ ; if the reverse inequality holds, then the opposite choice is preferred.
- (b)  $E(H_k, 0100)$  and  $E(H_k, 1100)$  are compared in a similar way.
- (c) If  $F(H_k, 1100) \leq F(H_k, 0000) - 2$  then  $E(H_k, 1100)$  is preferred; if  $F(H_k, 0000) \leq F(H_k, 1100)$  then  $E(H_k, 0000)$  is preferred. The only possible tie occurs precisely when  $F(H_k, 0000) = F(H_k, 1100) + 1$ . We will represent the better embedding, in this case, by  $E(H_k, **00)$ .

Running through all cases (a)–(c) yields the best embedding for  $H_k$ , with a possible tie between (0000) and (1100). We still have to decide how to permute the  $H_k$ 's, and how to rotate each individual  $H_k$ . This is done as follows. Assume first that no ties occurred, and for each  $H_k$  with  $k \neq l, r$ , we create a binary vector of two entries corresponding to the best embedding, together with an additional vector  $(b_l, b_r)$ , where  $b_l$  and  $b_r$  specify the status of the inside face of  $H_l$  and  $H_r$ , respectively. Now we have to order these vectors cyclically and rotate them so that the total number of consecutive vectors  $(\alpha, \beta)$  followed by  $(\gamma, \lambda)$  with  $\beta + \gamma \geq 1$  is minimum. For example, if the vectors are  $(0, 0)(0, 1)(0, 1)(0, 0)(0, 1)(0, 1)(1, 1)(1, 1)$ , the best ordering is  $(0, 1)(1, 0)(0, 0)(0, 0)(0, 1)(1, 1)(1, 1)(1, 0)$  of value 4. In general, is it easy to see that an optimal arrangement is obtained by putting all (1,1)'s in a string; if there is at least one (0, 1) put it at one end of the string; if there is another, put it at the other end; next pair up all remaining (0, 1)'s (at most one will not be paired up), and pair up all (0, 0)'s. Let  $n_{ij}$  equal the number of  $(i, j)$ 's; the value will be

$$\begin{aligned} n_{11} + 1 + \left\lceil \frac{n_{01}}{2} \right\rceil - 1 &= n_{11} + \left\lceil \frac{n_{01}}{2} \right\rceil \quad \text{if } n_{01} > 0, \\ n_{11} + 1 &\quad \text{if } n_{01} = 0, n_{00} > 0, \\ n_{11} &\quad \text{if } n_{01} = n_{00} = 0. \end{aligned}$$

Now, if a tie occurred in at least one  $H_k$ , i.e.,  $n_{**} > 0$ , using a (1, 1) embedding increases the  $n_{11}$  count by one but saves one internal face. Therefore, we either make all  $(*, *)$ 's into (1, 1)'s, or we make all of them into (0, 0)'s. That is, if  $n_{**} > 0$  and

$n_{01} > 0$  or  $n_{00} > 0$ , then make all  $(*, *)$ 's into  $(0, 0)$ 's, and if  $n_{01} = n_{00} = 0$ , then make them all into  $(1, 1)$ 's.

In this way, we compute the value  $m(b_l, b_r)$  of an optimal arrangement to account for faces between the  $H_i$ 's. Let  $f_k(b_l, b_r)$  be the number of internal faces used by each  $H_k$  in the optimal embedding. Thus,

$$F(H, xy00) = \min_{l,r} \left\{ \min_{b_l, b_r} \left\{ F(H_l, xb_l00) + F(H_r, b_r y00) + \sum_{k \neq l,r} f_k(b_l, b_r) + m(b_l, b_r) \right\} \right\}.$$

The above procedure can be implemented easily in a quadratic amount of work. We next sketch how to improve on it so that the  $F(\cdot)$  parameters are obtained in linear time. Assume that we want to compute  $F(H, xy00)$  (the general case is simpler), and fix  $b_l$  and  $b_r$  (there are four cases), without yet fixing  $H_l$  or  $H_r$ . We first consider the case where there are *no* ties in any of the blocks. Then the quantity  $m(b_l, b_r)$  is well defined. We can also select a "best" block to act as a left end, namely, select  $k$  such that

$$F(H_k, xb_l00) - f_k(b_l, b_r)$$

is minimized, where we use the notation given above. Similarly, we can choose a "best" right block. It is not difficult to show that these blocks should indeed occupy the ends.

We now pass to the case of ties. If either

- (i)  $b_l + b_r \leq 1$ , or
- (ii)  $n_{01} + n_{00}$  is 0 or at least 3,

Then the ties will always be resolved, and we proceed as in the previous paragraph. Otherwise, we also try all possible ways of using, as end blocks, the (at most two) blocks whose best embedding is of type 01 or 00. This only adds a constant number of additional cases. Hence, this case can indeed be computed in linear time.







(3) *Messy Case.* Assume we have computed the m.m.b.'s. Let  $W$  be an m.m.b., with extremes  $v, w$ ; we want to describe the behavior of  $W$  with a small gadget.

Suppose  $W$  is an interior m.m.b.; that is, the extremes of  $W$  are contained in the interior of  $H'$ . We can compare the possible embeddings of  $W$  as follows:

- (a) For fixed  $z$  and  $w$ , we compare each pair  $(xyzw)$  and  $(x'y'zw)$  precisely as in case (2) of DYN. For example, if  $F(W, 1000) \leq F(W, 1100)$ , then  $E(W, 1000)$  is preferred over  $E(W, 1100)$  (and if the inequalities are reversed, otherwise). Recall that we have a tie between  $E(W, 0000)$  and  $E(W, 1100)$  if  $F(W, 0000) = 1 + F(W, 1100)$ .
- (b) If  $F(W, 1000) \geq F(W, 00zw)$  with  $z + w \geq 1$ , then  $E(W, 00zw)$  is preferred over  $E(W, 1000)$ . The reason for this is that if in an optimal embedding of  $H$  we use  $E(W, 1000)$ , we can instead use  $E(W, 00zw)$  and keep everything else fixed to obtain a feasible embedding that must also be optimal.
- (c) If  $F(W, 0000) \leq F(W, 0011)$  then  $E(W, 0000)$  is preferred; if  $F(W, 0011) + 1 \leq F(W, 0000)$  then  $E(W, 0011)$  is preferred.
- (d) If  $F(W, 1100) + 2 \leq F(W, 0011)$  then  $E(W, 1100)$  is preferred; if  $F(W, 0011) \leq F(W, 1100)$  then  $E(W, 0011)$  is preferred, with a tie if  $F(W, 1100) + 1 = F(W, 0011)$ .
- (e) In a similar way, we compare  $E(W, 1100)$  with  $E(W, 00zw)$  where  $z + w = 1$ , with a tie if  $F(W, 1100) + 1 = F(W, 00zw)$ . We may even have a three way tie between  $E(W, 1100)$ ,  $E(W, 0010)$  and  $E(W, 0001)$ .

Now suppose we are able to select a best embedding  $E(W, x^*y^*z^*w^*)$ ; that is, there are no ties for best. Then we replace  $W$  with the appropriate graph in Table 1

TABLE 1  
Replacement graphs with no ties.

(0000)	
(1000)	
(1100)	
(0011)	
(0010)	
(0001)	

(where black vertices represent vertices of  $D$ ). Let  $\hat{H}$  be the new graph. It is easy to prove that  $F(\hat{H}, xyzw) + F(W, x^*y^*z^*w^*) = F(H, xyzw)$ . Thus, we solve the problem on  $\hat{H}$  first and then add the weight  $F(W, x^*y^*z^*w^*)$ .

By induction, we can replace any set of interior m.m.b.'s (with no ties for best embedding) whenever they have no common extremes. If, on the other hand, say  $W_1, \dots, W_k$  have a common extreme  $v \in D$ , then: (i) if the best embedding for each  $W_k$  prescribes that  $v$  not be covered, then leave  $v$  uncovered in each replacement graph, and (ii) if, in at least one  $W_j$ , the best embedding covers  $v$ , then use the same replacement for each  $W_i$ , except that  $v$  is removed from  $D$ .

A few complications arise if a particular interior m.m.b.  $W$  has ties for best embeddings. The list of all possible ties is given here:

$$(T1) \quad F(W, 0000) = F(W, 1100) + 1,$$

$$(T2) \quad F(W, 0011) = F(W, 1100) + 1,$$

$$(T3) \quad F(W, 1100) + 1 = F(W, 0010),$$


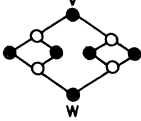
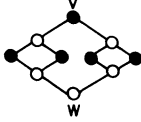
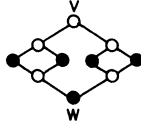
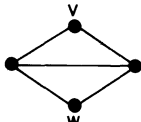

$$(T4) \quad F(W, 1100) + 1 = F(W, 0001),$$

$$(T5) \quad F(W, 1100) + 1 = F(W, 0010) = F(W, 0001),$$

$$(T6) \quad F(W, 0010) = F(W, 0001).$$

Table 2 contains the replacement graph to be used for each tie, together with the appropriate weight to be added after solving. It is not difficult to verify that these are all correct; here we will do so for the most complicated case (T2). Let  $\hat{H}$  be obtained from  $H$  by replacing  $W$  with the corresponding graph in case (T2). Now in any optimal embedding  $E(H, xyzw)$ , at least one internal face of the replacement graph will be

TABLE 2  
Replacement graphs with ties.

CASE	GRAPH	WEIGHT
T1		$F(W, 1100)$
T2		$F(W, 1100) - 1$
T3		$F(W, 1100) - 1$
T4		$F(W, 1100) - 1$
T5		$F(W, 1100)$
T6		$F(W, 0010)$

NOTICE THAT ALL WEIGHTS ARE NONNEGATIVE

used. It is easy to see that, without loss of generality, it is either the face containing  $v$  and  $w$  (but none of the other faces), or the two other faces (but not the one containing  $v$  and  $w$ ). Given our choice for the weight,  $F(W, 1100) - 1$ , this gives the correct result.

Thus, we can replace interior m.m.b.'s with small graphs, with the small caveat concerning extremes that are common to several m.m.b.'s as given before. Noninterior m.m.b.'s are dealt with similarly:

(i) Suppose we want to compute  $F(H, 0000)$ ; then we replace every m.m.b. using the above rules.

(ii) Suppose we are trying to compute  $F(H, 00zw)$  where  $z + w \geq 1$ ; say,  $F(H, 0010)$  (i.e.,  $u_1$  is not covered). Then we proceed exactly as in (i), except that now if an m.m.b.  $W$  contains  $u_1$  as an extreme, we use the best embedding for  $W$  of the form  $E(W, xy1w)$ , and all faces of the resulting graph  $\hat{H}$  that are incident on  $u_1$  are rejected. (There is a small wrinkle if there is an m.m.b.  $W_1$  with extremes  $u_1$  and  $v, v \in D$ , and such that  $v$  is an extreme of only one other m.m.b.,  $W_2$ . In this case we may have to force  $v$  to be covered by  $W_1$  or  $W_2$ . Since there are at most two vertices such as  $v$ , this yields only four or fewer problems.)

In cases (i) and (ii), we use Algorithm XTND in the resulting graph  $\hat{H}$  after the replacement. If XTND finds a cover of cardinality  $k$  or less, then we add the sum of the weights of the replacement graphs to the cardinality of a minimum cover of  $\hat{H}$ . If, on the other hand, XTND finds that  $\hat{H}$  has no covers of cardinality  $k$  or less, then



the corresponding embedding  $E(H, 0000)$  or  $E(H, 0010)$  should not be considered. (Similarly with  $F(H, 0001)$  and  $F(H, 0011)$ .)

(iii) The final case occurs for  $F(H, xyzw)$  where  $x + z \geq 1$ ; say, we want to compute  $F(H, 10zw)$ . We proceed in two stages. Let  $p_1$  and  $p_2$  be the two paths from  $u_1$  to  $u_2$  in the outer face of  $H'$ . Then we obtain  $\hat{H}$  as for computing  $F(H, 00zw)$ , and we remove from  $D$  all vertices in the segment of the outer face of  $\hat{H}$  corresponding to  $p_1$ . We then apply XTND to this graph, add the weights of the replacement graphs to the returned cardinality, and denote the obtained quantity by  $m(p_1)$ . Similarly compute  $m(p_2)$ . Then  $m(H, z, w) = \min(m(p_1), m(p_2))$  is the minimum number of faces needed to cover  $D \cap H$  (with the exception of  $u_1$  and  $u_2$  as indicated by  $z$  and  $w$ ) where we do not require that we cover one of the segments of the outer face.

Now if  $m(H, z, w) < F(H, 00zw)$ , then we are done; set  $m(H, z, w) = F(H, 10zw)$ . If  $m(H, z, w) = F(H, 00zw)$ , then we proceed to the second stage. Let  $W$  be an arbitrary m.m.b. whose corresponding edge in  $H'$  is contained in  $p$  or  $p'$ . Let  $\hat{H}(W)$  be the graph obtained from  $H$  by replacing  $W$  with the best embedding of the form  $E(W, 1y00)$ , and all other m.m.b.'s replaced with graphs as for cases (i) and (ii).

Notice that the replacement graph for  $W$  will contain a vertex of  $D$  in the outer face of  $\hat{H}(W)$  (see Table 1); call this vertex  $w'$ , and set  $f(W)$  to be the min cardinality of face cover of  $D \cap \hat{H}(W)$ , with all faces incident to  $w'$  rejected plus weight of replacement graphs. Then  $\min_w f(W)$  finds the best embedding of  $H$  that does not cover at least one vertex (which is not an extreme of an m.m.b.) in the outer face. In a similar way, we compute all quantities  $g(v)$ , where  $v \in D$  is extreme of an m.m.b. in the outer face of  $H'$  that we want not to cover. Then  $\min\{\min_w f(W), \min_v g(v)\}$  is the quantity  $F(H, 1000)$  we seek.

There is one shortcut that we can take to improve the algorithm above. Suppose there are at least  $k$  possible graphs  $W$  for which  $F(W, 1y00) \leq k$  that can be used to compute  $\min_w f(W)$ . Now, the fact that we have reached Stage 2 implies that  $F(H, 10zw) \geq F(H, 00zw)$ . However, any internal face of  $H$  can cover at most one of the vertices of  $D$  counted above; otherwise,  $H'$  would contain a cutset of two vertices in one of the segments  $p_1$  and  $p_2$ , which is impossible. Therefore,  $F(H, 00zw) > k$ , and we can avoid Stage 2 altogether. Similar considerations apply towards computing  $\min_v g(v)$ . Consequently, we can assume that each of the minimums in Stage 2 at most has  $O(k)$  arguments.

## REFERENCES

- [Ba] B. BAKER, *Approximation algorithms for NP-complete problems on planar graphs*, Proc. 24th Annual Symposium on the Foundations of Computer Science, (1983), pp. 265-273.
- [BM] D. BIENSTOCK AND C. C. MONMA, *On the complexity of embedding planar graphs to minimize certain distance measures*, submitted.
- [CNP] G. CORNUEJOLS, D. NADDEF AND W. R. PULLEYBLANK, *Halin graphs and the traveling salesman problem*, Math. Programming, 26 (1983), pp. 287-294.
- [EMV] R. E. ERICKSON, C. L. MONMA AND A. F. VEINOTT, JR., *Send-and-split method for minimum-concave-cost network flows*, Math. Oper. Res., to appear.
- [FHS] M. FELLOWS, F. HICKLING AND M. SYSLO, *A topological characterization and hard graph problems*, Extended Abstract, University of New Mexico, Albuquerque, NM.
- [GJ1] M. R. GAREY AND D. S. JOHNSON, *The rectilinear Steiner tree problem is NP-complete*, SIAM J. Appl. Math., 32 (1977), pp. 835-859.
- [GJ2] ———, *Computers And Intractability: A Guide To The Theory Of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.

- [GJS] M. R. GAREY, D. S. JOHNSON AND L. STOCKMEYER, *Some simplified NP-complete graph problems*, Theoret. Comput. Sci., 1 (1976), pp. 237-267.
- [HT1] J. E. HOPCROFT AND R. E. TARJAN, *Dividing a graph into triconnected components*, this Journal (1973), pp. 135-158.
- [HT2] ———, *Efficient planarity testing*, J. Assoc. Comput. Mach., 21 (1974), pp. 549-568.
- [LT] R. J. LIPTON AND R. E. TARJAN, *Applications of the planar separator theorem*, this Journal, 9 (1980), 615-627.
- [Pr] J. S. PROVAN, *Convexity and the Steiner tree problem*, Technical Report, TR-85/3, University of North Carolina, January 1985.
- [RS] N. ROBERTSON AND P. D. SEYMOUR, *Graph minors. III. Planar tree-width*, J. Combin. Theory Ser. B, 36 (1984), pp. 49-64.
- [Se] P. D. SEYMOUR, personal communication, August 1986.
- [Wh] H. WHITNEY, *Two-isomorphic graphs*, Amer. J. of Math. Soc., 34 (1932), pp. 339-362.
- [Wi] P. WINTER, *Steiner problems in networks: a survey*, Networks, to appear.

## FAST SIMULATION OF TURING MACHINES BY RANDOM ACCESS MACHINES\*

JYRKI KATAJAINEN<sup>†</sup>, JAN VAN LEEUWEN<sup>‡</sup> AND MARTTI PENTTONEN<sup>†</sup>

**Abstract.** We prove that a  $T(n)$  time-bounded,  $S(n)$  space-bounded and  $U(n)$  output-length-bounded Turing machine can be simulated in  $O(T(n) + (n + U(n)) \log \log S(n))$  time by a random access machine (with no multiplication or division instructions) under the logarithmic cost criterion.

**Key words.** Turing machine, random access machine, simulation, computational complexity

**1. Introduction.** In the theory of computation one uses both the Turing Machine (TM) and the Random Access Machine (RAM) as standard models of effective computing (see e.g., [1]). Whereas the models are vastly different in detail, it is well known that the machines are “equivalent” in computational strength. More precisely, one can show that the machines are polynomially related in the sense of computational complexity theory (see [1, § 1.7] or [2]): a TM can simulate a RAM in  $O(T(n)^2)$  time and a RAM can simulate a TM in  $O(T(n) \log T(n))$  time, where  $T(n)$  is the time complexity of the simulated machine and RAMs are assumed to use the so-called logarithmic cost criterion. In the result, RAMs are assumed without explicit “single” multiplication or division instructions in their instruction set. Slot and van Emde Boas [15] have shown that TMs and RAMs can simulate one another within only a constant factor of extra space.

When we speak of the simulation of one machine by another, we require that on the same input the latter machine produce the same output as the former one. In general, the simulating machine passes through an analogous computation, but it may also contain some auxiliary computations intermixed. These auxiliary steps are, of course, included in the complexity of the simulating machine.

Several studies have attempted to refine or lower the simulation costs between the two models, especially for the case of simulating RAMs by TMs (see e.g., Wiedermann [16] for some recent results). In this paper we consider the efficient simulation of TMs by RAMs. Let  $T(n)$  denote the time complexity,  $S(n)$  the space complexity and  $U(n)$  an upperbound on the length of the longest output on inputs of length  $n$ . The following results are known.

**THEOREM A** (Folklore, see e.g., [1, § 1.7]). *A TM can be simulated in  $O(T(n) \log S(n))$  time by a RAM (with no multiplication or division instructions) under the logarithmic cost criterion.*

**THEOREM B** (Paul [11, § 3.3]). *A TM can be simulated in  $O(T(n) + n \log n + U(n) \log T(n))$  time by a RAM (with no multiplication or division instructions) under the logarithmic cost criterion.*

Theorem B shows that TMs can be simulated by RAMs with no essential time-loss provided  $T(n) \geq n \cdot \log n$  and  $U(n) \leq T(n)/\log T(n)$ . Note, however, that [11, § 3.3] assumes stronger RAMs with shift instructions, that is (multiplication and) division by 2. Related results are also found in literature: Dymond and Tompa [4] proved that

---

\* Received by the editors March 5, 1986; accepted for publication (in revised form) April 16, 1987.

<sup>†</sup> Department of Computer Science, University of Turku, 20500 Turku, Finland.

<sup>‡</sup> Department of Computer Science, University of Utrecht, P.O. Box 80012, 3508 TA Utrecht, The Netherlands.

a TM can be simulated in time  $\sqrt{T(n)}$  by a parallel RAM. Hopcroft, Paul and Valiant [7] simulate a TM in time  $T(n)/\log T(n)$  by a unit-cost RAM. Robson [13] speeds up a TM computation by a probabilistic RAM.

In this paper we improve Theorem A to Theorem C as follows.

**THEOREM C.** *A TM can be simulated in  $O(T(n) \log \log S(n))$  time by a RAM (with no multiplication or division instructions) under the logarithmic cost criterion.*

We also can improve Theorem B to Theorem D as follows.

**THEOREM D.** *A TM can be simulated in  $O(T(n) + (n + U(n)) \log \log S(n))$  time by a RAM (with no multiplication, division or shift instructions) under the logarithmic cost criterion.*

This is, indeed, an improvement of Theorem B because in the case  $\log n < \log \log S(n)$ ,  $T(n)$  is the dominating term. This theorem improves also Theorem C, because  $T(n) \geq n$  and  $T(n) \geq U(n)$ . None of the results assume that  $T(n)$ ,  $S(n)$  or  $U(n)$  are constructible.

As an example of the use of Theorem C we mention the following corollary.

**COROLLARY E.** *Any linear time TM can be simulated in  $O(n \log \log n)$  logarithmic time by a RAM (with no multiplication or division instructions).*

It follows that e.g. the reversal of a string of  $n$  inputs can be output by a RAM in  $O(n \log \log n)$  units of logarithmic time. We can apply the above corollary also to the string-matching problem, where the task is to find all occurrences of a given pattern of length  $m$  from the text of length  $n$ ,  $m \leq n$ . The string-matching can be done in  $O(n)$  time on a TM as shown by Fischer and Paterson [5] (see also [6]), and therefore in  $O(n \log \log n)$  units of logarithmic time on a RAM.

In language acceptance the size of the output is constant. Hence, by Theorem D we also have

**COROLLARY F.** *Any language accepted by a  $T(n)$  time-bounded,  $S(n)$  space-bounded TM can be accepted in  $O(T(n) + n \log \log S(n))$  time on a RAM.*

The paper is organized as follows. In § 2 we recapitulate some basic definitions. In § 3 we prove Theorem C and Theorem D. Finally, in § 4 we discuss how these improvements were achieved and how the results could probably be improved further.

**2. Machine models.** We define TMs and RAMs such that they appear as instances of the same abstract model, following the guidelines of [14]. The machines have very similar input, output and control structures, but differ in the structure and the use of the memory. The definition of TMs and RAMs is included to fix the particular instruction sets.

**2.1. Turing machines.** We describe the “parts” of a Turing machine without much formal notation. We assume that the input, output and work-tape alphabet is  $\{0, 1\}$  and refer to the individual symbols as bits. A (multitape) TM consists of the following parts (compare [1, § 1.6]):

(i) *a one-way read-only input tape*, containing a bit string followed by an endmarker #.

(ii) *a one-way write-only output tape*, where a bit string will be written.

(iii) *k two-way read-write work-tapes* (“memory”), containing bits in successive memory cells. The tapes are two-way infinite. On each tape there is a separate read-write head that can be activated for reading, writing or moving one tape-cell to the left or to the right.

(iv) *a TM program*, which is a finite sequence of labelled or unlabelled instructions from a fixed instruction set (see below). No two instructions should carry the same label.

The instruction set of a TM contains eight instruction types:

(1) **input**  $\lambda_0, \lambda_1, \lambda_{\#}$ : causes a “next” input symbol  $\alpha$  to be read, and the input head moves one cell to the right (except on  $\#$ ). Depending on whether  $\alpha$  is 0, 1 or  $\#$ , control is transferred to the instruction with label  $\lambda_0, \lambda_1, \lambda_{\#}$ .

(2) **output**  $\beta$ : causes a bit  $\beta$  to be output, and the output head moves one cell to the right.

(3) **jump**  $\lambda$ : transfers control to the instruction with label  $\lambda$ .

(4) **halt**: halts the program.

(5) **head**  $i$ : activates the read-write head on the  $i$ th work-tape ( $1 \leq i \leq k$ ). Only one read-write head will be active at a time.

(6) **write**  $\beta$ : causes a bit  $\beta$  to be written in the tape-cell designated by the active read-write head.

(7) **branch**  $\lambda_0, \lambda_1$ : causes the bit  $\beta$  to be read from the tape-cell designated by the active head. Depending on whether  $\beta$  is 0 or 1 control is transferred to the instruction with label  $\lambda_0$  or  $\lambda_1$ .

(8) **move**  $\delta$  (with  $\delta \in \{L, R\}$ ): moves the active read-write head one cell to the left or to the right depending on whether  $\delta$  is  $L$  or  $R$ .

We assume that initially all work-tapes contain 0 in every cell, and that head 1 is active. The computation starts from the first instruction and thereafter the instructions of a program are executed in their successive order unless a jump instruction orders otherwise.

The *time complexity*  $T(n)$  of a TM is the largest number of instructions executed in halting computations on inputs of length  $n$ . The *space complexity*  $S(n)$  is the largest number of cells occupied on any work-tape in halting computations on inputs of length  $n$ . The *output complexity*  $U(n)$  is the length of the longest output produced in halting computations on inputs of length  $n$ .

Because a TM with a two-way infinite tape can be simulated by a TM with a one-way infinite tape in real time (see e.g., [8, § 7.5]), we shall assume that the work-tapes of a TM are one-way infinite, say infinite to the right. Initially all read-write heads are positioned on the leftmost cell of their work-tape. By the standard construction used in the above simulation [8, § 7.5], we can further assume that a read-write head is never moved off the left end of the work-tape. (Thus the computation is stopped by a halt instruction, not by the fall of a read-write head.) Although in the construction the tape alphabet is enlarged, it is straightforward to return into the binary alphabet (see also [8, § 7.8]).

**2.2. Random access machines.** In describing the random access machine, we only emphasize the parts that are different from those of a TM. Parts (i) and (ii) are very similar for a RAM but instead of (iii) one has the following set-up (compare [1, § 1.2]):

(iii') a special register called the *accumulator* (AC) and a countable sequence of ordinary *registers* (“memory”) indexed by the nonnegative integers (used as addresses). Each register can hold an arbitrary nonnegative integer in binary notation. Only data stored in the AC can be operated upon.

A RAM *program* is defined as in (iv), but the instruction set of a RAM differs from the instruction set of a TM. In instructions, the contents of register  $j$  are denoted by  $\langle j \rangle$ . The instruction set of a RAM contains twelve instruction types:

(1')–(4'): similar to the instructions (1)–(4) of a TM.

(5') **jzero**  $\lambda$ : transfers control to the instruction with label  $\lambda$  if  $\langle \text{AC} \rangle = 0$ , and continues to next instruction otherwise.

(6') **load**  $= j$ : loads the integer  $j$  into AC.

(7') **load**  $j$ : loads  $\langle j \rangle$  into AC.

- (8') **load** \* $j$ : loads  $\langle\langle j \rangle\rangle$  into AC (“indirect addressing”).
- (9') **store**  $j$ : stores  $\langle AC \rangle$  into register  $j$ .
- (10') **store** \* $j$ : stores  $\langle AC \rangle$  into register  $\langle j \rangle$ .
- (11') **add**  $j$ : adds  $\langle j \rangle$  to the current value in AC.
- (12') **sub**  $j$ : subtracts  $\langle j \rangle$  from the current value in AC. In order to keep the contents of the AC nonnegative, we assume that subtraction is proper, i.e., the result is 0 whenever  $\langle AC \rangle \leq \langle j \rangle$ .

We assume that all registers, including the AC, initially contain 0. Memory need not be used contiguously.

We do not simply count the number of instructions executed in a RAM program but use the so-called *logarithmic cost criterion*: the “time” charged for an instruction is equal to the sum of the sizes (in bits) of the integers (addresses and data) involved in its execution. Note that the size of a positive integer  $m$  is  $\lceil \log(m+1) \rceil \sim \log m$ , and the size of zero is 1. The *time complexity*  $T(n)$  of a RAM is the largest amount of time, measured according to the logarithmic cost criterion, used in halting computations on inputs of length  $n$ . See Slot and van Emde Boas [15] for notions of space complexity for RAMs.

It will be convenient to use various extensions to the basic RAM instruction set, provided that the execution time is adequately measured by the logarithmic cost criterion. By using subtractions and a trick introduced in [13, pp. 495–496], one can easily show that this is the case for comparison instructions. It should be noted, however, that the properness of the subtraction operation is not needed anywhere in the subsequent proofs because we always know which of the two operands is greater. Also in some algorithms it is convenient to have a RAM with  $k$  separate *memories* (or *arrays* as called by Cook and Reckhow [2]),  $k > 1$ , each consisting of a countable sequence of registers indexed  $0, 1, 2, \dots$ . We call this a “multimemory” RAM.

LEMMA 2.2.1. *Every  $T(n)$  time-bounded multimemory RAM can be simulated in  $O(T(n))$  time by an ordinary RAM.*

*Proof.* The technique was essentially given in [2]. The idea is simply to interleave the RAM memories into one, using addresses  $i + kj - 1$  for register  $j$  of the  $i$ th memory ( $1 \leq i \leq k, j \geq 0$ ). Those addresses can be computed in  $O(k \log j)$  time, which multiplies the time bound by a constant factor.  $\square$

It is important to state explicitly the basic instruction set of the RAM. However, for the sake of the readability, we extend it with some Pascal-like control structures that have obvious translations to the basic RAM instructions.

**3. The simulation of a TM by a RAM.** Consider a  $T(n)$  time-bounded,  $S(n)$  space-bounded TM. The simple idea underlying Theorem A is to represent the cells of the work-tapes in consecutive registers of a RAM, with additional registers containing current read-write head positions. Every step of the TM is easily simulated in  $O(\log S(n))$  time on a RAM, assuming the logarithmic cost criterion. In the simulation underlying Theorem B, a saving in the cost per step is achieved by precomputing in a table the action of the TM on all blocks of a suitable size. A subcomputation corresponding to the size of the block reduces to a table look-up.

We will also use blocking to balance the costs of the address and the contents of a memory location. At first we keep the idea of step-wise simulation. We use blocking merely to localize the active region of the work-tape during a time-interval. For step-wise simulation, the active block is swapped to a low-indexed region of the memory in order to save in the access time under logarithmic cost criterion. (It is interesting to compare this technique to the swapping of pages to and from disk in paged virtual

memory operating systems, see e.g., Deitel [3]. Another analogy is a hardware cache. The cache idea was also used by Loui in [9].)

A serious problem is determining the optimal block size together with efficient algorithms for unpacking and packing blocks. A further problem is that  $T(n)$  and  $S(n)$  need not be constructible. For simplicity, we denote these values by  $T$  and  $S$ . In § 3.1 we assume that  $n$  and  $S(n)$  are known at the beginning of the computation, and determine the optimal block size in this case. In § 3.2 we remove this assumption and notice that the same time bound holds even though the block size is determined dynamically during the computation. Ultimately, in § 3.3 we improve the table look-up method of [11]. Interestingly, the packing and unpacking techniques developed in §§ 3.1 and 3.2 will now be useful in reading input blocks and writing output blocks.

**3.1. The static step-wise simulation using blocked memory.** We assume now that  $n$  and  $S(n)$  are known in advance. We will begin the basic simulation algorithm together with the necessary blocking and deblocking algorithms. By the time analysis of this algorithm we determine the optimal block size. For simplicity, we speak of one TM tape only; if there are many work-tapes, they are treated analogously, independently.

The basic idea of the simulation is to divide the tape into  $S/b$  blocks of  $b$  cells. Hence,  $b$  successive cells of the TM are represented by a number (in the range  $0, \dots, 2^b - 1$ ) in a register of the RAM. By Lemma 2.2.1 we can assume that the RAM keeps the “blocked” representation of the work-tape in a separate memory. The position of the tape head of the TM is indicated by an address to the active block (a number in the range  $1, \dots, S/b$ ) together with an address within a block ( $0, \dots, b - 1$ ), both stored in fixed RAM registers. One simulation cycle, corresponding to  $b$  steps of the TM, consists of accessing the active block with two neighbours, unpacking them to low-indexed registers, directly simulating the next  $b$  steps of the TM, and packing the updated bits back to the same registers. The neighbouring blocks are taken along in order to guarantee that in all cases  $b$  simulation steps can be taken staying in the unpacked zone. This unpacked  $3b$  bit zone, kept in a separate memory, is called the *window*.

The simulation of the single TM instructions is quite obvious; it is done as in the proof of Theorem A. It is easy to construct a procedure simulating an instruction of the TM, updating the contents of the window, the head address and the current instruction label. Now we can represent the simulation in the form of a RAM program as follows:

```

procedure simulate
  {Suppose that the block size  $b$  is given.}
  activeblock := 1
  {the first block is active, with empty left neighbour}
  head := the first address of the middle block of the window
  loop {until a halt instruction in the simulation}
    loadwindow(activeblock,  $b$ )
    for  $b$  times do simulate an instruction
    storewindow(activeblock,  $b$ )
    if head moved to neighbour then
      update head and activeblock addresses

```

The procedure `loadwindow` fetches the contents of the active block and its neighbours into low-indexed registers, and unpacks the  $b$ -bit integers. The procedure `storewindow` packs the window blocks, and stores them by overwriting their older copies.

We shall now attack the problem of packing and unpacking the blocks efficiently. As our RAM model does not include division or shift instructions, we have to invent another method for finding the bit representation of a number and vice versa. We will see that unpacking and packing can be done efficiently with precomputed tables.

As a first attempt one could decode numbers to bit-strings by building a table that gives the decoding directly. For example the table could contain the  $b$  bits of a number  $n (<2^b)$  in the registers  $nb, nb + 1, \dots, nb + b - 1$ . A disadvantage of this method is that, while bits are obtained directly, the access of them may cost  $O(\log n)$ . For this reason loading a window takes  $O(b^2)$  time. By a similar analysis as what follows, one can see that this would give  $O(T\sqrt{\log S})$  simulation algorithm. However, we can do unpacking and packing in  $O(b \log b)$  time.

The efficient decoding of a number to its bit representation and vice versa is based on a divide-and-conquer strategy with precomputed shift tables. We will first build the necessary tables and then give the unpacking and packing algorithms.

We assume that each table is stored in its own memory. We will need tables *lshift*, *rshift*, *origin* and *power*. By *lshift(i)*, *rshift(i)*,  $\dots$ , we denote the contents of the register  $i$  reserved for *lshift*, *rshift*,  $\dots$ .

The tables *lshift* and *rshift* in Fig. 1 contain as subtables shift tables for 1-bit numbers, 2-bit numbers, 4-bit numbers etc. The divide-and-conquer strategy implies that  $b$ -bit numbers are shifted  $b/2$  bits to the right or  $b$  bits to the left. The entries of the tables are numbers rather than bit strings. Thus, for example, the number  $75 = 01001011_2$  has the right shift  $4 = 0100_2$  and  $4 = 0100_2$  has the left shift  $64 = 01000000_2$ . The origin table expresses where subtables begin: The shift tables for  $2^i$ -bit numbers begin at *origin(i)*.

<i>origin</i> :	register	0	1	2	3	4	$\dots$	$i$						
	contents	0	2	6	22	278	$\dots$	$2^{2^0} + 2^{2^1} + \dots + 2^{2^{i-1}}$						
<i>rshift</i> :	register	0	1	2	3	4	5	6	7	8	$\dots$	21	$\dots$	<i>origin(i) + j</i>
	block size	1 bit	2 bits				4 bits				$\dots$	$2^i$ bits		
	block value	0	1	0	1	2	3	0	1	2	$\dots$	15	$\dots$	$j$
	contents	0	0	0	0	1	1	0	0	0	$\dots$	3	$\dots$	$j \text{ div } 2^{2^{i-1}}$
<i>lshift</i> :	register	0	1	2	3	4	5	6	7	8	$\dots$	21	$\dots$	<i>origin(i) + j</i>
	block size	1 bit	2 bits				4 bits				$\dots$	$2^i$ bits		
	block value	0	1	0	1	2	3	0	1	2	$\dots$	15	$\dots$	$j$
	contents	0	2	0	4	8	12	0	16	32	$\dots$	240	$\dots$	$j \cdot 2^{2^i}$

FIG. 1. The origin, rshift and lshift tables.

We have to first analyze how much the building of the tables costs.

LEMMA 3.1.1. *The tables origin, rshift and lshift up to block size  $b (=2^k)$  can be built in logarithmic time  $O(b2^b)$ .*

*Proof.* Assuming that the values *origin(i-2)* and *origin(i-1)* are already computed, the following program will compute the  $i$ th origin value,  $i \geq 2$ .

```

procedure buildorigin( $i$ )
 $t := \text{origin}(i-1) - \text{origin}(i-2)$   { $t := 2^{2^{i-2}}$ }
 $\text{origin}(i) := \text{origin}(i-1)$ 
for  $t$  times do  $\text{origin}(i) := \text{origin}(i) + t$ 
    
```

Clearly, its time complexity is  $O(2^{2^{i-2}} \cdot 2^{i-1})$ .



When constructing the  $i$ th  $rshift$  and  $lshift$  subtables we can use the origin values for  $i-1$ ,  $i$  and  $i+1$ .

```

procedure build $rshift$ ( $i$ )
 $j := origin(i)$ ;  $x := 0$ ;  $t := origin(i) - origin(i-1)$   { $t := 2^{2^{i-1}}$ }
for  $t$  times do
  for  $t$  times do  $rshift(j) := x$ ;  $j := j + 1$ 
   $x := x + 1$ 

procedure build $lshift$ ( $i$ )
 $j := origin(i)$ ;  $x := 0$ ;  $t := origin(i+1) - origin(i)$   { $t := 2^{2^i}$ }
for  $t$  times do  $lshift(j) := x$ ;  $j := j + 1$ ;  $x := x + t$ 

```

The execution of both procedures requires  $O(2^{2^i} \cdot 2^i)$  time. Thus the tables up to  $k$  can be constructed in time  $O(\sum_{i=1}^k 2^{2^i} \cdot 2^i) = O(2^k \cdot 2^{2^k}) = O(b2^b)$ .  $\square$

We also need powers of 2 for unpacking numbers to bit strings, and for packing bit strings to numbers. It is useful to precompute them, too, in the table *power*.

LEMMA 3.1.2. *The table  $power(i) = 2^i$  up to  $k$ th power can be built in  $O(k^2)$  logarithmic time.*

*Proof.* A new power can be computed by doubling the previous one by addition. This method gives the time bound  $O(k^2)$ .  $\square$

Now we are ready to present the unpacking and packing algorithms.

LEMMA 3.1.3. *Assuming that the tables  $lshift$ ,  $rshift$ ,  $origin$  and  $power$  up to the block size  $b$  are available, it is possible to compute the  $b$ -bit representation of an integer  $n < 2^b$ , and the numeric value of a  $b$ -bit string, both in  $O(b \cdot \log b)$  time.*

*Proof.* The procedure  $unpack(n, j, a)$  unpacks a number  $n < 2^{2^j}$  to its  $2^j$ -bit representation beginning at the  $a$ th register of the window. The procedure is as follows:

```

procedure  $unpack(n, j, a)$ 
if  $j = 0$  then  $window(a) := n$ 
else  $n_1 := rshift(origin(j) + n)$ ;  $n_2 := n - lshift(origin(j-1) + n_1)$ 
   $unpack(n_1, j-1, a)$ ;  $unpack(n_2, j-1, a + power(j-1))$ 

```

For clarity, we have written the algorithm in recursive form. The recursion can be eliminated by using one memory as a stack where the second recursive call is stored while the first is executed. While unpacking a number  $n < 2^j$  there are never more than  $\log j$  calls in the stack. In order to balance access cost it is economical to initiate the stack at the address  $\log b$  and let it grow downwards. If we denote by  $t(x)$  the logarithmic time of unpacking an  $x$ -bit number, by analyzing the program we get

$$t(1) = k_1 \log b,$$

$$t(x) = 2t(x/2) + k_2x + k_3 \log b$$

which gives  $t(b) = O(b \cdot \log b)$ .

The procedure  $pack(a, j, n)$  computes the numeric value of the bit string  $window(a)$ ,  $window(a+1)$ ,  $\dots$ ,  $window(a+2^j-1)$ . Also it is written recursively:

```

procedure  $pack(a, j, n)$ 
if  $j = 0$  then  $n := window(a)$ 
else  $pack(a, j-1, n_1)$ ;  $pack(a + power(j-1), j-1, n_2)$ 
   $n := lshift(origin(j-1) + n_1) + n_2$ 

```

The recursion is controlled as in unpacking. Also the time analysis is analogous.  $\square$

We can now obtain a preliminary version of Theorem C:

THEOREM 3.1.4. *Assuming that  $n$  and  $S(n)$  are known, a  $T(n)$  time-bounded,  $S(n)$  space-bounded TM can be simulated in  $O(T(n) \log \log S(n))$  logarithmic time by a RAM.*

*Proof.* The total time of the static simulation is bounded by

$$T_{\text{RAM}} = b2^b + T/b(\log S + b \cdot \log b + b \cdot \log b + b \cdot \log b + \log S)$$

where  $b2^b$  is needed for the construction of the tables,  $T/b$  is the number of cycles in the simulation loop,  $\log S$  is the cost of loading and storing the blocks,  $b \cdot \log b$  is the cost of unpacking and packing the blocks of the window, and another  $b \cdot \log b$  is needed for the elementary simulation steps. By choosing  $b = c \cdot \log S$ ,  $c < 1$ , we get  $T_{\text{RAM}} = O(T \cdot \log \log S)$ .  $\square$

**3.2. The dynamization.** Until now we have assumed that the space requirement  $S$  of the TM on an input is known in advance, which is not the case. Now we shall remove this assumption by using the well-known technique (see, e.g., [10]) which consists of using one space bound while sufficient and increasing it when necessary. As long as the space bound

$$S_0, S_1, \dots, S_i = 2^{2^{i+1}}, \dots$$

is sufficient, the block size  $b_0, b_1, \dots, b_i = 2^i, \dots$ , is used respectively. Every time a space bound is exceeded, the simulating memory must be reorganized by combining the blocks pairwise. As the size of the blocks grows also the tables must be grown accordingly. It is worth noting that, unlike in [11], input head need not be reset and the computation be restarted from the beginning when the block's size is increased.

The dynamic simulating algorithm gets the following form.

**procedure** simulate

activeblock := 1

head := the first address of the middle block of the window

$b := 1$ ;  $S := 4$

Initialize tables  $l$ shift,  $r$ shift, origin and power;

**loop** {until a **halt** instruction in the simulation}

**while**  $\lceil \log(S+1) \rceil \leq 4b$  **do**

    loadwindow(activeblock,  $b$ )

**for**  $b$  times **do**

      Simulate an instruction

**if** head on a previously unvisited cell **then**  $S := S + 1$

    storewindow(activeblock,  $b$ )

**if** head moved to neighbour **then**

      update head and activeblock addresses

$b := b + b$

  Combine successive blocks pairwise

  Update head and activeblock corresponding to the reorganization

  Update  $l$ shift,  $r$ shift, origin and power to the new block size

The analysis of the dynamic simulation is basically the same as in the previous section, but now there is the extra work of reorganizing the memory. In spite of using wrong block sizes at the beginning, the program still behaves asymptotically fast. Before going to the time analysis of the program, we prove a small counting lemma, also used in [2], [12].

**LEMMA 3.2.1.** *A binary counter of any event occurring  $t$  times during the computation can be maintained in  $O(t)$  time. As a by-product we get easily the length of the binary representation of the count, which is  $\lceil \log(t+1) \rceil$ .*

*Proof.* A memory is reserved for the binary representation of the count. For each bit of the count (without leading zeros), two bits are used. The  $i$ th least significant bit 0 or 1 is represented by  $00_2$  or  $01_2$  in the register  $i$ . If the actual value of the count is  $t$ ,  $2^{j-1} \leq t \leq 2^j$ , then  $11_2$  in the register  $j$  marks the end of the representation.

Each counting step is simulated by a sweep over the representation in the memory, beginning at the register 0.  $01$ 's are replaced by  $00$ 's, until  $00$  is met which becomes  $01$ . However, if  $11$  is met instead of  $00$ , it is replaced by  $01$  and  $11$  is stored in the next register. At any moment, the address of  $11$  corresponds to the logarithm of the count, and the registers below it correspond to the bit representation of the count.

The linear time bound is seen as follows. During a count up to  $t = 2^j$ , the length of the sweep is  $t/2$  times 1,  $t/4$  times 2,  $t/8$  times 3 and so forth. Hence the total logarithmic time is proportional to

$$2^{j-1} \cdot 1 \cdot \log 1 + \dots + 2^{j-i} \cdot i \cdot \log i + \dots + 1 \cdot j \cdot \log j = O(t)$$

as can be seen by the quotient test applied to the infinite series  $2^{-1} \cdot 1 \cdot \log 1 + \dots + 2^{-i} \cdot i \cdot \log i + \dots$ .  $\square$

The complexity analysis of the dynamic simulation is basically the same as in the previous subsection, but we have to take into account the following differences. A counter must be maintained in order to determine when block size must be increased. Whenever the block size is increased, the simulating memory must be reorganized by combining blocks pairwise. This is extra work, following from the fact that nonoptimal block size was used. Fortunately, the amount of extra work caused by reblockings and nonoptimal block size proves to be insignificant.

By Lemma 3.2.1, no more than  $O(S)$  time is used in the computation of  $S$ . The time needed for the comparison  $\lceil \log(S+1) \rceil \leq 4b$  is  $O(\log b)$  which is covered by the  $O(b \cdot \log b)$  used in the simulation. Whenever the test fails,  $b$  is doubled and tables are updated. Note that by Lemmas 3.1.1 and 3.1.2 the total time needed in the construction of the tables is  $O(b2^b) = O(S)$ , because  $2b \leq \log S < 4b$ . Each doubling of  $b$  is followed by reblocking. This is most easily done by the  $l$ shift table in  $O((S_i/b_i) \cdot b_i)$  time. The superexponential growth of the series  $\{S_i\}$  implies that all reblockings can be done in time  $O(S)$ .

For the final analysis of the simulation, assume that  $T_i$  steps of the TM are simulated using block size  $b_i$ . Hence the total time of the simulation is bounded by

$$\begin{aligned} \sum_{i=0}^{\log \log S} (T_i/b_i)(\log(S_i) + b_i \log b_i) &= \sum_{i=0}^{\log \log S} T_i \log b_i \\ &\leq \log b \sum_{i=0}^{\log \log S} T_i = T \cdot \log b = T \cdot \log \log S. \end{aligned}$$

Hence, the time bound  $O(T \cdot \log \log S)$  holds also for the dynamic simulation. We have proved

**THEOREM 3.2.1.** *A  $T(n)$  time-bounded and  $S(n)$  space-bounded TM can be simulated in  $O(T(n) \log \log S(n))$  time on a RAM without multiplication and division instructions.*

**3.3. Simulation with precomputed transition tables.** A further improvement in the simulation is achieved by combining the fast packing and unpacking algorithms of § 3.1 with the use of precomputed transition tables as in [7], [11].

As in § 3.1, we shall first assume that the optimal block size is known at the beginning of the simulation. The computation is speeded up by precomputing in a table all subcomputations of length  $b$ . For this purpose, the work-tape is divided into

blocks of size  $b$ , as in previous constructions. Now, input and output must also be handled blockwise, because a subcomputation of length  $b$  may read  $b$  bits and write  $b$  bits.

In the new construction, the simulation itself is simply table look-up and takes, as we will see, only  $O(T(n))$  logarithmic time. Paul [11] organizes the table as a heap (like in heapsort) and uses shift operations in the calculation of the pointers. We store the table as a multidimensional array and calculate the addresses by precomputed shift tables, avoiding the use of shift operations. For input and output blocking, the packing and unpacking techniques of § 3.1 are used. We shall see that, if  $U(n)$  is the maximum output length for inputs of length  $n$ ,  $O((n + U(n)) \log \log S(n))$  is sufficient for input and output, when it is done to and from low addresses, not higher than  $O(\log S(n))$ . Hence, the total time of the simulation is  $O(T(n) + (n + U(n)) \log \log S(n))$ .

In order to guarantee that large enough input blocks are always available, and there is not too frequent need for filling the blocks, we will use an input buffer of length  $2b$ . It is filled every time when more than half of the bits have been consumed. The same idea is useful for outputting.

We shall precompute the transition table

$$\text{transit}(i, x, l, p, v_{-1}, v_0, v_1, j, y) = (i', l', p', v'_{-1}, v'_0, v'_1, d, j', y')$$

where all entries are integers:  $i, j, p, p' < b$ ;  $i', j' < 2b$ ;  $x, y, y' < 2^{2b}$ ;  $l, l'$  are program line numbers;  $v_k, v'_k < 2^b$ ; and  $d = -1, 0, 1$ . The intuitive meaning of the table is the following: If  $i$  bits of the contents  $x$  of the input buffer have been consumed before, the TM is at its instruction  $l$ , the work-tape head is at the position  $p$  of the block  $v_0$  whose neighbours are  $v_{-1}$  and  $v_1$ , and there are already  $j$  bits of  $y$  in the output buffer, then after  $b$  elementary steps of the TM,  $i' (< 2b)$  bits of  $x$  have been consumed, TM is at line  $l'$ , the work-tape blocks have changed to  $v'_k$  and the head is at the position  $p'$  of  $v'_d$ , and there are  $j'$  bits of  $y'$  in the output buffer. With this notation, we can write the *static* simulating algorithm as follows:

**procedure** simulate

$i := 2b - 1$ ;  $j := 0$  {input and output buffers are empty}

$l := 1$ ; activeblock := 1;  $p := 0$ ;

**loop** {until the simulation of a **halt** causes an exit}

**if**  $i \geq b$  **then** fill input buffer to a new  $x$  with  $i = 0$ ;

**for**  $k = -1, 0, 1$  **do**  $v_k := \text{work-tape}(\text{activeblock} + k)$

$(i, l, p, v_{-1}, v_0, v_1, d, j, y) := \text{transit}(i, x, l, p, v_{-1}, v_0, v_1, j, y)$

**for**  $k = -1, 0, 1$  **do**  $\text{work-tape}(\text{activeblock} + k) := v_k$

  activeblock := activeblock +  $d$

**if**  $j \geq b$  **then** output the  $j$  bits of  $y$  and make  $j := 0$ ;

Some points of the program need closer examination.

First, as  $b$  successive steps of the TM are composed, the main loop is iterated  $T/b$  times.

If  $i \geq b$ , the input buffer is filled as follows. First the  $2b$  bits of  $x$  are unpacked by Lemma 3.1.3 in time  $O(b \cdot \log b)$ . Then the unused  $2b - i$  bits are moved to positions  $0, 1, \dots$ , followed by  $i$  new bits read from the input tape. This is easily done in  $O(b \cdot \log b)$  time. Output is treated analogously.

Loading and storing work-tape blocks obviously takes  $O(\log S)$  time. The transition table can easily be linearized to  $2^{kb}$  registers, where  $k$  is a constant. By the shift tables similar to those in § 3.1, additions and subtractions, the address of a table entry can be computed, and the value can be decoded in  $O(b)$  time.

We observe that input and output take  $O(\log b)$  time for each bit, and hence  $O((n + U(n)) \log b)$  time in total. The cost of the simulation is of order  $(T/b)(\log S + b)$ . Hence, by choosing  $b = O(\log S)$ , we see that  $O((n + U(n)) \log \log S(n) + T(n))$  time is sufficient for simulation.

Finally, we shall show that the table can be built in  $O(S)$  time. It can be done in the obvious way: For each table entry, simulate  $b$  steps of the TM and store the result in a register. The generation of the entries is controlled by nine nested loops, one for each argument of the table. To compute the value of each table entry, it is first unpacked to low-indexed registers in time  $O(b \cdot \log b)$ , then  $b$  steps of the TM are simulated in time  $O(b \cdot \log b)$ , and finally the result is packed in time  $O(b \cdot \log b)$  and stored in time  $O(b)$ . Hence, the construction of the table takes  $O(2^{kb} \cdot b \cdot \log b)$  time, which is  $O(S)$ , if  $b = c \cdot \log S$  with a constant  $c < 1/k$  is chosen.

Thus, we have proved Theorem D in the static case. As in § 3.2, the efficiency of the static construction can be achieved even though the optimal block size is not known in advance. We let the block size grow in the series  $b_0, b_1, \dots, b_i = 2^i, \dots$ , when the space requirement reaches the bounds

$$S_0, S_1, \dots, S_i = 2^{2^{i+a}}, \dots,$$

respectively. In  $S_i$ , the constant  $a$  is chosen such that  $k < 2^a$ . The *dynamic* simulating program is as follows.

**procedure** simulate

$i := 2b - 1; j := 0$

$l := 1; \text{activeblock} := 1; p := 0;$

$b := 1; S := 2^{2^a}$  { $a$  is a small constant}

Initialize  $l\text{shift}$ ,  $r\text{shift}$ , origin, power and transit tables;

**loop** {until a **halt** in the simulation}

**while**  $\lceil \log(S + 1) \rceil \leq 2^{a+1} b$  **do**

**if**  $i \geq b$  **then** fill input buffer;

**for**  $k = -1, 0, 1$  **do**  $v_k := \text{work-tape}(\text{activeblock} + k)$

$(i, l, p, v_{-1}, v_0, v_1, d, j, y) := \text{transit}(i, x, l, p, v_{-1}, v_0, v_1, j, y)$

    Increase  $S$  when necessary;

**for**  $k = -1, 0, 1$  **do**  $\text{work-tape}(\text{activeblock} + k) := v_k$

$\text{activeblock} := \text{activeblock} + d$

**if**  $j \geq b$  **then** flush the output buffer;

$b := b + b;$

  Combine successive memory blocks pairwise;

  Update  $p$  and activeblock corresponding to the reorganization;

  Update  $l\text{shift}$ ,  $r\text{shift}$ , origin and power;

  Construct transit for the block size  $b$ ;

There is very little new in the analysis of the program. During the whole simulation,  $O(S)$  time is used in the construction of the tables  $l\text{shift}$ ,  $r\text{shift}$ , origin and power. Every time when a new transit table is constructed,  $O(S_i)$  time is spent, hence  $\sum S_i = O(S)$  in total.

As the block size never exceeds  $O(\log S)$ , input and output packing, as well as unpacking, can be done in  $O(\log \log S)$  time per bit,  $O((n + U(n)) \log \log S(n))$  time in total. As the simulation for each block size is linear, the total simulation is linear, too.

Hence, we have proved

**THEOREM 3.3.1.** *Any  $T(n)$  time-bounded,  $S(n)$  space-bounded and  $U(n)$  output length-bounded TM can be simulated in  $O(T(n) + (n + U(n)) \log \log S(n))$  logarithmic time by a RAM.*

**4. Discussion.** We have improved the simulations of TMs by RAMs that can be found in [1] and [11]. We first improved the strategy of [1] by introducing blocking, a well-known technique used for speed-ups. Essential in our constructions was how to pack and unpack blocks efficiently. We did it with a precomputed table. The same tabulation technique was also used to improve the method of [11]. First, it allowed a natural look-up method, and second, with it input and output could be done more efficiently.

It is interesting to observe that in the simulation of Theorem D, faster input and output would improve the time bound. With a better input/output pattern the simulation could perhaps be sped up further. Whether such an improvement is possible remains as an open problem.

**Acknowledgments.** We are grateful to the referee for remarks and references that have improved the representation.

#### REFERENCES

- [1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [2] S. A. COOK AND R. A. RECKHOW, *Time bounded random access machines*, J. Comput. System Sci., 7 (1973), pp. 354-375.
- [3] H. M. DEITEL, *An Introduction to Operating Systems*, Addison-Wesley, Reading, MA, 1984.
- [4] P. W. DYMOND AND M. TOMPA, *Speedups of deterministic machines by synchronous parallel machines*, J. Comput. System Sci., 30 (1985), pp. 149-161.
- [5] M. J. FISCHER AND M. S. PATERSON, *String-matching and other products*, in Complexity of Computation, R. M. Karp, ed., SIAM-AMS Proceedings 7, Amer. Math. Soc., Providence, RI, 1974, pp. 113-125.
- [6] G. GALIL AND J. SEIFERAS, *Time-space-optimal string matching*, J. Comput. System Sci., 26 (1983), pp. 280-294.
- [7] J. HOPCROFT, W. PAUL AND L. VALIANT, *On time versus space and related problems*, in Proc. 16th Annual IEEE Symposium on the Foundations of Computer Science, 1975, pp. 57-64.
- [8] J. E. HOPCROFT AND J. D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.
- [9] M. C. LOUI, *Minimizing pointers into trees and arrays*, J. Comput. System Sci., 28 (1984), pp. 359-378.
- [10] M. H. OVERMARS, *The design of dynamic data structures*, Lecture Notes in Computer Science 156, Springer-Verlag, Berlin, 1983.
- [11] W. J. PAUL, *Komplexitätstheorie*, Teubner Verlag, Stuttgart, 1978.
- [12] M. PENTTONEN AND J. KATAJAINEN, *Notes on the complexity of sorting in abstract machines*, BIT, 25 (1985), pp. 611-622.
- [13] J. M. ROBSON, *Fast probabilistic RAM simulation of single tape Turing machine computations*, Inform. and Control, 63 (1984), pp. 67-87.
- [14] A. SCHÖNHAGE, *Storage modification machines*, this Journal, 9 (1980), pp. 490-508.
- [15] C. SLOT AND P. VAN EMDE BOAS, *On tape versus core: an application of space efficient hash functions to the invariance of space*, in Proc. 16th Annual ACM Symposium on the Theory of Computing, ACM, New York, 1984, pp. 391-400.
- [16] J. WIEDERMANN, *Deterministic and nondeterministic simulation of the RAM by the Turing machine*, in Proc. IFIP Congress 83, R. E. A. Mason, ed., North-Holland, Amsterdam, 1983, pp. 163-168.

## DESIGNING A UNIFORM RANDOM NUMBER GENERATOR WHOSE SUBSEQUENCES ARE $k$ -DISTRIBUTED\*

MASANORI FUSHIMI†

**Abstract.** A method for designing a uniform random number generator based on  $M$ -sequence is described. It generates a  $k$ -distributed sequence  $\{x_t; t=0, 1, 2, \dots\}$  such that its decimated sequences  $\{x_{nt}; t=0, 1, 2, \dots\}$  are also  $k$ -distributed for several values of  $n$ . The sequence is obtained by permuting the bits in Tausworthe sequence; a suitable permutation is found by solving a special 0-1 integer linear programme, a set covering problem. The constraint matrix of the programme is obtained by applying a modified Gaussian elimination to another 0-1 matrix. Two simple heuristic algorithms for the programme are given and demonstrated to find good approximate solutions to an illustrative example.

**Key words.** Tausworthe sequence, decimated sequence,  $k$ -distribution, set covering problem, heuristic algorithm

**AMS (MOS) subject classification.** 65C10, 12C10, 15A03, 90C50

**1. Introduction.** When one uses a sequence  $\{x_t; t=0, 1, 2, \dots\}$  of uniform random numbers, it often happens that numbers are used in batches. For example, in a simulation dealing with three random variables  $X$ ,  $Y$  and  $Z$ , three numbers in the sequence  $\{x_t\}$  may be used at each step in the simulation. In such applications it is important that not only the original sequence  $\{x_t\}$  but also the subsequences consisting of every third term of  $\{x_t\}$  are random. In general, if the simulation requires  $n$  random numbers at a time, the  $n$  subsequences  $\{x_{nt+i}; t=0, 1, 2, \dots\}$ ,  $0 \leq i \leq n-1$ , must be random. Since a random number generator may be used for various applications, it is desirable that the above property hold for various values of  $n$ .

Knuth [9, p. 71] says "experience with linear congruential sequences has shown that these derived sequences rarely if ever behave less randomly than the original sequence, unless  $n$  has a large factor in common with the period length," although there is no theoretical proof that this is always the case. Then how about other uniform number generators? What we shall be concerned with in this paper is the design of uniform random number generators with the above property based on the maximal-length linearly recurring sequence modulo 2 ( $M$ -sequence for short). In § 2, we will review some uniform random number generators based on  $M$ -sequences. Section 3 will show that our problem can be formulated as a special form of 0-1 integer linear programme, in fact, a set covering problem, for which approximate solutions can be obtained quite easily by simple algorithms. We will show in § 4 two such algorithms as well as the fact that the constraint matrix of the programme can be obtained by using a modified Gaussian elimination method. The final section will illustrate our procedure by an example which leads to a uniform random number generator useful for practical applications.

**2. Uniform random number generators based on  $M$ -sequence.** An  $M$ -sequence is a sequence of 0's and 1's generated by a linear recurrence relation

$$(2.1) \quad a_t = c_1 a_{t-1} + c_2 a_{t-2} + \dots + c_p a_{t-p} \pmod{2}$$

whose characteristic polynomial

$$f(D) = 1 + c_1 D + c_2 D^2 + \dots + c_p D^p, \quad c_p = 1,$$

\* Received by the editors May 29, 1985; accepted for publication (in revised form) April 17, 1987.

† Department of Mathematical Engineering and Instrumentation Physics, Faculty of Engineering, University of Tokyo, Bunkyo-ku, Tokyo 113, Japan.

is primitive over the Galois field  $GF(2)$ , given any initial values  $(a_1, a_2, \dots, a_p) \neq (0, 0, \dots, 0)$ .  $\{a_i\}$  is a periodic sequence with period  $2^p - 1$ .

Tausworthe [11] proposed to construct a sequence  $\{x_i\}$  of  $l$ -bit binary numbers using the  $M$ -sequence  $\{a_i\}$  as follows:

$$(2.2) \quad x_i = 0.a_{\sigma i+1}a_{\sigma i+2} \cdots a_{\sigma i+l},$$

where  $\sigma$  is a constant which satisfies the following conditions.

$$(2.3) \quad \sigma \geq l \quad \text{and} \quad \gcd(\sigma, 2^p - 1) = 1.$$

Properties of the sequence  $\{x_i\}$  so constructed were investigated by Tausworthe himself and some others [2], [12], [13].

Lewis and Payne [10] proposed to construct an apparently different sequence  $\{y_i\}$ , called the GFSR sequence, using the  $M$ -sequence generated by the primitive trinomial  $f(D) = D^p + D^q + 1$  as follows.

$$(2.4) \quad y_i = 0.a_i a_{i+\tau} a_{i+2\tau} \cdots a_{i+(l-1)\tau},$$

where  $\tau$  is a "judiciously selected" constant. The crucial point of their idea is that the sequence  $\{y_i\}$  can be generated very fast by using the recurrence relation  $y_i = y_{i-p} \oplus y_{i-q}$ , where  $\oplus$  means the bitwise addition modulo 2.

Kashiwagi [8] and Fushimi [4] have shown the following equivalence relation between the class of Tausworthe sequences without the first one of the conditions (2.3) and that of GFSR sequences dropping the condition that  $f(D)$  is a trinomial. The key point in the following is the notion of "proper decimation" [7] of the  $M$ -sequence, in which "decimation" means selecting every  $n$ th term of the  $M$ -sequence, and "proper" means that  $n$  is relatively prime to the period  $2^p - 1$ . If  $\{a_i\}$  is an  $M$ -sequence whose characteristic polynomial  $f$  is of degree  $p$ , then a properly decimated sequence  $\{a_{ni}; i = 0, 1, 2, \dots\}$  is also an  $M$ -sequence whose characteristic polynomial, say  $f_n$ , is of degree  $p$ . Let  $n^{-1}$  denote the inverse of  $n$  with respect to the multiplication modulo  $2^p - 1$ , i.e.,  $n \cdot n^{-1} = 1 \pmod{2^p - 1}$ . In the following theorem, we use the notation  $\{x_i(f; \sigma)\}$  and  $\{y_i(f; \tau)\}$  to specify the primitive polynomials and the parameters used to construct  $\{x_i\}$  and  $\{y_i\}$ . Finally we write  $\{x_i\} \simeq \{y_i\}$  when the two sequences  $\{x_i\}$  and  $\{y_i\}$  are equivalent in the sense they are the same sequence except possibly the location of the starting point.

**THEOREM 1.** *If  $f$  is a primitive polynomial of order  $p$ , and  $\sigma$  and  $\tau$  are relatively prime to  $2^p - 1$ , then the following equivalence relations hold.*

$$\{x_i(f; \sigma)\} \simeq \{y_i(f_\sigma; \sigma^{-1})\},$$

$$\{y_i(f; \tau)\} \simeq \{x_i(f_\tau; \tau^{-1})\}.$$

It follows from the theorem that  $\{x_i(f; \sigma)\}$  can also be generated very fast if  $f_\sigma$  is a trinomial or a polynomial with few terms; if, for example,  $f_\sigma = D^p + D^r + 1$ , then  $\{x_i(f; \sigma)\}$  can be generated by the recurrence relation  $x_i = x_{i-p} \oplus x_{i-r}$ .

Fushimi and Tezuka [5] investigated the  $k$ -distribution, i.e., multidimensional equidistribution, property<sup>1</sup> of a generalization of the GFSR sequence

$$(2.5) \quad y'_i = 0.a_{i+\tau_1} a_{i+\tau_2} a_{i+\tau_3} \cdots a_{i+\tau_l},$$

and established a necessary and sufficient condition for the sequence  $\{y'_i\}$  to be  $k$ -distributed, which is reproduced below in a slightly different form.

<sup>1</sup>The reader who is unfamiliar with the notion of  $k$ -distribution and its relation to randomness is referred to Knuth [9, § 3.5].



**THEOREM 2.**  $\{y'_i\}$  is  $k$ -distributed if and only if the  $kl$  terms of the basic  $M$ -sequence contained in  $y'_i, 0 \leq i \leq k-1$ , are linearly independent.

It follows from the theorem that the maximum possible order of equidistribution for  $\{y'_i\}$  is

$$(2.6) \quad m = [p/l],$$

where  $[\cdot]$  is the greatest integer function, since the maximum number of linearly independent terms of the  $M$ -sequence is  $p$ .

It is clear from Theorem 1 and the arguments in [5] that Theorem 2 also holds for Tausworthe sequence as well as any properly decimated sequence  $\{x_{nt}; t = 0, 1, 2, \dots\}$ , in which case the words "linearly independent" mean the following. Each of the  $kl$  bits of  $x_{nt}, 0 \leq t \leq k-1$ , can be expressed as a linear combination of the initial  $p$  terms of the basic  $M$ -sequence, i.e., in the form  $e_1 a_1 + e_2 a_2 + \dots + e_p a_p \pmod{2}$ , by (repeated, if necessary,) application(s) of the recurrence relation (2.1), and thus the unique weight vector  $(e_1, e_2, \dots, e_p)$  is associated with each bit of  $x_{nt}, 0 \leq t \leq k-1$ . The phrase "linearly independent" implies that these weight vectors are linearly independent over  $GF(2)$ .

At this point we introduce the following notation.

$$A_{n,j} = \{a_{\sigma_{nt+j}} | 0 \leq t \leq m-1\}, \quad 1 \leq j \leq l,$$

$$A_n = \bigcup_{j=1}^l A_{n,j}.$$

Thus  $A_{n,j}$  is the set of the terms of the basic  $M$ -sequence which constitute the  $j$ th bits of the first  $m$  terms of the decimated sequence  $\{x_{nt}\}$ . In addition we use the symbol  $E_n$  to denote the  $lm \times p$  matrix whose  $m(j-1) + t + 1$ st row is the weight vector associated with  $a_{\sigma_{nt+j}}$ .

Fushimi [3] considered a generalization  $\{x'_t\}$  of Tausworthe sequence  $\{x_t\}$ , which is obtained by a rearrangement of the bits of  $\{x_t\}$  as follows:

$$(2.7) \quad x'_t = 0.a_{\sigma_{t+j(1)}} a_{\sigma_{t+j(2)}} \dots a_{\sigma_{t+j(l)}},$$

where  $\{j(1), j(2), \dots, j(l)\}$  is a permutation of  $\{1, 2, \dots, l\}$  independent of  $t$ . It is clear that Theorem 2 is also applicable to the sequence  $\{x'_t\}$  and any properly decimated sequence of it. We remark that the sequence  $\{x'_t\}$  can be generated exactly as fast as the sequence  $\{x_t\}$  once the initial values  $\{x'_t; 0 \leq t \leq p-1\}$  are given (see the remark following Theorem 1).

In what follows we also use  $\{x_t\}$  and  $\{x'_t\}$ , and consider which permutation is appropriate for our purpose. It must be noted that the maximum orders of equidistribution of  $\{x_{nt}\}$  and  $\{x'_{nt}\}$  are equivalent for any permutation of bits since the number of linearly independent vectors in a given set of vectors is invariant under any permutation of the vectors in the set. Then one may naturally raise a question: in what sense can we improve the sequence  $\{x_{nt}\}$  by considering the rearrangement of bits? The answer is as follows. If we form the sequences of  $l'$ -bit ( $l' < l$ ) numbers by reading the first  $l'$  bits of the terms in the sequences  $\{x_{nt}\}$  and  $\{x'_{nt}\}$ , respectively, then the orders of equidistribution for these  $l'$ -bit sequences may be different. To be more specific, suppose the sequence  $\{x_{nt}\}$  is  $m$ -distributed if the terms in the sequence are read to  $l'_1$ -bit accuracy,<sup>2</sup> but not  $m$ -distributed if read to more than  $l'_1$ -bit accuracy. Then the sequence

<sup>2</sup> In such a case, we will say the sequence is  $m(l'_1)$ -distributed.

$\{x'_{ni}\}$  is considered to be better than  $\{x_{ni}\}$  if the former is  $m\langle l'_2 \rangle$ -distributed for some  $l'_2 > l'_1$ . Thus it is worthwhile to ask for a permutation for which  $\{x'_{ni}\}$  is  $m\langle l' \rangle$ -distributed with  $l'$  as large as possible.

**3. A 0-1 integer programme.** Now let  $N$  be a set of integers relatively prime to  $2^p - 1$ , and we require all the decimated sequences  $\{x'_{ni}\}$ ,  $n \in N$ , be  $m\langle l' \rangle$ -distributed. There are an immense number of integers which are relatively prime to  $2^p - 1$  when  $p$  is large, and there is no obvious way to solve the problem if  $N$  consists of all of them. On the other hand, we usually use the decimated sequences  $\{x'_{ni}\}$  with relatively small  $n$ 's. Thus it is natural, from practical points of view, to take  $N$  as a set of small integers relatively prime to  $2^p - 1$ .

Our problem can be stated as follows.

*Design problem.* Given  $f$ ,  $\sigma$  and  $l$ , to find a permutation  $\{j(1), j(2), \dots, j(l)\}$  of  $\{1, 2, \dots, l\}$  for which the  $l'$  in the following statement is maximum: all the decimated sequences  $\{x'_{ni}\}$ ,  $n \in N$ , are  $m\langle l' \rangle$ -distributed, where  $m$  is defined by (2.6).

In order to solve this problem, we must first find minimal linear dependence relations,<sup>3</sup> for each  $n \in N$ , among the  $ml$  terms in  $A_n$ , or equivalently among the row vectors of  $E_n$ , and this can be accomplished, as will be explained in detail in the next section, by applying Gaussian elimination method to  $E_n$ . To each minimal linear dependence relation thus found, we associate an  $l$ -dimensional row vector  $\mathbf{g} = (g_1, g_2, \dots, g_l)$  as follows. If the terms in, say,  $A_{n,j_1}, A_{n,j_2}, \dots, A_{n,j_\nu}$  are linearly dependent, then  $g_{j_1} = g_{j_2} = \dots = g_{j_\nu} = 1$  and all the other components of  $\mathbf{g}$  are set equal to 0. Then we form a matrix  $G$  whose rows consist of  $\mathbf{g}$ 's corresponding to all the minimal dependence relations found for all  $n \in N$ .

Using  $G$  as the coefficient matrix of the constraints, we formulate the following 0-1 integer linear programming problem.

$$(3.1) \quad \text{ILP: minimize} \quad z_0 = \sum_{j=1}^l z(j)$$

$$(3.2) \quad \text{subject to} \quad G\mathbf{z} \geq \mathbf{1}, \quad \text{and}$$

$$(3.3) \quad z(j) = 0 \text{ or } 1, \quad 1 \leq j \leq l.$$

Here,  $\mathbf{z}$  is an  $l$ -dimensional column vector whose  $j$ th component is  $z(j)$ , and  $\mathbf{1}$  is the column vector whose components are all equal to 1.

**THEOREM 3.** *The 0-1 integer linear programming problem ILP is equivalent to the design problem.*

*Proof.* Let  $G^*$  be a matrix obtained by applying permutation  $\{j(1), j(2), \dots, j(l)\}$  to the columns of  $G$ , i.e., the  $k$ th column of  $G^*$  is the  $j(k)$ -th column of  $G$ , and  $G_0^*$  and  $G_1^*$  be the submatrices of  $G^*$  consisting of the first  $l'$  and the last  $(l - l')$  columns of  $G^*$ , respectively.

We have shown in the previous section that a necessary and sufficient condition for the sequence  $\{x'_{ni}\}$  defined by (2.7) to be  $m\langle l' \rangle$ -distributed is that there is no linear dependence relation among the  $ml'$  terms of the  $M$ -sequence contained in the leading  $l'$  bits of  $x'_{ni}$ ,  $0 \leq i \leq m - 1$ . This is equivalent to the condition that there is at least one 1 in every row of  $G_1^*$  since every row of  $G$  or  $G^*$  represents a *minimal* linear dependence relation. The inequality (3.2) is nothing but this condition. Since our objective is to maximize  $l'$ , our objective function is (3.1). Q.E.D.

<sup>3</sup> A linear dependence relation among a set of vectors is said to be minimal if no proper subset of the vectors is linearly dependent.

#### 4. Methods for solving the design problem.

**4.1. Algorithms for finding approximate solutions to ILP.** The 0-1 integer linear programming problem ILP we formulated in the previous section is a well-known problem in combinatorics, i.e., a set covering problem. It is one of NP-hard problems, i.e., there are no algorithms known at present to find an optimal solution to it with time complexity of a polynomial order of  $l$ , although several algorithms have been proposed using standard techniques for solving integer linear programmes, e.g., a branch-and-bound method and a cutting plane method. Approximate solutions to it, however, may be obtained in a very short time by simple heuristic algorithms. We will list two of them below.

Let  $J_0$  be the set of indices  $j$  such that  $z(j) = 0$ . Algorithm P starts with  $J_0 = \emptyset$ , when all the inequalities in (3.2) are satisfied but the value of the objective function  $z_0$  is maximum, and decreases  $z_0$  by iteratively including a  $j$  in  $J_0$  in the increasing order of the column sum of  $G$  as long as the constraint (3.2) is satisfied. On the contrary, Algorithm D starts with  $J_0 = \{1, 2, \dots, l\}$ , when all the inequalities in (3.2) are violated, and decreases the number of violated inequalities by iteratively excluding a  $j$  from  $J_0$  until all the inequalities are satisfied. At each iteration, such a  $j$  that decreases the number of violated inequalities most is excluded from  $J_0$ . Details of both algorithms are as follows.

##### ALGORITHM P.

1. Compute the column sum  $s(j)$  of  $G$  for  $j = 1, 2, \dots, l$ .
2. Sort  $s(j)$ 's in increasing order. Assume we have got  $s(j_1) \leq s(j_2) \leq \dots \leq s(j_l)$ .
3. Set  $z(j) = 1$  for  $j = 1, 2, \dots, l$ .  $J_0 \leftarrow \emptyset$ .
4. *for*  $i := 1$  *to*  $l$  *do*  
     *begin*  
          $z(j_i) := 0$ ;  
         *if*  $GZ \geq 1$  *is violated then*  $z(j_i) := 1$  *else*  $J_0 \leftarrow J_0 \cup \{j_i\}$   
     *end*

##### ALGORITHM D.

1.  $J_0 \leftarrow \{1, 2, \dots, l\}$ .  $G_0 \leftarrow G$ .  $k \leftarrow 0$ .
2.  $k \leftarrow k + 1$ .
3. Compute the column sum  $s_k(j)$  of  $G_{k-1}$  for each  $j \in J_0$ .
4. Let  $j^*$  be a maximizer of  $s_k(j)$  among  $j \in J_0$ .
5.  $J_0 \leftarrow J_0 - \{j^*\}$ .
6. Let  $G_k$  be the matrix consisting of all the row vectors of  $G_{k-1}$  whose  $j^*$ th component equals 0.
7. If  $G_k$  is not empty then go to step 2.

**4.2. How to find the constraints.** We shall be concerned in this subsection with a method of obtaining the constraint matrix  $G$  in (3.2). As was explained in the previous section, it is easily constructed once we know all the minimal linear dependence relations among the row vectors in  $E_n$  for each  $n \in N$ , but this problem itself is nonpolynomial in the number of the rows of  $E_n$ . We will present below an algorithm for solving this problem. The algorithm consists of two stages: first, we find *some* dependence relations by using a modification of Gaussian elimination method; second, we generate *enough* dependence relations to solve the problem by combining the relations obtained in the first stage. It is to be noted that we do not necessarily generate *all* the dependence relations exploiting the special structure of the problem.

Gaussian elimination is basically a method for solving systems of linear equations by performing a sequence of basic row operations, i.e., a multiplication of a row by

a constant and an addition of a row to another row, and usually the “history” of row operations is not recorded. In our case, however, where the objective is to find the linear dependence relations among the rows, the history of row operations is the crucial information. Here the word “history” means the record of the information which row was added to which row at each step. (Note that we need not perform a multiplication because the arithmetics are done on  $GF(2)$ .) In order to record the history, we introduce the variables  $h(i', i)$  and set  $h(i', i) = 1$  if the  $i'$ th row is added to the  $i$ th row at some step of elimination, and set  $h(i', i) = 0$  otherwise.

Now fix an  $n \in N$ , and let the  $i$ th row of  $E_n$  be denoted by  $\mathbf{e}_i^{(0)}$ ,  $1 \leq i \leq lm$ . Linear dependence relations among the row vectors of  $E_n$ , if any, are found by the following procedure, in which  $\mathbf{e}_i$  denotes an  $l$ -dimensional row vector whose  $j$ th component is  $e_{ij}$ .

```

procedure GAUSS;
  begin
    for  $i := 1$  to  $lm$  do  $\mathbf{e}_i := \mathbf{e}_i^{(0)}$ ;
    for  $i := 1$  to  $lm$  do
      begin
        if  $\mathbf{e}_i = \mathbf{0}$  then DEPENDENCE( $i$ )
        else if  $i < lm$  then
          begin
             $j_0 :=$  the minimum  $j$  for which  $e_{ij} = 1$ ;
            for  $k := i + 1$  to  $lm$  do
              if  $e_{kj_0} = 1$  then
                begin
                   $\mathbf{e}_k := \mathbf{e}_k + \mathbf{e}_i \pmod{2}$ ;
                   $h(i, k) := 1$ 
                end
              else  $h(i, k) := 0$ 
            end
          end
        end
      end
    end
  end

```

The procedure DEPENDENCE( $i$ ) is called when  $\mathbf{e}_i = \mathbf{0}$ , which is the case if and only if the vector  $\mathbf{e}_i^{(0)}$  is equivalent to some linear combination of the vectors  $\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}, \dots, \mathbf{e}_{i-1}^{(0)}$ . A method for finding a linear dependence relation among these vectors shall be illustrated by a simple example. Suppose  $i = 6$  and the relevant part of the history  $h(i', i)$  is as shown in Table 1, which means the following.<sup>4</sup>

$$(4.1a) \quad \mathbf{e}_6 = \mathbf{e}_1 + \mathbf{e}_3 + \mathbf{e}_4 + \mathbf{e}_5 + \mathbf{e}_6^{(0)},$$

TABLE 1  
An example of the history  $h(i', i)$ .

		1	2	3	4	5	6	
$i'$	1							
	2		1	1	1	0	1	
	3			0	0	1	0	
	4				0	0	1	
	5					1	1	
	6						1	

<sup>4</sup> In (4.1) and the following three equations, equality sign is to be understood on  $GF(2)$ .

$$(4.1b) \quad \mathbf{e}_5 = \mathbf{e}_2 + \mathbf{e}_4 + \mathbf{e}_5^{(0)},$$

$$(4.1c) \quad \mathbf{e}_4 = \mathbf{e}_1 + \mathbf{e}_4^{(0)},$$

$$(4.1d) \quad \mathbf{e}_3 = \mathbf{e}_1 + \mathbf{e}_3^{(0)},$$

$$(4.1e) \quad \mathbf{e}_2 = \mathbf{e}_1 + \mathbf{e}_2^{(0)}.$$

Substituting (4.1b) in (4.1a), we get

$$\mathbf{e}_6 = \mathbf{e}_1 + \mathbf{e}_2 + \mathbf{e}_3 + \mathbf{e}_5^{(0)} + \mathbf{e}_6^{(0)},$$

which, upon substitution of (4.1d), leads to

$$\mathbf{e}_6 = \mathbf{e}_2 + \mathbf{e}_3^{(0)} + \mathbf{e}_5^{(0)} + \mathbf{e}_6^{(0)}.$$

If we substitute (4.1e) in the above expression and use the fact that  $\mathbf{e}_1 = \mathbf{e}_1^{(0)}$ , we finally obtain the following linear dependence relation.

$$\mathbf{0} = \mathbf{e}_6 = \mathbf{e}_1^{(0)} + \mathbf{e}_2^{(0)} + \mathbf{e}_3^{(0)} + \mathbf{e}_5^{(0)} + \mathbf{e}_6^{(0)}.$$

Thus, in general, a linear dependence relation can be obtained by the following successive substitution.

```

procedure DEPENDENCE(i);
  begin
     $d_1 := d_2 := \dots := d_{i-1} := 0; d_i := 1;$ 
    for  $j := i$  downto 2 do
      if  $d_j \neq 0$  then
        for  $i' := j - 1$  downto 1 do  $d_{i'} := d_{i'} + h(i', j) \pmod{2}$ 
      end
    end
  
```

If  $d_{i_1} = d_{i_2} = \dots = d_{i_\nu} = d_i = 1$  and the other  $d$ 's are 0 after the above procedure is executed, we know that there is a linear dependence relation

$$(4.2) \quad \mathbf{e}_{i_1}^{(0)} + \mathbf{e}_{i_2}^{(0)} + \dots + \mathbf{e}_{i_\nu}^{(0)} + \mathbf{e}_i^{(0)} = \mathbf{0}.$$

It is clear, by the property of Gaussian elimination, that the set of vectors  $\{\mathbf{e}_{i_1}^{(0)}, \mathbf{e}_{i_2}^{(0)}, \dots, \mathbf{e}_{i_\nu}^{(0)}\}$  is independent, and that (4.2) is a minimal linear dependence relation.

There remains a question whether the minimal linear dependence relations found by the above procedure are all that exist among the row vectors of  $E_n$ . The answer is no in general, and the reason is as follows. Suppose there are two minimal linear dependence relations, (4.2) and (4.3) below.

$$(4.3) \quad \mathbf{e}_{k_1}^{(0)} + \mathbf{e}_{k_2}^{(0)} + \dots + \mathbf{e}_{k_\lambda}^{(0)} + \mathbf{e}_k^{(0)} = \mathbf{0}.$$

If  $\{i_1, i_2, \dots, i_\nu, i\} \cap \{k_1, k_2, \dots, k_\lambda, k\} \neq \emptyset$ , there is a possibility that the linear dependence relation obtained by adding (4.2) and (4.3) modulo 2 happens to be minimal, and the similar argument applies when there are more than two linear dependence relations. Hence we must, in principle, check all the combinations of the linear dependence relations found by procedure GAUSS in order to find all the linear dependence relations among the rows of  $E_n$ , which would be extremely time consuming. It must be remembered, however, that what we need finally is not the linear dependence relations among the rows of  $E_n$ , i.e., among the elements in  $A_n$ , but the dependence relations among the columns  $A_{n,1}, A_{n,2}, \dots, A_{n,l}$ . Hereafter we will use the words “ $b$ -dependence” and “ $c$ -dependence” as abbreviations for “linear dependence among the elements (bits) in  $A_n$ ” and “linear dependence among the columns” respectively.

It often happens, as will be illustrated in the next section, that several  $b$ -dependence relations give rise to the same  $c$ -dependence relation, and this observation leads to a rather efficient procedure for finding all the  $c$ -dependence relations, which will be best illustrated by an example in the next section.

**5. An illustrative example.** As an example, we take a primitive trinomial  $f(D) = D^{521} + D^{32} + 1$ , which was also used by many authors, see, e.g., [1]-[3], [5]. The period of the  $M$ -sequence generated by this trinomial,  $2^{521} - 1$ , is a Mersenne prime, hence all the decimated sequences are also  $M$ -sequences. We have chosen as the set  $N$  the first sixteen natural numbers:

$$N = \{1, 2, 3, \dots, 16\}.$$

The bit length  $l$  and the parameter  $\sigma$  of the Tausworthe sequence (2.2) are chosen to be 32. The maximum possible order of equidistribution,  $m$ , for the sequences  $\{x_{ni}\}$  is equal to  $\lfloor 521/32 \rfloor = 16$ .

For each  $n \in N$ , we constructed the matrix  $E_n$ , to which procedure GAUSS was applied. It was found that there exists no linear dependence relation among the rows of  $E_n$  for  $n = 1, 2, 4, 8, 9$  and  $16$ . Table 2 shows the maximum  $l'$ , for each  $n \in N$ , such that Tausworthe sequence (2.2) is  $m(l')$ -distributed.

TABLE 2  
Maximum  $l'$  such that the decimated sequence of the Tausworthe sequence in the illustrative example is  $16(l')$ -distributed.

$n$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$l'$	32	32	23	32	9	27	27	32	32	18	28	28	27	31	27	32

Now we will show the  $b$ -dependence relations found by GAUSS and interrelations among them. For  $n = 3$ , 45  $b$ -dependence relations were found among the  $lm = 512$  rows of  $E_3$ , or equivalently among the bits in  $\{x_{3t}; 0 \leq t \leq 15\}$ . The first<sup>5</sup> one of them is shown in Fig. 1, which indicates that the shaded bits are linearly dependent. It is clear that any set of four bits whose configuration is equivalent to the one in Fig. 1 is also linearly dependent. Let  $\mathcal{R}(\Delta t, \Delta j)$  denote the  $b$ -dependence relation among the four bits obtained by translating the shaded bits in Fig. 1 by  $\Delta t$  downward and by  $\Delta j$  rightward. For example,  $\mathcal{R}(4, 8)$  means the  $b$ -dependence relation among the four bits at  $(t, j) = (9, 9), (14, 18), (15, 18)$  and  $(4, 32)$ . We shall say that the two  $b$ -dependence relations are essentially the same (different) if their configurations are equivalent (different). There are 45 ( $= 5 \times 9$ )  $b$ -dependence relations which are essentially the same as the one in Fig. 1, including itself, i.e.,  $\mathcal{R}(\Delta t, \Delta j), 0 \leq \Delta t \leq 4$  and  $0 \leq \Delta j \leq 8$ , and they are all that were found by GAUSS when  $n = 3$ .

It is evident that, for any fixed  $\Delta j$ , five  $b$ -dependence relations,  $\mathcal{R}(\Delta t, \Delta j), 0 \leq \Delta t \leq 4$ , or any other  $b$ -dependence relations obtained by adding these modulo 2, give rise to the same  $c$ -dependence relation. It is also clear that any two relations  $\mathcal{R}(\Delta t_1, \Delta j_1)$  and  $\mathcal{R}(\Delta t_2, \Delta j_2)$  have no bit in common if  $\Delta j_1 \neq \Delta j_2$ , hence a minimal  $b$ -dependence relation cannot be obtained by combining these.

<sup>5</sup> The word "first" means "the  $i$  was minimum when procedure DEPENDENCE( $i$ ) was called in procedure GAUSS."



the sequence number of  $c$ -dependence relations for each  $n$ , and “ $r$ ” means the number of rows of the constraint matrix  $G$  to be generated by each  $c$ -dependence relation. Thus 99 rows of  $G$  are generated in all.

We have applied the Algorithms P and D to our set covering problem with the constraint matrix  $G$  obtained above. They have found solutions with  $z_0 = 14$  ( $l' = 18$ ) and  $z_0 = 13$  ( $l' = 19$ ), respectively.

It is well known, see e.g. [6], and easy to see that if the two rows  $\mathbf{g}_i$  and  $\mathbf{g}_{i'}$  of  $G$  satisfy the dominance relation  $\mathbf{g}_i \geq \mathbf{g}_{i'}$ , then  $\mathbf{g}_i$  may be deleted from the constraint. In our example, 48 rows can be deleted by this dominance relation. Asterisks in Table 3 show the  $c$ -dependence relations corresponding to the remaining 51 rows. We have applied again the Algorithms P and D to the set covering problem with the reduced constraint matrix, obtaining solutions with  $z_0 = 13$  ( $l' = 19$ ) and  $z_0 = 14$  ( $l' = 18$ ), respectively.

One of the solutions thus found with  $z_0 = 13$  ( $l' = 19$ ) is such that  $J_0 = \{j | z(j) = 0\} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 19, 20, 21, 22, 23, 24, 29, 30, 31, 32\} = \{j(1), j(2), \dots, j(19)\}$  and  $J_1 = \{j | z(j) = 1\} = \{10, 11, 12, 13, 14, 15, 16, 17, 18, 25, 26, 27, 28\} = \{j(20), j(21), \dots, j(32)\}$ . Therefore, if we construct the sequence  $\{x'_i\}$  by

$$x'_i = 0.a_{32i+j(1)}a_{32i+j(2)} \cdots a_{32i+j(32)}$$

using the permutation  $\{j(1), j(2), \dots, j(32)\}$  defined above, all the decimated sequences  $\{x'_n\}$ ,  $1 \leq n \leq 16$ , are 16-distributed when they are read to 19-bit accuracy.

Finally we briefly describe a method of implementing the generator thus obtained, using similar techniques as Tootill et al. [12], [13]. Let  $\{X_t\} = \{2^{32}x_t\}$  and  $\{X'_t\} = \{2^{32}x'_t\}$  be sequences of 32-bit integer random numbers. Since  $\sigma (= 32)$  is a power of 2, we have  $f_\sigma(D) = D^{521} + D^{32} + 1$  so that  $\{X'_t\}$  can be generated by the recurrence

$$X'_t = X'_{t-521} \oplus X'_{t-32}.$$

To initialize  $\{X'_t\}$ , we first set the initial values of the sequence  $\{X_t\}$ , i.e.,  $\{X_t; 0 \leq t \leq 520\}$ , and then perform the permutation of the bits of these initial values. Since  $521 (= 32 \times 16 + 9)$  initial values of the basic  $M$ -sequence  $\{a_t\}$  can be given arbitrarily, we can set  $X_t$  ( $0 \leq t \leq 15$ ) and the leading 9 bits of  $X_{16}$  arbitrarily. Practically, however, it would be convenient for one to give all the bits of  $X_t$  ( $0 \leq t \leq 16$ ) arbitrarily by using, for example, a congruential generator, and then to modify  $X_{16}$  by the following formula based on the “principle of complementary shifting” [13, p. 478]:

$$X_{16} = M^{32}((L^{23}X_{t-17} + R^9X_0) \oplus X_{15}).$$

Here  $L^{23}$  denotes the logical left shift 23 bits, and  $M^{32}$  means taking the least significant 32 bits by a masking operation. Then the remaining 504 initial values,  $\{X_t; 17 \leq t \leq 520\}$ , are generated by use of the recurrence

$$X_t = M^{32}((L^{23}X_{t-17} + R^9X_{t-16}) \oplus X_{t-1}).$$

Now that the initial values of the sequence  $\{X_t\}$  are set, we have only to permute the bits of  $X_t$  to obtain  $X'_t$  for each  $t$ ,  $0 \leq t \leq 520$ . This can be conveniently done in FORTRAN by using double length integer variables. A sample program for a computer with the word length 32 bits is given below, in which LSHFR(K, J) is a generic function for the logical right shift  $J$  bits operation.

**Acknowledgment.** The author is grateful to H. Imai of the University of Tokyo for valuable discussions. He also appreciates the constructive comments given by an anonymous referee.



```

SUBROUTINE PERMUT(X)
INTEGER X(521), XJ0(2), XJ1(2), XJ0D*8, XJ1D*8
EQUIVALENCE (XJ0, XJ0D), (XJ1, XJ1D)
DO 1 I=1, 521
XJ0(1) = X(I)
XJ0(2) = 0
XJ1(1) = X(I)
C   CONSTRUCTS THE LEADING 19 BITS
XJ0D = LSHFR(XJ0D, 4)
XJ0(1) = LSHFR(XJ0(1), 4)
XJ0D = LSHFR(XJ0D, 6)
XJ0(1) = LSHFR(XJ0(1), 9)
XJ0D = LSHFR(XJ0D, 9)
C   CONSTRUCTS THE REMAINING 13 BITS
XJ1(1) = LSHFR(XJ1(1), 4)
XJ1D = LSHFR(XJ1D, 4)
XJ1(1) = LSHFR(XJ1(1), 6)
XJ1D = LSHFR(XJ1D, 9)
XJ1(2) = LSHFR(XJ1(2), 19)
C   CONCATENATES THE TWO PARTS
X(I) = XJ0(2) + XJ1(2)
1   CONTINUE
RETURN
END

```

## REFERENCES

- [1] A. C. ARVILLIAS AND A. E. MARITSAS, *Partitioning the period of  $m$ -sequences and application to pseudorandom number generation*, J. Assoc. Comput. Mach., 25 (1978), pp. 675–686.
- [2] H. BRIGHT AND R. ENISON, *Quasi-random number sequences from a long-period TLP generator with remarks on application to cryptography*, Comput. Surveys, 11 (1979), pp. 357–370.
- [3] M. FUSHIMI, *Increasing the orders of equidistribution of the leading bits of the Tausworthe sequence*, Inform. Process. Lett., 16 (1983), pp. 189–192.
- [4] —, *A reciprocity theorem on the random number generation based on  $m$ -sequences and its applications*, Trans. Inform. Process. Soc. Japan, 24 (1983), pp. 576–579. (In Japanese.)
- [5] M. FUSHIMI AND S. TEZUKA, *The  $k$ -distribution of the generalized feedback shift register pseudorandom numbers*, Comm. ACM, 26 (1983), pp. 516–523.
- [6] R. S. GARFINKEL AND G. L. NEMHAUSER, *Integer Programming*, John Wiley, New York, 1972.
- [7] S. W. GOLOMB, *Shift Register Sequences*, Holden-Day, San Francisco, 1967.
- [8] H. KASHIWAGI, *On some properties of TLP random numbers generated by  $m$ -sequence*, Trans. Soc. Instrument and Control Engineers, 18 (1982), pp. 828–832. (In Japanese.)
- [9] D. E. KNUTH, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, 2nd ed.*, Addison-Wesley, Reading, MA, 1981.
- [10] T. G. LEWIS AND W. H. PAYNE, *Generalized feedback shift register pseudorandom number algorithms*, J. Assoc. Comput. Mach., 20 (1973), pp. 456–468.
- [11] R. C. TAUSWORTHE, *Random numbers generated by linear recurrence modulo two*, Math. Comp., 19 (1965), pp. 201–209.
- [12] J. P. R. TOOTILL, W. D. ROBINSON AND A. G. ADAMS, *The runs up-and-down performance of Tausworthe pseudo-random number generators*, J. Assoc. Comput. Mach., 18 (1971), pp. 381–399.
- [13] J. P. R. TOOTILL, W. D. ROBINSON AND D. J. EAGLE, *An asymptotically random Tausworthe sequence*, J. Assoc. Comput. Mach., 20 (1973), pp. 469–481.

## SORTING AND RECOGNITION PROBLEMS FOR ORDERED SETS\*

U. FAIGLE† AND GY. TURÁN‡

**Abstract.** How many questions are needed to decide whether an unknown ordered set is isomorphic to a fixed ordered set  $P_0$ ? This recognition problem is considered, together with some related computational problems concerning ordered sets.

**Key words.** sorting, recognition, identification, order, complexity, algorithm

**AMS(MOS) subject classifications.** 68E05, 68C25

**Introduction.** The standard sorting problem is to identify an unknown order knowing that it is a linear order. This formulation suggests the following generalization to ordered sets, called the  $P_0$ -*sorting problem*: determine an unknown order, knowing that it is isomorphic to a fixed “pattern” order  $P_0$  (thus  $P_0$  is a chain in the classical case).

A related question is the  $P_0$ -*recognition problem*: decide whether the unknown order is isomorphic to  $P_0$ . The *identification problem* asks for determining the unknown order without any a priori information.

In this paper we discuss the complexity of these problems. The model of computation is the usual decision tree model with the only modification that now every node has three sons since two elements can turn out to be incomparable as well. For the sorting and recognition problems we use the worst-case measure of complexity. This measure is of no use for the identification problem as clearly every identification algorithm has to ask every pair in order to identify an antichain. Instead, we introduce a refined measure of complexity which assigns a function  $C_A(P)$  to every algorithm  $A$  giving the worst-case behavior of  $A$  on orders isomorphic to  $P$  for every order  $P$ .

In § 2 we observe some connections between the complexities of the above problems. We need the notion of an *essential set* (consisting of covering and critical pairs), essentially defined in Rabinovitch and Rival [8] and Kelly [4], to describe these connections.

An example is given of an adversary argument to bound the sorting complexity in § 3. In § 4 the recognition complexity of Boolean algebras is determined to be  $\Theta(n \log_2 n)$ . It is shown that the minimal recognition complexity of orders with height 1 on  $n$  elements is asymptotically  $n \log_2 n$ . (Also, an  $\Omega(n \log_2 n)$  lower bound holds for ordered sets with width less than  $n^{1-\varepsilon}$  for any  $\varepsilon > 0$ , in particular for orders with bounded width.)

Section 5 contains identification algorithms. The first uses Dilworth decomposition, the second is based on a recent result of Linial and Saks [6] on the existence of central elements; both generalize sorting by insertion. These algorithms are optimal for orders of bounded width but they can perform badly in general. The third algorithm is optimal for orders of height 1, the “other extreme.” The merging of these algorithms is still not optimal as is shown by their behavior on Boolean algebras.

---

\* Received by the editors February 5, 1985; accepted for publication (in revised form) April 20, 1987. This research was supported by the joint research project “Algorithmic Aspects of Combinatorial Optimization” of the Hungarian Academy of Sciences (Magyar Tudományos Akadémia) and the German Research Association (Deutsche Forschungsgemeinschaft, SFB 21).

† Institut für Ökonometrie und Operations Research, Abteilung Operations Research, Rheinische Friedrich-Wilhelms-Universität Bonn, Bonn, West Germany. Present address, Department of Applied Mathematics, University of Twente, Enschede, Netherlands.

‡ Department of Mathematics, Statistics and Computer Science, University of Illinois, Chicago, Illinois 60637. This author was on leave from Automata Theory Research Group of the Hungarian Academy of Sciences, Szeged, Hungary.

Some remarks and open problems are given in § 6. In particular, it is pointed out that the problems discussed are special cases of more general problems which are ordered set versions of graph property recognition problems in the sense discussed in Bollobás [1].

**1. Definitions.** Ordered sets  $P$  on  $n$  elements are assumed to have a fixed ground set  $\{a_1, \dots, a_n\}$ ; the order relation is  $<$ ; incomparability is denoted by  $\parallel$ . An element  $x$  is *isolated* if  $x \parallel y$  for every  $y$ . The pair  $(a_i, a_j)$  is a *covering pair* if  $a_i < a_j$  and there is no  $a_k$  with  $a_i < a_k < a_j$  (these pairs form the Hasse diagram of  $P$ ). The *width*  $w(P)$  (resp. the *height*  $h(P)$ ) of  $P$  is the size of a maximum antichain in  $P$  (resp. the length of a maximum chain in  $P$ ).

An *ideal*  $I$  (resp. a *filter*  $F$ ) is a subset of elements subject to  $x \in I, y < x$ , which implies  $y \in I$  (resp.  $x \in F, y > x$  implies  $y \in F$ ).  $I_P(x) = \{y: y < x\}$  is the ideal generated by  $x$ ,  $F_P(x) = \{y: y > x\}$  denotes the filter generated by  $x$ .  $N_P$  is the total number of ideals in  $P$  and  $N_P(x)$  is the number of ideals containing  $x$ .

The theorem of Dilworth [2] relates the width of  $P$  to a chain cover of the ground set: a chain cover of  $P$  with  $k$  chains exists if and only if  $k \geq w(P)$ .

Let  $P_0$  be an ordered set on  $n$  elements.

The  $P_0$ -*sorting problem* is to determine an unknown order  $P$  knowing only that  $P$  is isomorphic to  $P_0$ .

An *algorithm*  $A$  to solve the  $P_0$ -sorting problem is a *ternary decision tree* with nonleaves labeled " $a_i: a_j$ " for some  $1 \leq i < j \leq n$  and outgoing edges labeled " $a_i < a_j$ ," " $a_i \parallel a_j$ ," " $a_i > a_j$ ." A leaf is either labeled by an order  $P \simeq P_0$  or by a sign  $xx$ . If it is labeled by  $P$ , then  $P$  is the only order isomorphic to  $P_0$  consistent with the answers obtained along the path leading to that leaf. If it is labeled by  $xx$ , then there is no order isomorphic to  $P_0$  satisfying the answer on the corresponding path. For  $P \simeq P_0$ ,  $A(P)$  is the number of questions used to find  $P$  (the length of the unique path leading to the leaf labeled  $P$ ).

The *complexity* of  $A$  is

$$C_A^s := \max \{A(P): P \simeq P_0\}$$

and

$$C^s(P_0) := \min \{C_A^s: A \text{ is a } P_0\text{-sorting algorithm}\}$$

is the *sorting complexity* of  $P_0$ .

The  $P_0$ -*recognition problem* is to decide if an unknown order  $P$  is isomorphic to  $P_0$ .

An *algorithm*  $A$  to solve the recognition problem is again a ternary tree with nonleaves and edges labeled as above. Leaves are labeled with "yes" or "no." If a leaf is labeled "yes," then all orders satisfying the answers given along the path leading to the leaf must be isomorphic to  $P_0$ . If it is labeled "no," then there can be no order isomorphic to  $P_0$  consistent with these answers.

The *complexity*  $C_A^r$  of  $A$  is the depth of the tree and

$$C^r(P_0) := \min \{C_A^r: A \text{ is a } P_0\text{-recognition algorithm}\}$$

is the *recognition complexity* of  $P_0$ .

The *identification problem* is to determine an unknown order without any a priori information.

An *algorithm*  $A$  to solve the identification problem is a sequence  $A = (A_1, A_2, \dots)$ , where  $A_n$  is an algorithm to solve the identification problem on  $n$  elements. Thus  $A_n$  is a ternary tree as above, with leaves labeled by orders on  $\{a_1, \dots, a_n\}$  subject to a

leaf labeled  $P$  if  $P$  is the only ordered set consistent with the answers on the corresponding path.

$A(P)$  is the number of questions used by  $A_n$  to identify  $P$  and

$$C_A(P) := \max \{A(P') : P' \simeq P\}$$

is the *complexity* of  $A$  to identify orders isomorphic to  $P$ . Note that here the complexity of  $A$  is a function.

An identification algorithm is defined to be *optimal* if for every identification algorithm  $B$

$$C_A(P) = O(C_B(P)).$$

(Notation:  $f(P) = O(g(P))$  means  $|f(P)/g(P)| \leq \alpha$  for some constant  $\alpha > 0$ ,  $f(P) = \Omega(g(P))$  if  $g(P) = O(f(P))$  and  $f(P) = \Theta(g(P))$  if  $f(P) = O(g(P))$  and  $f(P) = \Omega(g(P))$ ,  $f(n) = o(g(n))$  if  $f(n)/g(n) \rightarrow 0$  as  $n \rightarrow \infty$ .)

If  $P$  is an ordered set on  $\{a_1, \dots, a_n\}$ , then  $E = E_1 \cup E_2 \subseteq \{a_1, \dots, a_n\} \times \{a_1, \dots, a_n\}$  is an *essential set* for  $P$  if

- (a)  $(a_i, a_j) \in E_1$  implies  $a_i < a_j$  and  $(a_i, a_j) \in E_2$  implies  $a_i \parallel a_j$ ,
- (b)  $P$  is the only order satisfying (a).

(Thus  $E$  is a set of comparabilities and incomparabilities which uniquely determines  $P$ .) The size of a minimum essential set is denoted by  $e(P)$ .

**2. Some observations.** In this section we collect some useful facts about the problems defined above.

LEMMA 1. For every order  $P_0$ ,

$$\max(C^s(P_0), e(P_0)) \leq C^r(P_0) \leq C^s(P_0) + e(P_0).$$

*Proof.* Consider a  $P_0$ -recognition algorithm and apply it to an order  $P \simeq P_0$ . Then arriving at the leaf,  $P$  must be identified. Indeed, if the relation between  $a_i$  and  $a_j$  were not determined then it is easy to see that  $a_i \parallel a_j$  and, say,  $a_i > a_j$  would both be consistent. It is impossible, however, that both relations yield orders isomorphic to  $P_0$  (i.e., the number of incomparable pairs is not the same). Thus every recognition algorithm can be used for sorting and hence  $C^s(P_0) \leq C^r(P_0)$ . Furthermore,  $e(P_0) \leq C^r(P_0)$  because in the above case enough questions must be asked to determine  $P$  uniquely.

A recognition algorithm for  $P_0$  can be obtained as follows: assume that the unknown order is isomorphic to  $P_0$  and apply a sorting algorithm. Then check the elements of a minimal size essential set to justify the assumption. Thus  $C^r(P_0) \leq C^s(P_0) + e(P_0)$ .  $\square$

LEMMA 2. If  $A$  is an identification algorithm then for every order  $P_0$ ,

$$C_A(P_0) \geq C^r(P_0).$$

*Proof.* Every identification algorithm can be used as a  $P_0$ -recognition algorithm: run  $A$  for  $C_A(P_0)$  steps. Then every  $P \simeq P_0$  is already identified. If the order is not identified yet it cannot be isomorphic to  $P_0$ .  $\square$

LEMMA 3. An identification algorithm  $A$  is optimal iff

$$C_A(P_0) = O(C^r(P_0)).$$

*Proof.* ( $\leftarrow$ ) Follows from Lemma 2 above.

( $\rightarrow$ ) If  $C_A(P_0)/C^r(P_0)$  is not bounded, choose, for every  $k$ , an order  $P_k$  on  $n_k$  elements with  $C_A(P_k) > k \cdot C^r(P_k)$  such that  $n_1 < n_2 < \dots$ . Let  $B$  be an identification algorithm obtained from  $A$  by merging  $A_{n_k}$  with an optimal recognition algorithm for

$P_k$  (merging means asking questions of the two algorithms alternatingly). It follows from the proof of Lemma 1 that B in fact is an identification algorithm and  $C_B(P_k) \leq 2C^r(P_k)$ . Thus  $C_A(P_k) > (k/2)C_B(P_k)$  and A is not optimal. (Also note that B is a “better” algorithm as  $C_B(P) \leq 2C_A(P)$  holds for every  $P$ .)  $\square$

The next lemma is a reformulation of observations made by Rabinovitch and Rival [8] and Kelly [4].

A pair  $(a_i, a_j)$  is *critical* if  $a_i \parallel a_j$ ,  $x > a_i$  implies  $x > a_j$ , and  $y < a_j$  implies  $y < a_i$ .

LEMMA 4. *For every  $P_0$ , the covering and critical pairs form the unique minimal essential set.*

*Proof.* These pairs are necessary as their status is not implied by the other relations. Covering pairs imply all comparabilities. Incomparability of critical pairs implies all incomparabilities. (See [8], [4].)  $\square$

In the sequel this unique minimal essential set will be denoted by  $E(P_0)$ .

**3. Sorting.** There are simple examples with small sorting complexity. For example, if  $P_0$  has a unique maximal element and all other elements are incomparable, then  $C^s(P_0) = \lceil (n-1)/2 \rceil$ .

If  $\#(P_0)$  denotes the number of orders isomorphic to  $P_0$  on the ground set  $\{a_1, \dots, a_n\}$  then clearly

$$C^s(P_0) \geq \log_3 \#(P_0).$$

Partitioning the elements into antichains according to their height (i.e., the length of the longest chain ending with them) and considering the sizes  $s_1, \dots, s_n$  of these antichains we get

$$\#(P_0) \geq \frac{n!}{s_1! \cdots s_n!}$$

and with  $s_i \leq w(P_0)$  this implies

$$C^s(P_0) \geq n \log_3 n - n \log_3 w(P_0) - 5n.$$

Later we shall see (Corollary 18 in § 5) that

$$C^s(P_0) \leq w(P_0) \cdot 2n \log_2 n + 3nw(P_0).$$

Thus, when  $w$  is fixed, the sorting complexity is determined within a constant factor.

Another obvious example where the information-theoretic bound is not sharp, besides the example above, is given by the  $n$ -element order containing only one comparable pair. Here  $\#(P_0) = n(n-1)$  and  $C^s(P_0) = \binom{n}{2}$ .

A special case which may be of some interest is the order consisting of independent comparable pairs (a “matching”), where the information-theoretic bound is  $\Omega(n \log_2 n)$ .

The following theorem determines the sorting complexity of a matching. The upper bound was observed by M. Aigner.

THEOREM 5. *Let  $n$  be even and  $P_0$  be the parallel composition of  $n/2$  2-element chains. Then for the sorting complexity of  $P_0$*

$$C^s(P_0) = \frac{n^2}{4}.$$

*Proof.* For the lower bound we use the “greedy” adversary strategy: assume that at every stage of the algorithm the answer to a question “ $a_i : a_j$ ” is always “ $a_i \parallel a_j$ ” unless adding this incomparability to the information provided already leaves no order

isomorphic to  $P_0$  consistent with these data. In the latter case the answer is “ $a_i > a_j$ .” (It is easy to see that the direction of the inequality in the answer is arbitrary; it is only fixed for being definite.)

Assume that the algorithm stops by determining the order  $P \simeq P_0$  with comparable pairs  $a_1 > a_2, a_3 > a_4, \dots, a_{n-1} > a_n$ . Then it is clear that each of these pairs must have been asked by the algorithm as otherwise reversing one inequality would also be consistent with every other answer. Letting  $E(>)$  denote the set of these comparable pairs, we assume that they have been asked by the algorithm in the above order. Further, let  $E(\parallel)$  be the set of pairs  $(a_i, a_j)$  asked by the algorithm with answer “ $a_i \parallel a_j$ ” obtained and  $\bar{E}$  be the set of unasked pairs.

Just before  $(a_1, a_2)$  was asked, the set of pairs not asked yet contained  $E(>) \cup \bar{E}$  and as the adversary uses the greedy strategy, it must have been the case that every perfect matching (corresponding to the comparable pairs) in this set of pairs contains  $(a_1, a_2)$ . Hence every perfect matching in  $E(>) \cup \bar{E}$  contains  $(a_1, a_2)$ .

The same argument shows that every perfect matching in  $E(>) \cup \bar{E}$  contains  $(a_3, a_4), \dots, (a_{n-1}, a_n)$  as well, thus  $E(>) \cup \bar{E}$  contains exactly one perfect matching.

Now we use the following result of Heteyi [3] (see also [7, Problem 7.24]):

LEMMA 6 [3]. *If a graph on  $n$  vertices has exactly one perfect matching then it has at most  $n^2/4$  edges.*

This implies  $|E(\parallel)| \geq n(n-2)/4$ . As all pairs in  $E(\parallel)$  and  $E(>)$  have been asked by the algorithm, the lower bound follows.

A sorting algorithm can be given as follows: starting with an arbitrary element  $a_1$ , the only element  $a_2$  comparable to it can be found with  $n-1$  questions. Elements  $a_1$  and  $a_2$  can then be discarded. To find the next comparable pair,  $n-3$  questions are needed, etc. Altogether  $(n-1) + (n-3) + \dots + 1 = n^2/4$  questions are required.

**4. Recognition.** In this section we first determine the recognition complexity of Boolean algebras (this example will be used in the next section). Then we show that the minimal recognition complexity of orders with height 1 on  $n$  elements is asymptotically  $n \log_2 n$ . Finally we construct orders of height 4 with linear size essential set (implying that the proof for height 1 does not extend to arbitrary bounded height).

In the next section it is shown (Corollary 18) that

$$C^r(P_0) \leq w(P_0) \cdot 2n \log_2 n + 3nw(P_0)$$

and, if  $h(P_0) = 1$ , then

$$C^r(P_0) \leq 2e(P_0).$$

This means that the recognition complexity is determined for orders of bounded width or of height 1 up to a constant factor. (Note that we do not have an analogous statement for the sorting complexity of orders of height 1.)

THEOREM 7. *Let  $n = 2^m$  and  $B_n$  be the Boolean algebra on  $m$  atoms. Then for the recognition complexity of  $B_n$*

$$C^r(B_n) = \Theta(n \log_2 n).$$

LEMMA 8.  $e(B_n) = (n/2 + 1) \log_2 n$  (if  $m > 2$ ).

*Proof.* We use Lemma 4. Every element has degree  $m$  in the Hasse diagram. Critical pairs form a matching of atoms and co-atoms.  $\square$

LEMMA 9. *If  $P$  is an arbitrary ordered set on  $n$  elements, then a maximal (nonextendible) chain can be found in  $n \cdot \lceil \log_2 (h(P) + 2) \rceil$  steps.*

*Proof.* Assume we found a chain  $x_1 < \dots < x_k$ . Take an unused element  $y$  and start inserting  $y$  into the chain by binary search. If the search is successful,  $y$  is inserted in at most  $\lceil \log_2(k+1) \rceil$  steps. If  $y \parallel x_i$  for some  $1 \leq i \leq k$  then the chain cannot be extended by  $y$  and hence  $y$  can be excluded from later comparisons.  $\square$

LEMMA 10.  $C^s(B_n) \leq n(\log_2 n + 2 \lceil \log(\log n + 2) \rceil + 2)$ .

*Proof.* We describe a  $B_n$ -sorting algorithm.

Find a maximal chain  $x_0 < \dots < x_m$  in  $n \lceil \log(\log n + 2) \rceil$  steps. Then  $x_1$  is an atom. Compare every element to  $x_1$ ; those incomparable to  $x_1$  form a Boolean algebra with  $m-1$  atoms. Repeat the same process altogether  $m$  times. Finally the atoms  $y_1, \dots, y_m$  are identified and we used at most  $2n(1 + \lceil \log(\log n + 2) \rceil)$  steps.

Now compare every element with  $y_1, \dots, y_m$ . This determines the Boolean algebra since

$$u \geq v \text{ iff } v \geq y_i \text{ implies } u \geq y_i \text{ for every atom } y_i.$$

The last phase needs at most  $n \log_2 n$  steps.  $\square$

*Proof of Theorem 7.* The upper bound follows directly from Lemmas 1, 8 and 10. By fixing atoms  $b_1, \dots, b_m$  all the other elements are already determined. Thus

$$\begin{aligned} C^r(B_n) &\geq C^s(B_n) \geq \log_3 \#(B_n) \geq \log_3 \left( \binom{n}{m} (n-m)! \right) = n \log_3 n + o(n \log n) \\ &= \Omega(n \log_2 n). \end{aligned} \quad \square$$

We now turn to orders of height 1.

LEMMA 11. *For every  $\varepsilon > 0$  there exists an  $n_0$  such that if  $P_0$  is an ordered set on  $n$  elements with  $n \geq n_0$  and  $h(P_0) = 1$  then*

$$C^r(P_0) \geq (1 - \varepsilon)n \log_2 n.$$

For the proof of Lemma 11, we use the following lemma.

LEMMA 12. *For every  $\varepsilon > 0$  there exists an  $n_0$  such that if  $P_0$  is an ordered set on  $n$  elements with  $n \geq n_0$  and  $h(P_0) = 1$  then*

$$e(P_0) \geq (1 - \varepsilon)n \log_2 n.$$

*Proof.* Let  $P_0 = A \cup B$  where the elements in  $A$  are maximal and the elements in  $B$  are minimal (isolated elements are distributed arbitrarily). Assume  $|A| = l$ ,  $|B| = k$ ,  $k \leq \lfloor n/2 \rfloor$ . Then every pair  $(x, y)$  with  $x \in B$ ,  $y \in A$  belongs to  $E(P_0)$  (every such pair is either critical or covering). Consider now  $x, y \in A$ . Then  $x \parallel y$  and  $(x, y)$  or  $(y, x)$  is a critical pair iff  $I_{P_0}(x)$  and  $I_{P_0}(y)$  are comparable. If  $I_{P_0}(x) = I_{P_0}(y)$  then  $(x, y)$  is critical.

Let  $H_1, \dots, H_t$  ( $t = 2^k$ ) be the subsets of  $B$  and  $s_i$  ( $1 \leq i \leq t$ ) be the number of elements  $x$  in  $A$  with  $I_{P_0}(x) = H_i$ . Then there are

$$r = \sum_{i=1}^t \binom{s_i}{2}$$

pairs  $(x, y)$  with  $I_{P_0}(x) = I_{P_0}(y)$  ( $x, y \in A$ ,  $x \neq y$ ). As  $\sum_{i=1}^t s_i = l$ , the convexity of the function  $f(x) = x^2$  implies that the above sum is minimized if all the  $s_i$  are

$$\lfloor l/2^k \rfloor \text{ or } \lceil l/2^k \rceil.$$

Thus

$$r \geq \binom{\lfloor l/2^k \rfloor}{2} \cdot 2^k.$$

Hence

$$\begin{aligned} e(P_0) &\geq k \cdot l + \binom{\lfloor l/2^k \rfloor}{2} \cdot 2^k \\ &\geq k \cdot l + \binom{\lfloor n/2^{k+1} \rfloor}{2} \cdot 2^k \\ &\geq k \cdot l + \frac{(n-2^{k+1})(n-2^{k+2})}{8 \cdot 2^k}. \end{aligned}$$

If

$$(*) \quad n \geq 4 \cdot 2^{k+1}$$

then

$$e(P_0) \geq k \cdot l + \frac{1}{32} \cdot \frac{n^2}{2^k}.$$

We distinguish two cases. If

$$k \geq \log_2 n - \log_2 \log_2 n - 5$$

then

$$e(P_0) \geq (\log_2 n - \log_2 \log_2 n - 5)(n - \log_2 n) \geq (1 - \varepsilon)n \log_2 n$$

when  $n$  is sufficiently large. If

$$k < \log_2 n - \log_2 \log_2 n - 5$$

then  $(*)$  holds and

$$2^k < \frac{n}{32 \log_2 n}.$$

Thus

$$e(P_0) \geq \frac{1}{32} \cdot \frac{n^2}{2^k} > n \log_2 n. \quad \square$$

*Proof of Lemma 11.* Using Lemma 1, the lemma follows immediately from Lemma 12 above.  $\square$

LEMMA 13. *For every  $\varepsilon > 0$ , there exists an  $n_0$ , such that if  $n \geq n_0$  then there exists an order  $P_0$  on  $n$  elements with  $h(P_0) = 1$  and recognition complexity*

$$C_r(P_0) \leq (1 + \varepsilon)n \log_2 n.$$

The proof is given in the next section after Corollary 22.

THEOREM 14. *Let  $f(n) := \min \{C'(P_0) : |P_0| = n, h(P_0) = 1\}$ . Then*

$$\frac{f(n)}{n \log_2 n} \rightarrow 1 \quad \text{as } n \rightarrow \infty.$$

*Proof.* Follows directly from Lemmas 11 and 13.  $\square$

Lemma 12 might suggest that  $e(P_0) = \Omega(n \log_2 n)$  for orders with  $h(P) \leq k$  for every fixed  $k$ . The next theorem shows that this is not true.

THEOREM 15. *There exist orders of height 4 with  $e(P) = (4 + o(1))|P|$ .*



*Proof.* Take two disjoint sets  $A, B$  with  $|A| = |B| = 2m$  and define

$$\mathcal{H}_1 := \{\{x\} : x \in A \cup B\};$$

$$\mathcal{H}_2 := \{H : |H| = m \text{ and } H \subseteq A \text{ or } H \subseteq B\};$$

$$\mathcal{H}_3 := \{H : H = H_1 \cup H_2 \text{ with } |H_1| = |H_2| = m, H_1 \subseteq A, H_2 \subseteq B\};$$

$$\mathcal{H}_4 := \{H : |H| = 3m, (A \cup B) - H \in \mathcal{H}_2\};$$

$$\mathcal{H}_5 := \{A \cup B - \{x\} : x \in A \cup B\};$$

$$\mathcal{H} := \bigcup_{i=1}^5 \mathcal{H}_i;$$

$$P_{\mathcal{H}} := \mathcal{H} \text{ ordered by inclusion.}$$

Then clearly  $P_{\mathcal{H}}$  is of height 4 and has

$$8m + 4 \binom{2m}{m} + \binom{2m}{m}^2 = \binom{2m}{m}^2 (1 + o(1))$$

elements. Furthermore, the Hasse-diagram of  $P_{\mathcal{H}}$  contains

$$8m \binom{2m-1}{m-1} + 4 \binom{2m}{m}^2 = 4 \binom{2m}{m}^2 (1 + o(1))$$

edges. The first term counts edges incident to maximal and minimal elements. The second term counts edges incident to  $\mathcal{H}_3$ : each  $H \in \mathcal{H}_3$  contains exactly two sets in  $\mathcal{H}_2$  and is contained in exactly two sets from  $\mathcal{H}_4$ .

The critical pairs in  $P_{\mathcal{H}}$  are the pairs  $(A \cup B - \{x\}, \{x\})$  as in the case of the full Boolean algebra.

Thus

$$e(P) = 4 \binom{2m}{m}^2 (1 + o(1)),$$

proving the theorem.  $\square$

(Note that if we delete some elements of  $\mathcal{H}_3$  so that each set in  $\mathcal{H}_2$  and  $\mathcal{H}_4$  has still neighbors in  $\mathcal{H}_3$ , the necessary properties of  $P_{\mathcal{H}}$  are preserved. Thus for every  $n$  there exist height 4 orders with linear size essential set.)

To close this section we note that from the bound given at the beginning of § 3 it follows that

$$C^r(P_0) = \Omega(n \log_2 n)$$

if  $w(P_0) \leq n^{1-\varepsilon}$  for any  $\varepsilon > 0$ , suggesting problem 2 in the last section.

**5. Identification.** We describe some identification algorithms and give bounds for their performance.

The first two algorithms generalize sorting by insertion. Inserting an element  $x$  into an ordered set  $P'$  means to determine which elements are smaller, larger or incomparable to  $x$ .

Both algorithms A and B proceed in  $n$  stages on orders on  $\{a_1, \dots, a_n\}$ . After stage  $i$  the suborder  $P_i$  on  $\{a_1, \dots, a_i\}$  of  $P$  is determined. Stage  $i+1$  consists of inserting  $a_{i+1}$  into  $P_i$ . We only give the insertion methods.

*Insertion in algorithm A.* Choose a minimum chain cover  $C_1, \dots, C_{l_i}$  of  $P_i$  and insert  $a_{i+1}$  into each chain. Insertion into a chain is done by a straightforward modification of binary search.

To describe algorithm B we note that the elements of  $\{a_1, \dots, a_i\}$  that are smaller than  $a_{i+1}$  in  $P$  form an ideal  $I_i$  in  $P_i$  (resp. the elements which are larger form a filter  $F_i$  of  $P_i$ ).

*Insertion in algorithm B.* Determine  $I_i$  and  $F_i$  by algorithm C below.

Obviously, the two tasks are equivalent so we only formulate the algorithm to determine  $I_i$ .

An element  $z$  of  $P'$  is called a *central element* if for every  $z'$  in  $P'$

$$\left| \frac{N_{P'}(z)}{N_{P'}} - \frac{1}{2} \right| \leq \left| \frac{N_{P'}(z')}{N_{P'}} - \frac{1}{2} \right|.$$

Algorithm C below determines the ideal  $P'(x) = \{y: y \in P', y < x\}$  for an already identified order  $P'$  and a new element  $x$  recursively.

ALGORITHM C.

Given  $P'$  and  $x$  choose a central element  $z$  of  $P'$  and compare it with  $x$ .

If  $x > z$  then put  $P'(x) := I_{P'}(z) \cup \{z\} \cup P''(x)$ , where  $P'' = P' - (I_{P'}(z) \cup \{z\})$ .

If  $x \not> z$  then put  $P'(x) := P''(x)$ , where  $P'' = P' - (F_{P'}(z) \cup \{z\})$ .

We need the following result of Linial and Saks [6].

LEMMA 16 [6]. *If  $z$  is a central element then*

$$\delta_0 \leq \frac{N_{P'}(z)}{N_{P'}} \leq 1 - \delta_0,$$

where

$$\delta_0 = \frac{1}{4}(3 - \log_2 5) \approx 0.17.$$

THEOREM 17. *A and B are correct identification algorithms. For every ordered set  $P$  it holds that*

$$C_A(P) \leq 2nw(P)^{\frac{1}{2}} + \log_2(n + w(P)) - \log_2 w(P)$$

and

$$C_B(P) \leq 7.46n \log_2 N_P.$$

*Proof.* The correctness of A is obvious. The correctness of B follows from observing that if  $x > z$  then  $x > z'$  for every  $z' < z$  in  $P'$  and if  $x \not> z$  then  $x \not> z'$  for every  $z' > z$  in  $P'$ .

The bound for A holds since insertion into a chain of length  $m$  is easily seen to require at most  $2 \log_2(m+1) + 1$  steps and at each stage  $l_i \leq w(P)$  by Dilworth's theorem. (The computation uses the concavity of  $\log_2(x)$ .)

The bound for B is obtained as in [6] noting that the choice of a central element always reduces the number of ideals by a factor of  $(1 - \delta_0)$  at least, using Lemma 16 above.  $\square$

COROLLARY 18. For every order  $P_0$

$$C^s(P_0) \leq C^r(P_0) \leq w(P_0)2n \log_2 n + 3nw(P_0).$$

*Proof.* Immediate from Lemmas 1 and 2.  $\square$

Comparing A and B we note that A is simpler to compute. The bound of B is not worse than that of A but we do not know of any examples where it performs much better.

Now we describe another identification algorithm, efficient for orders of height

1. We use the following variables:

- $P'$  denotes the actual order obtained up to a certain stage of the algorithm (including the comparabilities and incomparabilities deduced);
- MAX (resp. MIN) is the set of elements that are maximal (resp. minimal) in  $P'$ ; (where now  $x$  is maximal (resp. minimal) if  $x \prec z$  for every  $z$  and  $x > y$  for some  $y$  (resp.  $x \succ z$  and  $x < y$  for some  $y$ ));
- ISO is the set of elements that are known to be isolated in  $P$  (thus  $x \in \text{ISO}$  iff it is already known to be incomparable to every element);
- $U$  is the set of "unprocessed" elements ( $U = P - (\text{MAX} \cup \text{MIN} \cup \text{ISO})$ ).

A pair  $(x, y)$  is called *undetermined* in  $P'$  if neither  $x \parallel y$ ,  $x > y$  nor  $x < y$  is in  $P'$ ; otherwise it is determined.

ALGORITHM D.

1.  $P' := \emptyset$ , MAX :=  $\emptyset$ , MIN :=  $\emptyset$ , ISO :=  $\emptyset$ ,  $U := \{a_1, \dots, a_n\}$ .
2. If  $U = \emptyset$  compare every undetermined pair  $x, y \in \text{MAX}$  subject to  $I_P(x) \subseteq I_P(y)$  and every undetermined pair  $x, y \in \text{MIN}$  subject to  $F_P(x) \subseteq F_P(y)$ , go to 6.
3. Choose a  $z \in U$  and compare it to every  $x$  subject to  $(z, x)$  is undetermined in  $P'$ . Update  $P'$ , MAX, MIN, ISO,  $U$ . If  $h(P') > 1$ , go to 5.
4. If  $|\text{MAX}| \leq |\text{MIN}|$  compare every  $x \in \text{MAX}$  with every  $y$  s.t.  $(x, y)$  is undetermined in  $P'$ . Update  $P'$ , MAX, MIN, ISO,  $U$ . If  $h(P') > 1$  go to 5. If  $|\text{MIN}| \leq |\text{MAX}|$  compare every  $x \in \text{MIN}$  with every  $y$  s.t.  $(x, y)$  is undetermined in  $P'$ . Update  $P'$ , MAX, MIN, ISO,  $U$ . If  $h(P') > 1$  go to 5, else go to 2.
5. Compare every undetermined pair.
6. Stop.

Informally, the algorithm proceeds as follows. Sets MAX, MIN and ISO contain elements already known to be maximal, minimal or isolated. If there is an element  $z$  not contained in any of them, it is compared to every element. To update necessary information, elements in the smaller of the two new sets MIN and MAX (or elements in both sets, if they are of equal size) are compared to every other element. If  $\text{MAX} \cup \text{MIN} \cup \text{ISO}$  contains every element, all remaining undetermined pairs are compared. If at any point the order turns out to have height  $> 1$ , all pairs are compared.

PROPOSITION 19. Algorithm D is a correct identification algorithm.

*Proof.* If  $h(P') > 1$  during an execution of steps 3 or 4 then every undetermined pair is compared in step 5. So the order is obviously identified. Otherwise the algorithm reaches the execution of step 2: after an execution of step 3  $z$  belongs to MAX, MIN or ISO. So  $|U|$  decreases.

If step 2 is reached then the elements in ISO are already compared with every other element. Furthermore, every pair  $(x, y)$  with  $x \in \text{MAX}$ ,  $y \in \text{MIN}$  is determined as can be seen from the previous execution of step 4. Thus the only undetermined pairs  $(x, y)$  must satisfy either  $x, y \in \text{MAX}$  or  $x, y \in \text{MIN}$  (we have  $P = \text{MAX} \cup \text{MIN} \cup$

ISO at that point). If  $x, y \in \text{MAX}$  (resp.  $x, y \in \text{MIN}$ ) and  $I_P(x)$  and  $I_P(y)$  are incomparable sets then  $(x, y)$  is determined as it must be true that  $x \parallel y$ . Therefore all undetermined pairs are compared in step 2 and the order is identified indeed.  $\square$

**THEOREM 20.** *If  $h(P) = 1$  then  $C_D(P) \leq 2e(P)$ .*

*Proof.* Let  $P$  be an order of height 1. Since every pair is compared at most once, it is sufficient to show that the number of nonessential pairs ever compared by the algorithm is at most  $e(P)$ .

The definition of critical pairs implies that every pair compared in step 2 is essential. (In fact, only pairs from the larger of the two sets MAX, MIN are compared at that step.) If in step 3 the element  $z$  turns out to be isolated, then every pair containing  $z$  is critical and in this case MAX and MIN are not modified.

Therefore, the algorithm can compare nonessential pairs  $(x, y)$  only during those executions of steps 3 and 4 where the element  $z$  turns out to be maximal or minimal. (Furthermore, for every such pair either both  $x, y$  are maximal or both are minimal.) An execution of steps 3 and 4, where  $z$  turns out to be maximal or minimal, is called a *phase*.

Let  $\text{MAX}_i, \text{MIN}_i, \text{ISO}_i$  and  $U_i$  denote the values of the sets MAX, MIN, ISO and  $U$  after phase  $i$ . Let  $z_i$  be the element selected in phase  $i$  and  $V_i$  be the set of elements comparable to  $z_i$ .  $\square$

**LEMMA 21.** *For every  $i$ :*

- (a) *if  $|\text{MAX}_i| \leq |\text{MIN}_i|$ , then after phase  $i$  every pair containing an element from  $\text{MAX}_i$  is determined;*
- (b) *if  $|\text{MIN}_i| \leq |\text{MAX}_i|$ , then after phase  $i$  every pair containing an element from  $\text{MIN}_i$  is determined.*

*Furthermore, there are sets  $\text{MAX}_i^*, \text{MIN}_i^*$  with the following properties:*

- (c)  $\text{MAX}_i^* \cup \text{MIN}_i^*$  covers all pairs  $(u, v)$  compared by the algorithm in phases  $1, \dots, i$  subject to  $\{u, v\} \cap \text{ISO}_i = \emptyset$ ;
- (d)  $|\text{MAX}_i^* \cup \text{MIN}_i^*| \leq 2 \min(|\text{MAX}_i|, |\text{MIN}_i|)$ ;
- (e)  $\text{MAX}_i^* \subseteq \text{MAX}_i, \text{MIN}_i^* \subseteq \text{MIN}_i$ ;
- (f) *if  $|\text{MAX}_i| \leq |\text{MIN}_i|$  then  $\text{MAX}_i^* = \text{MAX}_i$ ;*
- (g) *if  $|\text{MIN}_i| \leq |\text{MAX}_i|$  then  $\text{MIN}_i^* = \text{MIN}_i$ .*

*Proof.* By induction on  $i$ . For  $i = 0$  all sets are empty. We distinguish several cases.

*Case 1.*  $|\text{MAX}_i| < |\text{MIN}_i|$ .

*Case 1.1.*  $z_{i+1}$  is a maximal element.

Then in step 3 of phase  $i+1$ ,  $z_{i+1}$  is added to MAX and  $V_{i+1}$  is added to MIN. (It may be the case that  $V_{i+1} \subseteq \text{MIN}_i$ .)

*Case 1.1.1.*  $|\text{MAX}_i \cup \{z_{i+1}\}| < |\text{MIN}_i \cup V_{i+1}|$ .

Then using (a) by induction, no further comparisons are performed in step 4 of phase  $i+1$ . Putting  $\text{MAX}_{i+1}^* := \text{MAX}_i^* \cup \{z_{i+1}\}$ ,  $\text{MIN}_{i+1}^* := \text{MIN}_i^*$ , all conditions remain satisfied.

*Case 1.1.2.*  $|\text{MAX}_i \cup \{z_{i+1}\}| = |\text{MIN}_i \cup V_{i+1}|$ .

(This may occur only if  $V_{i+1} \subseteq \text{MIN}_i$ .) In this case, no further comparisons are performed in the first "if" of step 4 of phase  $i+1$ , but there may be comparisons involving elements from  $\text{MIN}_i$  in the second "if" of step 4 of phase  $i+1$ . However,  $\text{MIN}_{i+1} = \text{MIN}_i$ ,  $\text{MAX}_{i+1} \subseteq \text{MAX}_i \cup \{z_{i+1}\}$  and thus  $|\text{MIN}_{i+1}| \leq |\text{MAX}_{i+1}|$ , so  $\text{MAX}_{i+1}^* := \text{MAX}_i^* \cup \{z_{i+1}\}$ ,  $\text{MIN}_{i+1}^* := \text{MIN}_i$  satisfies the requirements.

*Case 1.2.*  $z_{i+1}$  is a minimal element.

By (a),  $V_{i+1} \cap \text{MAX}_i = \emptyset$ , otherwise  $z_{i+1}$  would be contained in  $\text{MIN}_i$ . So in step 3 of phase  $i+1$ , new maximal element(s) will be added to MAX.

*Case 1.2.1.*  $|\text{MAX}_i \cup V_{i+1}| < |\text{MIN}_i \cup \{z_{i+1}\}|$ .

Then in step 4 of phase  $i + 1$ , all undetermined pairs containing an element from  $V_{i+1}$  are compared and no other pairs. We can choose  $\text{MAX}_{i+1}^* := \text{MAX}_i^* \cup V_{i+1}$ ,  $\text{MIN}_{i+1}^* := \text{MIN}_i^* \cup \{z_{i+1}\}$ .

Case 1.2.2.  $|\text{MAX}_i \cup V_{i+1}| = |\text{MIN}_i \cup \{z_{i+1}\}|$ .

In the first "if" of step 4 of phase  $i + 1$ , all undetermined pairs involving an element from  $V_{i+1}$  are compared. Possibly all undetermined pairs involving an element from  $\text{MIN}_i$  are compared in the second "if" of step 4. We can choose  $\text{MAX}_{i+1}^* := \text{MAX}_i^* \cup V_{i+1}$ ,  $\text{MIN}_{i+1}^* := \text{MIN}_i \cup \{z_{i+1}\}$ .

Case 1.2.3.  $|\text{MAX}_i \cup V_{i+1}| > |\text{MIN}_i \cup \{z_{i+1}\}|$ .

Then in step 4 of phase  $i + 1$ , undetermined pairs containing an element from  $\text{MIN}_i$  are compared and we can choose  $\text{MAX}_{i+1}^* := \text{MAX}_i^*$ ,  $\text{MIN}_{i+1}^* := \text{MIN}_i \cup \{z_{i+1}\}$ .

Case 2.  $|\text{MAX}_i| = |\text{MIN}_i|$ .

Case 2.1.  $z_{i+1}$  is a maximal element.

By (b) of the induction hypothesis,  $V_{i+1} \cap \text{MIN} = \emptyset$  and we get two cases.

Case 2.1.1.  $|V_{i+1}| = 1$ .

Let  $V_{i+1} = \{z'_{i+1}\}$ , then all undetermined pairs involving  $z'_{i+1}$  are compared in the second "if" of step 4 of phase  $i + 1$ . We can choose  $\text{MAX}_{i+1}^* := \text{MAX}_i^* \cup \{z'_{i+1}\}$ ,  $\text{MIN}_{i+1}^* := \text{MIN}_i^* \cup \{z'_{i+1}\}$ .

Case 2.1.2.  $|V_{i+1}| > 1$ .

In this case no further comparisons are performed in step 4 of phase  $i + 1$  and we can put  $\text{MAX}_{i+1}^* := \text{MAX}_i^* \cup \{z_{i+1}\}$ ,  $\text{MIN}_{i+1}^* := \text{MIN}_i^*$ .

The remaining cases require analogous arguments, so we give the definitions of the sets  $\text{MAX}_{i+1}^*$ ,  $\text{MIN}_{i+1}^*$  only.

Case 2.2.  $z_{i+1}$  is a minimal element.

Case 2.2.1.  $|V_{i+1}| = 1$ .

$$\text{MAX}_{i+1}^* := \text{MAX}_i^* \cup \{z'_{i+1}\}, \quad \text{MIN}_{i+1}^* := \text{MIN}_i^* \cup \{z_{i+1}\}.$$

Case 2.2.2.  $|V_{i+1}| > 1$ .

$$\text{MAX}_{i+1}^* := \text{MAX}_i^*, \quad \text{MIN}_{i+1}^* := \text{MIN}_i^* \cup \{z_{i+1}\}.$$

Case 3.  $|\text{MAX}_i| > |\text{MIN}_i|$ .

(Note that step 4 is not symmetric in MAX and MIN.)

Case 3.1.  $z_{i+1}$  is a maximal element.

Case 3.1.1.  $|\text{MAX}_i \cup \{z_{i+1}\}| > |\text{MIN}_i \cup V_{i+1}|$ .

$$\text{MAX}_{i+1}^* := \text{MAX}_i^* \cup \{z_{i+1}\}, \quad \text{MIN}_{i+1}^* := \text{MIN}_i^* \cup V_{i+1}.$$

Case 3.1.2.  $|\text{MAX}_i \cup \{z_{i+1}\}| = |\text{MIN}_i \cup V_{i+1}|$ .

$$\text{MAX}_{i+1}^* := \text{MAX}_i \cup \{z_{i+1}\}, \quad \text{MIN}_{i+1}^* := \text{MIN}_i^* \cup V_{i+1}.$$

Case 3.1.3.  $|\text{MAX}_i \cup \{z_{i+1}\}| < |\text{MIN}_i \cup V_{i+1}|$ .

$$\text{MAX}_{i+1}^* := \text{MAX}_i \cup \{z_{i+1}\}, \quad \text{MIN}_{i+1}^* := \text{MIN}_i^*.$$

Case 3.2.  $z_{i+1}$  is a minimal element.

Case 3.2.1.  $|\text{MAX}_i \cup V_{i+1}| > |\text{MIN}_i \cup \{z_{i+1}\}|$ .

$$\text{MAX}_{i+1}^* := \text{MAX}_i^*, \quad \text{MIN}_{i+1}^* := \text{MIN}_i^* \cup \{z_{i+1}\}.$$

Case 3.2.2.  $|\text{MAX}_i \cup V_{i+1}| = |\text{MIN}_i \cup \{z_{i+1}\}|$ .

$$\text{MAX}_{i+1}^* := \text{MAX}_i, \quad \text{MIN}_{i+1}^* := \text{MIN}_i^* \cup \{z_{i+1}\}.$$

Returning to the proof of Theorem 20, let  $i$  be the last phase,  $\text{MAX}^* := \text{MAX}_i^*$  and  $\text{MIN}^* := \text{MIN}_i^*$ . The rest of the argument is symmetric in  $\text{MAX}$  and  $\text{MIN}$ , so assume  $|\text{MAX}| \leq |\text{MIN}|$ . Then Lemma 21 implies  $\text{MAX}^* = \text{MAX}$  and as  $|\text{MAX}^*| + |\text{MIN}^*| \leq 2|\text{MAX}|$ , also  $|\text{MIN}^*| \leq |\text{MAX}^*|$  holds. All nonessential pairs compared by the algorithm are either pairs of maximal elements or pairs of minimal elements covered by  $\text{MIN}^*$ . So their number is at most

$$\begin{aligned} & \binom{\text{MAX}}{2} + \left( \binom{\text{MIN}}{2} - \binom{\text{MIN} - \text{MIN}^*}{2} \right) \\ & \leq \binom{\text{MAX}}{2} + \binom{\text{MIN}}{2} - \binom{\text{MIN} - \text{MAX}}{2} = \text{MAX} \cdot (\text{MIN} - 1) \\ & < \text{MAX} \cdot \text{MIN} \leq e(P). \quad \square \end{aligned}$$

(We remark that the example of a matching shows that the factor 2 in the bound is sharp for the algorithm.)

**COROLLARY 22.** *If  $h(P) = 1$  then  $C_D(P) \leq 2 \cdot C^r(P)$  and  $C^r(P) \leq 2 \cdot e(P)$ .*

*Proof.* Immediate from Lemmas 1 and 2.  $\square$

Now we prove Lemma 13 of previous section.

*Proof of Lemma 13.* We construct the order  $P_0$  as follows:  $P_0$  has  $\lceil (1 + \varepsilon) \log_2 n \rceil$  maximal elements and  $n - \lceil (1 + \varepsilon) \log_2 n \rceil$  minimal elements. Each minimal element has  $\lceil (1 + \varepsilon)/2 \cdot \log_2 n \rceil$  upper covers so that the set of upper covers is different for each minimal element. This condition is easy to satisfy as there are

$$\binom{\lceil (1 + \varepsilon) \log_2 n \rceil}{\lceil \frac{1 + \varepsilon}{2} \log_2 n \rceil} \geq c_\varepsilon \cdot \frac{n^{1 + \varepsilon}}{\sqrt{\log_2 n}}$$

choices. Furthermore, it can also be required that the set of lower covers be different for each maximal element and each be of size between, e.g.,  $n/4$  and  $3n/4$ . Then

$$e(P_0) = \lceil (1 + \varepsilon) \log_2 n \rceil (n - \lceil (1 + \varepsilon) \log_2 n \rceil).$$

Thus a  $2(1 + \varepsilon)n \log_2 n$  recognition algorithm for  $P_0$  is given by running  $D$  for  $C_D(P_0)$  steps. (If the order is not identified up to this point, it cannot be isomorphic to  $P_0$ .)

A slight modification gives the smaller upper bound. A pair  $x > y$  can be found in  $n - 1$  steps;  $y$  can be compared to everything in at most  $(n - 2)$  steps. Then  $y$  must have  $\lceil (1 + \varepsilon)/2 \log_2 n \rceil$  upper covers (otherwise  $P \neq P_0$ ). Comparing the upper covers of  $y$  to everything we get every minimal element except at most one. Comparing the remaining at most  $\lceil (1 + \varepsilon)/2 \log_2 n \rceil + 1$  elements to everything,  $P$  is identified whenever  $P \approx P_0$ .  $\square$

Finally we point out that algorithms B and D are not sufficient to produce an optimal algorithm. Let algorithm  $E$  be obtained by merging B and D (in the sense of the proof of Lemma 3).

Then for the Boolean algebras  $B_n$  we have

$$C^r(B_n) = O(n \log_2 n)$$

from Theorem 7 and

$$C_E(B_n) = \Omega\left(\frac{n^2}{\log_2 n}\right).$$

Indeed, if the antichain of size

$$\binom{m}{m/2} = \Omega\left(\frac{m}{\sqrt{\log_2 n}}\right)$$

is examined first then every pair is compared in order to perform the insertion in B, and D is of no help in this case.

**6. Some open problems.** (1) It would be interesting to find good bounds for the sorting complexity in general.

(2) The special cases discussed in § 4 suggest that perhaps the recognition complexity is  $\Omega(n \log n)$  for every order  $P_0$ .

(3) Is it true that there is no optimal identification algorithm in the strong sense of optimality defined here (cf. Lemma 3)? Are there other measures of complexity for identification algorithms?

(4) The definition of sorting and recognition problems introduced here is actually a special case of a more general one. One can consider sorting and recognition problems for arbitrary classes of orders (in this paper we considered classes that consist of isomorphic copies of a fixed order  $P_0$ ). It seems reasonable to assume that these classes are closed under isomorphism, i.e., they are *ordered set properties*. In this framework the identification problem is just the sorting problem with respect to the class of all orders. The recognition problems arising this way are natural ordered set versions of analogous problems concerning graphs (see Bollobás [1, Chap. 8] for an excellent survey). The ordered set properties are substantially different as there are many “easy” properties with complexity  $o(n^2)$  like having a unique maximal element, being a linear order, having bounded width or setup number. Nevertheless it is not difficult to give  $\Omega(n^2)$  lower bounds, e.g., for connectivity, bounded height, being a lattice, an interval order, etc. What properties can be proved to be “elusive,” i.e., to have complexity  $\Omega(n^2)$ ?

**Acknowledgment.** We are grateful to M. Aigner for his remarks (in particular for pointing out the upper bound in Theorem 5) and to a referee for suggesting a simplification of the proof of Theorem 20.

*Note added in proof.* There are some further results concerning problems 2 and 3. It was shown that interval orders have recognition complexity  $\Omega(n \log n)$  and there is an optimal identification algorithm for semiorders (U. Faigle and Gy. Turán, *The complexity of interval orders and semiorders*, Discrete Math., to appear). It was also shown that almost all orders have recognition complexity  $\Omega(n \log n)$  (H. J. Prömel, *Counting unlabeled structures*, J. Combin. Theory Ser. A, 44 (1987), pp. 83–93). Recently M. Saks solved problem 2. In general, he showed that every  $n$  element order has recognition complexity at least  $\frac{2}{3}n \log_2 n + o(n \log n)$  (M. Saks, *Recognition problems for transitive relations*, to be published).

#### REFERENCES

- [1] B. BOLLOBÁS (1978), *Extremal Graph Theory*, Academic Press, New York.
- [2] R. P. DILWORTH (1950), *A decomposition theorem for partially ordered sets*, Annals Math., 51, pp. 161–166.
- [3] G. HETYEI (1964), unpublished.
- [4] D. KELLY (1981), *On the dimension of partially ordered sets*, Discrete Math., 35, pp. 135–156.
- [5] D. E. KNUTH (1973), *The Art of Computer Programming, Vol. 3, Sorting and Searching*, Addison-Wesley, Reading, MA.
- [6] N. LINIAL AND M. E. SAKS (1983), *Information bounds are good for search problems on ordered structures*, Proc. 24th Annual IEEE Conference on the Foundations of Computer Science, pp. 473–475.
- [7] L. LOVÁSZ (1979), *Combinatorial Problems and Exercises*, Akadémiai Kiadó.
- [8] L. RABINOVITCH AND L. RIVAL (1979), *The rank of a distributive lattice*, Discrete Math., 25, pp. 275–279.

## EXPECTED PERFORMANCE OF $m$ -SOLUTION BACKTRACKING\*

DAVID M. NICOL†

**Abstract.** This paper derives upper bounds on the expected number of search tree nodes visited during an  $m$ -solution backtracking search, a search which terminates after some preselected number  $m$  problem solutions are found. The search behavior is assumed to have a general probabilistic structure. Our results are stated in terms of node expansion and contraction. A visited search tree node is said to be *expanding* if the mean number of its children visited by the search exceeds 1, and is *contracting* otherwise. We show that if every node expands, or if every node contracts, then the number of search tree nodes visited by a search has an upper bound which is linear in the depth of the tree, in the mean number of children a node has, and in the number of solutions sought. Bounds linear in the depth of the tree are derived for the case where the upper portion of the tree contracts while the lower portion expands. We derive linear bounds for some special cases of an expanding upper portion and contracting lower portion; however, in the general case we have exponentially complex bounds. While previous analyses of 1-solution backtracking have concluded that the expected performance is always linear in the tree depth, our model allows super-linear expected performance. By generalizing previous work in the expected behavior of backtracking, we are better able to identify classes of trees which can be searched in linear expected time.

**Key words.** random search, random trees, backtracking, depth-first search

**AMS(MOS) subject classifications.** 68C25, 68E05

**1. Introduction.** Backtracking (or depth-first search) is a powerful tool for quickly finding solutions to certain types of problems whose potential solution space is huge. Such a search may terminate upon finding one solution, or may attempt to discover all solutions. We will study  $m$ -solution searches, which terminate once a preselected number  $m$  solutions are discovered (or when it is determined that fewer than  $m$  solutions exist). Backtracking is used to solve problems which seek an  $n$ -vector  $\langle X_1, \dots, X_n \rangle$  where each  $X_i$  is an element of a finite set  $S_i$ , such that some condition  $C(\langle X_1, \dots, X_n \rangle)$  is satisfied. For example, backtracking is often used to solve the  $n$ -Queen's problem. In this case, each vector coordinate position represents a chess board column,  $X_i = j$  means that a queen is placed in column  $i$  and row  $j$ , and the condition  $C$  is that no queen attacks any other. A backtracking search exploits necessary subconditions for a solution. The necessary subconditions  $\{C_j\}$  are chosen so that if  $C_j(\langle X_1, \dots, X_j \rangle)$  is satisfied, then  $C_i(\langle X_1, \dots, X_i \rangle)$  is satisfied for  $i$  such that  $1 \leq i < j$ . Given a candidate partial solution  $\langle X_1, \dots, X_k \rangle$ , the search checks condition  $C_k(\langle X_1, \dots, X_k \rangle)$ . If that condition is satisfied, the partial solution is extended to  $\langle X_1, \dots, X_{k+1} \rangle$  for some  $X_{k+1} \in S_{k+1}$ , and the extended candidate partial solution is tested against  $C_{k+1}$ . If on the other hand  $C_k(\langle X_1, \dots, X_k \rangle)$  is not satisfied, then an alternate value  $\hat{X}_k \in S_k$  which has not yet been appended to  $\langle X_1, \dots, X_{k-1} \rangle$  (if one exists) is chosen, and the partial solution  $\langle X_1, \dots, X_{k-1}, \hat{X}_k \rangle$  is checked against  $C_k$ . If all elements of  $S_k$  have already been appended to  $\langle X_1, \dots, X_{k-1} \rangle$  and have failed to satisfy  $C_k$ , the algorithm "backtracks" and attempts new extensions to

---

\* Received by the editors August 6, 1986; accepted for publication April 22, 1987. This research was supported by the National Aeronautics and Space Administration under NASA contract NAS1-18107 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, Virginia 23665.

† Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, Virginia 23665 and The College of William and Mary, Williamsburg, Virginia 23185.



$\langle X_1, \dots, X_{k-2} \rangle$ . Most basic algorithm texts discuss the details of backtracking, for example, see [5].

The search space of a backtracking search can be represented by a tree. A tree node is uniquely identified by the path between it and the tree root; a path of length  $j$  identifies a unique partial solution with  $j$  coordinate assignments; a node's set of children represent all possible extensions to the partial solution represented by the node. Leaf nodes represent potential full solutions. Viewed in this way, backtracking is a depth-first search of the tree with pruning. Pruning occurs if a tested partial solution fails to satisfy the appropriate condition: the entire subtree rooted in the failing node is pruned from the search space, and the search "backtracks." Backtracking's success lies in this (problem-dependent) ability to dynamically prune large portions of the search tree. However, backtracking's worst case performance is exponential in the size of the problem; nevertheless, many problems do not exhibit this worst case performance. A more meaningful measure is the *expected* performance. The usual method for determining the expected or average performance of an algorithm is to assume that problem parameters are random, or that the outcome of a decision is random (for examples, see [6]). One way of introducing randomness into backtracking is to suppose that the success or failure of a visited node is random. This approach is adopted by [12], and is the one we will explore. While a probabilistic model may not describe any particular backtracking search very accurately, it is useful for determining performance as a function of general search space characteristics. As pointed out in [7], such a model can also reveal the sensitivity of performance to the search space parameters.

A number of researchers have considered the expected performance of backtracking, or related problems. Extensive analysis of branch and bound methods on random search trees is found in [11]. That work employs the theory of branching processes [4], which provides a very uniform probability structure for the search tree. In [7], the problem of finding a least cost path in a random binary tree is considered; this analysis also employs the theory of branching processes. Backtracking searches which seek all solutions to random CNF satisfiability problems are considered in [2]. The randomness introduced to the search model there is intimately related to the problem type (CNF satisfiability), allowing a tight analysis of that problem. Close approximations for the mean and variance of the number of nodes visited in a depth-first search which terminates with one solution are derived in [12]. Our search model closely resembles theirs, but our approach is quite different. The analysis in [12] assumes that the search tree is binary, and that the randomness in node evaluation outcomes is uniform across the entire tree; every visited node has probability  $p$  of being a successful partial solution. The search tree has an extremely uniform structure under these assumptions, allowing [12] to identify recurrence relations and closely approximate their solution. Renewal theory is employed in [14] to derive approximate expected bounds on memory requirements for general branch-and-bound methods.

All of the research mentioned above introduces randomness in a very uniform way. This is accepted practice and is usually required for analytic tractability. For the purposes of general behavioral description, uniform randomness is adequate, unless the model deviates strongly from known behavior. For backtracking, this is exactly the case. Intuition suggests that in general, the larger the size of the partial solution, the harder it will be to extend that solution. Yet under the assumption of uniform randomness, the likelihood of pruning a node and its subtree is the same anywhere in the search tree. Furthermore, the assumption of uniform randomness has led to surprising results. Both [12] and [11] derive bounds on the expected number of search nodes visited during a 1-solution search; these bounds are linear in the tree depth

regardless of the probability parameter  $p$ . Under their modeling assumptions, we must conclude that the average number of nodes visited by a 1-solution search is always linear in the depth of the search tree.

Uniform randomness does not accurately model our expectations about pruning behavior; furthermore, its assumption produces results whose strength is counter-intuitive. These observations have led us to consider a more general probabilistic model of backtracking's behavior. In doing so, we reaffirm much of the strength of the results derived in [12] and [11]. However, unlike the models in [12] and [11], our model allows super-linear complexity under the normally anticipated behavior of a backtracking search. By generalizing the probabilities which model searching behavior, we are thus better able to classify searches which will have linear expected performance. We will allow a general probability structure on a tree with depth  $D$ , whose nodes have random numbers of children. Defining natural notions of node expansion and contraction, we will show that if all nodes expand, or if all nodes contract, then there exist linear upper bounds on the number of nodes visited in a backtracking search for  $m$  solutions. These bounds are derived in terms of the probability structure, the number of solutions sought and the depth of the search tree. We then derive linear upper bounds for any search tree whose upper portion has all contracting nodes, and whose lower portion has all expanding nodes. We also derive linear upper bounds for trees having an expanding upper portion and contracting lower portion, where the degree of contraction is bounded in a depth independent way. If the degree of contraction cannot be bounded, we show that super-linear complexity can be expected.

**2. Model definition.** A backtracking search is a depth-first search, with the provision that it does not traverse any subtree rooted in a failed node. We model the decision to prune by supposing that every explored node has an *extension probability* of representing a successful partial solution. The extension probability for a leaf node is the probability that the leaf node represents a full solution. We also assume that a given visited node's success is independent of any other node's at the same depth. It is important to observe that this probability is conditioned on the event of the node being visited by the search. Given that a node is successful, we suppose that it has a random number of children, and that the mean number of children is  $n$  (this assumption is similar to one in [11]). We allow each node's distribution of children to be unique, but assume that each distribution is "new better than used in expectation," or NBUE [10]. A nonnegative random variable  $Y$  is NBUE if  $E[Y - a | Y > a] \leq E[Y]$  for all  $a \geq 0$ . Common examples of continuous NBUE distributions are hypo-exponentials, normals and certain classes of gamma and Weibull distributions. An appropriately constructed discrete approximation to a continuous NBUE random variable will be NBUE, and the degenerate constant random variable is NBUE. Treating  $Y$  as a life-time of some object,  $Y$  is *not* NBUE if a used object's expected remaining life-time after living  $a > 0$  time units is larger than its unconditional expectation  $E[X]$ ; mixtures of exponential random variables fall into this class [10]. We say that a node with extension probability  $p$  *expands* if  $p > 1/n$ , because given that the node is visited, the expected number of its children which are visited is

$$pn + (1 - p) \cdot 0 > 1.$$

We say that a search tree is expanding if each of its nodes expands. Likewise, if  $p \leq 1/n$ , we say that the node contracts; we say that the search tree contracts if every one of its nodes contracts.

Figures 1, 2 and 3 illustrate some of these ideas. Figure 1 depicts a small search tree, where each node is labeled with its extension probability. Figure 2 shows a

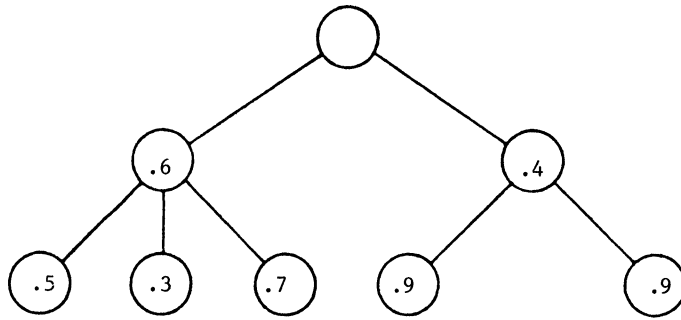


FIG. 1. Search tree with extension probabilities.

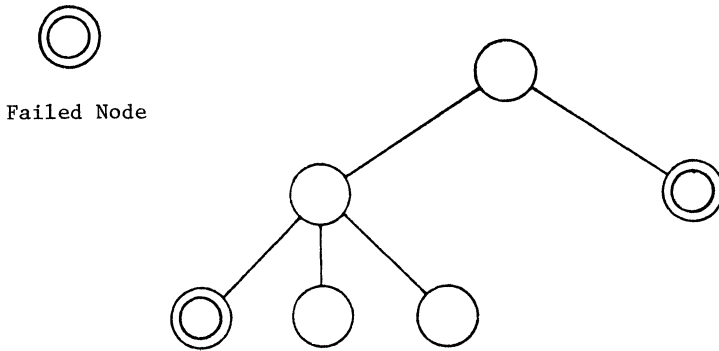


FIG. 2. Realization of full search, two solutions.

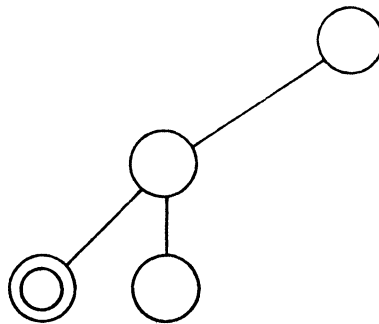


FIG. 3. Realization of 1-solution search.

possible realization of the nodes searched, where the entire left subtree is traversed, but the right child of the root fails, so that none of its children are visited. Assuming that  $n = 2$ , we see that the root's left child is expanding ( $0.6 > \frac{1}{2}$ ) while the root's right child is contracting ( $0.4 < \frac{1}{2}$ ). It is important to note that the definition of expanding and contracting nodes are in terms of the *mean* value  $n$ , not the number of children actually realized by a parent. Figure 3 depicts the nodes searched during a 1-solution search which discovers that the second leaf node visited is a successful full solution.

We will alternately consider two types of backtracking searches. A *full* search is one which terminates only after all solutions have been found. We let  $N$  denote the

random number of nodes visited during a full search, we also let  $N_j$  denote the random number of nodes visited at depth  $j$ . For many problems, we are satisfied with finding only one solution, and may substantially reduce the number of nodes visited by stopping with the first solution. We call a search which terminates with the first solution (or exhausts the search tree) a 1-solution search. Let  $T$  denote the random number of nodes visited during a 1-solution search, and let  $T_j$  denote the random number of nodes visited at depth  $j$  during a 1-solution search. Our approach also supports generalization of 1-solution searching, an  $m$ -solution search. An  $m$ -solution search might be used in a situation where it is too costly to search for all solutions, but a sizable sampling of solutions is desired. The random variables  $T(m)$  and  $T_j(m)$  are the immediate  $m$ -solution analogues to  $T$  and  $T_j$ . We will use  $E[N]$  and  $E[T(m)]$  as measures of backtracking's average complexity. As noted in [11] and [12], the average *time* complexity of searching may be larger, being dependent on the complexity of the evaluation of the necessary subconditions.

Our results are stated in terms of bounds on the maximal and minimal extension probabilities among all search tree nodes. We let  $p_j^{\max}$  denote the maximum extension probability among all nodes at depth  $j$ ; we similarly define the depth-dependent minimum  $p_j^{\min}$ . For every depth  $J$  (recall that the tree has depth  $D$ ) we define

$$p_{+J}^{\max} = \max_{1 \leq j \leq J} \{p_j^{\max}\}, \quad p_{+J}^{\min} = \min_{1 \leq j \leq J} \{p_j^{\min}\},$$

$$p_{J+}^{\max} = \max_{J < j \leq D} \{p_j^{\max}\}, \quad p_{J+}^{\min} = \min_{J < j \leq D} \{p_j^{\min}\},$$

and finally,  $p^{\min} = p_{+D}^{\min}$  and  $p^{\max} = p_{+D}^{\max}$ . Table 1 summarizes our model definitions.

Under our model assumptions, it is possible to find extension probabilities which lead to exponentially complex 1-solution searches. Suppose that the extension probabilities for all nodes at depths  $1, 2, \dots, D-1$  are equal to 1, and that the solution probabilities at depth  $D$  are all equal to 0. The 1-solution search will never find a solution, but will visit every node in the search tree. This degenerate case clearly has complexity which grows exponentially in the depth of the tree. While this example is not likely to represent any interesting problem particularly well, it does illustrate that the complexity of a 1-solution search can be high. By identifying classes of trees which yield good expected complexity, we are better able to identify classes of trees which do not.

TABLE 1  
*Model parameters.*

Notation	Definition
$D$	Depth of search tree
$N$	Number of nodes visited in full search
$m$	Number of solutions sought
$T(m)$	Number of nodes visited in $m$ -solution search
$n$	Mean number of children of successful node
$p_{+J}^{\max}$	Maximum extension probability at depths up to $J$
$p_{+J}^{\min}$	Minimum extension probability at depths up to $J$
$p_{J+}^{\max}$	Maximum extension probability at depths greater than $J$
$p_{J+}^{\min}$	Minimum extension probability at depths greater than $J$
$p^{\max}$	Maximum extension probability among all nodes
$p^{\min}$	Minimum extension probability among all nodes

**3. Summary of results.** Our analysis will consider four classes of search trees. We first treat the class of contracting trees ( $p^{\max} \leq 1/n$ ), and show by Lemma 1 that  $E[N] \leq nD$ . We then examine the class of expanding trees ( $p^{\min} > 1/n$ ), and give in Lemma 2 an upper bound on  $E[T(m)]$ . This bound is linear in  $D$ , in  $m$ , and in  $n$ . Together, Lemmas 1 and 2 address the model assumptions considered in [12]. While our bounds are not as tight as the results in [12], they are considerably more general. Lemmas 3, 4 and 5 treat trees with mixtures of expanding and contracting nodes. In Lemma 3 we give a bound which is marginally linear in  $D$ ,  $m$  and  $n$  for the situation where an upper portion (near the root) of the tree has contracting nodes, and the lower portion has expanding nodes. This indication of good performance is not surprising; contracting nodes near the root tend to prune large portions of the tree. These results serve as analytic vindication of the search rearrangement strategies studied in [1], [3] and [13] (a formal analysis of average behavior under this strategy for the CNF satisfiability problem is reported in [9]). Search rearrangement strives to reorder the sequence of variable assignments so as to increase the likelihood of pruning at shallow levels of the search tree. Finally, Lemmas 4 and 5 treat intuitively appealing situations where it is relatively easier to extend small partial solutions than it is to extend larger partial solutions. We can model this phenomenon with search trees having an expanding upper portion and contracting lower portion. When the extent and degree of contraction is bounded independently of the tree depth, the bound on  $E[T(m)]$  is marginally linear in  $D$ , polynomial in  $n$  and exponential in the extent of contraction. Thus we see that our general model allows linear performance in a situation we expect to encounter in practice, but which is not well described by previous average performance analysis. Furthermore, the constraints required to achieve this linearity are a guide towards identifying trees which do not have linear expected performance. We discuss that issue further in § 7.

Our results are stated below. Derivations are given in following sections.

LEMMA 1 (Contracting Trees). *If  $p^{\max} \leq 1/n$ , then  $E[N] < nD$ .  $\square$*

LEMMA 2 (Expanding Trees). *If there is an  $\varepsilon > 1$  such that  $p^{\min} \geq \varepsilon/n$ , then*

$$E[T(m)] < \frac{n(D+m-1)}{\varepsilon-1}. \quad \square$$

LEMMA 3 (Contracting/Expanding Trees). *If there is an  $I \geq 1$  and  $\varepsilon > 1$  such that  $p_{+I}^{\max} \leq 1/n$  and  $p_{I+}^{\min} \geq \varepsilon/n$ , then*

$$E[T(m)] < In + \frac{n(D-I+m-1)}{\varepsilon-1}. \quad \square$$

LEMMA 4 (Expanding/Contracting Trees). *If there is a  $\delta > 1$  and  $J \geq 1$  such that  $p_{+(D-J)}^{\max} \leq \delta/n$  and  $p_{(D-J)+}^{\max} \leq 1/n$ , then*

$$E[N] \leq n \left( \frac{\delta^{D-J}-1}{\delta-1} \right) + nJ\delta^{D-J-1}. \quad \square$$

LEMMA 5 (Expanding/Contracting Trees). *If there are  $\varepsilon > 1$ , and  $J, C \geq 1$  such that  $p_{+(D-J)}^{\min} \geq \varepsilon/n$  and  $1/Cn \leq p_{(D-J)+}^{\min} \leq 1/n$ , then*

$$E[T(m)] < \frac{m(Cn)^{J+1} + n(D-J-1)}{\varepsilon-1} + m(Cn)^{J+1}nJ. \quad \square$$

**4. Contracting trees (Lemma 1).** We first derive an upper bound on  $E[N]$ , the expected number of nodes visited during a full search. Recalling that  $N_j$  is the random

number of nodes visited at depth  $j$  during a full search, note that

$$E[N] = \sum_{j=1}^D E[N_j].$$

In order to calculate  $E[N_j]$ , observe that if  $N_{j-1} = s$  for a given full search, then

$$(1) \quad E[N_j | N_{j-1} = s] \leq sp^{\max} n.$$

This follows because every visited node has an extension probability less than  $p^{\max}$ , and a successful node has  $n$  children on the average. Taking the expectation with respect to  $N_{j-1}$ ,

$$E[N_j] \leq E[N_{j-1}]p^{\max} n.$$

In a full search, every node at depth 1 is visited, so  $E[N_1] = n$ . From inequality (1),

$$E[N_2] \leq n^2 p^{\max},$$

and in general,

$$(2) \quad E[N_j] \leq n^j (p^{\max})^{j-1}.$$

The expected number of nodes visited during a full search is thus bounded from above by

$$E[N] \leq \sum_{j=1}^D n^j (p^{\max})^{j-1}.$$

Recalling the identity [8]

$$\sum_{j=0}^k r^j = \frac{1-r^{k+1}}{1-r} \quad \text{for } r \neq 1,$$

we see that

$$(3) \quad E[N] \leq \begin{cases} nD & \text{if } np^{\max} = 1, \\ n \frac{1-(np^{\max})^D}{1-np^{\max}} & \text{otherwise.} \end{cases}$$

It is also clear from (2) that our bound is an increasing function of  $p^{\max}$ .

A contracting search tree is characterized by  $p^{\max} \leq 1/n$ . Thus Lemma 1 follows from (3) above.

LEMMA 1. *If  $p^{\max} \leq 1/n$ , then*

$$E[N] \leq nD. \quad \square$$

When the extension probabilities are all small enough, the total number of nodes visited in a full search (and hence any search) is bounded from above by  $nD$ . This bound is marginally linear in  $n$  and in  $D$ . The key reason for this linearity is the high probability of large subtrees being pruned as a result of extension failures at shallow depths.

**5. Expanding trees (Lemma 2).** We next consider an expanding search tree. Inequality (3) provides an exponentially large upper bound on  $E[N]$  when  $np^{\max} > 1$ ;

furthermore, this bound is tight if all extension probabilities are identical. However, the number of nodes visited in an  $m$ -solution search may be substantially smaller than that of a full search. Under the assumption at  $p^{\min} \geq \varepsilon/n$  for some  $\varepsilon > 1$ , we will next show that the number of nodes visited by an  $m$ -solution search is bounded above by a function which is marginally linear in  $D$ , in  $n$  and in  $m$ .

The tactic we adopt in deriving this bound is to embed the search tree into an infinitely wide tree, and then analyze an  $m$ -solution search on the infinite tree from the bottom up. An  $m$ -solution search on an infinite tree with expanding nodes will always find  $m$  solutions; by analyzing a search there we avoid having to condition on whether or not exactly  $m$ -solutions are discovered in the original search tree. The bottom up analysis is supported by two basic observations. One is that we can bound  $E[T_D(m)]$  without worrying about the behavior of the search at depths other than  $D$ . The second observation is that in an  $m$ -solution search of the infinite tree, the knowledge of how many nodes have been visited at depth  $j+1$  allows us to bound the number of nodes which have been visited at depth  $j$ . The bound on  $E[T_D(m)]$  allows us to quantify the bound on  $E[T_{D-1}(m)]$ ; repeating this back-substitution we can quantify  $E[T_j(m)]$  for each  $j$ , and sum these bounds to bound  $E[T(m)]$ .

We embed an expanding search tree into an infinite tree as follows. If the search tree root has  $K$  children, we may view the tree as a collection of  $K$  major subtrees  $B_1, B_2, \dots, B_K$ , each being rooted in a child of the search tree root. Now consider a tree with depth  $D$  consisting of a root and a countably infinite number of major subtrees, each with depth  $D-1$ . We assume that for  $j=1, 2, \dots, K$ , the  $j$ th major subtree in the infinite search tree is probabilistically identical to the  $j$ th major subtree in the finite search: their corresponding nodes have identical extension probabilities. Subtrees  $K+1, K+2, \dots$ , in the infinite tree are taken to be arbitrary expanding subtrees. The infinite tree so constructed does not correspond to an expansion of a given problem; rather, we use its probabilistic properties as a convenient tool. We can couple the behavior of a search on the original tree with a search on the infinite tree by requiring the evaluation outcomes in the original tree to be identical to the infinite tree's outcomes in its first  $K$  subtrees. Thus the tree expansions in the original tree and in the first  $K$  major subtrees of the infinite tree are taken to be identical. If exactly  $m$  solutions are discovered in the original tree then exactly  $m$  "solutions" are discovered in the first  $K$  subtrees of the infinite tree; if fewer than  $m$  solutions are discovered in the original tree, its search will terminate while the search on the infinite tree continues on major subtrees  $K+1, K+2, \dots$ , until  $m$  "solutions" are found. From this definition it is clear that for every search, at every depth  $j$ , the number of nodes visited in the infinite tree at depth  $j$  is an upper bound on the number of nodes visited by the search in the finite tree. We will therefore bound  $E[T(m)]$  by bounding the expected number of nodes visited in an  $m$ -solution search of the infinite tree. At the risk of abusing our notation, we will now let  $T_j(m)$  denote the number of nodes visited at depth  $j$  in an  $m$ -solution search of the infinite tree;  $T(m)$  is similarly redefined. Any upper bounds we derive on the infinite tree apply equally well to the finite problem tree. Our notational modification is understood to apply only to this section. For the remainder of this section, we will refer only to searches of the infinite tree; consideration of this tree is also limited to this section.

We initially consider the behavior of an  $m$ -solution search at the tree's deepest level  $D$ . Any time a leaf node is visited it is found to be a solution with probability no less than  $p^{\min}$ , it is otherwise found to fail. Because of the independence in leaf node evaluation outcomes, we can view the search behavior at depth  $D$  as a sequence of independent (but not necessarily identical) Bernoulli trials [10], each with success

probability greater than or equal to  $p^{\min}$ . If the success probabilities at each node visit were identically  $p^{\min}$ , the number of visits to depth  $D$  required to find a solution would be a geometric random variable with mean  $1/p^{\min}$ . Because each visit to depth  $D$  has probability of success at least as large as  $p^{\min}$ ,  $1/p^{\min}$  is an upper bound on the mean number of visits required to find one solution;  $m/p^{\min}$  bounds the mean number of visits required to find  $m$  solutions. Recalling that  $p^{\min} \geq \varepsilon/n$  by definition of an expanding tree, we have

$$(4) \quad E[T_D(m)] \leq \frac{m}{p^{\min}} \leq \frac{mn}{\varepsilon}.$$

We next derive upper bounds on  $E[T_j(m)]$  for each  $j$ , bounding  $E[T_j(m)]$  in terms of  $E[T_{j+1}(m)]$ . Following that, we will use (4) to quantify these upper bounds, and sum the bounds over all depths to bound  $E[T(m)]$ . Our discussion is facilitated by another definition. Let  $S_j(m)$  be the number of *successful* nodes visited in an  $m$ -solution search. We will first bound  $E[T_j(m)]$  in terms of  $E[S_j(m)]$ . Note that the last node visited by an  $m$ -solution search is always a successful leaf node, implying that the last node visited at depth  $j$  by a search is successful (being an ancestor of the last solution). Given that  $S_j(m) = k$ , it follows that  $E[T_j(m)] \leq k/p^{\min}$ , since the mean number of nodes at depth  $j$  visited between successful visits at that depth is no greater than  $1/p^{\min}$ . It follows immediately that

$$(5) \quad E[T_j(m)] \leq \frac{E[S_j(m)]}{p^{\min}}.$$

We now bound  $E[S_j(m)]$  from above in terms of  $E[T_{j+1}(m)]$ . Call a node *activated* if its parent has been explored and is successful. Note that nodes are activated in groups, the group size being the number of children spawned by the parent. Furthermore, group sizes are independent random variables, each with mean  $n$ . If we focus our attention during a search on the activity at depth  $j+1$ , we will see a succession of groups activated, with group sizes being random variables  $\{Y_k\}$ . From this viewpoint, the search behavior at depth  $j+1$  is very much like a renewal process [10], except that inter-renewal times need not have identical distributions. We call this stochastic process a *quasi-renewal* process. “Time” in this process is the number of nodes visited by the search at depth  $j+1$ , minus 1 (so that time begins at 0); a quasi-renewal occurs at “time”  $t$  if node  $t$  (the  $t+1$ st node visited) is the first node in its group to be visited (excluding node 0). Now define  $S_j(m, t)$  to be the number of successful nodes visited at depth  $j$  by “time”  $t$ . As the search progresses, at any given “time”  $t$  the number of quasi-renewals which have occurred is exactly the number of parents of depth  $j+1$  nodes already visited, minus 1:  $S_j(m, t) - 1$ . The renewal function  $R(t)$  studied by renewal theory is the expected number of renewals which occur by time  $t$ . Thus for the problem at hand,  $R(t) = E[S_j(m, t)] - 1$ . In [10, p. 275] it is shown that the renewal function for a quasi-renewal process with NBUE inter-quasi-renewal times each having mean  $n$  is bounded from above by the renewal function  $R(t) = t/n$  of a Poisson process with mean inter-renewal time  $n$ . Since the  $\{Y_k\}$ ’s are NBUE by assumption, it follows that

$$E[S_j(m, t)] \leq \frac{t}{n} + 1.$$

“Time” stops at value  $T_{j+1}(m)$ , when the search terminates. The inequality above then



implies that

$$E[S_j(m)] \leq \frac{E[T_{j+1}(m)]}{n} + 1.$$

By substituting this bound into inequality (5) and noting that  $p^{\min} \geq \varepsilon/n$ , we find that

$$(6) \quad E[T_j(m)] \leq \frac{E[T_{j+1}(m)]}{\varepsilon} + \frac{n}{\varepsilon},$$

which establishes our goal of a bound on  $E[T_j(m)]$  in terms of  $E[T_{j+1}(m)]$ .

Inequality (6) gives us a sequence of bounds as we vary  $j$  from  $D-1$  to 1. We quantify these inequalities by employing the bound on  $E[T_D(m)]$  given by (4), and then repeatedly apply inequality (6) to obtain bounds on  $E[T_j(m)]$  for  $j = D-1, D-2, \dots, 1$ . We know that

$$E[T_D(m)] \leq \frac{nm}{\varepsilon}.$$

Substituting this bound for  $E[T_D(m)]$  into inequality (6) we obtain

$$E[T_{D-1}(m)] \leq \frac{nm}{\varepsilon^2} + \frac{n}{\varepsilon}.$$

Repeating this process, we find that for  $0 \leq k \leq D-1$

$$(7) \quad \begin{aligned} E[T_{D-k}(m)] &\leq \frac{nm}{\varepsilon^{k+1}} + n \sum_{j=1}^k \left( \frac{1}{\varepsilon^j} \right) \\ &= \frac{nm}{\varepsilon^{k+1}} + \frac{n(1-1/\varepsilon^k)}{\varepsilon-1}. \end{aligned}$$

We can then bound  $E[T(m)]$  by summing the upper bounds of the different depths given by (7):

$$\begin{aligned} E[T(m)] &\leq \sum_{k=0}^{D-1} \left( \frac{nm}{\varepsilon^{k+1}} + \frac{n(1-1/\varepsilon^k)}{\varepsilon-1} \right) \\ &= \frac{1}{\varepsilon-1} \left[ nm \left( 1 - \frac{1}{\varepsilon^D} \right) + n(D-1) - n \left( \frac{\varepsilon}{\varepsilon-1} \right) \left( 1 - \frac{1}{\varepsilon^D} \right) \right]. \end{aligned}$$

A less strict, but less involved bound follows immediately from the fact that  $\varepsilon > 1$ .

LEMMA 2. *If there is an  $\varepsilon > 1$  such that  $p^{\min} \geq \varepsilon/n$ , then*

$$(8) \quad E[T(m)] < \frac{n(D+m-1)}{\varepsilon-1}. \quad \square$$

This bound is marginally linear in  $n$ ,  $D$  and  $m$ . Furthermore, its change with respect to  $m$  is interesting. Doubling  $m$  increases the bound by less than a factor of two. This supports our intuitive understanding that after one solution is found, the additional cost of finding a second solution is (in expectation) less. The bound given in (8) is very sensitive to  $\varepsilon$  when  $\varepsilon$  is close to 1. Yet for  $\varepsilon \geq 1.1$ ,  $E[T(m)]$  is no greater than  $10n(D+m-1)$ , which seems quite reasonable.

**6. Contracting/expanding trees (Lemma 3).** Lemmas 1 and 2 give us means of bounding the expected performance of an  $m$ -solution search on contracting and on expanding trees. We next employ these bounds to derive bounds for trees which

contract at shallow depths, and expand at deep depths. As expected, we obtain these bounds by considering the expanding and contracting regions separately, and combine the results for an overall bound. We derive a bound on  $E[T(m)]$  which is linear in  $n$ , in  $D$  and in  $m$ .

We suppose there is a depth  $I$  such that  $p_{+I}^{\max} \leq 1/n$ ; and  $p_{I+}^{\min} \geq \varepsilon/n$  for some  $\varepsilon > 1$ . Consider the search tree truncated at depth  $I$ ; Lemma 1 states that the expected number of nodes visited by a full search of the truncated tree is no greater than  $nI$ , and (2) shows that the expected number of depth  $I$  nodes visited by a full search is bounded by  $n$ . It follows that the expected number of successful depth  $I$  nodes found in a full search, and hence an  $m$ -solution search, is no greater than 1. We can bound the expected number of nodes visited by an  $m$ -solution search at depths  $I+1, \dots, D$  by considering the subtrees rooted in successful nodes at depth  $I$ . Each such subtree has depth  $D-I$ , and extension probabilities (excluding the subtree root) which all exceed  $1/n$ . Now consider a *total subtree search* of depths  $I+1, \dots, D$ , which conducts an  $m$ -solution search of every subtree rooted in a successful node at depth  $I$ . Clearly the expected number of nodes visited by such a search is greater than the expected number of visited nodes in a usual  $m$ -solution search: the usual  $m$ -solution search will stop with the first  $m$  solutions, while the total subtree search will continue on to explore all subtrees with successful roots at depth  $I$ . The expected number of successful nodes at depth  $I$  is no greater than 1, so that the expected number of nodes at depths greater than  $I$  visited by a total subtree search is bounded from above by

$$\frac{n(D-I+m-1)}{\varepsilon-1}.$$

This bound follows from the application of Lemma 2 to one subtree rooted in depth  $I$ . We have already bounded the number of nodes visited at depths  $1, \dots, I$  by  $nI$ . The expected number of nodes visited in an  $m$ -solution search is therefore bounded as described in Lemma 3.

LEMMA 3. *If there is an  $I \geq 1$  and  $\varepsilon > 1$  such that  $p_{+I}^{\max} \leq 1/n$  and  $p_{I+}^{\min} \geq \varepsilon/n$ , then*

$$(9) \quad E[T(m)] \leq nI + \frac{n(D-I+m-1)}{\varepsilon-1}. \quad \square$$

This bound has the same marginal properties in  $n$ ,  $D$  and  $m$  as does the bound given by Lemma 2.

**7. Expanding/contracting trees (Lemmas 4 and 5).** We earlier gave an example of a tree which expanded in all but the last depth, and then contracted completely at the leaf nodes. This tree has the worst possible performance because it searches all nodes, but finds no solutions. This tree falls in the class of trees we next consider, trees which expand in upper levels and contract in lower levels. However, we are able to show that if the degree and extent of contraction is bounded, then we achieve marginal linear performance in the depth of the tree. Our bound is more sensitive to  $n$ : it increases as a polynomial in  $n$  and in the degree of contraction. This bound is exponential in the extent of the contracting region.

We suppose there is a depth  $D-J$  such that  $p_{+(D-J)}^{\min} \geq \varepsilon/n$  for some  $\varepsilon > 1$  and  $1/n \geq p_{(D-J)+}^{\max} \geq 1/Cn$  for some  $C \geq 1$ . The parameter  $J$  describes the extent of the contracting region ( $J$  depth levels), and  $C$  describes the degree of the contraction. We derive two bounds for this case. Our bound on  $E[N]$  is useful when the extension probabilities are small or when the expanding portion of the tree is limited; the other bound is useful if the contracting region is small.

To bound  $E[N]$ , we make the additional assumption that  $p^{\max} \leq \delta/n$ . By (3), the expected number of nodes visited at depths  $1, \dots, D-J$  by a full search is bounded from above by

$$n \left( \frac{\delta^{D-J} - 1}{\delta - 1} \right).$$

As a consequence of (2), the expected number of successful nodes at depth  $D-J$  is bounded from above by  $\delta^{D-J-1}$ . By Lemma 1, the expected number of nodes visited in a subtree rooted in a successful depth  $D-J$  node is bounded from above by  $nJ$ . Lemma 4 combines these observations.

LEMMA 4. *If there is a  $\delta > 1$  and  $J \geq 1$  such that  $p_{+(D-J)}^{\max} \leq \delta/n$  and  $p_{(D-J)+}^{\max} \leq 1/n$ , then*

$$(10) \quad E[N] \leq n \left( \frac{\delta^{D-J} - 1}{\delta - 1} \right) + nJ\delta^{D-J-1}. \quad \square$$

The key parameters in this inequality are  $\delta$  and  $J$ . When  $\delta$  is small, say  $\delta < 2$ , the exponential growth of (10) in  $D-J$  is slow. If  $D-J$  is small, then the troublesome expanding portion of the tree is limited, and (10) may yield a reasonably small bound. If neither of these cases is satisfied, we should consider another bound on  $E[T(m)]$ , which we derive next.

The assumption that  $p_{(D-J)+}^{\min} \geq 1/Cn$  bounds the degree to which the search tree contracts. We can use this restriction to bound  $E[T(m)]$ . The first step is to look at a subtree rooted in a visited depth  $D-J$  node and find a lower bound on the probability  $p_S$  of finding a solution in that subtree. One potential path to discovering a solution in this subtree occurs if the first  $J+1$  successive visits in a depth-first search of the subtree (we include the root here) each discover successful nodes. The last successful node is the full solution. The probability of this occurrence is at least

$$p_L = \frac{1}{(Cn)^{J+1}} \leq p_S.$$

This bound holds for every subtree (and hence every probability of finding a solution in a subtree). We will say that a depth  $D-J$  node is *ultimately successful*, or *u-successful*, if it is an ancestor of a successful complete solution.  $p_L$  is a lower bound on the probability of a visited depth  $D-J$  node being *u-successful*. The behavior of a 1-solution search at depths  $1, 2, \dots, D-J$  is probabilistically identical to the behavior of a 1-solution search on a modified tree having depth  $D-J$  and the same extension probabilities as the original search tree, except that depth  $D-J$  nodes' extension probabilities are replaced by their *u-success* probabilities. Since a subtree rooted in depth  $D-J$  may contain multiple solutions, an  $m$ -solution search of the original tree will visit fewer nodes at depths  $1, \dots, D-J$  than will an  $m$ -solution search of the modified tree. The arguments presented in § 5 establish that the mean number of nodes visited at depth  $D-J$  by an  $m$ -solution search of the modified tree is bounded from above by  $m/p_L = m(Cn)^{J+1}$ . Then the same type of analysis which leads to equation (8) shows that

$$(11) \quad \sum_{j=1}^{D-J} E[T_j(m)] < \frac{m(Cn)^{J+1} + n(D-J-1)}{\varepsilon - 1}.$$

Reconsidering the original depth  $D$  tree, we use Lemma 1 to see that the expected number of nodes visited by a total search of a subtree rooted in depth  $D-J$  is bounded

from above by  $nJ$ .  $m(Cn)^{J+1}nJ$  is thus an upper bound on the expected number of nodes visited at depths  $D-J+1, \dots, D$  in an  $m$ -solution search of the original tree. Summing this bound with the one given by (11), we have the following lemma.

LEMMA 5. *If there are  $\varepsilon > 1$ , and  $J, C \geq 1$  such that  $p_{+(D-J)}^{\max} \geq \varepsilon/n$  and  $1/Cn \leq p_{(D-J)+}^{\min} \leq 1/n$ , then*

$$E[T(m)] < \frac{m(Cn)^{J+1} + n(D-J-1)}{\varepsilon - 1} + m(Cn)^{J+1}nJ. \quad \square$$

This bound is reasonably small only if  $J$  is small. We can expect marginal linear performance in the depth of the tree, provided there are depth independent bounds on the number of tree levels which contract and the degree to which nodes in that region contract. Lemma 5 also hints at the relative sensitivity of performance to changes in the different model parameters. The bound grows as a polynomial in  $n$  and in  $C$ ; it grows exponentially in  $J$ .

Reconsider the example of the tree which visited all nodes and found no solutions. If we slightly change the example so that a leaf node has some fixed probability  $p > 0$  of being a solution, then Lemma 5 gives us marginal linear performance in the depth of the tree (supporting empirical data presented in [12].) This is more of an indictment of asymptotic results than it is an indication of good performance. If  $J$  is fixed and quite large, Lemma 5 can still give us linear performance in  $D$ . That does not mean that we can expect to find a solution quickly.

The example highlighting exponential complexity hinged on having extension probabilities equal to zero. There exist trees with all nonzero extension probabilities which have exponential performance asymptotically. It is not difficult to see (or prove) that as the depth of a contracting tree increases, the probability of discovering a solution in that tree approaches zero. Now consider a tree which expands in the upper half of its levels and contracts in the lower half. In the limit of increasing tree depth, a full search of the tree's upper half occurs because the search is always pruned in the lower half before reaching a solution. Since the upper half is expanding, the number of nodes visited there increases exponentially in the tree depth. Clearly the dividing line between expanding and contracting tree regions need not be at  $D/2$  to cause this type of asymptotic behavior; we can expect asymptotic exponential complexity in the general case of such a tree. This example re-illustrates the need for extending past work in average performance analysis of backtracking, particularly as empirical data suggests that common problems exhibit expanding/contracting search trees [13]. Using uniform probability structures, previous performance models conclude that all 1-solution searches have expected linear complexity. The example above shows that under a general model, there are 1-solution searches with super-linear complexity. Our general model is more selective in its identification of trees leading to low complexity and specifically excludes the example outlined above from the class of searches with linear complexity.

**8. Summary.** Previous analysis of the average case performance of 1-solution backtracking searches have assumed uniform probability structures and have subsequently shown that 1-solution backtracking always has complexity linear in the depth of the search tree. However, using a more general probability model, it is possible to construct 1-solution backtracking searches which have expected super-linear complexity. This paper investigates the implications of assuming a general probability structure for the average case analysis. Using the notions of "expanding" and "contracting" nodes, we have derived expected linear complexity bounds on  $m$ -solution searches

of trees with all nodes contracting and trees with all nodes expanding. We present linear bounds on trees which have an upper portion contracting with the lower portion expanding. We also derive linear bounds on special cases of trees with an expanding upper portion and a contracting lower portion and show that this latter class generally yields super-linear complexity. By extending previous work in the average complexity of backtracking, we have both provided further assurance of good expected performance under appropriate conditions and indirectly shown where classes of searches with high complexity lie.

**Acknowledgments.** Use of the terms “expanding” and “contracting” is due to a discussion with Harold Stone. I also thank Shahid Bokhari and Bob Voigt for suggestions regarding the presentation of this paper.

## REFERENCES

- [1] J. R. BITNER AND E. M. REINGOLD, *Backtrack programming techniques*, Comm. ACM, 18 (1975), pp. 651–655.
- [2] C. A. BROWN AND P. W. PURDOM, *An average time analysis of backtracking*, this Journal 10 (1981), pp. 583–593.
- [3] ———, *An empirical comparison of backtracking algorithms*, in Proc. IEEE Trans. Pattern Analysis and Machine Intelligence, PAMI-4, 3 (1982), pp. 309–316.
- [4] T. HARRIS, *The Theory of Branching Processes*, Springer, Berlin, 1963.
- [5] E. HOROWITZ AND S. SAHNI, *Fundamentals of Computer Algorithms*, Computer Science Press, Rockville, MD, 1978.
- [6] R. M. KARP, *The probabilistic analysis of some combinatorial search algorithms* in Algorithms and Complexity, J. F. Traub, ed., Academic Press, New York, 1976, pp. 1–19.
- [7] R. M. KARP AND J. PEARL, *Searching for an optimal path in a tree with random costs*, Artificial Intelligence, 21 (1983), pp. 99–117.
- [8] D. E. KNUTH, *The Art of Computer Programming, Vol. 1*, Addison-Wesley, Menlo Park, CA, 1973.
- [9] P. W. PURDOM, JR., *Search rearrangement backtracking and polynomial average time*, Artificial Intelligence, 21 (1983), pp. 117–133.
- [10] S. ROSS, *Stochastic Processes*, John Wiley, New York, 1983.
- [11] D. R. SMITH, *Random trees and the analysis of branch and bound procedures*, J. ACM, 31 (1984), pp. 163–188.
- [12] H. S. STONE AND P. SIPALA, *The average complexity of depth-first search with backtracking and cutoff*, IBM J. Res. and Develop., 30 (1986), pp. 242–258.
- [13] H. S. STONE AND J. M. STONE, *Efficient search techniques—an empirical study of the N-Queens problem*, IBM J. Res. Develop., to appear.
- [14] B. W. WAH AND C. F. YU, *Stochastic modeling of branch-and-bound with best-first search*, IEEE Trans. Software Engrg., SE-11, (1985), pp. 922–934.

## APPROXIMATE PARALLEL SCHEDULING. PART I: THE BASIC TECHNIQUE WITH APPLICATIONS TO OPTIMAL PARALLEL LIST RANKING IN LOGARITHMIC TIME\*

RICHARD COLE† AND UZI VISHKIN‡

**Abstract.** We define a novel scheduling problem; it is solved in parallel by repeated, rapid, approximate reschedulings. This leads to the first optimal logarithmic time PRAM algorithm for list ranking. Companion papers show how to apply these results to obtain improved PRAM upper bounds for a variety of problems on graphs, including the following: connectivity, biconnectivity, Euler tour and  $st$ -numbering, and a number of problems on trees.

**Key words.** PRAM algorithm, list ranking, approximate scheduling, expander graphs, optimal speed-up

**AMS(MOS) subject classifications.** 68C05, 68C25

**1. Introduction.** The model of parallel computation used in this paper is a member of the parallel random access machine (PRAM) family. A PRAM employs  $p$  synchronous processors all having access to a common memory. In this paper we use an exclusive-read exclusive-write (EREW) PRAM. The EREW PRAM does not allow simultaneous access by more than one processor to the same memory location for read or write purposes. See [Vi-83] for a survey of results concerning PRAMs.

Let  $\text{Seq}(n)$  be the fastest known worst-case running time of a sequential algorithm, where  $n$  is the length of the input for the problem at hand. Obviously, the best upper bound on the parallel time achievable using  $p$  processors, without improving the sequential result, is of the form  $O(\text{Seq}(n)/p)$ . A parallel algorithm that achieves this running time is said to have *optimal speed-up* or more simply to be *optimal*. A primary goal in parallel computation is to design optimal algorithms that also run as fast as possible. Some authors use optimal to mean that as well as achieving an  $O(\text{Seq}(n))$  processor time product, the parallel running time is as small as possible (typically  $O(\log n)$ ).

Most of the problems we consider can be solved by parallel algorithms that obey the following framework. Given an input of size  $n$  the parallel algorithm employs a *reducing* procedure to produce a smaller instance of the same problem (of size  $\leq n/2$ , say). The smaller problem is solved recursively until this brings us below some threshold for the size of the problem. An alternative procedure is then used to complete the parallel algorithm. We refer the reader to [CV-86d] where this algorithmic technique, which is called accelerating cascades, is discussed. Typically, we need to reschedule

---

\* Received by the editors October 16, 1986; accepted for publication (in revised form) May 4, 1987. A preliminary version of this paper appeared in the Twenty-seventh Annual Symposium on Foundations of Computer Science, Toronto, Canada, October 27-29, 1986. Copyright 1986, Institute for Electrical and Electronics Engineers (Inc.).

† Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. The research of this author was supported in part by National Science Foundation grants DCR-84-01633 and CCR-8702271, Office of Naval Research grant N00014-85-K-0046 and by an IBM faculty development award.

‡ Courant Institute of Mathematical Sciences, New York University, New York, New York 10012 and Tel Aviv University, Tel Aviv, Israel. The work of this author was supported in part by National Science Foundation grants NSF-DCR-8318874 and NSF-DCR-8413359, Office of Naval Research grant N00014-85-K-0046 and by the Applied Mathematical Science subprogram of the office of Energy Research, U.S. Department of Energy under contract DE-AC02-76ER03077 and by the Foundation for Research in Electronics, Computers and Communication, administered by the Israeli Academy of Sciences and Humanities.

the processors in order to apply the reducing procedure efficiently to the smaller sized problem. Suppose the input for a problem of size  $n$  is given in an array of size  $n$ . A natural approach is to compress the smaller problem instance into a smaller array, of size  $\leq n/2$ . This is often done using a prefix sum algorithm (it takes  $O(\log n)$  time on  $n/\log n$  processors to compute the prefix sums for  $n$  inputs stored in an array). Thus if we need to reschedule the processors repeatedly it is unclear how to achieve logarithmic time. Sometimes the rescheduling need not be performed very often: In [CV-86a], [C-86] the authors show that for some problems (list ranking and selection)  $\log^* n$  reschedulings suffice. Alternatively, one can use a fast random algorithm to perform the rescheduling, or at least an approximate rescheduling. (By approximate rescheduling we mean that we may not be able to partition the work evenly among the processors, but only approximately evenly.) Thus the need for rescheduling does not preclude  $O(\log n)$  time optimal random algorithms. One of the main contributions of this paper is to provide an algorithm for performing approximate rescheduling deterministically in  $O(1)$  time. This is used to solve a novel scheduling problem, called the *duration-unknown task scheduling* problem. The solution to the scheduling problem leads to a logarithmic time optimal deterministic parallel algorithm for list ranking. A related rescheduling procedure is one of the tools that leads to a logarithmic time connectivity algorithm which is optimal unless the graph is extremely sparse.

The *duration-unknown task scheduling* problem is defined as follows.  $n$  tasks are given, each of length between 1 and  $e \log n$ ,  $e$  a constant; the total length of the tasks is bounded by  $cn$ ,  $c$  a constant. (A task can be thought of as a program.) However, we do not know, in advance, the lengths of the individual tasks; in fact, they may vary, depending on the order of execution of the tasks. The problem is to schedule the  $n$  tasks on an EREW PRAM of  $n/\log n$  processors so that the tasks are completed in  $O(\log n)$  time; it is solved in § 3.

We now discuss how to design algorithms that take advantage of this task scheduling algorithm. Given a problem, our job is to design a “protocol” for solving the problem by using a set of short tasks (each of length between 1 and  $e \log n$ ). This provides an important new opportunity for the designer of a protocol which is based on using the scheduling algorithm: the designer of the protocol need not know anything about the order of execution of the tasks. Such an opportunity for designing parallel tasks, without knowing in advance their lengths, with the guarantee that they will be scheduled efficiently, sounds very promising. However, this opportunity cannot be separated from a considerable difficulty in designing such a protocol: we have no control over the order of execution of the tasks, so we must ensure that the protocol works correctly regardless of the order of execution. We note that this style of protocol design may be useful for distributed systems that are not tightly synchronized; here too, we have to be sure that the protocol works correctly regardless of the order of execution. Section 4 demonstrates how to design such a protocol for the following problem.

#### LIST RANKING.

*Input.* A linked list of length  $n$ . It is given in an array of length  $n$ , not necessarily in the order of the linked list. Each of the  $n$  elements (except the last element in the linked list) has the array index of its successor in the linked list.

*The problem.* For each element, compute the number of elements following it in the linked list.

*Result.* On the EREW PRAM,  $O(\log n)$  time using  $n/\log n$  processors.

*Previous results.* On the EREW PRAM,

(i)  $O(n/p \cdot \log n/\log n/p)$  time using  $p < n$  processors [KRS-85] (this was the first optimal algorithm to achieve sublinear time).

- (ii)  $O(\log n \log^* n)$  time using  $n/(\log n \log^* n)$  processors [CV-86a].
- (iii) for fixed  $k$ ,  $O(k \log n)$  time using  $n \log^{(k)} n / \log n$  processors [CV-86a].

Our new result is asymptotically better than (ii). However, as it involves considerably larger constants, for all practical purposes the older result seems stronger.

The following randomized algorithms are also known.

(i)  $O(\log n \log^* n)$  time using  $n/(\log n \log^* n)$  processors [Vi-84]. The journal version of this paper uses the new parallel prefix sums algorithm of [CV-86e] to achieve  $O(\log n)$  time using  $O(n/\log n)$  processors.

(ii)  $O(\log n)$  time using  $O(n/\log n)$  processors is implicitly achieved by [MR-85].

*Comment.* The list ranking problem is often encountered as a subproblem in other parallel algorithms. The Euler tour technique on trees [TV-85], [Vi-85] reduces a variety of tree functions into list ranking. [TV-85] used list ranking to reduce graph biconnectivity to graph connectivity. List ranking plays a central role in the “accelerated centroid decomposition (ACD)” parallel method for evaluating tree expressions [CV-86b]. Recently, in [SV-86] Schieber and Vishkin have shown how to apply the new list ranking algorithm for responding to queries regarding lowest common ancestors of pairs of vertices on a tree.

Part 2 of this research (the paper [CV-86c]) shows how to apply the approximate scheduling method together with the new list ranking algorithm in order to derive improved PRAM upper bounds for a variety of problems on graphs, including the following: connectivity, biconnectivity, Euler tour and  $st$ -numbering.

As can be seen, the results presented here improve on previous work in [CV-86a]. The main contributions of [CV-86a] were the deterministic coin tossing technique and a methodology for scheduling that used as few reschedulings as possible. While the details of their reschedulings were quite intricate, the rescheduling procedure itself was standard. We make two main contributions here. First, we provide a new approach to the rescheduling problem. Second, we show how to solve the list ranking problem in the novel framework imposed by our solution to the rescheduling problem. In addition, the companion papers show anew the central role played by the list ranking problem.

We note that a preliminary version of this paper appeared in [CV-86e].

In § 3 we present the duration-unknown task scheduling algorithm. In § 4 we give the new list ranking algorithm. In § 5 we provide a summary of the results obtained and pose an open problem.

**2. Preliminaries.** We give below a useful and simple scheme, due to Brent [B-74], for designing parallel algorithms. Later, we discuss implications of this scheme for the formulation of complexity results regarding the performance of parallel algorithms.

**THEOREM (Brent).** *Any synchronous parallel algorithm of time  $t$  that consists of a total of  $x$  elementary operations can be implemented by  $p$  processors in time  $\lceil x/p \rceil + t$ .*

*Remark.* Brent’s theorem is stated for parallel models of computation where not all computational overheads are taken into account. Specifically, the proof of Brent’s theorem ignores the problem of assigning processors to their jobs. Often, in the present paper, it is straightforward to overcome this implementation problem without increasing the running time or the number of processors in order of magnitude. Therefore, we allow ourselves to switch freely from a result of the form “ $O(x)$  operations and  $O(t)$  time” to “ $x/t$  processors and  $O(t)$  time” (and vice versa, which is always correct). However, we avoided doing this where there are difficulties with the implementation problem.



**3. Approximate scheduling.** We solve the following dynamic scheduling problem: the *duration-unknown task scheduling* problem (which is encountered in the list ranking problem). Our solution requires repeated rapid rescheduling of the processors. The mechanism that provides each rescheduling is a *redistribution* procedure (it solves the *object redistribution* problem). We start by defining the two problems. Then we give the redistribution procedure. Following this, we show how to solve the task scheduling problem using repeated applications of the redistribution procedure. Finally, we prove the correctness of the redistribution procedure.

We define the *duration-unknown task scheduling* problem (for short, the task scheduling problem), as follows. We are given  $n$  tasks. A task is a program that is to be run by one processor. The *length* of a task is its sequential running time (we will measure this as a specific number of  $O(1)$  time steps, which we call *real steps*). The lengths of individual tasks may be dependent on the scheduling; in fact, the length of an individual task may not be known until its execution terminates; however, we are guaranteed that no task has length greater than  $e \log n$ , and that the total length of all the tasks is bounded by  $cn$ , regardless of the scheduling, for some constants  $c$  and  $e$ . The problem is to execute these tasks on  $n/\log n$  processors in  $O(\log n)$  time. We are allowed to schedule and reschedule the tasks as we wish, so long as the scheduling occupies only  $O(\log n)$  time.

We define the *object redistribution* problem, as follows. We are given  $r$  objects, partitioned among  $p$  collections of objects. We are also given one processor per collection. Loosely speaking, the problem is to redistribute the objects among the collections so that they are more evenly distributed. For a more precise description we need some definitions.  $size_i$ , the *size* of collection  $i$ ,  $1 \leq i \leq p$ , is the number of objects in collection  $i$ ; note that  $r = \sum_{i=1}^p size_i$ . The *weight* of collection  $i$  is  $size_i^2$ , the square of its size; let  $W = \sum_{i=1}^p size_i^2$  be the total weight of all the collections. Let  $WMIN_r$  denote the minimum possible weight over all possible distributions of  $r$  objects among  $p$  collections; we note that  $p \lfloor r/p \rfloor^2 \leq WMIN_r \leq p \lceil r/p \rceil^2 \leq 4WMIN_r$ . When the value of  $r$  is clear from the context, we simply write  $WMIN$ . Let  $f$  and  $g$  be constants ( $f \geq g = 298$ , with  $f$  specified more precisely later in this section). If  $W$  is bounded by either  $f p$  or  $g WMIN$  then the collections are said to be *balanced* (i.e. either there are few objects present, or objects are roughly evenly distributed). We distinguish the following *classes* of collections: class 0 comprises those collections of size 0 or 1; class  $i$ ,  $i > 0$ , comprises those collections whose size is in the range  $[2^i, 2^{i+1})$ .

The *object redistribution problem* is the following: Redistribute the objects (using the redistribution procedure) so that the following properties hold:

- (1) The total weight is not increased.
- (2) The maximum number of objects in any one collection does not increase.
- (3) If the collections are unbalanced, the total weight of the collections is reduced by a multiplicative constant factor.
- (4) The redistribution takes  $O(1)$  time.

We describe the redistribution procedure and then show how to implement it in  $O(1)$  time. Later we show how to use this procedure to solve the task scheduling problem.

The redistribution procedure will perform a cascade of redistributions of objects from heavier collections to lighter collections. As we will see, this provides the desired weight reduction. We note that we have little control over the distribution of objects at the start of the redistribution procedure, and hence we have little control over the distribution of collections among the classes. Nonetheless, for any class containing many collections, we want to be able to perform redistributions of objects from most

of these collections to other collections of considerably smaller size. This motivates us to consider expander graphs.

**DEFINITION.** A graph  $G = (V, E)$  is a  $(d, \epsilon, \gamma)$ -expander graph,  $\epsilon < 1$ , if for any subset  $U \subseteq V$ , with  $|U| \leq \epsilon|V|$ , the set  $N(U)$  of neighbors of vertices in  $U$  has size  $|N(U)| \geq \gamma|U|$ , and  $G$  has vertex degree  $d$ .

It is convenient to assume that  $d$  is a power of 2 (if not, we can add additional edges to  $G$ ; this will not affect the expander property). Given any  $\epsilon < 1$ , it is known that there exist expander graphs with  $\gamma = (1 - \epsilon)/\epsilon$ , for sufficiently large  $d$  (where  $d$  is a function of  $\epsilon$  only). Further, such an expander graph can be built in  $O(\log |V|)$  time using  $|V|/\log |V|$  processors, for each fixed  $\epsilon$ , as we show in Appendix B. (See [LPS-86], [JM-85], [GG-81] for results on expander graphs.)

*Remark.* Peleg and Upfal recently applied expander graphs to a similar but not fully comparable problem [PU-86].

*The redistribution procedure.* For each collection we create a node; these nodes are connected by a  $(d, \epsilon, (1 - \epsilon)/\epsilon)$ -expander graph. We perform the following object redistribution. Suppose there is an edge connecting collections  $C_1$  and  $C_2$ . Let  $C_1$  and  $C_2$  be in classes  $i_1$  and  $i_2$ , respectively. If  $i_1 > i_2 + 1$ , then a number of objects are removed from  $C_1$  and added to  $C_2$ . More precisely, let  $\beta = 2^{i_1}$  (then  $C_1$  contains at least  $\beta$  objects and fewer than  $2\beta$  objects); if  $\beta \geq 8d$  then the number of objects transferred is  $\beta/8d$ ; otherwise, no objects are moved. If such an object transfer from  $C_1$  to  $C_2$  occurs, we call  $C_1$  a *giving* collection and  $C_2$  a *receiving* collection. A collection can be both giving and receiving.

We note that an attempted redistribution from a giving collection of size less than  $8d$  has no effect. Thus we would like the collections to be balanced if  $W \leq (8d)^2 n / \log n$ . This is achieved by requiring  $f \geq (8d)^2$ .

Next, we describe the data structure we use to implement the redistribution in  $O(1)$  time.

*Implementation of the redistribution procedure.* The objects are stored at the leaves of complete binary trees. Formally, suppose a collection has  $\alpha$  objects. If  $\alpha = 1$  then this single object forms a (complete binary) tree. Otherwise, suppose  $2^i \leq \alpha < 2^{i+1}$ ,  $i \geq 1$ . Then,  $2^i$  of these objects form a complete binary tree and the remaining  $\alpha - 2^i$  objects recursively form complete binary trees. So, in each collection, the objects are stored in a set of complete binary trees, no two of the same size. We maintain the following pointers: for each internal node in such a tree, pointers to its leftmost and rightmost leaves; in addition, we keep the leaves in each tree in a linked list, in left to right order (these pointers are needed for the solution to the scheduling problem). Finally, for each collection, we keep its trees in a doubly linked list, in increasing order by size.

The object transfer can be performed in  $O(1)$  time since it only involves manipulating the top  $O(\log d)$  levels of the largest tree in  $C_1$  (followed by a constant amount of tree reconstruction in  $C_1$  and  $C_2$ ). More precisely, for each collection with at least  $8d$  objects, we divide the largest tree in the collection (of size  $\beta \geq 8d$ ) into  $8d$  equal sized subtrees. Up to  $d$  of these trees are transferred in the object redistribution. The remaining trees of size  $\geq \beta/8d$  in the collection are then repeatedly paired together (only equal sized trees are paired), until there is at most one tree of each size. Since this involves at most  $8d + \log 8d$  trees it takes  $O(1)$  time using one processor per collection. The pairing process is then repeated with the trees that are transferred to the collection. There are at most  $d + \log 4d$  trees involved in the latter pairing; so it also takes  $O(1)$  time using one processor per collection. We note that care must be taken to maintain the pointers for the tree nodes as the trees are manipulated; however, this is straightforward.

The theorem below claims the correctness of the redistribution procedure.

*The redistribution procedure (RP) Correctness Theorem.* The redistribution procedure solves the object redistribution problem (i.e., it satisfies the four properties of the problem).

So far we have only shown that the redistribution procedure needs  $O(1)$  time (property (4)). The second half of this section is devoted to the proof of the other three properties of the RP Correctness Theorem. However, in the description of the algorithm for task scheduling and its proof, which follow, we will already assume the RP Correctness Theorem.

*The algorithm for the duration-unknown task scheduling problem.* We solve the task scheduling problem as follows. When applying the redistribution procedure, we view the tasks as the objects; also, we set  $p = n/\log n$  and define  $r$  to be the number of tasks at hand. Initially, we distribute  $\log n$  tasks to each collection. Each collection will never contain more than  $\log n$  tasks. We perform  $O(\log^{(2)} n)$  of the following stages.

(i) For each processor, perform  $\log n/\log^{(2)} n$  real steps on the tasks in its collection in  $O(\log n/\log^{(2)} n)$  time.

(ii) Apply the redistribution procedure. This requires  $O(1)$  time.

The following lemma justifies going to the final stage at this point; the lemma will be proved later.

LEMMA 3.1. *After  $h \log^{(2)} n$  stages there are at most  $fn/\log n$  incomplete tasks, for some constant  $h$ .*

*The final stage.* Distribute the remaining (at most  $fn/\log n$ ) tasks evenly among the processors (this is done using a standard prefix sum computation, as in [FL-80], in a further  $O(\log n)$  time using  $n/\log n$  processors). Each processor will then complete its allotted tasks by performing at most  $fe \log n$  more real steps.

Thus the task scheduling problem can be solved in  $O(\log n)$  time on  $n/\log n$  processors.

*Remark.* There are several implementation details of the scheduling algorithm that should be mentioned. First, the links in the trees containing the tasks allow us to access each successive task in  $O(1)$  time, starting at the leftmost task (leaf) in the smallest tree. Thus we can perform  $\log n/\log^{(2)} n$  real steps, in part (i), in  $O(\log n/\log^{(2)} n)$  time. Second, it is convenient, in the list ranking application, to ensure that having started one task, the processor completes it before beginning another. All we have to do is ensure that we do not redistribute the tasks that processors are currently executing. This is easily done by redistributing from the right end of trees in the distribution procedure. We will then never redistribute the current task, which is at the leftmost end of some tree. Third, we need to explain how to reformat the trees, at the end of part (i) of a typical stage, so that they are of the form assumed by the redistribution procedure. Let the task being processed at the end of part (i) be at leaf  $\nu$  of tree  $T$ . To ensure all the trees are complete and distinct we need to partition  $T$  into complete subtrees (recall that  $T$  is the smallest tree associated with the collection; thus we are guaranteed that if we create distinct sized complete binary trees from  $T$  then no two trees in the collection will have the same size and we get a proper set of complete binary trees). To partition  $T$ , we follow a path from  $\nu$  to the root of  $T$ . Each maximal right subtree that we encounter is made into a new tree. This traversal takes  $O(\log^{(2)} n)$  time.

*Complexity.* Assuming the RP Correctness Theorem and Lemma 3.1 we can conclude that the task scheduling problem can be solved in  $O(\log n)$  time on  $n/\log n$  processors.

Next, we prove Lemma 3.1.

*Proof of Lemma 3.1.* Consider a single stage and consider the end of part (i) of that stage. There are three possibilities.

Case 1. The collections are not balanced.

Case 2. The collections are balanced; so either

Case 2a. the total weight is bounded by  $gWMIN$  but not by  $fn/\log n$ , or

Case 2b. the total weight is bounded by  $fn/\log n$ .

In Case 1, part (ii) reduces the total weight by a constant factor by property (3) of the RP Correctness Theorem. Since the initial weight is  $n \log n$ , this can happen only  $O(\log^{(2)} n)$  times, say at most  $h_1 \log^{(2)} n$  times, for some constant  $h_1$  (for at that point the weight will have been reduced to  $fn/\log n$ ; i.e., the collections are balanced).

We claim that in Case 2a at least  $1/(16g) \cdot n/\log n$  of the collections are not empty. This is seen as follows. We first note that  $r$ , the number of objects present, is at least  $p(=n/\log n)$  for otherwise we would have  $WMIN \leq p$ . This would imply  $gWMIN \leq fp$  (remember  $f \geq g$ ), and therefore contradicts the assumption of Case 2a. Next, suppose  $x$  of the collections are nonempty. Then  $W$ , the total weight of all the collections, is at least  $x \cdot \lfloor r/x \rfloor^2$ ; also  $W \leq gWMIN$  and  $WMIN \leq p \cdot \lceil r/p \rceil^2$ . That is,  $x \lfloor r/x \rfloor^2 \leq gp \lceil r/p \rceil^2$ , and since  $r \geq p \geq x$ , we have  $r^2/4x \leq 4gr^2/p$ , or  $x \geq p/(16g)$ .

For each nonempty collection, in part (i) of the stage,  $\log n/\log^{(2)} n$  real steps were performed on the tasks at hand. Thus, in Case 2(a), at least  $n/(16g \log^{(2)} n)$  real steps were performed on the tasks in part (i) of this stage. We conclude that there are at most  $16gc \log^{(2)} n$  instances of Case 2(a).

In Case 2(b), there are at most  $fn/\log n$  tasks remaining.

Let  $h = h_1 + 16gc$ . We have shown that following  $h \log^{(2)} n$  iterations there can be at most  $fn/\log n$  tasks remaining incomplete.  $\square$

It remains to prove that properties (1)–(3) of the RP Correctness Theorem hold. Property (2) is clear. In the rest of this section we prove properties (1) and (3). We start by providing an outline of their proofs. We then prove individual lemmas.

For purposes of the analysis, it is convenient to serialize the redistributions that occur during the redistribution procedure (in fact, the redistributions are performed simultaneously); we refer to a redistribution between two collections as a *redistribution step*. We consider each redistribution step to be an ordered pair comprising the weight of the receiving collection followed by the weight of the giving collection. The redistribution steps are ordered lexicographically, and are performed in this order. Thus the redistribution steps into the lightest receiving collection are performed first; among these redistribution steps, the one from the lightest giving collection is performed first.

Consider a single redistribution step involving giving collection  $C_g$ , of size  $s_g$  at the start of the redistribution procedure, and receiving collection  $C_r$ , of size  $s_r$  at the start of the redistribution procedure.

LEMMA 3.2. *The weight reduction produced by the redistribution step is at least  $s_g^2/32d$ .*

COROLLARY 3.1. *Any sequence of redistribution steps reduces the total weight (property (1) of the RP Correctness Theorem).*

COROLLARY 3.2. *Let  $W_g$  be the total weight of the giving collections, counting multiplicities, at the start of the redistribution procedure. Then the weight reduction for these collections, due to the redistribution procedure, is at least  $W_g/32d$ .*

The remainder of the proof is concerned with proving property (3) of the RP Correctness Theorem. So henceforth we assume that the collections are unbalanced. We show that in this case  $W_g$  is a constant fraction of  $W$ .

We need additional notation and definitions. Let  $S_i$  be the set of collections in class  $i$ . Let  $L_i = \bigcup_{k \geq i} S_k$ . Let  $|S_i|$  (resp.  $|L_i|$ ) denote the number of collections in  $S_i$  (resp.  $L_i$ ), and let  $wt(S_i)$  (resp.  $wt(L_i)$ ) denote the sum of the weights of the collections

in  $S_i$  (resp.  $L_i$ ). On the average there are  $s/p$  objects in each one of the  $p$  collections. Define the variable *average* to be  $\lfloor \log_2 s/p \rfloor$ . We refer to class *average* as the average class. Let  $\alpha$  denote the smallest  $i > \text{average} + 2$  such that  $|S_i| > |S_{i-1}|/32$ , if any. We define a set  $S_i$ ,  $i \geq \alpha$ , to be *giving* if there are at least  $|S_i|$  giving collections in  $S_i$ , counting multiplicities. We implement the expander graph so that whenever  $|S_i| \geq 1/32|S_{i-1}|$  and  $|S_i| \geq |L_{i+1}|$  then  $S_i$  is giving.

We now give a proof outline.

LEMMA 3.3.  $\alpha$  exists.

LEMMA 3.4. The weight of  $L_\alpha$  is at least  $W/2$ .

LEMMA 3.5. The weight of the giving collections in  $L_\alpha$ , counting multiplicities, is at least  $1/40 \text{ wt}(L_\alpha) > 1/2^6 \text{ wt}(L_\alpha)$ .

COROLLARY 3.3. The weight of the giving collections in  $L_\alpha$ , counting multiplicities, is at least  $1/2^7 W$ .

We conclude, from Corollaries 3.2 and 3.3 the following.

THEOREM 3.1. If the collections are not balanced then, in  $O(1)$  time, the redistribution algorithm reduces the total weight by a multiplicative factor of at least  $1 - (1/32d)$  ( $1/2^7 \geq 1 - (1/2^{12}d)$ ).

Theorem 3.1 demonstrates property (3) of the RP Correctness Theorem, and thus completes its proof.

We now elaborate on the proof of Lemma 3.5, which is a consequence of several sublemmas. We need additional definitions. For Lemma 3.5, we only consider sets  $S_i$  with  $i \geq \alpha$ . We need to classify these sets according to whether they must contain many giving collections. Thus, we define a set  $S_i$  to be *large* if  $|S_i| \geq |S_{i-1}|/32$ . Likewise, it is *small* if  $|S_i| < |S_{i-1}|/32$ . We will show that the weight of the large sets is at least half the weight of  $L_\alpha$ . The neighbors, for giving collections, are provided by the edges of an expander graph. Thus if  $|L_{i+1}|$  is large compared to  $|S_i|$ , we may not be able to guarantee that there are many giving collections in  $S_i$ . So we define  $S_i$  to be *useful* if  $|S_i| \geq |L_{i+1}|$ . We will show that the weight of the useful large sets is at least one-fifth of the weight of the large sets. Finally, by choosing the right constant  $\varepsilon$  for the expander graph ( $\varepsilon = \frac{1}{36}$ ), we will ensure that each useful large set is a giving set.

LEMMA 3.5.1. The weight of the large sets is at least  $\frac{1}{2} \text{ wt}(L_\alpha)$ .

LEMMA 3.5.2. The weight of the useful large sets is at least  $\frac{1}{5}$  of the weight of the large sets.

LEMMA 3.5.3. A useful large set  $S_i$  has at least  $|S_i|$  neighbors in  $\cup_{j=0}^{i-2} S_j$ , i.e., it is a giving set.

*Proof of Lemma 3.5.* In Lemma 3.5.1 we showed that the weight of the large sets was at least  $\frac{1}{2} \text{ wt}(L_\alpha)$ . In Lemma 3.5.2 we showed that the weight of the useful large sets was at least one-fifth the weight of the large sets. In Lemma 3.5.3 we showed that each useful large set  $S_i$  has at least  $|S_i|$  giving collections, counting multiplicities. Finally, since the size of the collections in each useful large set  $S_i$  lie in the range  $[2^i, 2^{i+1})$ , we conclude that the giving collections in  $S_i$ , counting multiplicities, have weight at least  $\frac{1}{4} \text{ wt}(S_i)$ . Thus the weight of the giving collections in  $L_\alpha$  is at least  $\frac{1}{2} \cdot \frac{1}{5} \cdot \frac{1}{4} \text{ wt}(L_\alpha) = \frac{1}{40} \text{ wt}(L_\alpha)$ .  $\square$

It remains to prove Lemmas 3.2, 3.3, 3.4, 3.5.1, 3.5.2 and 3.5.3.

*Proof of Lemma 3.2. Claim.* Following the redistribution step,  $C_g$  has size at least  $\frac{7}{8} s_g$ , and  $C_r$  has size at most  $s_r + \frac{1}{8} s_g$ . (This follows from the lexicographic ordering of the redistribution steps, and the fact that each giving collection redistributes at most  $\frac{1}{8}$  of its objects.)

Let  $\frac{7}{8} s_g + a + b$  be the size of  $C_g$  immediately before the redistribution step, let  $s_r + c$  be the size of  $C_r$  immediately before the redistribution step, and let  $a$  be the number of items transferred. We know that: (1)  $s_r < s_g/2$  (since if  $s_g$  belongs to some

class  $i$  then  $s_r$  can belong only to a class  $j$ , where  $j \leq i - 2$ . (2)  $c + a \leq \frac{1}{8} s_g$ . (This follows from the above claim regarding the size of  $C_r$ .) (3)  $a + b \geq 0$  (this is trivial). And, (4)  $16da > s_g$  (since  $a$ , the number of elements transferred is  $> s_g/16d$ ). The weight reduction  $W_R$  is given by:

$$W_R = (\frac{7}{8} s_g + a + b)^2 + (s_r + c)^2 - (\frac{7}{8} s_g + b)^2 - (s_r + c + a)^2 = 2a(\frac{7}{8} s_g + b - s_r - c).$$

Observe that (2) and (3) above imply that  $b - c = -((c + a) - (b + a)) \geq -s_g/8$ . Using this and (1) above we get,

$$W_R \geq 2a(\frac{7}{8} s_g - s_g/2 - \frac{1}{8} s_g) = \frac{as_g}{2}.$$

By (4) above we get,

$$W_R \geq \frac{s_g^2}{32d} \quad \square$$

*Proof of Lemma 3.3.* We assume, in contradiction, that  $\alpha$  does not exist and show that the collections must have been balanced, which is contrary to assumption. To show this we observe the following:

(a) The total weight of the collections in classes  $0, \dots, average - 1$  is less than  $WMIN$ .

(b)  $wt(S_{average}) \leq 4WMIN$ ;  $wt(S_{average+1}) \leq 16WMIN$ ;  $wt(S_{average+2}) \leq 64WMIN$ .

(c)  $wt(S_i) \leq wt(S_{i-1})/2$  for  $i > average + 2$  (since otherwise  $\alpha$  would exist).

The total weight is therefore,

$$\sum_{i < average} wt(S_i) + wt(S_{average}) + wt(S_{average+1}) + wt(S_{average+2}) + \sum_{i > average+2} wt(S_i).$$

Observations (a), (b) and (c) above imply that this is

$$\leq WMIN + 4WMIN + 16WMIN + 64WMIN + 64WMIN = 149WMIN.$$

Since  $g \geq 298$  we conclude that the collections are balanced.  $\square$

*Proof of Lemma 3.4.* The proof of Lemma 3.3 shows that  $wt(L_\alpha) \geq W - 149WMIN$ . By assumption, since the collections are unbalanced,  $W \geq gWMIN \geq 298WMIN$ ; we conclude that  $wt(L_\alpha) \geq W/2$ .  $\square$

*Proof of Lemma 3.5.1.* Let  $S_i, S_{i+k}$  be large sets and suppose that for each  $j, 1 \leq j < k, S_{i+j}$  is not large. Then, as in the proof of Lemma 3.3, we can show that  $wt(S_i) \geq \sum_{j=1}^{k-1} wt(S_{i+j})$ . Let  $S_h$  be the large set of greatest index; we can also show that  $wt(S_h) \geq \sum_{j \geq 1} wt(S_{h+j})$ . Since  $S_\alpha$  is large the lemma follows.  $\square$

*Proof of Lemma 3.5.2.* Observe that the large set whose index is maximal must be useful. Consider a sequence of sets  $S_{i_{k+1+1}}, S_{i_{k+1+2}}, \dots, S_{i_0}$ , which satisfies the following: (1)  $S_{i_0}$  is the only set in the sequence which is both useful and large. (2) Either  $S_{i_{k+1}}$  is both useful and large; or  $i_{k+1}$  is  $\alpha - 1$ . (3)  $S_{i_k}, S_{i_{k-1}}, \dots, S_{i_1}, S_{i_0}$  is the subsequence of large sets in this sequence. We show that  $\sum_{j=1}^k wt(S_{i_j}) \leq 4wt(S_{i_0})$ . Since any large set must lie in such a sequence the lemma follows.

Let  $T_j$  be the union of the small sets between  $S_j$  and  $S_{j-1}$ . Let the sequence  $R_{i_h}, R_{i_{h-1}}, \dots, R_{i_0}$ , comprise the merge of the sequence  $S_{i_k}, \dots, S_{i_0}$ , and the nonempty sets in the sequence  $T_{i_k}, \dots, T_{i_1}$ . We actually prove that  $\sum_{j=1}^h wt(R_{i_j}) \leq 4wt(R_{i_0}) = 4wt(S_{i_0})$ . This result is an immediate consequence of the following claim: *Claim 1.* For  $1 \leq j \leq h, wt(R_{i_j}) \leq 2^{2-j} wt(R_{i_0})$ . This in turn, is an immediate consequence of the following two claims: *Claim 2.* For  $1 \leq j \leq h, |R_{i_j}| \leq 2^j |R_{i_0}|$ . *Claim 3.* For  $1 \leq j \leq h$ , the weight of any collection in  $R_{i_j}$  is at most  $2^{2-2j}$  of the weight of any collection in  $R_{i_0}$ . Claim 3 is immediate.

*Proof of Claim 2.* For each set  $R_j, j \geq 1$ , we prove the following *assertion*: the number of collections in  $\bigcup_{x=0}^{j-1} R_x \cup L_0$  is greater than the number of collections in  $R_j$ . Claim 2 follows by induction on  $j$ , from the assertion, when we note that  $|L_{i_0}| \leq |R_{i_0}|$ . The proof of the assertion breaks into two cases.

*Case 1.*  $R_j$  is a set  $S_{i_y}$ . Because  $S_{i_y}$  is not useful  $|S_{i_y}| < |L_{i_y+1}|$ , and the assertion follows.

*Case 2.*  $R_j$  is a set  $T_{i_x}$ . We note that  $|T_{i_x}| < \frac{1}{31}|S_{i_x}|$ . Also, since  $S_{i_x}$  is not useful,  $|S_{i_x}| < |T_{i_x}| + |L_{i_x-1}|$ . Thus  $|T_{i_x}| < \frac{1}{31}(|T_{i_x}| + |L_{i_x-1}|)$ . We deduce that  $|T_{i_x}| < |L_{i_x-1}|$ . Again, the assertion follows.  $\square$

*Proof of Lemma 3.5.3.* We note that  $|S_{i-1}| + |S_i| + |L_{i+1}| \leq 34|S_i|$  for  $S_i$  large and useful. There are two possibilities:

*Possibility 1.*  $|S_i| \leq p/36$ . Then  $S_i$  has at least  $35|S_i|$  neighbors, of which at least  $|S_i|$  lie outside  $S_{i-1} \cup S_i \cup L_{i+1}$ .

*Possibility 2.*  $|S_i| > p/36$ . Then  $S_i$  has at least  $35p/36$  neighbors. Observe that  $|\bigcup_{j=0}^{i-2} S_j| \geq p/2$  (since  $i > \text{average} + 2$ ). Therefore, at least  $17p/36$  of these neighbors are in  $\bigcup_{j=0}^{i-2} S_j$ . Also,  $i > \text{average} + 2$ , implies that  $|S_i| \leq p/4$ . Thus  $S_i$  has at least  $17/9|S_i| > |S_i|$  neighbors in  $\bigcup_{j=0}^{i-2} S_j$ . This completes the proof of Lemma 3.5.3.  $\square$

Note that we have required that  $f \geq g, f \geq (8d)^2$ . Thus, in the definition of balanced above, we set  $f = \max \{g, (8d)^2\}$ .

**4. List ranking.** We give an optimal  $O(\log n)$  time algorithm to solve the following generalization of list ranking.

*Input.* A linked list of  $n$  nodes, stored in an array with index range  $[0: n-1]$ . In addition, for each node, we are given the distance to its successor in the list (typically this distance is unity, though in general it need not be). For each node, we store the pointer and the distance to its successor in the arrays  $D(0: n-1)$  and  $R(0: n-1)$ , respectively. For the last node  $v$  in the list we have  $R(v) = 0$  and  $D(v) = \text{nil}$ .

*Problem.* Compute into array  $R$ , for each node  $u$ , the distance from  $u$  to the end of the list.

It is useful to assume that each node knows its predecessor in the list. We store the predecessors in array  $P$ .  $P$  can be computed in  $O(1)$  time using  $O(n)$  operations, in the obvious way.

The algorithm starts with a series of  $O(\log n)$  steps. Each step takes  $O(1)$  time. In each step we reduce the problem to a smaller subproblem by removing a set of nonadjacent nodes from the list. We remove node  $u$ , the successor of node  $v$  (that is  $u = D(v)$ ), by the following group of assignments.

$$\begin{aligned} R(v) &:= R(v) + R(u); \\ D(v) &:= D(D(v)) \\ \text{if } D(v) \neq \text{nil} &\text{ then } P(D(v)) := v \end{aligned}$$

At each step of the algorithm each processor is associated with a node. If these assignments are performed by a processor associated with node  $v$  this is called a *traversal* by node  $v$ ; if they are performed by a processor associated with node  $u$  this is called a *removal* by node  $u$ . Since, in any given step the removed nodes are nonadjacent, it is easy to perform the removals by an EREW algorithm.

Our algorithm has three stages.

*Stage 1.* In  $O(\log n)$  steps the input list will be reduced to a list of at most  $n/\log n + 1$  nodes. This will take  $O(\log n)$  time and  $O(n)$  operations.

*Stage 2.* We compute the list ranking on the remaining list (the *reduced* list) using Wyllie's list ranking algorithm [W-79]; it will perform  $O(n)$  operations in time  $O(\log n)$ .

*Stage 3.* We show how to “reconstruct” the list ranking of nodes that were removed in Stage 1; it is reconstructed from the list ranking of the reduced list, computed in Stage 2. We first observe that given the list ranking for the list present at the end of a step of Stage 1, the ranking for the nodes present at the start of the step can be computed in  $O(1)$  time. Specifically, let the vector  $R_{actual}$  contain the final list ranking (to be computed). Let  $u$  be a node, other than the last node, which was removed from the list at a step of Stage 1. Suppose that  $R_{actual}(D(u))$  is known. (Note that neither  $D(u)$  nor  $R(u)$  was changed by our algorithm after this removal). Then  $R_{actual}(u)$  can be computed by  $R_{actual}(u) := R(u) + R_{actual}(D(u))$ . Second we apply the idea of backtracking. Specifically, each processor “revisits” the operations it performed at Stage 1 from the most recent to the earliest; at the time of revisiting the operation of removing node  $u$  it simply computes  $R_{actual}(u)$ . We refer the reader to [CV-86a] for a more detailed discussion of the “backtracking” procedure required. The time complexity of Stage 3 is dominated by the time complexity of the forward steps of Stage 1.

So the total complexity of the algorithm is  $O(n)$  operations and  $O(\log n)$  time.

The rest of this section is concerned only with the traversals and removals of Stage 1. The goal is to obtain a reduced list of at most  $n/\log n + 1$  nodes (the reduced list will include the first node of the input list). We call the nodes in the reduced list *full* nodes. The traversals and removals are performed by a set of  $n - 1$  tasks, associated with each node in the list (except the first node). Each task performs at most  $5 \log n - 1$  steps of traversals and removals. Our main effort in this section is to show how to formulate these tasks so that we can use the duration-unknown task scheduling algorithm from § 3 to schedule them. We need the following definition.

**DEFINITION.** An  $r$ -ruling set of a linked list is a subset  $U$  of the nodes of the list such that

- (i) No two nodes of  $U$  are adjacent in the list.
- (ii) If  $v$  is a node in the list, the next node from  $U$  in the list is at most  $r$  edges (links) distant from  $v$ .

We recall that there is a  $4 \log n$ -ruling set algorithm that performs  $O(n)$  operations in  $O(1)$  time [CV-86a]. To make the paper self-contained we have described a slightly modified version of this algorithm in Appendix A. The algorithm has the following property: The first node in the list is always placed into the ruling set unless its successor is the last node in the list (i.e. the list contains exactly two nodes). We remark, that by definition, the ruling set contains the last node in the list. Also, the algorithm actually provides a *stronger result that we use below*: Suppose we assign a processor to each node in the list. Then, solely by looking at  $v$ , its predecessor and two successors, the processor can determine in  $O(1)$  time whether  $v$  is in the  $\log n$ -ruling set.

*The task of  $v$ , for any node  $v$  in the list.* At the beginning, the task of node  $v$  will be *waiting*. At some step of Stage 1 a processor will be assigned to node  $v$  and the task will become *active*. The task will remain active until it is completed. Upon completing this task the processor will be able to determine whether  $v$  is in the reduced list (i.e. whether  $v$  is a *full node*) or not. If  $v$  is not in the reduced list then node  $v$  either: performs a removal, or “marks itself for removal,” or is already removed.

On becoming active the task determines if node  $v$  has been removed from the present linked list. If so, in one step, the processor completes the task of node  $v$  without removing any node and  $v$  is not a full node. So, suppose that  $v$  is in the present list. The processor of  $v$  will complete its task with the decision that  $v$  is a full node if and when the following two conditions hold:

- (i) Node  $v$  has performed at least  $\log n$  traversals.



(ii) The successor of node  $v$  in the present list is not marked for removal. Each step comprises three **synchronized** substeps.

*Substep 1.*

**If** node  $v$  has performed at least  $\log n$  traversals and the successor of node  $v$  in the present list is not marked for removal  
**then** the task is completed (without even proceeding to Substep 2);  $v$  is a full node.

*Substep 2.*

**If** the successor of  $v$  has completed and is a full node  
**then if** the predecessor of  $v$  is not active  
**then**  $v$  performs a removal; the task is completed;  $v$  is not a full node.  
**else** mark  $v$  for removal; the task is completed;  $v$  is not a full node.

*Substep 3.*  $v$  belongs to a “chain” of presently active nodes; the chain is of length at least one and is followed by a node that is not full (either a node whose task is waiting or a node marked for removal).

Use the stronger version of the  $\log n$ -ruling set algorithm to find whether  $v$  is in a  $\log n$ -ruling set with respect to this chain. (Note: the last node in the chain is always placed in the ruling set. Specifically, in the trivial case, where the chain consists of a single node, this node is in the ruling set.)

**If**  $v$  is not in the ruling set and is not the first node in the chain  
**then** mark  $v$  for removal; the task is completed;  $v$  is not a full node.  
**else if**  $v$  is not in the ruling set and is the first node in the chain  
**then**  $v$  performs a removal; the task is completed;  $v$  is not a full node.  
**else** (\*  $v$  is in the ruling set \*)  $v$  performs a traversal.

Let us analyze the algorithm.

The reduced list comprises nodes that have traversed at least  $\log n$  nodes each, plus the first node in the input list. This implies:

LEMMA 4.1. *The length of the reduced list is at most  $n/\log n + 1$ .*

LEMMA 4.2. *The task of each node includes at most  $5 \log n - 1$  traversals.*

*Proof.* We need the following observation. Consider the time at which node  $u$  is marked for removal (but not removed). There must be a chain of length  $x \leq 4 \log n + 1$  nodes which ends at (and includes)  $u$  and satisfies the following: (1) The last  $x - 1$  nodes of this chain are marked for removal simultaneously. (2) The first node of the chain is active and has performed less than  $\log n$  traversals.

Now consider node  $v$  that has performed at least  $\log n$  traversals. After the time at which node  $v$  performed its  $\log n$ th traversal, its successors could have formed a chain of at most  $4 \log n - 1$  nodes marked for removal. This chain cannot grow while node  $v$  traverses the chain. Finally, Substep 1 implies that node  $v$  completes its task after traversing the  $\leq 4 \log n - 1$  nodes in this chain. Lemma 4.2 follows.  $\square$

Each removed node was subject to exactly one traversal or removal operation in one of the  $n - 1$  tasks. This implies the following corollary.

COROLLARY 4.1. *There are  $n - 1$  tasks. Each task is of length  $O(\log n)$  and the total length of the tasks is  $O(n)$ . Further, a task, once active, remains active until it is completed.*

Corollary 4.1 describes exactly the problem solved by the duration-unknown task scheduling algorithm of § 3. Thus these tasks require  $O(\log n)$  time and  $O(n)$  operations.

We have shown the following.

**THEOREM 4.1.** *There is an EREW PRAM algorithm for list ranking that runs in  $O(\log n)$  time using  $n/\log n$  processors, which is optimal.*

**5. Summary.** This paper has identified two scheduling problems, namely the *object redistribution* problem and the *duration-unknown task scheduling* problem. The object redistribution problem is solved optimally in  $O(1)$  time on  $n/\log n$  processors (by the *redistribution procedure*), and the task scheduling problem is solved in  $O(\log n)$  time on  $O(n/\log n)$  processors; it used the redistribution procedure as a subroutine, repeatedly. A key feature of the redistribution procedure is that it makes use of expander graphs, resulting in large constants in the running time.

The other major problem solved in this paper is the list ranking problem. An optimal deterministic algorithm is given:  $O(\log n)$  time and  $O(n/\log n)$  processors.

We pose the following open problem. Can a solution to the task scheduling problem, that does not involve the use of expander graphs, be found?

**Appendix A. The  $4 \log n$ -ruling set algorithm.** The algorithm we describe is a simplified presentation of the algorithm given in [CV-86a]; this presentation is due to Goldberg, Plotkin and Shannon [GPS-87].

*Assumptions about the input representation.* The vertices are given in an array of length  $n$ . The entries of the array are numbered from 0 to  $n-1$ . Each entry number is called the *label* of its vertex. The labels are represented as binary strings of length  $\lceil \log n \rceil$ . We refer to each binary symbol (bit) of this representation by a number between 0 and  $\lceil \log n \rceil - 1$ . The rightmost (least significant) bit is called bit number 0 and the leftmost bit is called bit number  $\lceil \log n \rceil - 1$ . Each vertex has a pointer to the next vertex in the list (representing its outgoing edge). For simplicity we assume that  $\log n$  is an integer.<sup>1</sup>

We show how to find a  $4 \log n$ -ruling set.

First, we show how to relabel the vertices. The new label of a vertex is called the *color* of the vertex. Let  $a = a_{k-1}, \dots, a_0$  and  $b = b_{k-1}, \dots, b_0$  be the labels of vertex  $u$  and its successor, respectively. Let  $j$  be the index of the least significant binary digit in which  $a$  and  $b$  differ. The *color of vertex  $u$*  is  $\langle j, a_j \rangle$ . We consider the color to be a binary number between 0 and  $2k-1$ . It is easy to see that the colors satisfy the following two properties:

- (1) There are at most  $2 \log n$  different colors.
- (2) Each pair of adjacent vertices are colored differently.

A remark in [CV-86a] explains how a vertex can compute its label in constant time using standard operations.

Second, we show how to select a  $4 \log n$ -ruling set. We define a vertex to be a local maximum if its color is larger than the colors of its two neighbors. Then a  $(4 \log n - 2)$ -ruling set can be chosen as follows: select all vertices that are local maxima. Since a monotone chain can have length at most  $2 \log n$ , it follows that we have selected a  $(4 \log n - 2)$ -ruling set. A slight modification of this algorithm creates a  $4 \log n$ -ruling set that contains the last vertex of the list, and if it is not adjacent to the last vertex, the first vertex also, again in  $O(1)$  time.

**Appendix B. Building the expander graph.** For our construction we need  $\varepsilon = \frac{1}{36}$ . In other words, for  $|U| \leq |V|/36$ , we need that  $|N(U)| \geq 35|U|$ . We use an expander graph based on the construction described following Theorem 2 of [JM-85]. There, a different definition of expander graphs, which is due to Margulis [M-75] is given. The

<sup>1</sup> The base of all logarithms in the Appendix is 2.

definition follows: A bipartite graph  $K = (V_1, V_2, E)$  is an  $(n, k, \delta)$  expander graph if  $|V_1| = |V_2| = n$ , the degree is  $k$ , and for each subset  $U$  of  $V_1$ ,  $|N(U)| \geq (1 + \delta(1 - |U|/|V_1|))|U|$ . (Comment: actually in [JM-85] Jimbo and Maruoka do not restrict the degree; they merely require  $|E| \leq kn$ ; however, their constructions all obey this less liberal definition.) To obtain an expander obeying the definition of § 3, simply identify the sets  $V_1$  and  $V_2$ ; for a given pair of values  $\varepsilon$  and  $\delta$  it is easy to deduce the corresponding  $\gamma$ .

Jimbo and Maruoka [JM-85] give an expander graph  $K$  with  $k = 5$  and  $\delta > \frac{1}{10}$ . From this graph, using standard methods, we can obtain the expander graph  $G$  needed for our construction, as follows. We "iterate" the construction  $r$  times, for some constant  $r$  (which depends only on  $\varepsilon$ ), creating graph  $H$ . Namely, the vertices of  $H$  comprise the disjoint vertex sets  $V_1, V_2, \dots, V_{r+1}$ ; we place a copy of the edges of  $K$  between  $V_i$  and  $V_{i+1}$ , for  $1 \leq i \leq r$ .  $G$  has vertex sets  $V_1$  and  $V_{r+1}$ . If in  $H$ , vertices  $v \in V_1$  and  $w \in V_{r+1}$  are connected by a path of length  $r$ , then in  $G$ ,  $v$  and  $w$  are joined by an edge.  $K$  has vertex degree 5; thus  $G$  has vertex degree at most  $5^r$ . Finding the constant  $r$  is easy: For  $|U| \leq \frac{35}{36}|V_1|$  the graph  $K$  satisfies that  $|N(U)| \geq (1 + \frac{1}{10} \frac{1}{36})|U| \geq (1 + \frac{1}{360})|U|$ . It is easy to see that in  $G$ ,  $|N(U)| \geq \min\{\frac{35}{36}|V_1|, (1 + \frac{1}{360})^r|U|\}$ . Determining  $r$  is now straightforward. (A more careful argument provides a much tighter bound.) There are two more issues to be considered in order to adapt the construction of [JM-85] to our needs:

(1) In [JM-85] Jimbo and Maruoka provide expander graphs for which  $|V_1| = |V_2| = m^2$ , where  $m$  is an integer. We actually wanted an expander graph on  $n$  vertices, not on  $m^2$  vertices. So let  $(m-1)^2 < n \leq m^2$ . Instead of using an expander graph on  $n$  vertices, we use the expander graph on  $m^2$  vertices. This implies the algorithm assumes  $m^2$  processors are available; but such an algorithm is readily simulated on  $n$  processors, slowing it down by a factor of at most 2.

(2) We comment on the complexity of constructing the edges of  $K$  (and implicitly of  $G$ ). Each vertex has degree five. Each edge can be readily determined in  $O(1)$  time by a single processor.

**Acknowledgments.** The authors thank the referees for their comments; in particular, the comments that led to a clearer presentation of § 3.

## REFERENCES

- [B-74] R. P. BRENT, *The parallel evaluation of general arithmetic expressions*, J. Assoc. Comput. Mach., 21 (1974), pp. 201-206.
- [C-86] R. COLE, *An optimal parallel selection algorithm*, Courant Institute Tech. Rpt. 209, New York University, New York, 1986.
- [C-86a] ———, *Parallel merge sort*, Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 511-516.
- [CV-86a] R. COLE AND U. VISHKIN, *Deterministic coin tossing with applications to optimal parallel list ranking*, Inform. and Control, 70 (1986), pp. 32-53.
- [CV-86b] ———, *The accelerated centroid decomposition technique for optimal parallel tree evaluation in logarithmic time*, Courant Institute Tech. Rpt. 242, New York University, New York, 1986.
- [CV-86c] ———, *Approximate parallel scheduling. Part II: applications to optimal parallel graph algorithms in logarithmic time*, in preparation.
- [CV-86d] ———, *Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms*, Proc. 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 206-219.
- [CV-86e] ———, *Approximate and exact parallel scheduling with applications to list, tree and graph problems*, Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 478-491.
- [FL-80] M. FISHER AND L. LADNER, *Parallel prefix computation*, J. Assoc. Comput. Mach., 27 (1980), pp. 831-838.

- [GG-81] O. GABBER AND Z. GALIL, *Explicit constructions of linear sized superconcentrators*, J. Comput. System Sci., 22 (1981), pp. 407–420.
- [GPS-87] A. GOLDBERG, S. PLOTKIN AND G. SHANNON, *Parallel symmetry-breaking in sparse graphs*, Proc. 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 315–324.
- [JM-85] S. JIMBO AND A. MARUOKA, *Expanders obtained from affine transformations*, Proc. 17th Annual Symposium on Theory of Computing, 1985, pp. 88–97.
- [KRS-85] C. P. KRUSKAL, L. RUDOLPH AND M. SNIR, *Efficient parallel algorithms for some graph problems*, Proc. Internat. Conference on Parallel Processing, 1985, pp. 180–185.
- [LPS-86] A. LUBOTZKY, R. PHILLIPS AND P. SARNAK, *Ramanujan conjecture and explicit constructions of expanders and super-concentrators*, Proc. 18th Annual ACM Symposium on Theory of Computing, pp. 240–246.
- [M-75] G. A. MARGULIS, *Explicit constructions of concentrators*, Problemy Peredachi Informatsii, 9 (1973), pp. 71–80. (In Russian.) Problems Inform. Transmission (1975), pp. 325–332. (In English.)
- [MR-85] G. MILLER AND J. REIF, *Parallel tree contraction and its application*, Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 478–489.
- [Pi-86] N. PIPPENGER, *Sorting and selecting in rounds*, manuscript.
- [PU-86] D. PELEG AND E. UPFAL, *The token distribution problem*, Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 418–427.
- [R-86] J. REIF, *An optimal parallel algorithm for integer sorting*, Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, pp. 1986, 496–503.
- [SV-86] B. SCHIEBER AND U. VISHKIN, *On finding lowest common ancestors: simplification and parallelization*, manuscript.
- [TV-85] R. E. TARJAN AND U. VISHKIN, *An efficient parallel biconnectivity algorithm*, SIAM J. Comput., 14 (1985), pp. 862–874.
- [Vi-83] U. VISHKIN, *Synchronous parallel computation—a survey*, TR 71, Dept. of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, 1983.
- [Vi-84] ———, *Randomized speed-ups in parallel computation*, Proc. 16th ACM Symposium on Theory of Computing, 1984, pp. 230–239. For a journal version of this paper see *Randomized speed-ups for list ranking*, J. Parallel and Distributed Computing, to appear.
- [Vi-85] ———, *On efficient parallel strong orientation*, Inform. Process. Lett., 20 (1985), pp. 235–240.
- [W-79] J. C. WYLLIE, *The complexity of parallel computation*, TR 79-387, Department of Computer Science, Cornell University, Ithaca, NY, 1979.

## AN $O(n \log \log n)$ -TIME ALGORITHM FOR TRIANGULATING A SIMPLE POLYGON\*

ROBERT E. TARJAN†‡ AND CHRISTOPHER J. VAN WYK†

**Abstract.** Given a simple  $n$ -vertex polygon, the *triangulation problem* is to partition the interior of the polygon into  $n-2$  triangles by adding  $n-3$  nonintersecting diagonals. We propose an  $O(n \log \log n)$ -time algorithm for this problem, improving on the previously best bound of  $O(n \log n)$  and showing that triangulation is not as hard as sorting. Improved algorithms for several other computational geometry problems, including testing whether a polygon is simple, follow from our result.

**Key words.** amortized time, balanced divide and conquer, heterogeneous finger search tree, homogeneous finger search tree, horizontal visibility information, Jordan sorting with error-correction, simplicity testing

**AMS(MOS) subject classifications.** 51M15, 68P05, 68Q25

**1. Introduction.** Let  $P$  be an  $n$ -vertex simple polygon, defined by a list  $v_0, v_1, \dots, v_{n-1}$  of its vertices in clockwise order around the boundary. (The interior of the polygon is to the right as one walks clockwise around the boundary.) We denote the boundary of  $P$  by  $\partial P$ . We assume throughout this paper (without loss of generality) that the vertices of  $P$  have distinct  $y$ -coordinates. For convenience we define  $v_n = v_0$ . The *edges* of  $P$  are the open line segments whose endpoints are  $v_i, v_{i+1}$  for  $0 \leq i < n$ . The *diagonals* of  $P$  are the open line segments whose endpoints are vertices and that lie entirely in the interior of  $P$ . The *triangulation problem* is to find  $n-3$  nonintersecting diagonals of  $P$ , which partition the interior of  $P$  into  $n-2$  triangles.

If  $P$  is convex, any pair of vertices defines a diagonal, and it is easy to triangulate  $P$  in  $O(n)$  time. If  $P$  is not convex, not all pairs of vertices define diagonals, and even finding one diagonal, let alone triangulating  $P$ , is not a trivial problem. In 1978, Garey, Johnson, Preparata and Tarjan [10] presented an  $O(n \log n)$ -time triangulation algorithm. Since then, work on the problem has proceeded in two directions. Some authors have developed linear-time algorithms for triangulating special classes of polygons, such as monotone polygons [10] and star-shaped polygons [31]. Others have devised triangulation algorithms whose running time is  $O(n \log k)$  for a parameter  $k$  that somehow quantifies the complexity of the polygon, such as the number of reflex angles [13] or the "sinuosity" [5]. Since these measures all admit classes of polygons with  $k = \Omega(n)$ , the worst case running time of these algorithms is only known to be  $O(n \log n)$ . Determining whether triangulation can be done in  $o(n \log n)$  time, i.e. asymptotically faster than sorting, has been one of the foremost open problems in computational geometry.

In this paper we propose an  $O(n \log \log n)$ -time triangulation algorithm, thereby showing that triangulation is indeed easier than sorting. The paper is a revised and corrected version of a conference paper [27] which erroneously claimed an  $O(n)$ -time algorithm. The goal of obtaining a linear-time algorithm remains elusive, but our

---

\*Received by the editors September 8, 1986; accepted for publication (in revised form) April 29, 1987. Typeset on July 28, 1987 at AT&T Bell Laboratories, Murray Hill, New Jersey.

†AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

‡Department of Computer Science, Princeton University, Princeton, New Jersey 08544. The work of this author was partially supported by National Science Foundation grant DCR-8605962.

approach suggests some directions in which to look and clarifies the difficulties that must be overcome.

The starting point for our algorithm is a reduction of the triangulation problem to the problem of computing visibility information along a single direction, which we take to be horizontal. A *vertex-edge visible pair* is a vertex and an edge that can be connected by an open horizontal line segment that lies entirely inside  $P$ . Similarly, an *edge-edge visible pair* is a pair of edges that can be connected by an open horizontal line segment that lies entirely inside  $P$ . Fournier and Montuno [9] showed that triangulating  $P$  is linear-time equivalent to computing all vertex-edge visible pairs. The reduction of triangulation to computing visible pairs was independently obtained by Chazelle and Incerpi [5]. What we shall actually produce is an  $O(n \log \log n)$ -time algorithm for computing visible pairs, which by this reduction leads to an  $O(n \log \log n)$ -time triangulation algorithm.

Our visibility algorithm computes not only vertex-edge visible pairs but also possibly some edge-edge visible pairs. It is reassuring that the total number of visible pairs of either kind is linear.

LEMMA 1. *There are at most  $2n$  vertex-edge visible pairs and at most  $2n$  edge-edge visible pairs.*

*Proof.* Each vertex can be in at most two vertex-edge visible pairs, for a total over all vertices of at most  $2n$ . Partition  $P$  into trapezoids and triangles by drawing a horizontal line segment between each visible pair (of either kind) through the interior of  $P$ . (See Figure 1.) Each edge-edge visible pair corresponds to the bottom boundary of exactly one such trapezoid or triangle, the top boundary of which is either one or two line segments corresponding to vertex-edge visible pairs (in the case of a trapezoid) or a vertex that is in no visible pairs (in the case of a triangle). A vertex

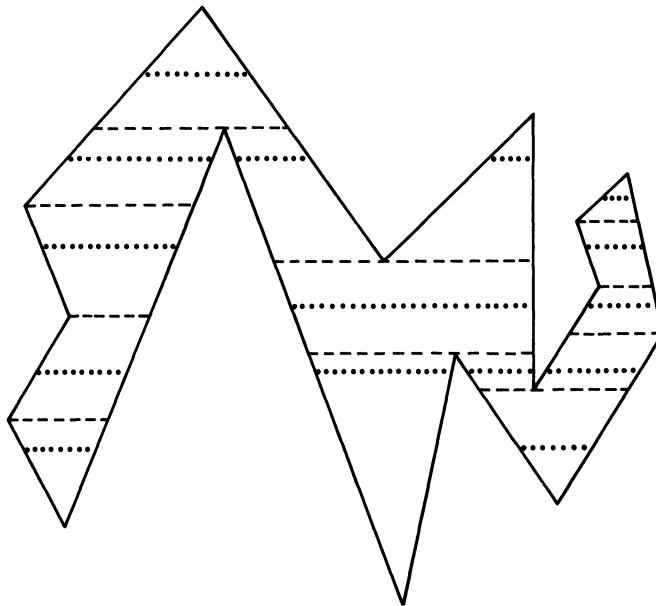


FIG. 1. A simple polygon  $P$ , showing visibility information: dashed horizontal lines correspond to vertex-edge visible pairs; dotted horizontal lines correspond to edge-edge visible pairs.

gives rise to at most two top boundary segments of trapezoids or to at most one top boundary of a triangle. Thus there are at most  $2n$  trapezoids and triangles whose bottom boundaries correspond to edge-edge visible pairs, and hence at most  $2n$  such pairs.

□

The second cornerstone of our method is the intimate connection between visibility computation and the Jordan sorting problem. For a simple polygon  $P$  and a horizontal line  $L$ , the *Jordan sorting problem* is to sort the intersection points of  $\partial P$  and  $L$  by  $x$ -coordinate, given as input only a list of the intersections in the order in which they occur clockwise around  $\partial P$ . (The list of vertices of  $P$  is *not* part of the input.) Hoffman, Mehlhorn, Rosenstiehl and Tarjan [14] have presented a linear-time Jordan sorting algorithm, which actually works for any simple curve, open or closed. This algorithm, which we call “the Jordan sorting algorithm,” requires that  $\partial P$  actually cross  $L$  wherever it touches it, but the algorithm is easily modified to handle tangent points, provided that each intersection point is labeled in the input as being either crossing or tangent.

Computing visible pairs is at least as hard as Jordan sorting, in the sense made precise in the following lemma:

**LEMMA 2.** *Using an algorithm to compute vertex-edge visible pairs, one can solve the Jordan sorting problem for an  $n$ -vertex polygon  $P$  in  $O(n)$  additional time, given as input the polygon and the line  $L$  (and not the intersections).*

*Proof.* Compute all vertex-edge visible pairs for  $P$ . Next, turn  $P$  “inside out” by breaking  $P$  at its lowest vertex and drawing a box around it as shown in Figure 2, forming a polygon  $Q$  with  $n+5$  vertices. Compute the vertex-edge visible pairs for  $Q$ . These pairs specify vertex-edge visibilities on the outside of  $P$ , and indicate which vertices of  $P$  can see arbitrarily far left or right on the outside. Partition the inside and outside of  $P$  into trapezoids, triangles, and unbounded trapezoidal regions by drawing horizontal line segments corresponding to each visible pair. Given a line  $L$ , the intersection points of  $\partial P$  and  $L$  can be read off in increasing  $x$ -order by moving left to right through the regions that intersect  $L$ . The total time for this algorithm, not including the two visibility computations, is  $O(n)$ . □

Since any visibility computation does Jordan sorting implicitly, it is natural to try using Jordan sorting explicitly to compute visible pairs. This leads to the following divide-and-conquer visibility algorithm (see Figure 3):

*Step 1.* Given  $P$ , choose a vertex  $v$  of  $P$  that does not have maximum or minimum  $y$ -coordinate. If no such  $v$  exists, stop: there are no visible pairs to compute. Otherwise, let  $L$  be the horizontal line through  $v$ .

*Step 2.* Determine the intersection points of  $\partial P$  and  $L$  in the order in which they occur along the boundary of  $P$ .

*Step 3.* Jordan sort the intersection points and report the visible pairs that correspond to consecutive intersection points along  $L$ .

*Step 4.* Slice  $P$  along  $L$ , dividing  $P$  into a collection of subpolygons.

*Step 5.* Apply the algorithm recursively to each subpolygon computed in Step 4.

The Jordan sorting algorithm has the fortuitous side effect of computing enough extra information so that Step 4 is easy. The hard part of the computation is Step 2. There are two major bottlenecks in the algorithm, either of which will make a naive implementation run in quadratic time. First, it is possible for the algorithm to report redundant visibility pairs; indeed, the example in Figure 4 shows that it can report  $\Omega(n^2)$  nondistinct pairs.

We eliminate this bottleneck by modifying Step 2 to compute only some of the intersections of  $\partial P$  with  $L$ . This can cause the Jordan sorting algorithm used in Step

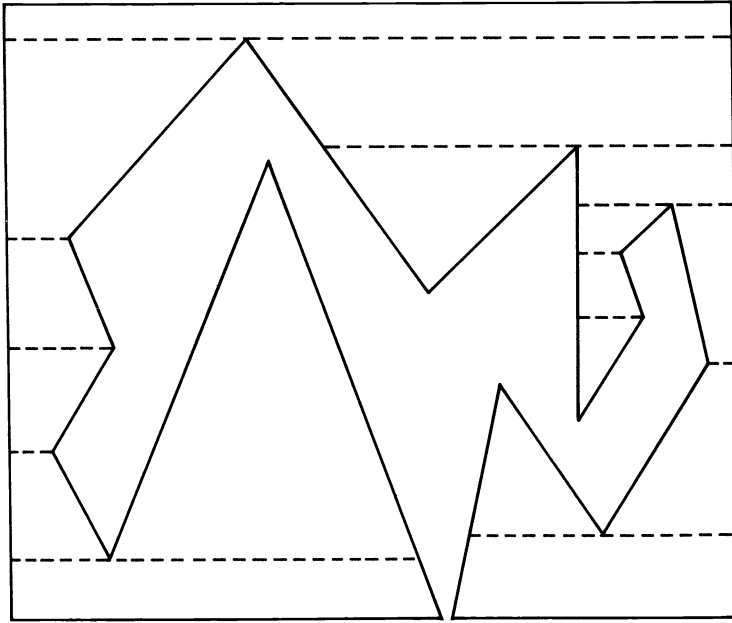


FIG. 2. Polygon  $P$  of Figure 1 turned "inside out," and showing exterior vertex-edge visible pairs.

3 to detect an error, since the sequence to be sorted need no longer consist of all intersections of a simple polygon with a line. Fortunately the sorting algorithm is incremental, and when it detects an error, we can restart it in a correct state by computing a few additional intersections and making local changes in its data structure. We call this augmented sorting method *Jordan sorting with error-correction*.

By computing only some of the intersections of  $\partial P$  and  $L$  and using Jordan sorting with error-correction, we obtain a visibility algorithm that reports only  $O(n)$  visible pairs and runs in  $O(n)$  time not counting the time needed to find intersections. This approach requires the use of a two-level data structure to represent polygon boundaries, but imposes no further constraints on the details of the data structure.

The second, far more serious bottleneck is the problem of actually finding the intersections. The line  $L$  divides  $\partial P$  into pieces. If each of these pieces ended up in a different subpolygon boundary, then we could obtain (with a little work) an overall  $O(n)$  time bound for the visibility algorithm by using finger search trees in the boundary data structure and appealing to the linearity of the following recurrence [20, p. 185]:

$$(1) \quad T(n) = \begin{cases} O(1) & \text{if } n = 1; \\ \max_{1 \leq k < n} \{T(k) + T(n-k) + O(1 + \log \min\{k, n-k\})\} & \text{if } n > 1. \end{cases}$$

Unfortunately the pieces of the original boundary do not stay apart but are regrouped to form the subpolygon boundaries. To beat the  $O(n \log n)$  time bound of previous triangulation algorithms, we need another idea, that of *balanced divide and conquer*. We refine the visibility algorithm to choose  $L$  judiciously, so that each of the subpolygon boundaries contains a relatively small number of pieces of the original



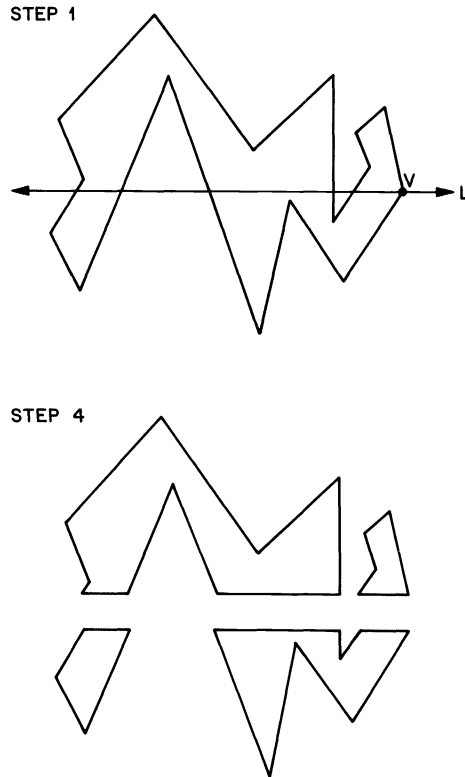


FIG. 3. Illustrating Steps 1 and 4 of the visibility algorithm.

boundary. Balanced divide and conquer combined with the use of finger search trees in the boundary data structure produces a visibility algorithm that runs in  $O(n \log \log n)$  time.

The remainder of this paper consists of five sections and an appendix. In Section 2 we review the Jordan sorting algorithm and modify it to do error-correction. In Section 3 we present a generic visibility algorithm, based on Jordan sorting with error-correction, that reports  $O(n)$  visible pairs. In Section 4 we refine the algorithm so that it uses balanced divide and conquer. In Section 5 we propose a data structure for representing the polygon boundary that consists of two levels of finger search trees. We show that with this data structure the visibility algorithm of Section 4 runs in  $O(n \log \log n)$  time. We close in Section 6 with some remarks, applications, and open problems. The appendix contains a discussion of finger search trees, which are needed not only in the visibility algorithm itself but also in the Jordan sorting algorithm.

**2. Jordan sorting with error-correction.** Let  $P$  be a simple polygon and let  $v$  be a vertex of  $P$ . Let  $L$  be the horizontal line through  $v$ , and let  $x_0 = v, x_1, \dots, x_{m-1}$  be the intersection points of  $\partial P$  and  $L$  in clockwise order around  $\partial P$ . (Since throughout this paper we are assuming that the vertices of  $P$  have distinct  $y$ -coordinates,  $\partial P$  and

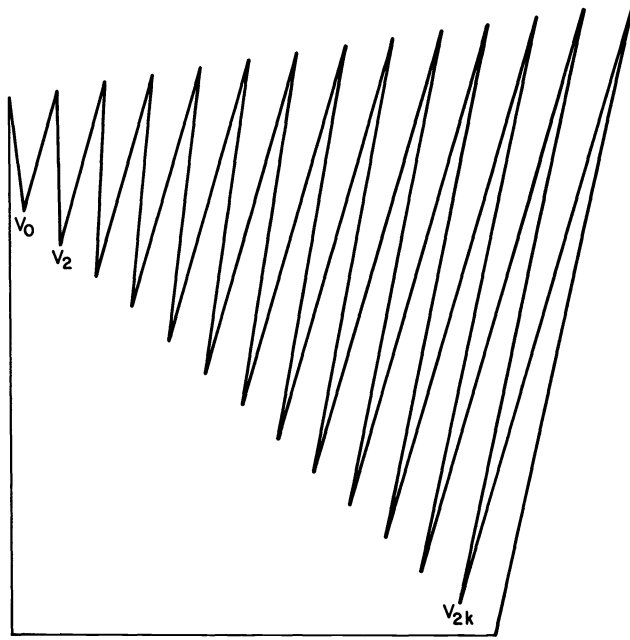


FIG. 4. This class of polygons can cause the naive algorithm to produce a quadratic amount of output. A first slice through  $v_0$  cuts off  $k+1$  triangles. Successive slices at  $v_{2i}$ ,  $i=1, 2, \dots, k$ , report  $k-i+1$  visible pairs, but only two are new each time.

$L$  intersect in a finite set of points.) Points  $x_1, x_2, \dots, x_{m-1}$  are crossing points of  $\partial P$  and  $L$ ; point  $x_0$  is either a crossing point or a tangent point. We impose a total order on the points  $x_i$  given by the order of their  $x$ -coordinates. We wish to sort  $x_0, x_1, \dots, x_{m-1}$  according to this total order.

The sequence  $x_0, x_1, \dots, x_{m-1}$  gives rise to two forests, in the following way. For convenience let  $x_m = x_0$ . Without loss of generality assume that the part of  $\partial P$  from  $x_0$  to  $x_1$  lies above  $L$ . For  $0 < i \leq m$ , let  $\ell_i = \min\{x_{i-1}, x_i\}$  and  $r_i = \max\{x_{i-1}, x_i\}$ . We say a pair  $\{x_{i-1}, x_i\}$  *encloses* a point  $x$  if  $\ell_i \leq x < r_i$ . We say two pairs  $\{x_{i-1}, x_i\}$  and  $\{x_{j-1}, x_j\}$  *cross* if  $\{x_{i-1}, x_i\}$  encloses exactly one of  $x_{j-1}$  and  $x_j$ ;  $\{x_{i-1}, x_i\}$  *encloses*  $\{x_{j-1}, x_j\}$  if it encloses both of  $x_{j-1}$  and  $x_j$ . The simplicity of  $P$  implies that if  $i \equiv j \pmod{2}$ , then the two pairs  $\{x_{i-1}, x_i\}$  and  $\{x_{j-1}, x_j\}$  do not cross. We call this the *noncrossing property*. The Hasse diagram of the “encloses” relation on the set of pairs  $\{\{x_{2i}, x_{2i+1}\} \mid 0 \leq i < m/2\}$  is a forest, called the *upper forest*. The Hasse diagram of “encloses” on the set of pairs  $\{\{x_{2i-1}, x_{2i}\} \mid 0 < i \leq m/2\}$  is also a forest, called the *lower forest*. We order each set of siblings in either forest by placing  $\{x_{i-1}, x_i\}$  before  $\{x_{j-1}, x_j\}$  if  $r_i \leq \ell_j$ . This makes each forest into an ordered forest. We make the two forests into trees by adding the dummy pair  $\{-\infty, \infty\}$  to each. Thus we obtain two trees, called the *upper tree* and the *lower tree*. (See Figure 5.) We call the set consisting of a parent in either tree and its children a *family*.

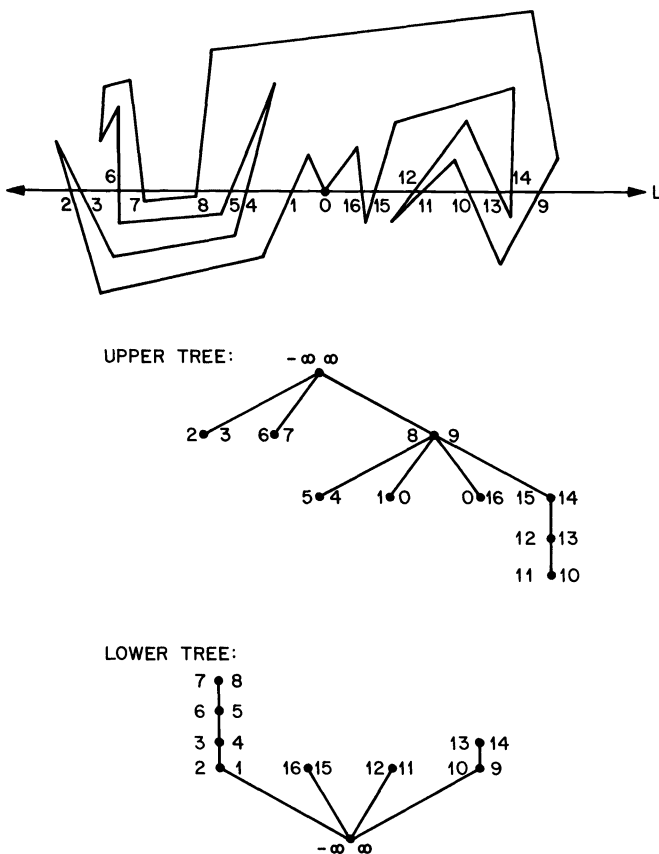


FIG. 5. Hasse diagram of the "encloses" relation with respect to line L. (Point  $x_i$  is labelled  $i$ .)

We shall restate the Jordan sorting algorithm [14] in a form suitable for extension to the visibility computation. The algorithm proceeds incrementally, processing the points  $x_1, x_2, \dots, x_m$  one at a time and building the upper tree, the lower tree, and a list of the points in sorted order. Initialization consists of making  $\{-\infty, \infty\}$  the only pair in both trees and defining the sorted list to be  $(-\infty, x_0, \infty)$ . The general step consists of processing point  $x_i$  by performing the following steps. Suppose  $i$  is odd, i.e.  $\{x_{i-1}, x_i\}$  is to be added to the upper tree. Assume  $x_{i-1} < x_i$ . (The case  $x_{i-1} > x_i$  is symmetric.)

- Step 1. Find the point  $x$  that follows  $x_{i-1}$  in the sorted list.
- Step 2. Find the pair  $\{x_{j-1}, x_j\}$  in the upper tree such that  $x \in \{x_{j-1}, x_j\}$ .
- Step 3. Apply the appropriate one of the following four cases (see Figure 6):
  - Case A ( $\ell_j < x_{i-1} < r_j < x_i$ ). Halt:  $\{x_{i-1}, x_i\}$  and  $\{x_{j-1}, x_j\}$  cross.
  - Case B ( $\ell_j < x_{i-1} < x_i \leq r_j$ ). Make  $\{x_{i-1}, x_i\}$  the new last child of  $\{x_{j-1}, x_j\}$ . If  $i < m$ , insert  $x_i$  after  $x_{i-1}$  in the sorted list.

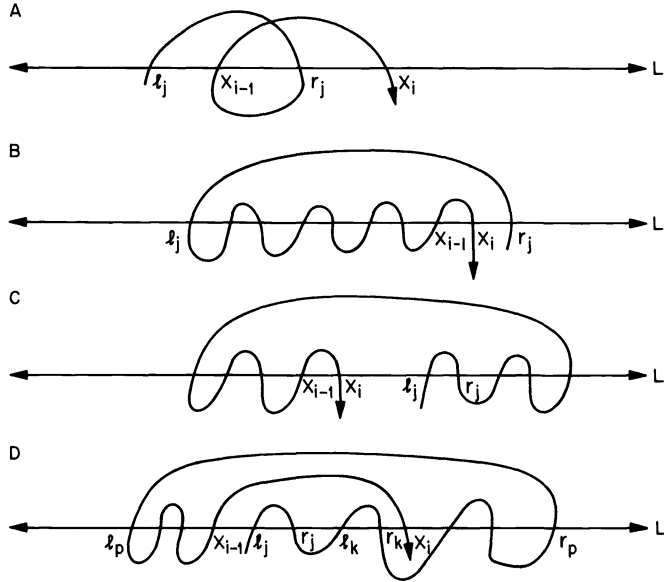


FIG. 6. The four cases for Jordan sorting.

Case C ( $x_i \leq \ell_j$ ). Insert  $\{x_{i-1}, x_i\}$  into the list of siblings of  $\{x_{j-1}, x_j\}$  just before  $\{x_{j-1}, x_j\}$ . If  $i < m$ , insert  $x_i$  after  $x_{i-1}$  in the sorted list.

Case D ( $x_{i-1} < \ell_j < x_i$ ). In the list of siblings of  $\{x_{j-1}, x_j\}$ , find the last one, say  $\{x_{k-1}, x_k\}$ , such that  $\ell_k < x_i$ . If  $r_k > x_i$ , halt:  $\{x_{i-1}, x_i\}$  and  $\{x_{k-1}, x_k\}$  cross. Otherwise, if  $\{x_{k-1}, x_k\}$  is the last child of its parent pair  $\{x_{p-1}, x_p\}$  and  $r_p < x_i$ , halt:  $\{x_{i-1}, x_i\}$  and  $\{x_{p-1}, x_p\}$  cross. If neither of these crossings is found, remove from the list of siblings of  $\{x_{j-1}, x_j\}$  the sublist from  $\{x_{j-1}, x_j\}$  to  $\{x_{k-1}, x_k\}$  (inclusive) and replace it by  $\{x_{i-1}, x_i\}$ . Make the removed sublist the list of children of  $\{x_{i-1}, x_i\}$ . If  $i < m$ , insert  $x_i$  after  $r_k$  in the sorted list.

Observe that if  $\{x_{i-1}, x_i\}$  and  $\{x_{j-1}, x_j\}$  are two pairs with  $i > j$  and  $i \equiv j \pmod 2$ , all four points  $x_{i-1}, x_i, x_{j-1}, x_j$  are distinct unless  $i = m$  and  $m$  is odd, in which case possibly  $x_i \in \{x_{j-1}, x_j\}$ . This means that Cases A-D exhaust the possible ordering relationships among the four points.

If  $i$  is even, i.e.  $\{x_{i-1}, x_i\}$  is to be added to the lower tree, the processing is analogous to the above, with a few changes needed to accommodate the fact that  $x_0$  is in no pair in the lower tree until  $x_m = x_0$  is processed (if then). The changes are as follows:

- (i) Just before Step 2, if  $x = x_0$ , replace  $x$  by the point that follows  $x_0$  in the sorted list.
- (ii) In Step 2, find the pair  $\{x_{j-1}, x_j\}$  that contains  $x$  in the lower tree (instead of in the upper tree).
- (iii) In Step 3, Cases B and C, if  $x_{i-1} < x_0 < x_i$ , insert  $x_i$  in the sorted list after  $x_0$  (instead of after  $x_{i-1}$ ).
- (iv) In Step 3, Case D, if  $r_k < x_0 < x_i$ , insert  $x_i$  in the sorted list after  $x_0$  (instead of after  $r_k$ ).

The sorting algorithm as stated tests for crossing pairs. If the input is guaranteed to be correct (i.e. to have the noncrossing property), we can simplify the algorithm by eliminating Case A and the two tests for crossing pairs in Case D.

Making the algorithm run in linear time requires the use of appropriate data structures. Each list of siblings in the upper and lower trees is represented by a homogeneous finger search tree (see the appendix) in which each leaf is a pair in the list. In addition, there are bidirectional pointers between each pair and its first and last children (in whichever tree contains the pair). Thus each family forms a doubly-linked circular list, with the additional property that any pair in the list can be accessed from any other pair  $d$  away in either direction in  $O(1 + \log d)$  time. Furthermore the amortized time<sup>1</sup> to insert a pair next to a given one in a family list is  $O(1)$ , and the amortized time to remove a sublist of  $d$  pairs from a list of  $s$  pairs, given the end pairs of the sublist, is  $O(1 + \log(\min\{d, s-d\} + 1))$ .

The running time of the Jordan sorting algorithm is dominated by the time spent doing "remove a sublist/insert a pair" operations on family lists. Let  $T(p,s)$  be the maximum amortized time needed to do a total of  $p$  such operations on an initial list of size  $s$  and on the removed sublists. Then  $T(p,s)$  obeys the following recurrence:

$$(2) \quad T(p,s) = \begin{cases} 0 & \text{if } p=0; \\ \max_{\substack{0 \leq i < p \\ 0 \leq d \leq s}} \{T(i,d+1) + T(p-i-1, s-d+1) \\ \quad + O(1 + \log(\min\{d, s-d\} + 1))\} & \text{if } p > 0. \end{cases}$$

A proof by induction shows that  $T(p,s) = O(p+s)$ . The list manipulation time of the Jordan sorting algorithm is at most  $T(\lceil m/2 \rceil, 1) + T(\lfloor m/2 \rfloor, 1)$ , from which it follows that the algorithm runs in  $O(m)$  time. For further details of the algorithm and the analysis see the original paper [14]. (In our restatement of the algorithm, we have modified the data structure slightly, the main change being to eliminate circular level links in the finger search trees. These changes do not affect the  $O(m)$  time bound.)

We now want to augment the Jordan sorting algorithm so that when it detects two crossing pairs, it can in certain cases restart itself in a corrected state that represents the partial sorting of a sequence modified to eliminate the crossing. In the application of Jordan sorting to the visibility computation, the sorting algorithm receives as input only a possibly noncontiguous subsequence of the sequence of intersections. In this subsequence, certain pairs are designated as *special*. (All other pairs are *normal*.)

To accommodate the operation of the triangulation algorithm, we impose on each special pair  $\{x_{i-1}, x_i\}$  the additional requirement that it enclose no given intersections other than  $x_{i-1}$  and  $x_i$ . We call this the *nonenclosure property*. On the other hand, special pairs are potentially modifiable: if  $\{x_{i-1}, x_i\}$  is a special pair, there may be additional intersections between  $x_{i-1}$  and  $x_i$  along  $\partial P$ . The Jordan sorting algorithm is allowed to request such additional intersections if it detects a crossing or a violation of the nonenclosure property.

The mechanism for providing additional intersections is a procedure named *refine*, whose input parameters consist of a special pair  $\{x_{i-1}, x_i\}$  and a point  $x$  enclosed by

---

<sup>1</sup>Amortized time is the time per operation averaged over a worst-case sequence of operations that begins with an empty data structure. For a discussion of this concept see the first author's survey paper [26].

$\{x_{i-1}, x_i\}$ . The procedure returns a *bracketing pair*  $x', x''$  such that  $x'$  and  $x''$  are intersections of  $\partial P$  and  $L$ , the four intersections occur in the order  $x_{i-1}, x', x'', x_i$  along  $\partial P$ , and the five points occur in the order  $x_{i-1}, x', x, x'', x_i$  (or its reverse) along  $L$ . If there is no such pair, *refine* returns nothing. If a pair is returned, the new sequence to be sorted is the old sequence with  $x'$  followed by  $x''$  inserted between  $x_{i-1}$  and  $x_i$ . Of the three pairs that replace  $\{x_{i-1}, x_i\}$ , the pairs  $\{x_{i-1}, x'\}$  and  $\{x'', x_i\}$  are special and the pair  $\{x', x''\}$  is normal. (This means that  $\{x', x''\}$  is the *closest* pair of intersections to  $x$ .)

We shall modify the Jordan sorting algorithm so that it can handle special pairs, using *refine* when possible to eliminate violations of the noncrossing and nonenclosure properties. The modification consists of the following three additions to the algorithm (see Figure 7):

(i) In Step 1, if  $\{x_{i-1}, x_i\}$  is special and  $x < x_i$ , call *refine* ( $\{x_{i-1}, x_i\}, x$ ). If *refine* returns no pair, halt:  $\{x_{i-1}, x_i\}$  violates the nonenclosure property. If *refine* returns a pair  $(x', x'')$ , insert  $x'$  and  $x''$  in the sequence to be sorted between  $x_{i-1}$  and  $x_i$  and restart the processing with  $x'$ .

(ii) In Step 3, Case B, if  $\{x_{j-1}, x_j\}$  is special, halt: the nonenclosure property has been violated. Even if it can be restored by refining  $\{x_{j-1}, x_j\}$ , this will produce a violation of the noncrossing property. (This also happens in Step 3, Case A: even if

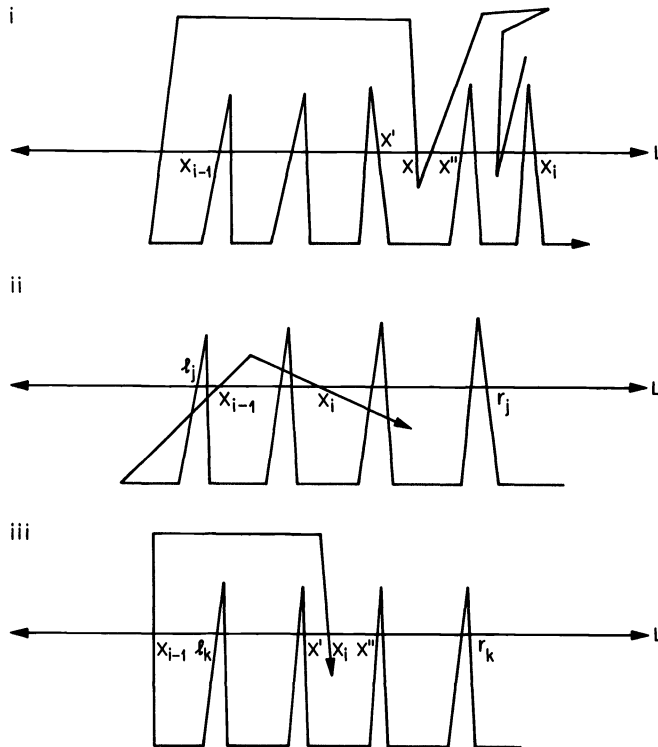


FIG. 7. Modifications to make Jordan sorting error-correcting.

$\{x_{j-1}, x_j\}$  is special and the crossing of  $\{x_{i-1}, x_i\}$  and  $\{x_{j-1}, x_j\}$  can be eliminated by refining  $\{x_{j-1}, x_j\}$ , this will produce a new crossing pair.)

(iii) In Step 3, Case D, if  $\{x_{k-1}, x_k\}$  is special and  $r_k > x_i$ , do not halt, but instead call *refine* ( $\{x_{k-1}, x_k\}, x_i$ ). If *refine* returns no pair, halt:  $\{x_{k-1}, x_k\}$  violates the nonenclosure property. If *refine* returns a pair  $(x', x'')$ , replace  $\{x_{k-1}, x_k\}$  in the upper forest by  $\{x_{k-1}, x'\}$  followed by  $\{x'', x_k\}$ . Insert  $\{x', x''\}$  in the appropriate place in the lower forest (as a sibling or child of the pair containing  $x_k$ , whichever is appropriate). Proceed as in the remainder of Step 3, Case D, using  $\{x_{k-1}, x'\}$  in place of  $\{x_{k-1}, x_k\}$  and also in place of  $\{x_{j-1}, x_j\}$  if  $\{x_{j-1}, x_j\} = \{x_{k-1}, x'\}$ . That is, if  $\{x_{j-1}, x_j\} = \{x_{k-1}, x_k\}$ , replace  $\{x_{j-1}, x'\}$  in its list of siblings by  $\{x_{i-1}, x_i\}$  and make  $\{x_{j-1}, x'\}$  a child of  $\{x_{i-1}, x_i\}$ . If  $\{x_{j-1}, x_j\} \neq \{x_{k-1}, x_k\}$ , replace the sublist from  $\{x_{j-1}, x_j\}$  to  $\{x_{k-1}, x'\}$  (inclusive) by  $\{x_{i-1}, x_i\}$ , and make the sublist the list of children of  $\{x_{i-1}, x_i\}$ . In either case insert  $x_i$  after  $x'$  in the sorted list (or after  $x_0$  if  $x' < x_0 < x_i$ ).

The correctness of the error-correcting Jordan sorting algorithm follows from the observation that, while the algorithm is running, a special pair  $\{x_{j-1}, x_j\}$  can enclose at most one intersection point  $x_i \notin \{x_{j-1}, x_j\}$ . To see this, suppose without loss of generality that  $\{x_{j-1}, x_j\}$  is a pair in the upper tree. An intersection  $x_i \notin \{x_{j-1}, x_j\}$  can be inserted between  $x_{j-1}$  and  $x_j$  in the sorted list because of the addition of a pair  $\{x_{i-1}, x_i\}$  to the lower tree, but the violation of the enclosure property will be detected when the point  $x_{i+1}$  is processed, as illustrated in Figure 7(ii).

The additions necessary to make the algorithm error-correcting cost only  $O(1)$  time per point processed and per refinement, not including the time spent inside calls of *refine*. (There are at most two insertions in sibling lists per refinement.) Thus the error-correcting algorithm runs in  $O(m)$  time, where  $m$  is the number of intersection points in the final refined sequence. In the next section, we shall see how error-correcting Jordan sorting can be used to compute visible pairs.

**3. An efficient visibility algorithm.** Our algorithm for computing visible pairs follows the outline laid out in Section 1. It is a divide-and-conquer method that cuts up the original polygon into subpolygons, cuts these into smaller subpolygons, and so on, until none of the subpolygons can be further divided. In order to present the details of the method, we must first discuss the structure of the subpolygons, which we call *visibility regions*. The interior of a visibility region is a simply connected subset of the original polygon interior contained between two horizontal lines, denoted by  $y = y_{\min}$  and  $y = y_{\max}$  (with  $y_{\min} < y_{\max}$ ). We require that the region boundary actually intersect both of these lines. (See Figure 8.)

The boundary of a visibility region consists of connected pieces of the boundary of the original polygon, called *boundary segments*, alternating with segments of the lines  $y = y_{\min}$  and  $y = y_{\max}$ . Each such horizontal segment that is not a single point corresponds to a visible pair. At most one vertex of the original polygon lies on each of the lines  $y = y_{\min}$  and  $y = y_{\max}$ . Although a visibility region is itself a simple polygon, when we speak of its vertices we mean *only* those that are vertices of the original polygon  $P$ . A boundary segment begins with a vertex or part of an edge, called a *partial edge*, and ends with a vertex or partial edge. We call the edge of the original polygon that contains such a partial edge an *end edge* of the segment.

We divide the boundary segments into three types:

*top*: no end edge or vertex intersects the line  $y = y_{\min}$ ;

*bottom*: no end edge or vertex intersects the line  $y = y_{\max}$ ;

*side*: one end edge or vertex intersects the line  $y = y_{\max}$  and one intersects the line  $y = y_{\min}$ .

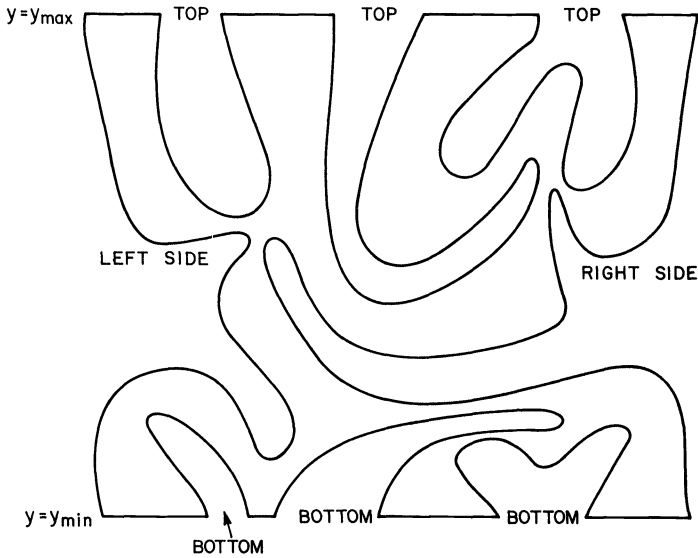


FIG. 8. Schematic illustration of a visibility region. Each curve represents a segment of the polygon boundary.

The degenerate case of a top or bottom boundary segment is a single vertex and no partial edges; the degenerate case of a side boundary segment is a single partial edge and no vertices. Clockwise around the boundary of a region, the boundary segments consist of four contiguous parts: a set of top segments, which together with the adjacent pieces of the line  $y = y_{\max}$  forms the *top* of the boundary; a side segment, which forms the *right side* of the boundary; a set of bottom segments, which together with the adjacent pieces of the line  $y = y_{\min}$  forms the *bottom* of the boundary; and another *side segment*, which forms the *left side* of the boundary. Both side segments must be present; either the top or the bottom or both can be empty.

We shall represent a visibility region by specifying  $y_{\min}$  and  $y_{\max}$  and the four parts of the boundary (left, right, top, and bottom). We represent the top and the bottom by lists of the boundary segments they contain, in clockwise order around the boundary. We represent the left and right sides by their single boundary segments. Finally, we represent each boundary segment by a list of the vertices in it, in clockwise order around the boundary, together with its end edges (if any). We leave unspecified the implementation of the lists that represent the boundary segments and the top and bottom boundaries of the region; this is the topic of Section 5.

Having discussed the structure of visibility regions, we now need to introduce some terminology concerning the intersections of the boundary of a region with a horizontal line. (See Figure 9.) Let  $V$  be a visibility region and let  $L$  be a horizontal line that intersects its interior. We partition the boundary segments of  $V$  into three types, depending on their relationship to  $L$ :

*shallow*: a segment that does not intersect  $L$ ;



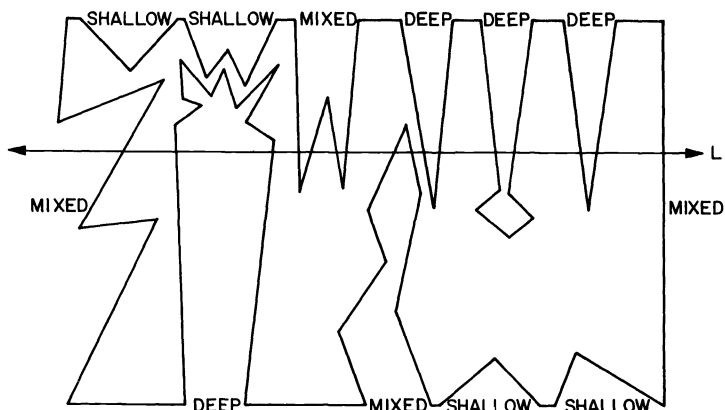


FIG. 9. Illustrating three kinds of boundary segment. The top group includes a deep section of three segments. Both top and bottom groups include shallow sections of two segments.

*deep*: a top segment whose vertices are all strictly below  $L$  or a bottom segment whose vertices are all strictly above  $L$ ;

*mixed*: any other segment.

A side segment is definitely mixed; a top or bottom segment can be of any type. We define a *shallow section* to be a (contiguous) sublist of shallow boundary segments in the list of boundary segments clockwise around  $\partial V$ ; we define a *deep section* similarly. A deep or shallow section consists entirely of top segments or entirely of bottom segments. Each of the partial edges of a deep section intersects  $L$  and these are the only intersections of the section with  $L$ . We classify the intersections of  $\partial V$  with  $L$  into two types:

*nonessential*: an intersection within a maximal deep section that is not the first or the last within the section;

*essential*: any other intersection.

The last issues we must discuss before presenting the visibility algorithm are the notion of a *special pair* and the effect of the *refine* procedure, both of which affect the running of the error-correcting Jordan sorting algorithm. A pair of intersections of  $\partial V$  and  $L$  is *special* if the intersections are the first and last in some deep section (along  $\partial V$ ) and *normal* otherwise. (Observe that the intersections of a deep section with  $L$  occur in the same order along  $L$  as they do along  $\partial V$ , or in reverse order.) A call *refine* ( $\{x_{i-1}, x_i\}, x$ ) has the following effect. Points  $x_{i-1}$  and  $x_i$  are the first and last intersections in some deep section, say  $S$ . If  $S$  can be split into two deep sections  $S_1$  and  $S_2$  with first and last intersections  $x_{i-1}, x'$  and  $x'', x_i$ , respectively, such that  $\{x', x''\}$  encloses  $x$ , then *refine* returns  $(x', x'')$ . If  $S$  cannot be so split, *refine* returns no pair. Observe that if a pair  $(x', x'')$  is returned,  $\{x_{i-1}, x'\}$  and  $\{x'', x_i\}$  are special pairs and  $(x', x'')$  is a normal pair, as required by the Jordan sorting algorithm.

There is one more crucial observation about special pairs. Consider a special pair  $\{x_{i-1}, x_i\}$  that comprises the first and last intersections of a single deep boundary segment. If the segment is a top segment  $T$ , then  $T$  together with the appropriate seg-

ment of the line  $y = y_{\max}$  forms a simple closed curve whose interior contains the line segment joining  $x_{i-1}$  and  $x_i$  and whose exterior contains the boundary of  $V$  other than  $T$ . By the Jordan curve theorem,  $\{x_{i-1}, x_i\}$  can enclose no intersections other than  $x_{i-1}$  and  $x_i$ . Thus every special pair either can be refined or has the nonenclosure property, as required by the Jordan sorting algorithm.

We are at last ready to discuss the visibility algorithm itself. The input to the algorithm is a single visibility region  $V$ . To apply the algorithm to the original polygon, we convert the polygon into a visibility region by dividing its boundary into two side boundary segments whose end vertices are the vertices of minimum and maximum  $y$ -coordinate. The  $y$ -coordinates of these two vertices become  $y_{\min}$  and  $y_{\max}$  for the region. The algorithm is the same as that in Section 1 except that it computes only the essential intersections of  $\partial V$  and  $L$  in Step 2 and uses Jordan sorting with error-correction in Step 3. That is, it consists of the following five steps:

*Step 1.* Given  $V$  with bounding lines  $y = y_{\min}$  and  $y = y_{\max}$ , choose a vertex of  $V$  having  $y$ -coordinate  $y_{\text{cut}}$  such that  $y_{\min} < y_{\text{cut}} < y_{\max}$ . If there is no such  $v$ , stop: there are no visible pairs to compute. Otherwise, let  $L$  be the line  $y = y_{\text{cut}}$ .

*Step 2.* Find the essential intersections of  $\partial V$  and  $L$  in the order in which they occur along  $\partial V$ .

*Step 3.* Use Jordan sorting with error-correction to sort by  $x$ -coordinate the essential intersections and any others introduced by refinement. Report the visible pairs corresponding to consecutive sorted intersections along  $L$ .

*Step 4.* Slice  $V$  along  $L$ , dividing  $V$  into a collection of subregions.

*Step 5.* Apply the algorithm recursively to each subregion formed in Step 4.

The observations made above concerning special pairs and refinement imply that the Jordan sorting step works correctly; any nonessential intersection occurs along  $\partial V$  between the members of a special pair and is available by refinement if needed.

The last detail we must fill in before undertaking an analysis of the algorithm is the effect of Step 4. The boundaries of the subregions are formed as follows. (See Figure 10.) Split  $\partial V$  at each of the intersections sorted in Step 3. Each of the pieces so formed corresponds to a pair in the upper or lower tree constructed by the Jordan sorting algorithm. Each family in each of the trees whose parent has odd depth (counting the dummy roots as being of depth zero) corresponds to a subregion. The boundary of the subregion consists of the pieces of  $\partial V$  that correspond to the pairs of the family, in the order in which the members of the family occur in the family list in the lower tree or the reverse of this order in the upper tree, interspersed with appropriate segments of the line  $y = y_{\text{cut}}$ , *with one crucial exception*: if  $D$  is a piece of  $\partial V$  that corresponds to a deep boundary section, before using  $D$  as part of a subregion boundary, each of its segments of the line  $y = y_{\min}$  or  $y = y_{\max}$  must be replaced by a corresponding segment of the line  $y = y_{\text{cut}}$ . Observe that this has *no* effect on the representation of the deep boundary section, which means that no change in the data structure representing the section is necessary. Furthermore the visibility regions cut off by this replacement and henceforth ignored are trapezoids, on which the visibility algorithm would terminate in Step 1 were it invoked on them. The visibility algorithm obtains its efficiency by avoiding any computation associated with these trivial trapezoids.

We define  $y_{\min}$  and  $y_{\max}$  for the subregions as follows. Consider a subregion above  $L$ , which corresponds to a family in the upper tree. The value of  $y_{\min}$  for this subregion is  $y_{\text{cut}}$ . The value of  $y_{\max}$  for the subregion is  $y_{\max}$  for the region if the parent of the family is at depth one in the upper tree, or otherwise the maximum  $y$ -coordinate of a vertex in the piece of  $\partial V$  corresponding to the parent of the family. In

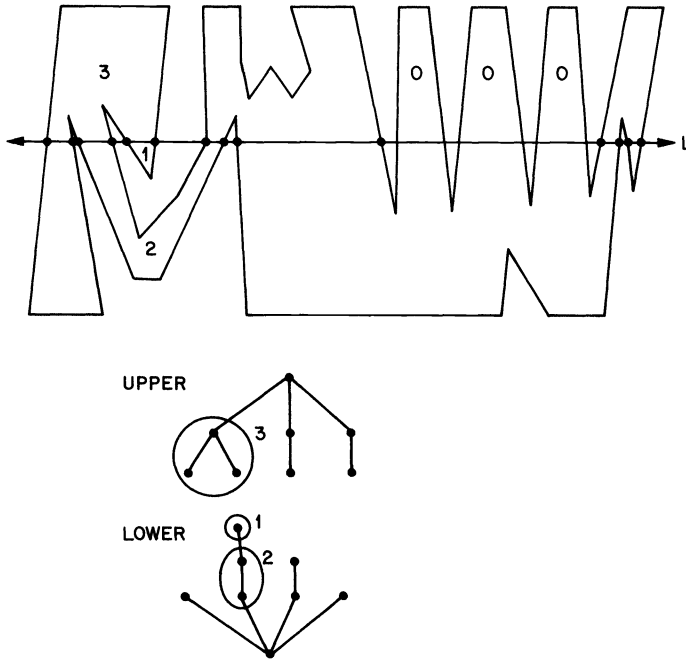


FIG. 10. *Assembling subregions. The family tree nodes associated with several regions are shown. Trapezoids labeled "0" are ignored completely.*

the latter case, the piece of  $\partial V$  corresponding to the parent consists of a single piece of a boundary segment of  $V$ . We split this piece at the vertex with maximum  $y$ -coordinate to form two pieces, which become the side boundary segments of the subregion; the top of the subregion is empty. The definitions are symmetric for subregions below  $L$ .

Let us restate the difference between the algorithm above and the one outlined in Section 1. In the former, a maximal deep section is treated as if it had only two intersections with  $L$  (the essential ones), until it is discovered that some intersection not in the section is enclosed by these two. This approximation to the truth works because the intersections in the section occur in the same order along the section as they do along  $L$  (or in reverse order). If no "foreign" intersections intervened, the algorithm of Section 1 would merely chop the section between each pair of contiguous boundary segments in Step 2 and put them back together in exactly the same order in Step 4. The new algorithm avoids this unnecessary work.

We now want to quantify the work saved by the new algorithm. Our main result is that the total number of visible pairs reported during the processing of an  $n$ -vertex polygon is  $O(n)$ . This implies that the time spent doing Jordan sorting, not including calls of *refine*, is  $O(n)$ .

LEMMA 3. *The processing of an  $n$ -vertex polygon requires at most  $n-2$  invocations of Steps 2-4.*

*Proof.* Each vertex except the ones of maximum and minimum  $y$ -coordinate can be selected as  $v$  in Step 1 at most once.  $\square$

LEMMA 4. Consider a single invocation of Step 3. Let  $k$  be the number of visible pairs reported during this invocation that were not reported during previous invocations. The total number of intersections sorted during this invocation, including those introduced by refinement, is  $O(k+1)$ .

*Proof.* First we count the essential intersections. We call an essential intersection  $x$  good if  $x$  is a vertex (i.e.  $x=v$ ) or if the two vertices preceding and following  $x$  along  $\partial V$ , say  $v'$  and  $v''$ , are in the same part of  $\partial V$  (top, bottom, left, or right) and not strictly on the same side of  $L$ . Otherwise  $x$  is bad. We claim that if  $x$  is a good intersection other than  $v$ , then the edge that contains  $x$  belongs to a newly reported visible pair. This is true if the part of  $\partial V$  from  $v'$  to  $v''$  consists of the line segment joining  $v'$  and  $v''$ , since the visible pair containing the edge from  $v'$  to  $v''$  that will be reported is the first one reported containing that edge. It is also true if the part of  $\partial V$  from  $v'$  to  $v''$  consists of a partial edge from  $v'$  to the line  $y = y_{\max}$  (or  $y = y_{\min}$ ), a segment of the line  $y = y_{\max}$  (or  $y = y_{\min}$ , respectively), and a partial edge from the line  $y = y_{\max}$  (or  $y = y_{\min}$ , respectively) to  $v''$ . To see this, suppose without loss of generality that  $v'$  is strictly below  $L$  and  $\partial V$  contains a partial edge from  $v'$  to the line  $y = y_{\max}$ . (See Figure 11.) Intersection  $x$  is on this partial edge. Along the line  $L$ , the edge from  $v'$  sees something other than the edge into  $v''$ , and thus will be contained in a newly reported visible pair, since if two edges see each other horizontally, the part of each edge that sees the other is connected. Thus in either case the claim is true. The claim implies that the number of good intersections is  $O(k+1)$ .

Consider the bad intersections. Any mixed boundary segment contains at most two bad intersections (the first and last in the segment) and, if it is a top or bottom segment, at least one good intersection. Any maximal deep section contains at most two bad intersections (the first and last in the section). Such a section either (i) is followed by a shallow segment, (ii) is followed by a mixed top or bottom boundary segment, or (iii) contains the last segment on its side. In case (i) the last intersection is good. The number of bad intersections in case (ii) is  $O(k+1)$ , since the mixed segment contains at least one good intersection. At most four bad intersections (the last two within each of the top and bottom boundaries) can fall into case (iii). There are also at most two bad intersections within each of the left and right sides. Thus the number of bad intersections is  $O(k+1)$ .

It remains for us to count the nonessential intersections introduced by refinement. Suppose that a call *refine*  $(\{x_{i-1}, x_i\}, x)$  returns a pair  $x', x''$ . Both of the edges that contain  $x'$  and  $x''$  will be contained in newly reported visible pairs, since along  $L$  they

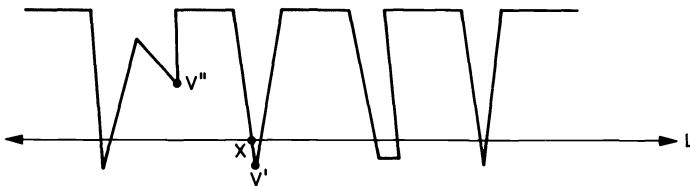


FIG. 11. The edge that contains  $x$  belongs to a newly reported visible pair.

see something other than each other. Thus the number of intersections introduced by refinement is  $O(k)$ .  $\square$

The following theorem summarizes our analysis of the algorithm so far.

**THEOREM 1.** *In processing an  $n$ -vertex polygon, the visibility algorithm reports  $O(n)$  visible pairs and spends  $O(n)$  time in Jordan sorting, not including its calls to refine. This includes all processing of subregions.*

*Proof.* Immediate from Lemmas 1, 3 and 4.  $\square$

**4. Use of balanced divide and conquer.** The visibility algorithm of Section 3 can in the worst case generate regions containing a total of  $\Omega(n^2)$  boundary segments (counting a boundary segment each time it occurs in a region). As we shall see in Section 5, obtaining an  $O(n \log \log n)$ -time implementation of the algorithm requires reducing the total number of boundary segments to  $O(n \log n)$ . We do this by refining the algorithm so that it uses a balanced divide-and-conquer strategy. The refinement consists of choosing the vertex  $v$  carefully in Step 1. Roughly speaking, we want to slice a region with many boundary segments so that at most a proper fraction of the segments end up in any one of the subregions. A choice that works and that can be made quickly is the following:

*Choose splitting vertex.* Let region  $V$  have  $t$  top boundary segments and  $b$  bottom boundary segments. Suppose  $t \geq b$ . (Otherwise, proceed symmetrically.) If  $t \leq 2$ , choose any vertex  $v$  whose  $y$ -coordinate is not in  $\{y_{\min}, y_{\max}\}$ . Otherwise, divide the list of top boundary segments into three sublists, with the first and last containing  $\lfloor t/3 \rfloor$  segments and the middle one the remainder. Among the segments in the middle sublist, choose as  $v$  the vertex whose  $y$ -coordinate is minimum.

We call the visibility algorithm refined to use this selection strategy the *balanced division algorithm*; we call the algorithm with an arbitrary selection strategy the *generic algorithm*. In the analysis to follow, we regard two boundary segments as distinct only if their vertex sets or their end edges (if there are any end edges) are different.

**LEMMA 5.** *In processing an  $n$ -vertex polygon, the generic visibility algorithm creates  $O(n)$  distinct boundary segments altogether.*

*Proof.* The only way the algorithm can create new boundary segments is by splitting an old boundary segment in two, at an intersection point that is input to the Jordan sorting algorithm or at the vertex of maximum or minimum  $y$ -coordinate in a subregion. By Theorem 1 there are  $O(n)$  such splitting points. Hence there are  $O(n)$  distinct boundary segments.  $\square$

**LEMMA 6.** *Let  $V$  be a region that has  $s$  boundary segments. If  $V$  is sliced using balanced division, then no subregion contains more than  $7s/8$  of the original boundary segments of  $V$ .*

*Proof.* Let  $V$  contain  $t$  top boundary segments and  $b$  bottom boundary segments. We have  $s = b + t + 2$ . Suppose without loss of generality that  $t \geq b$ . If  $t \leq 2$  the lemma is immediate, since some boundary segment is cut into new boundary segments distinct from the original; thus the original appears in no subregion. Suppose  $t \geq 3$ . Consider where the top boundary segments of  $V$  end up after slicing. A segment that is mixed with respect to the slicing line  $L$  is cut into new boundary segments distinct from the original; thus the original appears in no subregion. There are at most  $2 \lfloor t/3 \rfloor$  deep top boundary segments, which implies that each subregion below  $L$  contains at most  $b + 2t/3 + 2 \leq 7s/8$  of the boundary segments of  $V$ . A deep top segment cannot end up in a subregion above  $L$ ; only a part of it, constituting a distinct boundary segment, can. Thus among the top segments, only the shallow ones can end up in regions above  $L$ . A set of shallow segments can end up in the same subregion only if it forms a shallow section. By the choice of  $L$ , any such shallow section can contain at

most  $t - \lfloor t/3 \rfloor - 1 \leq 2t/3$  top boundary segments of  $V$ . Thus any subregion above  $L$  contains at most  $b + 2t/3 + 2 \leq 7s/8$  of the boundary segments of  $V$ .  $\square$

**THEOREM 2.** *When the balanced division algorithm processes an  $n$ -vertex polygon, the sum over all regions of the number of boundary segments per region is  $O(n \log n)$ .*

*Proof.* One way to prove this theorem is to write down a recurrence based on Lemma 6 and solve it. Instead, we shall use an amortization argument based on a credit analysis (see [26]). When the balanced division algorithm creates a distinct new boundary segment, we assign  $15 \log_{16/15} n$  credits to the segment. Each subsequent time that the segment appears in a subregion, we remove a credit. We shall show that the number of credits always remains nonnegative, which implies by Lemma 5 that the sum over all regions of the number of boundary segments is  $O(n \log n)$ .

To show that the number of credits remains nonnegative, we actually prove the following stronger *credit invariant*: a region that has  $s$  boundary segments has at least  $s \log_{16/15} s$  credits. The invariant is certainly true initially. Suppose it is true before some invocation of Steps 1-4. Let  $V$  be the region to be subdivided, and let  $s$  be its number of boundary segments. Before the subdivision,  $V$  has at least  $s \log_{16/15} s$  credits, of which we allocate  $\log_{16/15} s$  to each boundary segment. One of these pays for the appearance of the segment in  $V$ , leaving  $(\log_{16/15} s) - 1 = \log_{16/15} (15s/16)$  as its contribution to the credits of the subregion in which it appears. Consider a subregion  $V'$  formed by splitting  $V$ . Suppose its boundary contains  $p$  of the boundary segments of  $V$  and  $q$  newly created boundary segments; let  $s' = p + q$ . To verify that  $V'$  has  $s' \log_{16/15} s'$  credits, there are two cases to consider. If  $q > s'/15$ , then the total number of credits is at least the number of credits assigned to new segments:

$$15q \log_{16/15} n \geq s' \log_{16/15} n \geq s' \log_{16/15} s'.$$

If  $q \leq s'/15$ , then since  $p \leq 7s/8$  by Lemma 6, we have  $s' \leq 15s/16$ . The total number of credits is

$$\begin{aligned} p \log_{16/15} (15s/16) + 15q \log_{16/15} n &\geq p \log_{16/15} (15s/16) + 15q \log_{16/15} (15s/16) \\ &\geq s' \log_{16/15} (15s/16) \geq s' \log_{16/15} s'. \end{aligned}$$

By induction on the number of steps, the credit invariant is always true, from which the theorem follows.  $\square$

**5. Representation of the boundary using finger search trees.** We have now almost completed our presentation of the visibility algorithm. The task that remains is to choose a data structure for the lists that represent the boundary segments and boundary groups, and to analyze the effects of this choice. To represent both kinds of lists we use heterogeneous finger search trees (see the appendix). As we shall see, this gives an overall  $O(n \log \log n)$  running time for the balanced division visibility algorithm.

We represent each boundary segment by a heterogeneous finger search tree in which each leaf contains a vertex in the segment. Left-to-right order in the tree corresponds to the order of the vertices along the segment. In addition, if the segment has one or two end edges, we store these edges with the tree. Within the tree, we maintain two heap orders, both with respect to the  $y$ -coordinates of the vertices. One is by increasing  $y$ -coordinate, the other is by decreasing  $y$ -coordinate. That is, each node in the tree contains the maximum and minimum  $y$ -coordinates of all leaves reachable from it in the tree. (See Figure 12.) This allows us, for any given value of  $y$ , to find the leftmost (or rightmost) vertex with  $y$ -coordinate less than (or greater

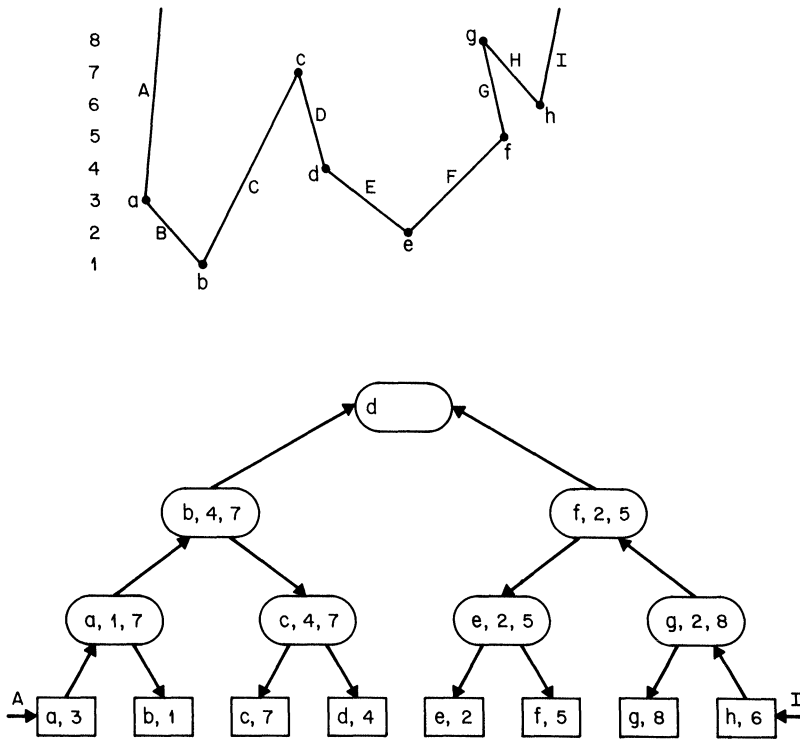


FIG. 12. A boundary segment and its representation as a heterogeneous finger search tree. (The drawing conventions are explained in Figure 22.)

than) the given value, in  $O(1 + \log(\min\{d, s-d\} + 1))$  time, where  $s$  is the total number of vertices stored in the tree and the one found is the  $d$ th. We can also find the vertex of maximum (or minimum)  $y$ -coordinate in the same time. The amortized time to split the tree at the  $d$ th out of  $s$  vertices is also  $O(1 + \log(\min\{d, s-d\} + 1))$ .

We represent each list of top boundary segments or bottom boundary segments constituting the top or bottom boundary of a region by a heterogeneous finger search tree in which each leaf represents a boundary segment. Left-to-right order in the tree corresponds to the order of the boundary segments clockwise around the boundary. Each leaf contains a pointer to the tree representing the corresponding boundary segment, as well as the end vertices or edges of the segment, and the maximum and minimum  $y$ -coordinates of the vertices within the segment. We think of each node in the tree as representing the section (sublist of boundary segments) corresponding to the set of leaves reachable from the node. We store in each node the first and last end vertices or edges of the corresponding section, the number of segments in the section, and the maximum and minimum  $y$ -coordinates of vertices in the section. (See Figure 13.) All these values can be updated bottom-up in the interior of the tree and top-down along the left and right paths.

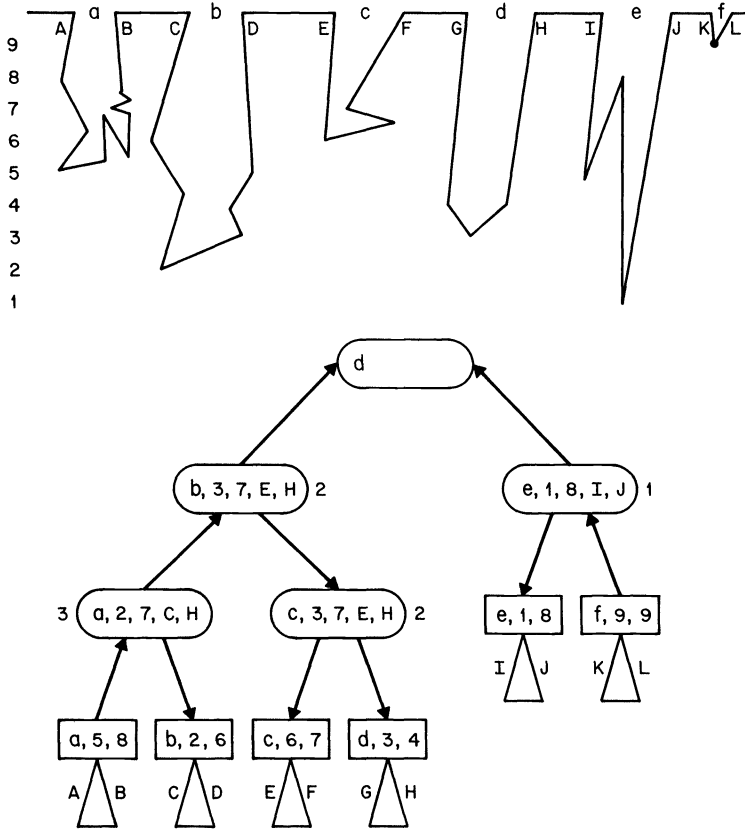


FIG. 13. A group of boundary segments and its representation as a heterogeneous finger search tree. (The drawing conventions are explained in Figure 22; capital letters represent end edges.)

All of the following operations can be performed in  $O(1 + \log(\min\{d, s-d\} + 1))$  time, where the segment found is the  $d$ th out of  $s$ :

- (i) Find the leftmost (or rightmost) segment in the tree that contains a vertex with  $y$ -coordinate less than (or greater than) a given value;
- (ii) Find the  $d$ th segment;
- (iii) Suppose all the segment vertices lie strictly above (or strictly below) a given horizontal line  $L$ , and that all end edges of the segments cross  $L$ . Given a value  $x$ , find the leftmost (or rightmost) segment in the tree having an edge whose intersection with  $L$  has  $x$  coordinate less than (or greater than)  $x$ . (All four possibilities, leftmost less than, leftmost greater than, etc., are allowed.)

In addition, inserting a new segment next to the  $d$ th out of  $s$  or splitting at the  $d$ th segment out of  $s$  takes  $O(1 + \log(\min\{d, s-d\} + 1))$  amortized time. Concatenating two trees takes  $O(1)$  amortized time.



Let us examine the manipulations of the boundary data structures required to carry out the steps of the balanced division visibility algorithm. We describe these step by step, including a timing estimate for some of the computations.

*Step 1.* Given  $V$ , determine the number of top and bottom boundary segments in its boundary, say  $t$  and  $b$ , respectively ( $O(1)$  time). Assume  $t \geq b$ . (The other case is symmetric.) If  $t \leq 2$ , choose a segment of the boundary that contains some vertex with  $y$ -coordinate strictly between  $y_{\min}$  and  $y_{\max}$ , and select as  $v$  the first such vertex in the segment ( $O(1)$  time). If  $t \geq 3$ , find the minimum  $y$ -coordinate of the vertices in the middle third of the top boundary segments (time to be analyzed below). This defines the slicing line  $L$ .

*Step 2.* Split the tree representing the top boundary between each pair of segments that differ in type among the types shallow, deep, and mixed (time to be analyzed below). This splits the top boundary list into mixed segments and maximal deep and shallow sections. Repeat this for the tree representing the bottom boundary (time to be analyzed below). For each mixed segment, split its tree between each pair of consecutive vertices, one on each side of  $L$  (time to be analyzed below). Split the tree that contains the vertex lying on  $L$  at that vertex, putting the vertex in both of the resulting trees (time to be analyzed below). Each of the splits performed corresponds to an essential intersection of the boundary with  $L$ . Form a list of these intersections, in the order in which they occur along the boundary, by examining the list of trees representing the deep sections and the new boundary segments that have been formed by splitting mixed segments ( $O(1)$  time per essential intersection).

*Step 3.* To execute a call *refine* ( $\{x_{i-1}, x_i\}, x$ ), examine the tree representing the deep section whose first and last intersections with  $L$  are  $x_{i-1}$  and  $x_i$ . Assume  $x_{i-1} < x_i$ . (The other case is symmetric.) Find in this tree the rightmost segment whose first intersection with  $L$  is less than  $x$  (time to be analyzed below). If the last intersection of this segment with  $L$  is greater than  $x$ , return no pair ( $O(1)$  time). Otherwise, split the section between this segment and the next one, and return as  $x'$  and  $x''$  the last intersection of this segment and the first intersection of the next one (time to be analyzed below).

*Step 4.* For each pair in the upper tree whose depth is odd and greater than one, split the tree representing the corresponding boundary segment at its vertex of maximum  $y$ -coordinate (time to be analyzed below). Put the splitting vertex in both of the resulting trees. Proceed symmetrically for the boundary segments corresponding to pairs in the lower tree. For each family in both trees, concatenate the boundary segments and deep sections corresponding to the pairs in the family to form the boundary of the subregion corresponding to the family ( $O(1)$  time per pair). Because the input polygon is simple, the order in which polygon vertices appear on the boundary of the subregion is consistent with their order in the original polygon. (See Figure 14.) This means that the concatenation will never need to reverse the order of boundary segments.

Now let us analyze the running time of this implementation of the balanced division algorithm. Observe that for each segment found within a top or bottom boundary, and for each vertex found within a segment, a split occurs at that segment or vertex. Since the timing estimates for finding and splitting are the same, the time spent splitting dominates the time spent finding such segments and vertices, including the time to find the  $y$ -coordinates of the splitting vertices in Step 1. Thus, by the timing estimates above and Theorem 1, the total running time of the algorithm is  $O(n)$  plus at most a constant times the time for tree insertions, splits, and concatenations.

It remains to estimate the time for tree update operations. Every insertion is at one end of a finger search tree and thus takes  $O(1)$  amortized time. The updates on

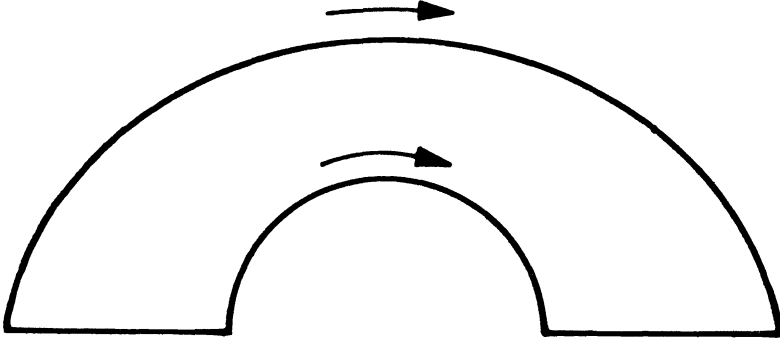


FIG. 14. A subregion that would cause trouble for Step 4. The arrows indicate the direction of the two boundary segments in the original polygon. Such subregions cannot occur for simple polygons.

the trees representing segments are only splits and insertions, of which there are  $O(n)$ . The total running time of these updates obeys a recurrence that is essentially the same as (1) (Section 1) and is thus  $O(n)$ .

The updates on the trees representing top and bottom boundaries include concatenations, and recurrence (1) does not apply. To analyze these operations, we first observe that the amortized time per concatenation or insertion is  $O(1)$ . Since the total number of such operations is  $O(n)$ , their total time is  $O(n)$ .

To analyze the  $O(n)$  splits, consider a particular visibility region  $V_i$  that has a total of  $s_i$  boundary segments. The total time to split the two trees representing the top and bottom boundaries is  $O(k_i + \sum_{j=0}^{k_i+1} \log s_{i,j})$ , where  $k_i$  is the total number of splits and  $s_{i,0}, s_{i,1}, \dots, s_{i,k_i+1}$  are the numbers of segments in each of the trees that are created by the splits. (Starting from two trees,  $k_i$  splits produce  $k_i + 2$  trees; if there is only one tree initially, we take  $s_{i,k_i+1} = 1$ .) We have  $s_{i,j} \geq 1$  for all  $j$ , and  $\sum_{j=0}^{k_i+1} s_{i,j} \leq s_i + 1$ . Since the logarithm is a concave function, the estimate of splitting time is maximized when all the pieces are of equal size, giving a bound of  $O(k_i + (k_i + 2) \log((s_i + 1)/(k_i + 2)))$ .

We evaluate the total time to perform all  $O(n)$  splits,  $O(\sum_i (k_i + (k_i + 2) \log((s_i + 1)/(k_i + 2))))$ , in two parts. Call  $V_i$  a *good region* if  $k_i \leq s_i / (\log n)^2$ , and *bad* otherwise. By Theorem 2,  $\sum_i s_i = O(n \log n)$ , so the total number of splits that occur while processing good regions,  $\sum_{k_i \leq s_i / (\log n)^2} k_i$ , is  $O(n \log n)$ ; using  $O(\log n)$  as a generous time bound for these splits, the total time to split good regions is  $O(n)$ . It remains to bound the time to split bad regions:

$$\begin{aligned} & O \left( \sum_{k_i \geq s_i / (\log n)^2} (k_i + (k_i + 2) \log((s_i + 1)/(k_i + 2))) \right) \\ &= O \left( \sum_{k_i \geq s_i / (\log n)^2} (k_i + (k_i + 2) \log \log n) \right) = O(n \log \log n), \end{aligned}$$

since  $\sum_i k_i = O(n)$ .

We conclude that the total running time of the visibility algorithm with balanced division is  $O(n \log \log n)$ .

**6. Remarks, applications and open problems.** We have presented an  $O(n \log \log n)$ -time algorithm for computing horizontally visible edge-vertex pairs

inside a simple polygon. By the linear-time reduction of triangulation to the visibility problem [5],[9] we obtain an  $O(n \log \log n)$ -time triangulation algorithm. The main ingredients of our algorithm are Jordan sorting, balanced divide and conquer, and finger search trees. It is intriguing to note that both the Jordan sorting algorithm and the visibility algorithm use finger search trees, but of two different kinds. The Jordan sorting algorithm requires fast access in the vicinity of any position in the tree but does not need to search on a secondary heap order. Homogeneous finger search trees satisfy these requirements. The visibility algorithm itself does need to search on secondary heap orders, but requires fast access only in the vicinity of the first and last positions. Heterogeneous finger search trees satisfy these requirements. Our algorithm exploits to the fullest the properties of these structures.

Finger search trees are sufficiently complicated that one would probably not want to use them in an actual implementation. The dynamic optimality conjecture of Sleator and Tarjan [23] suggests that the  $O(n \log \log n)$  time bound is still valid if splay trees (a form of self-adjusting search tree) are used in place of finger search trees. The use of splay trees might lead to a practical implementation of our algorithm, although this must be verified by experiment. Other minor changes in the algorithm might be useful in practice. We leave this as a topic for future research.

Our visibility algorithm can be modified to accommodate vertices having the same  $y$ -coordinate. To handle the resulting tangent intersection points in the Jordan sorting step (Section 2), we represent such a point  $x_i$  by a dummy pair  $(x_i, x_i)$  which we add to the lower tree if the tangency is on the top side of the splitting line  $L$ , or to the upper tree otherwise. The remaining changes to the algorithm are straightforward. (See e.g. [30].)

An efficient triangulation algorithm has a number of applications in computational geometry. These applications typically involve one (or possibly a few) triangulations and some linear-time pre- and postprocessing. Our triangulation algorithm gives  $O(n \log \log n)$ -time algorithms for these applications. Any improvement in the time to triangulate would give corresponding improvements in the applications. Such applications include:

- (i) several polygon decomposition problems [9] (where minimality, as in [17], is not required);
- (ii) regularizing (or triangulating) a planar subdivision that is given as a connected planar graph [8];
- (iii) computing the internal distance between two points in a polygon, and finding the point visibility polygon for a point inside the polygon [3];
- (iv) solving the single source shortest path problem inside a polygon, and computing internal visibility information from an edge inside a polygon [11];
- (v) testing two polygons for intersection, and decomposing simple splinegons [24] into a union of differences of unions of convex sets [7];
- (vi) determining translation separability of two simple polygons [1];
- (vii) finding a shortest watchman route in a simple rectilinear polygon ([6, Thm. 3]).

An important application of our visibility algorithm is to test whether an  $n$ -vertex polygon  $P$  is simple, and to exhibit a self-intersection of  $\partial P$  if it is not simple. We shall show how to modify our algorithm to perform these tasks in  $O(n \log \log n)$  time.

Even though the error-correcting Jordan sorting algorithm detects some instances of nonsimplicity as uncorrectable crossings or violations of the nonenclosure property, the successful completion of the visibility algorithm is not proof against nonsimplicity of the input polygon. Among the problems with which a guaranteed simplicity test

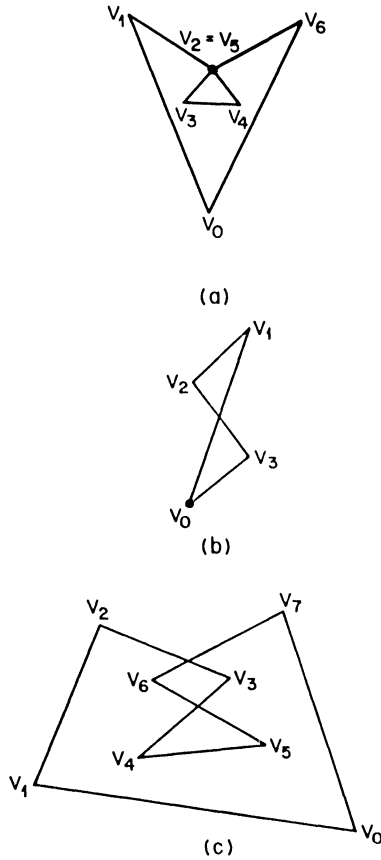


FIG. 15. Nonsimple polygons that cause no trouble in the algorithm for computing visibility information. In (c), a slice through  $v_5$  separates the polygon into three simple pieces.

must cope are polygons that are self-tangent at a vertex (Figure 15a), polygons for which an interior cannot be defined by any consistent labeling of the edges (Figure 15b), and nonsimple polygons that can be sliced into simple pieces (Figure 15c).

Our algorithm for testing simplicity is as follows. First we check that no two consecutive edges of  $P$  intersect in more than their common endpoint. Next, we run the visibility algorithm on the polygon  $P$  and on its “inside-out” partner  $Q$ , defined as in the proof of Lemma 2. We abort the algorithm and declare that  $P$  is nonsimple if one of the following cases occurs:

- (i) the Jordan sorting step finds an intersection point common to two parts of  $P$  other than an endpoint of two consecutive edges (see Figure 15a);
- (ii) the Jordan sorting step detects an uncorrectable crossing or violation of the nonenclosure property;

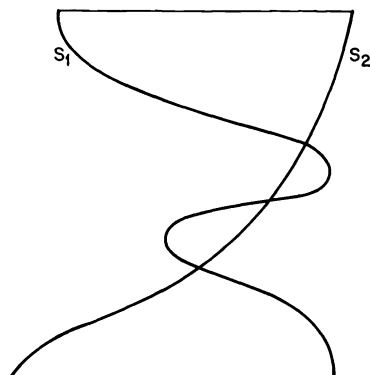


FIG. 16. The ordering of the four corners of this visibility region implies that it is not simple.

(iii) a subregion is constructed in Step 4 whose side boundary segments, say  $S_1$  and  $S_2$ , are known to cross, because the intersections of  $S_1$  and  $S_2$  with the top bounding line are in the opposite order from the order of their intersections with the bottom bounding line (see Figure 16).

If the visibility algorithm runs to completion on both  $P$  and  $Q$ , we declare that  $P$  is simple.

In our discussion of Step 4 in Section 4, we noted that if the polygon is simple, then the order of vertices along the boundary segments is consistent with the order of the boundary of the subregion being reassembled. It is conceivable if the polygon is not simple that Step 4 could be presented with a subregion whose boundary segments appear in an anomalous order, as in Figure 14. Fortunately, this situation cannot in fact occur: the existence of such a nested pair implies by the Jordan curve theorem that the current region being processed has a nonsimple boundary, and indeed that the segments of the boundary defined by the slicing line do not have the noncrossing property. Therefore, the nonsimplicity will be detected in Step 3.

**THEOREM 3.** *The simplicity-testing algorithm is correct.*

*Proof.* Certainly if the simplicity-testing algorithm declares that  $P$  is not simple then  $\partial P$  has a self-intersection. Suppose the algorithm reports that  $P$  is simple. The visibility computations in the algorithm produce two sets of regions. Let  $\mathbf{P}$  and  $\mathbf{Q}$  be the sets of regions produced when the visibility algorithm runs on  $P$  and  $Q$ , respectively; each region in  $\mathbf{P}$  and  $\mathbf{Q}$  is either a trapezoid or a triangle. Some of the regions in  $\mathbf{Q}$  are bounded by one or more edges of  $Q-P$ , i.e., by edges that were added to invert polygon  $P$ . Let  $\mathbf{Q}'$  be the set of regions formed by taking each region in  $\mathbf{Q}$  and extending it to infinity in the direction of any edge in  $Q-P$ . Regions in  $\mathbf{Q}'$  can be trapezoids, triangles, halfplanes, or infinite regions bounded by two horizontal lines and part of a side of  $P$ .

It is tempting to say that the regions in  $\mathbf{P}$  partition the "interior" of  $P$ , but we cannot say this, because we do not yet know that  $P$  has an interior. However, because of the way in which the regions in  $\mathbf{P}$  were produced, we know that they can be glued

together along shared horizontal visibility edges to form a region that is topologically equivalent to a disk. This is necessary but not sufficient for  $P$  to be simple (consider the polygon in Figure 15c). The visibility algorithm could be modified easily to produce this “gluing,” or, more properly, its dual graph, in which regions are vertices, and regions that share a horizontal visibility edge are joined by an edge in the dual graph.

Since the visibility algorithm succeeded, we also know that the regions in  $Q$  can be glued together along horizontal visibility edges to form a region that is topologically equivalent to the disk. This gluing can be extended naturally to  $Q'$ , which is topologically equivalent to the punctured plane. In what follows, we use the regions in  $P$  and  $Q'$  to construct a mapping from the plane onto itself.

Let  $C$  be a circle in the plane, and choose  $n$  distinct points on  $C$  corresponding to the vertices of  $P$ ; this induces a natural correspondence between points of  $\partial P$  and points of  $C$ . For each vertex-edge or edge-edge visible pair in  $P$  reported by the visibility algorithm, connect corresponding points on  $C$  by a path through the interior of  $C$ ; make all these paths disjoint except for corresponding endpoints. (The dual graph of the regions in  $P$  provides a natural way to do this constructively. Processing a vertex of degree one in the dual requires that we draw a path between two points on  $C$ ; that path divides the disk into two parts, one of which can be discarded and never enters into further computation of the mapping. Thus we can perform the complete construction by processing and deleting vertices of degree one from the dual until the dual is empty.) These paths divide the interior of  $C$  into regions corresponding to the regions of  $P$ . Similarly, for each piece of visibility information in  $Q'$ , construct a corresponding path joining two points of  $C$  and passing through the exterior of  $C$ . If the piece of information represents a vertex or edge that sees arbitrarily far to the left or right, the corresponding path leads from  $C$  to infinity. Make all the paths on the exterior of  $C$  noncrossing. This partitions the exterior of  $C$  into regions corresponding to the regions of  $Q'$ . (See Figure 17.)

We can construct a continuous mapping  $h$  of the plane onto itself that takes each region of the interior of  $C$  onto the corresponding region of  $P$  and each region of the exterior of  $C$  onto the corresponding region of  $Q'$ . The mapping is onto because every point in the plane is in some region of  $P$  or  $Q'$ : For any point  $x$ , move horizontally right from  $x$  until hitting  $\partial P$ . If the right side of  $\partial P$  is hit (with respect to clockwise order around  $\partial P$ ),  $x$  is in some region of  $P$ . Otherwise  $x$  is in some region of  $Q'$ . If  $\partial P$  is not hit,  $x$  is in some region of  $Q'$ .

Because  $P$  and  $Q'$  are finite and the preimages of distinct regions have disjoint interiors, the mapping  $h$  is a covering map from the plane onto itself: any point  $x$  in the plane has an open neighborhood  $N$  such that  $h^{-1}(N)$  is the disjoint union of a finite number of open sets [21]. Moreover, since the domain of  $h$  is connected, every point in its range is the image under  $h$  of the same number of points of its domain [21, ex. 8-3.5, p. 336]. Since any point in the open half plane that lies above the horizontal line through the highest vertex of  $P$  is the image under  $h$  of only one point,  $h$  is 1-1. Thus  $h$  is a homeomorphism of the plane onto itself. This proves that  $\partial P$  is homeomorphic to  $C$ , so  $P$  is simple.  $\square$

If the simplicity-testing algorithm reports that  $P$  is not simple, we can produce a witness to its nonsimplicity in  $O(n)$  additional time. If the nonsimplicity is detected in Case (i), the self-intersection is available immediately. Otherwise (if Case (ii) or (iii) occurs) we can find one or two bounding lines and two boundary segments  $S_1$  and  $S_2$  with ends on the bounding lines such that the two boundary segments are guaranteed to cross. There are seven possible configurations of the two boundary seg-

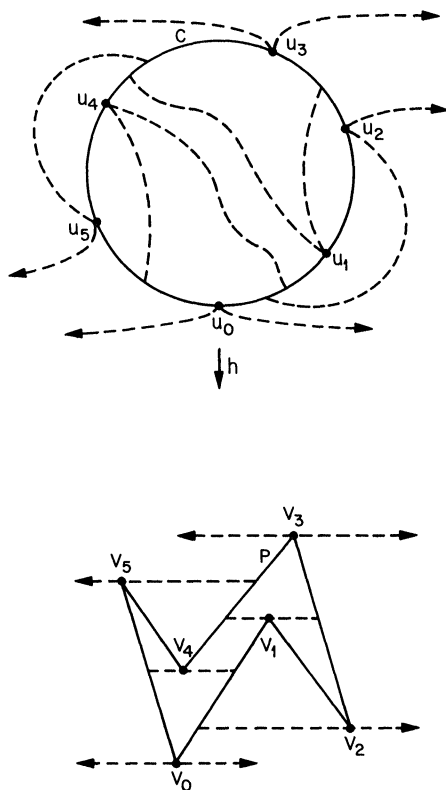


FIG. 17. Illustrating the mapping  $h$  constructed in the proof of Theorem 3. For  $0 \leq i \leq 5$ ,  $h(u_i) = v_i$ . Dashed paths in the upper figure correspond to vertex-edge visibility segments in the lower figure. Arrowheads indicate paths to infinity.

ments, illustrated in Figures 18 and 15b. We apply the following three steps repeatedly until an explicit crossing is found:

*Step 1.* Choose a vertex on  $S_1 \cup S_2$  not having maximum or minimum  $y$ -coordinate. Let  $L$  be the horizontal line through this vertex.

*Step 2.* Find all intersections of  $S_1$  and  $S_2$  with  $L$ , in the order in which they occur along  $S_1$  followed by  $S_2$ .

*Step 3.* Jordan sort the intersections. The Jordan sorting step must either find an explicit self-intersection or a violation of the noncrossing property. If such a violation is found, let  $S'_1$  and  $S'_2$  be the crossing subsegments. Replace  $S_1$  and  $S_2$  by  $S'_1$  and  $S'_2$ . If  $S'_1$  and  $S'_2$  are both single partial edges, report their intersection.

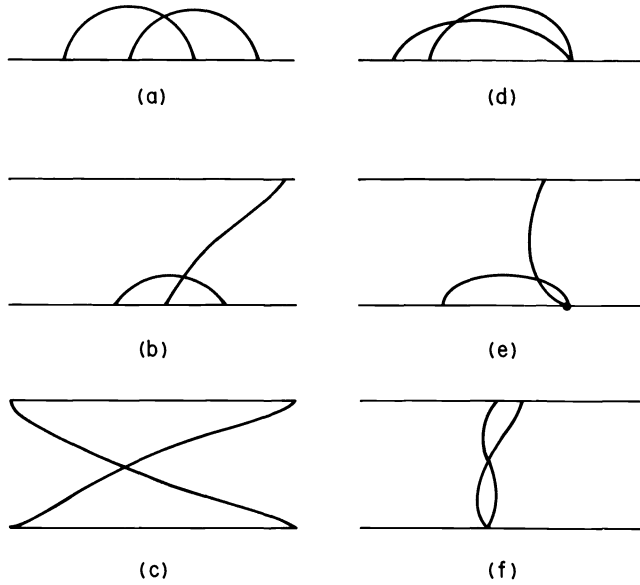


FIG. 18. Possible configurations of crossing boundary segments: (a) four points on one bounding line; (b) three points on one bounding line and one on the other; (c) two points on each bounding line; (d), (e), and (f) are degenerate cases of (a), (b), and (c), respectively. Figure 15b depicts a degenerate case of (f). Degenerate cases can be detected by examining the interior angle at the point or points of degeneracy.

An analysis like that in Sections 3–5 shows that this postprocessing takes  $O(n)$  time if the segments are represented by finger search trees. (Concatenation of finger search trees and balanced division are not needed.)

The algorithms for testing simplicity and producing a witness to nonsimplicity are easily extended to work on connected polygonal paths that are not closed. The first step of the extension is to draw a line through the two endpoints of the path, thus chopping the path into boundary segments that lie entirely on one side of the line, and to Jordan sort the points of intersection between the path and the line. If the Jordan sorting detects a violation of the noncrossing property, the algorithm for finding a crossing can be applied directly to the two boundary segments involved. Otherwise we use processing akin to that in Step 4, augmented to cope with boundary reversals as in Figure 14, to construct polygons that lie entirely on one side of the line; the path is simple if and only if each of those polygons is simple.

The algorithms for computing horizontal visibility information, for testing simplicity, and for producing a witness to nonsimplicity, can be extended to work on curves that obey certain mild restrictions. (See e.g. [22],[24].) To prepare the curve for processing add vertices to each edge to form a curve each of whose edges is monotone in the  $y$ -direction. The algorithms can now be run directly (given suitable procedures for



computing the intersection of a curved edge and a line), because the edges of the object have the property that if a collection of edges crosses two horizontal lines then the edges cross both lines in the same order. The other extension occurs in simplicity testing: we must check that the left and right sides of each trivial visibility region do not cross. If all of the output regions have this property, then the curve is simple; otherwise we have an immediate witness to its nonsimplicity. Both the preparation of the curves and the postprocessing of the output regions can be performed in  $O(n)$  time.

Several open problems remain. First and foremost, of course, is determining whether there is a linear-time triangulation algorithm, or even a  $o(n \log \log n)$ -time algorithm. Resolving this question seems to require a new idea. One possible approach is to invent a data structure for representing a polygonal curve that will allow fast computation of its intersections with an arbitrary horizontal line segment, or even with an arbitrary horizontal half line. Perhaps such a data structure can be built using information computed by the visibility algorithm called recursively on small pieces of the boundary, say of size  $O(\log n)$ . The result might be a visibility algorithm running in  $O(n \log^* n)$  time. Another open problem is to determine how fast all the self-intersections of a polygonal curve can be computed: can bounds better than those for an arbitrary collection of line segments [4] be obtained?

**Appendix. Finger search trees.** A *finger search tree* is a type of balanced search tree in which access in the vicinity of certain preferred positions, indicated by *fingers*, is especially efficient. Finger search trees were introduced by Guibas, McCreight, Plass and Roberts [12] and further developed by many other researchers [2],[15],[18],[28],[29]. We shall discuss two kinds of finger search trees with slightly different properties, *heterogeneous trees* and *homogeneous trees*. We base our development on a particular kind of balanced tree, the *red-black tree* [20],[25], although other kinds of balanced trees, such as *a,b-trees* [16],[19], form a suitable basis as well. We are mainly interested in amortized, not worst-case, complexity bounds.

For our purposes a *binary search tree* is a full binary tree in which each external node contains a distinct item selected from a totally ordered universe, with the left-to-right order of external nodes consistent with the total order on the items. Each internal node contains a *key*, which is an item greater than or equal to all items in its left subtree and less than all items in its right subtree. We can use the keys to search for the largest item in the tree less than or equal to a given one, by starting from the root and going to the left child if the item in the current node is greater than or equal to the given one, going to the right child otherwise, and repeating until an external node is reached. The desired item is either the one in the external node reached or the one in the preceding external node, which can be found by backing up the search path to a right child, starting from its left sibling, and going through right children to an external node. The time to search for an item is proportional to the tree depth.

A *red-black tree* is a binary search tree in which each node has one of two colors, *red* or *black*. The node colors obey the following constraints (see Figure 19):

- (i) all external nodes are black;
- (ii) all paths from the root to an external node contain the same number of black nodes;
- (iii) any red node, if it has a parent, has a black parent.

The depth of a red-black tree containing  $n$  items is  $O(\log n)$ . To insert a new item into a red-black tree, we search for the greatest item less than it in the tree. When the search reaches an external node, we replace this node by an internal node having two children, the old external node and a new external node containing the new

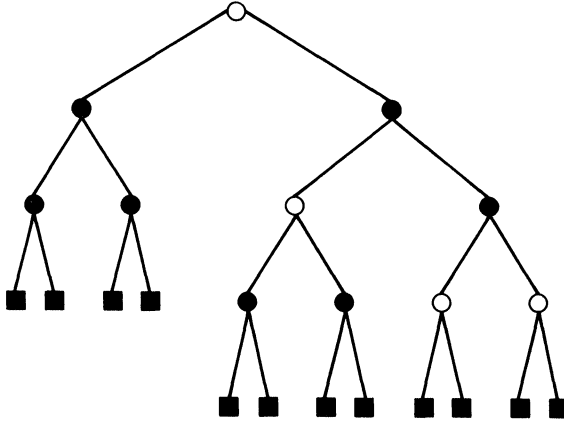


FIG. 19. A red-black tree; solid nodes are black; hollow nodes are red. (Only the colors of the nodes are shown.)

item. The new internal node contains as its key the smaller of the two items in its children. The new internal node is colored red. This may violate the red constraint (iii). To restore the red constraint, we proceed bottom-up along the search path, applying the recoloring transformation in Figure 20a until it no longer applies, followed by one application of Figure 20b, c, or d if necessary.

A deletion is similar. To delete an item, we find the external node containing it. We replace the parent of this node by the sibling of the node to be deleted. This may violate the black constraint (ii), producing a node that is *short*: all paths down from it to external nodes contain one fewer black node than paths down from its sibling. To restore the black constraint, we proceed bottom-up, applying the recoloring transformation of Figure 21a until it no longer applies, followed by one application of Figure 21b if necessary, and then possibly one application of Figure 21a, c, d, or e.

The worst-case insertion or deletion time in an  $n$ -item red-black tree is  $O(\log n)$ , but the amortized insertion/deletion time is only  $O(1)$ , not counting the time to search for the node at which the insertion or deletion takes place. (This is a restatement of a result of Huddleston and Mehlhorn [16] and Maier and Salveter [19] concerning  $a, b$ -trees.) To prove this, we define the *potential* of a red-black tree to be the number of black nodes with two black children plus twice the number of black nodes with two red children. We define the *actual time* of an insertion or deletion to be one plus the number of local transformations applied and the *amortized time* to be the actual time plus the net increase in potential caused by the operation. With these definitions, if we start with an empty tree, the total actual time for a sequence of insertions and deletions is at most the sum of the amortized times, since the initial potential is zero and the potential is always nonnegative. Furthermore the amortized time of an insertion or deletion is  $O(1)$ , since any of the transformations in Figures 20 and 21 increases the potential by  $O(1)$  and the nonterminal transformations 20a and 21a both decrease the potential by at least one.

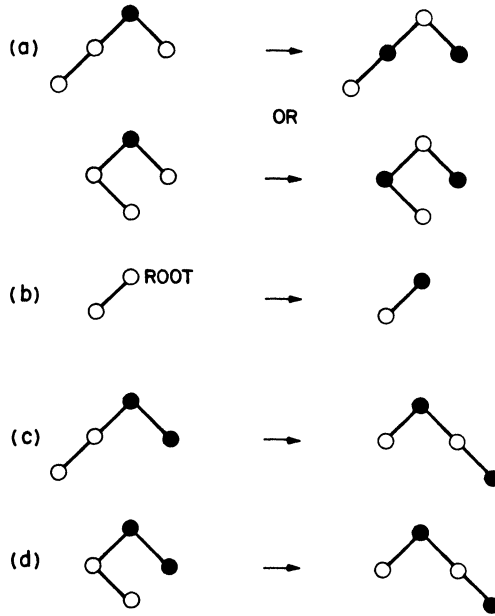


FIG. 20. *The rebalancing transformations in red-black tree insertion. Symmetric cases are omitted. All unknown children of red nodes are black. In cases (c) and (d) the bottommost black node shown can be external.*

In an ordinary binary search tree, each node points to its two children. We convert such a tree into a *heterogeneous finger search tree* by making each node along the left path<sup>2</sup> point to its parent instead of its left child, and each node along the right path point to its parent instead of its right child. Access to the tree is by two fingers pointing to the leftmost and rightmost external nodes. (See Figure 22.)

In an  $n$ -item heterogeneous search tree, we can search for an item  $d$  positions away from either end in  $O(1 + \log(\min\{d, n-d\} + 1))$  time, by searching up along the left and right paths concurrently until we find a subtree or two subtrees guaranteed to contain the desired item, and then searching down in this subtree or subtrees. Furthermore we can insert or delete an item  $d$  positions from either end in  $O(1 + \log(\min\{d, n-d\} + 1))$  amortized time. We can also search for an item based on its position or based on a secondary heap order. To accommodate search by position, we store in each internal node the number of external nodes reachable from it (by paths of pointers). To accommodate search based on a secondary heap order, we assume each item has an associated secondary value. We store in each internal node the minimum and maximum values of items reachable by paths of pointers. (See Figure 22.) By searching up along the left and right paths concurrently and then down into an appropriate subtree or subtrees, we can perform the following kinds of searches in  $O(1 + \log(\min\{d, n-d\} + 1))$  time:

- (i) Find the  $d$ th item in the tree;

<sup>2</sup>The *left path* in a binary tree is the path from the root through left children to an external node. The right path is defined symmetrically.

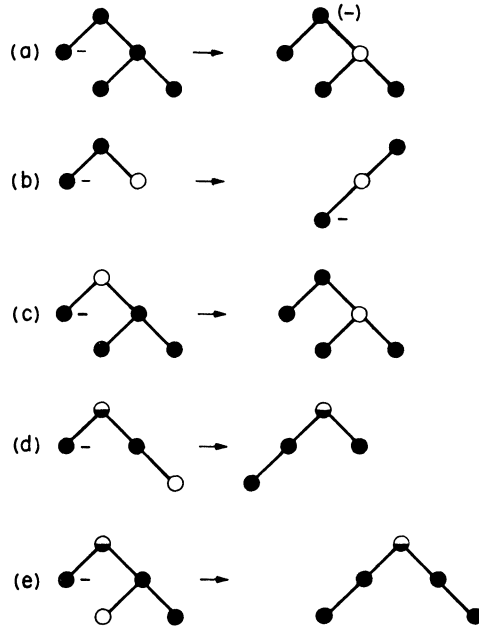


FIG. 21. *The rebalancing transformations in red-black tree deletion. The two ambiguous (half-solid) nodes in (d) have the same color, as do the two in (e). Minus signs denote short nodes. In (a), the top node after the transformation is short unless it is the root.*

(ii) Find the leftmost (or rightmost) item whose secondary value is at least (or at most) a given value, if the item found is the  $d$ th.

The auxiliary position and secondary value information must be updated when insertions and deletions are performed. This updating can be done bottom-up along the search path, i.e. bottom-up within the tree and top-down along the left or right path. The amortized time to insert or delete the  $d$ th item, including the search time, is  $O(1 + \log(\min\{d, n-d\} + 1))$ .

We now wish to extend our repertoire of update operations to include concatenation and splitting of trees. We shall discuss only the effect of these operations on the tree structure; it is easy to verify that the pointers, keys, and auxiliary position and secondary value information can be updated in the claimed time bounds. We define the *rank* of a node in a red-black tree to be the number of black internal nodes on any path from the node down to an external node; the rank of an external node is zero. We can compute the rank of a node in time proportional to the rank by walking down through the tree.

Concatenation is the simpler operation to describe. Suppose we wish to combine two trees  $T_1$  and  $T_2$  into a single tree; we assume that all items in  $T_1$  are less than all items in  $T_2$ . Let  $x_1$  with rank  $r_1$  and  $x_2$  with rank  $r_2$  be the roots of  $T_1$  and  $T_2$ , respectively. Assume  $r_1 \leq r_2$ . (The other case is symmetric.) To concatenate  $T_1$

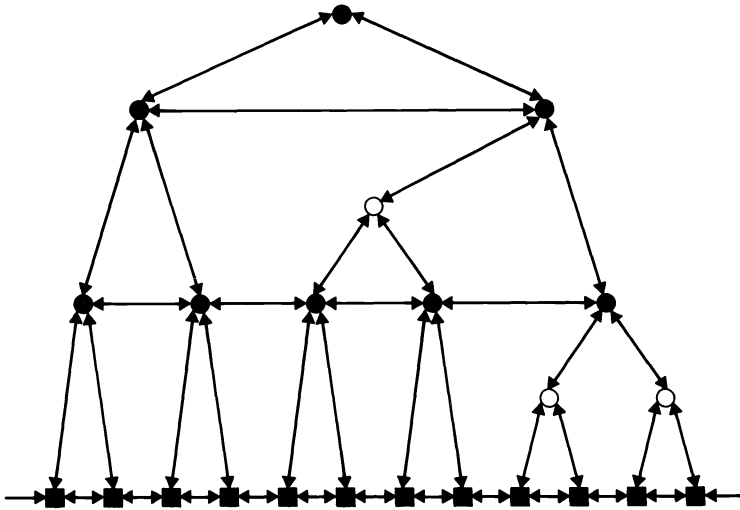


FIG. 22. A heterogeneous red-black finger search tree. The colors of nodes are not shown. The items are the letters *a* through *f*. The numbers in external nodes are secondary values. The numbers in internal nodes are the minimum and maximum secondary values reachable from the nodes. The numbers outside the nodes are the number of external nodes reachable from them.

and  $T_2$ , we walk up the left path of  $T_2$  until we reach a node, say  $y$ , with rank equal to  $r_1$ . We replace  $y$  in  $T_2$  by a new red node whose left child is  $x_1$  and whose right child is  $y$ , correcting any violation of the red constraint as in the case of an insertion. The amortized time for the concatenation is  $O(1 + \min\{r_1, r_2\})$ . If we change the definition of potential so that the potential of a tree is the rank of its root plus the number of black nodes with no black children, then the amortized time of a concatenation is  $O(1)$ ; the amortized time for inserting or deleting the  $d$ th item out of  $n$  remains  $O(1 + \log(\min\{d, n-d\} + 1))$ .

Suppose we wish to split a tree  $T$  containing  $n$  items at the  $d$ th item, dividing it into a tree  $T_1$  containing the first  $d$  items and a tree  $T_2$  containing the last  $n-d$  items. First we locate the  $d$ th item. Then we walk up along the search path to the left or right path, deleting every node along the search path except the external node containing the  $d$ th item. Assume we reach a node  $x$  on the left path. We concatenate the trees to the left of the search path (including the one consisting of the single node containing the  $d$ th item) in right-to-left order to form  $T_1$ . We concatenate the trees to the right of the search path whose roots are descendants of  $x$  to form a tree  $T'_2$ . If node  $x$  has no parent, then tree  $T'_2$  is the desired  $T_2$ . Otherwise, there remains another tree  $T''_2$  containing the parent of  $x$ , say  $y$ . Tree  $T''_2$  is missing a node, since node  $x$  was deleted. We replace node  $y$  by its right child, and repair the possible resulting shortness as in a deletion. Then we concatenate  $T_2$  and  $T'_2$  to form  $T_2$ . A careful analysis (see e.g. [20, pp. 214-216]) shows that the amortized time for splitting is  $O(1 + \log(\min\{d, n-d\} + 1))$  for either the new or the old definition of potential.

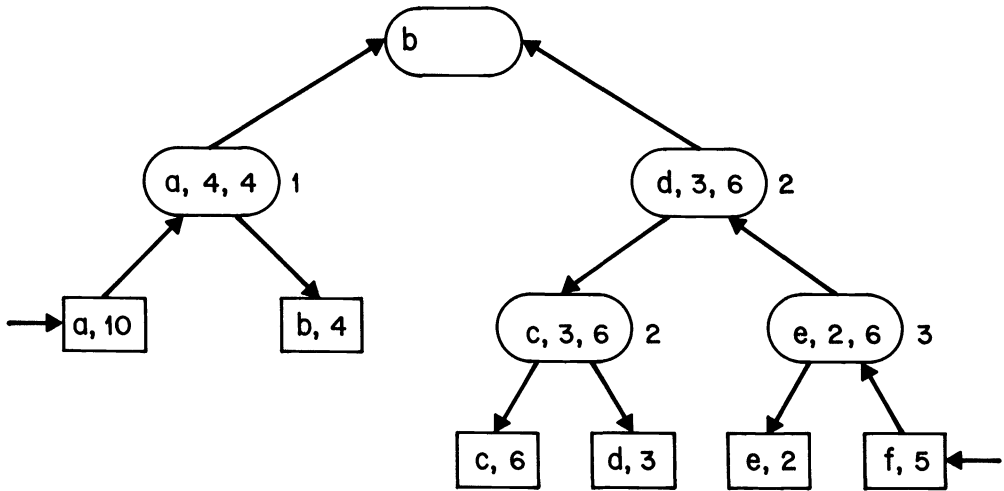


FIG. 23. *The pointers in a homogeneous red-black finger search tree.*

Heterogeneous finger search trees are used in the visibility algorithm to represent parts of region boundaries. The term “heterogeneous” refers to the fact that the pointer structure favors certain specific access positions, namely the two ends. In contrast, *homogeneous* finger search trees support fast access in the vicinity of *any* item. We make a red-black tree into a homogeneous finger search tree by adding additional pointers to it; namely, we make each node point to its two children and to its parent. Each black node also points to its *left* and *right neighbors*, the black nodes of the same rank that precede and follow the given node in symmetric order. (See Figure 23.) These extra pointers are called *level links*.

The level links support searching for a given item starting from an arbitrary item in the tree in  $O(1 + \log(\min\{d, n-d\} + 1))$  time, where  $d$  is the number of items between the two given items. The search proceeds up from the starting item following parent pointers and level links until a subtree or two in which the desired item must be located are found; then the search proceeds downward in the standard way. Simultaneously with the search from the given starting item, searches are performed starting from the first and last items in the list; all three searches terminate when the desired item is first located (or discovered not to be in the tree).

Homogeneous finger search trees support fast searches only with respect to the total order on the items, not searches by position or searches based on secondary heap order. On the other hand, the extra time needed to update level links is only  $O(1)$  per local transformation (of the kinds in Figures 19 and 20), and thus the amortized time bounds of insertion, deletion, concatenation, and splitting are the same in homogeneous trees as they are in heterogeneous trees. Furthermore, homogeneous trees support a more drastic splitting operation, called *three-way splitting*: given two items  $x$  and  $y$  in a tree  $T$ , remove from  $T$  the sublist of items from  $x$  to  $y$  (inclusive) to form two trees, one representing the removed sublist and the other representing the remaining

items. The amortized time for a three-way splitting operation is  $O(1 + \log(\min\{d, n-d\} + 1))$ , where  $d$  is the number of items in the removed sublist. See [14] for details on how a three-way split can be performed within this time bound.

**Acknowledgments.** We thank Brenda Baker for her many helpful comments on an earlier draft of this paper, and Bernard Chazelle for his thorough reading of this version of the paper.

#### REFERENCES

- [1] B. K. BHATTACHARYA AND G. T. TOUSSAINT, *A linear algorithm for determining the translation separability of two simple polygons*, Technical report SOCS 861, School of Computer Science, McGill University, Montreal, Canada, 1986.
- [2] M. R. BROWN AND R. E. TARJAN, *Design and analysis of a data structure for representing sorted lists*, this Journal, 9 (1980), pp. 594-614.
- [3] B. CHAZELLE, *A theorem on polygon cutting with applications*, Proc. 23rd Annual Symp. on Found. of Comput. Sci., 1982, pp. 339-349.
- [4] ———, *Intersecting is easier than sorting*, Proc. Sixteenth Annual ACM Symp. on Theory of Comput., 1984, pp. 125-134.
- [5] B. CHAZELLE AND J. INCERPI, *Triangulation and shape complexity*, ACM Trans. on Graphics, 3 (1984), pp. 135-152.
- [6] W.-P. CHIN AND S. NTAFOU, *Optimum watchman routes*, Proc. Second Annual Symp. on Computational Geometry, 1986, pp. 24-33.
- [7] D. P. DOBKIN, D. L. SOUVAINE AND C. J. VAN WYK, *Decomposition and intersection of simple splines*, Algorithmica, to appear.
- [8] H. EDELSBRUNNER, L. J. GUIBAS AND J. STOLFI, *Optimal point location in a monotone subdivision*, this Journal, 15 (1986), pp. 317-340.
- [9] A. FOURNIER AND D. Y. MONTUNO, *Triangulating simple polygons and equivalent problems*, ACM Trans. on Graphics, 3 (1984), pp. 153-174.
- [10] M. R. GAREY, D. S. JOHNSON, F. P. PREPARATA AND R. E. TARJAN, *Triangulating a simple polygon*, Inform. Process. Lett., 7 (1978), pp. 175-180.
- [11] L. GUIBAS, J. HERSHBERGER, D. LEVEN, M. SHARIR AND R. E. TARJAN, *Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons*, Algorithmica, to appear.
- [12] L. J. GUIBAS, E. M. MCCREIGHT, M. F. PLASS AND J. R. ROBERTS, *A new representation for linear lists*, Proc. Ninth Annual ACM Symp. on Theory of Comput., 1977, pp. 49-60.
- [13] S. HERTEL AND K. MEHLHORN, *Fast triangulation of a simple polygon*, Proc. Conf. Found. of Comput. Theory, New York, Springer-Verlag, Berlin, 1983, pp. 207-218.
- [14] K. HOFFMAN, K. MEHLHORN, P. ROSENSTIEHL AND R. TARJAN, *Sorting Jordan sequences in linear time using level-linked search trees*, Inform. and Control, 68 (1986), pp. 170-184.
- [15] S. HUDDLESTON, *An efficient scheme for fast local updates in linear lists*, Dept. of Information and Computer Science, University of California, Irvine, CA, 1981.
- [16] S. HUDDLESTON AND K. MEHLHORN, *A new data structure for representing sorted lists*, Acta Inform., 17 (1982), pp. 157-184.
- [17] J. M. KEIL, *Decomposing a polygon into simpler components*, this Journal, 14 (1985), pp. 799-817.
- [18] S. R. KOSARAJU, *Localized search in sorted lists*, Proc. Thirteenth Annual ACM Symp. on Theory of Comput., 1981, pp. 62-69.
- [19] D. MAIER AND C. SALVETER, *Hysterical B-trees*, Inform. Process. Lett., 12 (1981), pp. 199-202.
- [20] K. MEHLHORN, *Data Structures and Efficient Algorithms, Volume 1: Sorting and Searching*, Springer-Verlag, Berlin, 1984.
- [21] J. R. MUNKRES, *Topology: A First Course*, Prentice-Hall, Englewood Cliffs, NJ, 1975.
- [22] A. A. SCHÄFFER AND C. J. VAN WYK, *Convex hulls of piecewise smooth Jordan curves*, J. Algorithms, 8 (1987), pp. 66-94.
- [23] D. D. SLEATOR AND R. E. TARJAN, *Self-adjusting binary search trees*, J. Assoc. Comput. Mach., 32 (1985), pp. 652-686.
- [24] D. L. SOUVAINE, *Computational Geometry in a Curved World*, Ph.D. dissertation, Department of Computer Science, Princeton University, 1986.

- [25] R. E. TARJAN, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.
- [26] ———, *Amortized computational complexity*, SIAM J. Algebraic and Discrete Methods, 6 (1985), pp. 306-318.
- [27] R. E. TARJAN AND C. J. VAN WYK, *A linear-time algorithm for triangulating simple polygons*, Proc. Eighteenth Annual ACM Symp. on Theory of Comput., 1986, pp. 380-388.
- [28] A. K. TSAKALIDIS, *AVL-trees for localized search*, Automata, Languages, and Programming, 11th Colloquium, J. Paredaens, ed., Lecture Notes in Computer Science 172, Springer-Verlag, Berlin, 1984, pp. 473-485.
- [29] ———, *An optimal implementation for localized search*, A84/06 Fachbereich Angewandte Mathematik und Informatik, Universität des Saarlandes, Saarbrücken, West Germany, 1984.
- [30] C. J. VAN WYK, *Clipping to the boundary of a circular-arc polygon*, Computer Vision, Graphics, and Image Processing, 25 (1984), pp. 383-392.
- [31] T. C. WOO AND S. Y. SHIN, *A linear time algorithm for triangulating a point-visible polygon*, ACM Trans. on Graphics, 4 (1985), pp. 60-69.



## PREFACE

In June 1985, a workshop on the mathematical theory of security was held at the Massachusetts Institute of Technology's Endicott House. The workshop, which was run by Silvio Micali and Shafi Goldwasser, had three goals: to provide the necessary mathematical framework for rigorously discussing security issues, to select archetypal problems and provide concrete solutions, and to generate momentum and critical mass.

The participants in the workshop were invited to submit papers to this special issue. Most of the papers in this issue were presented at the workshop. The submitted papers went through the normal refereeing process; not all of them were accepted. Finally, after two years, the issue is ready for publication.

The papers cover the following topics: randomization, pseudorandom number generators, specific cryptographic protocols with certain provable security properties, some cryptanalytic attacks against specific protocols, and the theory of zero knowledge. In addition, there are papers on various notions of security, as well as relevant topics in complexity theory and hardware. Below, I discuss very briefly the main results of the papers in this issue.

Some cryptographic protocols require generating a random integer in a certain range with known factorization. The obvious method of selecting a random integer and then factoring it is usually unfeasible, as factoring is assumed to be hard. In "How to generate factored random numbers," Bach presents an efficient method. For a range of the form  $N/2 < x < N$ , the algorithm requires  $O(\log N)$  primality tests of integers not larger than  $N$ . It can thus be performed in random polynomial time.

In "RSA and Rabin functions: Certain parts are as hard as the whole," Alexi, Chor, Goldreich, and Schnorr prove the following property of the RSA and Rabin encryption functions. If  $E_N: Z_N \rightarrow Z_N$  is either one of the two functions, then the following computational tasks are equivalent under random polynomial-time transformations, given  $E_N(x)$ : (i) computing  $x$ ; and (ii) guessing the least significant bit of  $x$  with success probability  $\frac{1}{2} + 1/\text{poly}(n)$ , where  $n$  is the length of  $N$ . Assuming integer factoring is hard, they derive improved pseudorandom number generation and probabilistic encryption schemes as a result.

In "Privacy amplification by public discussion," Bennett, Brassard, and Robert consider the following setting: Two persons have an imperfect private channel and a perfect public channel. The authors design protocols that allow the two persons to assess the damage to the message sent and, in case it is not too severe, to extract from it an undamaged part about which the eavesdropper has no information.

Recently, there were several papers that defined a model for weak random sources and then showed how perfect or almost-perfect random bits can be generated using such sources. In "Unbiased bits from sources of weak randomness and probabilistic communication complexity," Chor and Goldreich define such a model, which generalizes previous models. They show how to extract random bits from such sources. They also derive results from probabilistic communication complexity and draw conclusions about the robustness of the complexity class *BPP*.

In "Reconstructing truncated integer variables satisfying linear congruences," Frieze, Hastad, Kannan, Lagarias, and Shamir propose a polynomial-time algorithm that finds small integer solutions to systems of linear congruences. They derive from it polynomial-time algorithms that reconstruct the values of variables  $x_1, \dots, x_k$ , when

there are some linear congruences relating them, together with certain bits from the binary representations of these values (e.g., the high-order bits, the low-order bits, or even an arbitrary window of consecutive bits). These algorithms yield two applications: predicting linear congruential generators with truncated outputs, and breaking the simplest version of Blum's protocol for exchanging secrets.

In "The knowledge complexity of interactive proof systems," Goldwasser, Micali, and Rackoff define the notion of zero-knowledge proofs as those proofs that convey no additional information aside from the correctness of the assertion being proved. The usual NP-proof that upholds the validity of the assertion (e.g., exhibiting a Hamiltonian tour) is not zero-knowledge since it conveys much more than the fact that the given string is in the language (e.g., it gives a tour which is much more than merely the fact that the graph is Hamiltonian). They give two examples of zero-knowledge proofs: one for the language of quadratic residues and another for its complement.

In "A digital signature scheme secure against adaptive chosen-message attacks," Goldwasser, Micali, and Rivest present a new signature scheme that is based on the assumed computational difficulty of integer factoring. The scheme has the strongest security property, as it is secure against adaptive chosen-message attack: an adversary who generates messages that may depend on previous signatures and who receives signatures for such messages cannot forge the signature of an additional message. (The task of forging a signature is equivalent to factoring.)

In "Minimum-knowledge interactive proofs for decision problems,"<sup>1</sup> Galil, Haber, and Yung introduce a protocol for two parties, the prover and the verifier, with the following properties: Following the protocol, the prover gives to the verifier a proof of the value, 0 or 1, of a particular Boolean predicate that is hard for the verifier to compute. This extends the "interactive proof systems" of Goldwasser, Micali, and Rackoff, which are only used to confirm that a certain predicate has value 1. The protocol is provably minimum-knowledge in the sense that it communicates no additional knowledge to the recipient aside from the value of the predicate. The protocol is result-indistinguishable: an eavesdropper, overhearing an execution of the protocol, does not learn the value of the predicate that is proved, or anything else.

In "Complexity measures for public-key cryptosystems," Grollmann and Selman use complexity theory to formulate two equivalent versions of cracking a public-key cryptosystem. They prove several complexity-theoretic results related to these formulations.

In "Solving simultaneous modular equations of low degree," Hastad shows how to solve polynomial modular equations. His algorithm is polynomial when the degree of the polynomial is small relative to the number of equations. Consequently, the use of the RSA with a small exponent as a public-key cryptosystem is not secure in a large network. A previously proposed protocol is broken using the new algorithm.

In "Unique extrapolation of polynomial recurrences," Lagarias and Reeds show how to correctly extrapolate the values of an unknown polynomial recurrence modulo  $M$  in  $k$  variables of degree at most  $d$ , where  $d$  and  $k$  are known and  $M$  is not known. A polynomial-time algorithm predicts the next value given the first  $n$  values, and is wrong for at most a constant number (that depends on  $d$ ,  $k$ , and  $M$ ) of values of  $n$ .

The first pseudorandom number generators gave methods that generate bits one at a time, such that any advantage in guessing the next bit generated was computationally equivalent to a hard problem. In "The discrete logarithm hides  $O(\log n)$  bits,"

---

<sup>1</sup> This paper was handled by a different editor.

Long and Wigderson show that obtaining any information about the  $O(\log |p|)$  most significant bits of  $x$ , given  $g^x \bmod p$ , is equivalent to computing the discrete logarithm mod  $p$ . So the method yields a pseudorandom bit generator, using seeds of length  $n$ , that generates  $\log n$  pseudorandom bits at once.

It is known how to construct a pseudorandom function generator from a pseudorandom bit generator. In “How to construct pseudorandom permutations from pseudorandom functions,” Luby and Rackoff show how to construct a pseudorandom invertible permutation generator from a pseudorandom function generator. An implication of this result is that any pseudorandom bit generator can be used to construct a private-key cryptosystem that is secure against chosen plaintext attack.

In “The notion of security for probabilistic cryptosystems,” Micali, Rackoff, and Sloan consider three formal definitions of security that have been proposed in the literature and prove that they are equivalent. They consider their result evidence that each one of the definitions captures the right notion of security.

In “A pipeline architecture for factoring large integers with the quadratic sieve algorithm,” Pomerance, Smith, and Tuler consider integer factoring from the practical, economic point of view. Specifically, their “complexity measure” of a given algorithm is the size of the largest numbers it can factor in one year while using equipment costing less than \$10 million. They describe an architecture and implementation of the quadratic sieve algorithm that would yield 144 decimal digits as the measure. Currently, the two best algorithms yield 101 and 126. The authors remark that the annual cost of factoring a 200-digit number with their strategy would be about  $\$10^{11}$ —or only 5 percent of the current U.S. national debt.

In “Efficient parallel pseudorandom number generation,” Reif and Tygar present a simple parallel pseudorandom bit generator, assuming that the multiplicative inverse problem is almost always not in the class *RNC*. Under this assumption, problems in *RNC* can be solved by using only  $n^\epsilon$  random bits to serve as a seed for the generator. They deduce some complexity-theoretic results.

At the time of the writing of this introduction the final versions of the two papers on zero knowledge (by Goldwasser, Micali, and Rackoff and by Galil, Haber, and Yung) were not ready. They may appear in a later issue of this journal.

—Zvi Galil<sup>2</sup>, *Editor of Special Issue on Cryptography*  
Computer Science Department  
Columbia University and Tel-Aviv University

---

<sup>2</sup> Research supported in part by National Science Foundation grants DCR-85-11713 and CCR-86-05353.

## HOW TO GENERATE FACTORED RANDOM NUMBERS\*

ERIC BACH†

**Abstract.** This paper presents an efficient method for generating a random integer with known factorization. When given a positive integer  $N$ , the algorithm produces the prime factorization of an integer  $x$  drawn uniformly from  $N/2 < x \leq N$ . The expected running time is that required for  $O(\log N)$  prime tests on integers less than or equal to  $N$ .

If there is a fast deterministic algorithm for primality testing, this is a polynomial-time process. The algorithm can also be implemented with randomized primality testing; in this case, the distribution of correctly factored outputs is uniform, and the possibility of an incorrectly factored output can in practice be disregarded.

**Key words.** factorization, primality, random variate generation

**AMS(MOS) subject classifications.** 1104, 11A51, 11Y05, 65C10

**1. Introduction.** Let  $N$  be a positive number, and suppose that we want a random integer  $x$  uniformly distributed on the interval  $N/2 < x \leq N$ . Further suppose that we do not want to output  $x$  in the usual decimal form, but rather as an explicit product of primes.

This is clearly possible if we are willing to factor  $x$ . However, the best known algorithms for factorization [8], [16] require roughly  $O(\log x)^{\sqrt{\log x}/\log \log x}$  steps on input  $x$ , so this approach is out of the question if  $N$  is large. In contrast, the method of this paper uses primality testing rather than factorization. Since there are efficient algorithms for determining primality [1], [10], [13], [17], the method is useful even when  $N$  is so large that factorization is infeasible.

The algorithm works by assembling random primes, but it is not clear a priori with what distribution these should be selected, nor how to efficiently implement a desired distribution on the primes. Much of the paper will deal with these questions, in a rather detailed fashion. However, if one is willing to overlook these technicalities, the resulting method can be easily sketched.

It selects a factor  $q$  of  $x$  whose length is roughly uniformly distributed between 0 and the length of  $N$ , then recursively selects the factors of a number  $y$  between  $N/2q$  and  $N/q$  and sets  $x = y \cdot q$ . It has now chosen  $x$  with a known bias; to correct this, it flips an unfair coin to decide whether to output  $x$  or repeat the whole process.

The results of this paper show not only that the distribution of  $x$  is uniform, but that this is a fast algorithm. A rough measure of its running time is the number of primality tests required; this quantity has expected value and standard deviation that are both  $O(\log N)$ —the same as required to generate a random prime of the same size.

This estimate is the basis for a finer analysis of the running time, which uses some assumptions about primality testing. If there is a deterministic polynomial-time prime test, as proved under the Extended Riemann Hypothesis by Miller [10], then the

---

\* Received by the editors April 25, 1983; accepted for publication (in revised form) August 6, 1985. Sections 1-8 of this article originally appeared as Chapter 2 of the author's Ph.D. dissertation, *Analytic Methods in the Analysis and Design of Number-Theoretic Algorithms*, © 1985 by the Massachusetts Institute of Technology Press. A preliminary version of this article was presented at the 15th Annual ACM Symposium on the Theory of Computing 1983.

† Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706. This research was supported by the National Science Foundation, grants MCS-8204506 and DCR-8504485, and the Wisconsin Alumni Research Foundation.

expected number of single-precision operations needed to generate  $x$  in factored form can be bounded by a polynomial in  $\log N$ .

If the method uses a probabilistic polynomial-time prime test such as those of Rabin [13] and Solovay and Strassen [17], a similar result holds. In this case, the distribution of correctly factored numbers is still uniform, and the possibility of producing an incompletely factored output can in practice be disregarded — all within an expected polynomial time bound.

The method has been implemented; on a medium-sized computer, it will generate a 120-digit number in about 2 minutes.

The rest of this paper is organized as follows. Section 2 gives a heuristic derivation of the algorithm, and § 3 gives a general discussion of random variate generation. Section 4 presents the algorithm in explicit form; its running time is analyzed in §§ 5–8. Finally, § 9 gives experimental results.

**2. Heuristics.** Later sections present a detailed algorithm; this one provides motivation and sketches a design based on heuristic arguments.

First, what is meant by a “random factor” of a number? If we write down all numbers between  $N/2$  to  $N$  in factored form, we will have an array that is roughly rectangular, because the juxtaposition of a number’s factors is about as long as the number itself. If the factorizations are arranged one per line, and given in binary notation, the picture will look something like this:

```

      ⋮
10 10 10 10011 1001110111 100001100111001
11 11 11 1011011101 10100010111000111
10 101 11010011 10111110111110101011
10111111 100000111101110001010101
10 10 11 1111111010011 100000111110011
100101 1101101 1100011111010101101
10 111 111 111 10010100011 11111101011
11 101 1011 1101 10111011110111001111
10 10 10 10 11101 11111 11100000000111101
11010100011 11101101001011011011
      ⋮

```

Choosing a random factorization is equivalent to picking a row at random from this list; if the list were perfectly rectangular, we could do this by choosing a bit at random and taking the row in which it appears.

Now suppose that we wanted to get the effect of this process by choosing a prime factor  $p$  first and selecting one of the remaining  $N/p$  possibilities uniformly. To do this, we would pick  $p$  with probability proportional to its “surface area,” that is, proportional to the total number of bits occurring in all copies of  $p$ .

This suggests selecting the first factor  $p$  with probability about  $\log p/p \log N$ , since  $p$  occurs in about  $1/p$  of the numbers, and a random bit of such a number will be in  $p$  about  $\log p/\log N$  of the time (ignoring repeated factors).<sup>1</sup>

It is instructive to see what effect this would have on the *length* of  $p$ . A weak form of the prime number theorem [6, Thm. 7] implies that for  $0 < x < N$ ,

$$\sum_{p \leq x} \frac{\log p}{p \log N} \cong \frac{\log x}{\log N}.$$

<sup>1</sup> Unless otherwise indicated, all logarithms in this paper are to the base  $e = 2.718281 \dots$

Therefore, if we chose  $p$  with the proposed probability, the length of  $p$  (relative to the length of  $N$ ) would be close to that produced by a uniform distribution, since for  $0 \leq \alpha \leq 1$ ,  $\log p / \log N < \alpha$  with roughly the same frequency in both cases.

One can justify this uniform length heuristic in another fashion. The factorization of numbers into primes is analogous to the decomposition of permutations into disjoint cycles; for instance, one can easily prove the “prime permutation theorem”: a random permutation on  $n$  letters is prime (a cycle) with probability  $1/n$ . This analogy extends to the distribution of factor lengths: Knuth and Trabb Pardo have shown that the relative length of the  $k$ th largest factor of a random number has the same asymptotic distribution as the relative length of the  $k$ th largest cycle in a random permutation [7, § 10]. Under this analogy, our prime selection technique corresponds to a process that selects a random letter in a random permutation and outputs the cycle to which it belongs. Results on random permutations [5, p. 258] imply that the length of this cycle is uniformly distributed.

Thus, to choose  $x$  uniformly with  $N/2 < x \leq N$ , we might proceed as follows. Select a length  $\lambda$  uniformly from  $(0, \log N)$  and pick the largest prime  $p$  with  $\log p \leq \lambda$ . Then recursively select  $y$  (the remaining bits of  $x$ ) to satisfy  $N/2p < y \leq N/p$ , and output  $x$ , as  $p$  times the prime factorization of  $y$ .

If the distribution of  $y$  were uniform, the probability of selecting  $x$  would be about

$$\sum_{p|x} \frac{\log p}{p \log N} \cdot \frac{1}{N/p - N/2p}.$$

This is  $2/N$ , the correct probability for a uniform distribution, times a bias factor of

$$\frac{1}{\log N} \sum_{p|x} \log p.$$

This bias should be close to 1, and it is, provided that  $x$  does not have too many repeated prime factors.

Thus, one would suspect that this method is almost right; however, a closer look at the algorithm reveals the complications listed below.

- 1) Merely picking the biggest prime less than some given value will not do; for one thing, the first member of a twin prime pair will be chosen less frequently than the second. A correct method must be insensitive to these local irregularities.
- 2) The bias factor is quite small for certain  $x$ , say powers of 2. This problem can be eliminated by also including prime power factors in the first step, but we must further decide how often these are chosen.
- 3) At the end of the algorithm,  $x$  will have been chosen with a certain bias, but the recursion will not work unless all  $x$ 's are equally likely. The odds must be changed somehow to make the eventual output uniform.
- 4) Finally, we imagine selecting  $y$ , the rest of  $x$ , from  $(N/2p, N/p]$  with probability  $2p/N$ . However, it is by no means certain, and in general not true, that there are  $N/2p$  integers in this range.

Dealing with these problems requires some machinery that will be developed in the next section.

**3. Doctoring the odds.** This section discusses a general technique for using one distribution to simulate another, called the “acceptance-rejection” method [11], [15]. It requires only a little information about the distributions, a source of uniform  $(0,1)$  random real numbers, and some extra time.

This method is usually applied in situations where everything is known about the distributions. In our case, we will only know the relative probabilities involved, hence we need the following definition.

Let  $(x_1, \dots, x_n)$  be a finite set. We will say that  $X$  has a finite distribution with odds  $(p_1, \dots, p_n)$  if  $X = x_i$  with probability  $p_i / \sum_{j=1}^n p_j$ . The odds of a distribution are only defined up to a multiplicative constant; this conforms to ordinary usage, in which odds of 2:1 and 10:5 are regarded as identical.

To see how to turn one distribution into another, consider an example. Suppose we have a coin that is biased in favor of heads with odds of 2:1, and we wish to make it fair. This can be done by the following trick. Flip the coin. If it comes up tails, say "tails"; if it comes up heads, say "heads" with probability 1/2, and with probability 1/2 repeat the process.

The stopping time can be analyzed by the following "renewal" argument. The process must flip the biased coin once no matter what happens, and after this first step, it has one chance in three of being born again. Thus the expected stopping time  $E(T)$  must satisfy  $E(T) = 1 + E(T)/3$ , so  $E(T) = 3/2$ . More generally,  $T = t$  with probability  $(2/3) \cdot (1/3)^{t-1}$ ; this is a geometric distribution with expected value 3/2. At each reincarnation, the process has no memory of its past, so the stopping time and the ultimate result are independent.

This example is not very useful, as it requires a fair coin to produce the effect of one; however, it points out some important features of the method. First, decisions are only made locally; after getting, say, heads, a decision can be made without knowing the other possible outcomes or even their total number. Second, only the odds matter; knowing only the relative probability of each outcome is sufficient for undoing the bias.

Here is the general version; we are given odds  $(p_1, \dots, p_n)$  but want odds  $(q_1, \dots, q_n)$ . Assume that  $q_i \leq p_i$  for all  $i$ ; we can use the following recipe:

PROCESS A: Acceptance-rejection method.

- (\*) Select  $X$  from the original distribution.
- Choose a real number  $\lambda$  from the  $U(0,1)$  distribution.
- If  $X = x_i$  and  $\lambda < q_i/p_i$ , output  $Y = x_i$ .
- If not, go back to (\*).

**THEOREM 1.** *Let  $X$  have a finite distribution with odds  $(p_1, \dots, p_n)$ . If  $q_i \leq p_i$  for  $1 \leq i \leq n$ , then the output of Process A has a finite distribution with odds  $(q_1, \dots, q_n)$ . The stopping time  $T$  and the output value  $Y$  are independent random variables. If  $P = \sum_{i=1}^n p_i$ ,  $Q = \sum_{i=1}^n q_i$ , then the stopping time is distributed geometrically with expected value  $P/Q$ .*

*Proof.* A direct calculation shows that the joint distribution of  $T$  and  $Y$  is

$$\Pr[T = t, Y = x_i] = \frac{q_i}{Q} \cdot \frac{Q}{P} \left(1 - \frac{P}{Q}\right)^{t-1}. \quad \square$$

This technique is at the heart of the method, in two ways:

At the top level, the algorithm generates  $x$ ,  $N/2 < x \leq N$ , with probability proportional to  $\log x$ . It accepts  $x$  with probability  $\log(N/2)/\log x$ , producing a uniform distribution.

To select a factor with approximately uniform length, the algorithm chooses prime powers  $q$  with odds  $\Delta_N(q)$  (defined below). To do this, it first chooses integers  $q$  in the following way: 2 and 3 each appear with odds 1/2, 4, 5, 6, and 7 each appear with odds 1/4, and so on. It turns out that  $\Delta_N(q) < 1/q$ , so acceptance-rejection is used twice: first to produce the distribution  $\Delta_N(q)$ , and then to throw away  $q$ 's that are not prime powers.

**4. The complete algorithm.** This section presents an explicit program for generating factored random integers, using the language of real numbers. It assumes the uniform (0,1) distribution as a source of randomness; § 7 will show how this can be simulated by a fair coin.

For real numbers  $a$  and  $b$ , let  $\#(a,b]$  denote the number of integers  $x$  satisfying  $a < x \leq b$ . For prime powers  $q = p^\alpha$ , and integers  $N$ , let

$$(1) \quad \Delta_N(q) = \frac{\log p}{\log N} \cdot \frac{\#(N/2q, N/q)}{N}.$$

Note that if  $[x]$  denotes the greatest integer  $\leq x$ , then  $\#(a/2, a] = [(a+1)/2]$ ; this implies the frequently used estimate

$$(2) \quad (a-1)/2 \leq \#(a/2, a] \leq (a+1)/2$$

and also shows that for  $q \leq N$ ,

$$\Delta_N(q) \leq \frac{N/q+1}{2N}.$$

The innermost part of the program selects random prime powers; using the above notation, it is defined below.

PROCESS F: Factor generator.

- (\*) Select a random integer  $j$  with  $1 \leq j \leq \log_2 N$ .  
 Let  $q = 2^j + r$ , where  $r, 0 \leq r < 2^j$ , is chosen at random.  
 Choose a random real number  $\lambda$  from the  $U(0, 1)$  distribution.  
 If  $q$  is a prime power,  $q \leq N$ , and  $\lambda < \Delta_N(q)2^{\lceil \log_2 q \rceil}$ , output  $q$ .  
 If not, go back to (\*).

The salient features of this process are given by the following result.

**THEOREM 2.** *Process F almost surely halts; the number of times (\*) is reached has a geometric distribution whose expected value is  $O(\log N)$ . It outputs a prime power  $q = p^\alpha$ ,  $2 \leq q \leq N$ , with odds  $\Delta_N(q)$ . The stopping time and the output value are independent.*

*Proof.* The first two steps select  $q$  with odds  $2^{-\lceil \log_2 q \rceil}$ , and since  $2^{\lceil \log_2 q \rceil} \Delta_N(q) \leq (N+q)/2N \leq 1$  for  $q \leq N$ , Theorem 1 implies that  $q$  is output with the stated probability. For the stopping time estimate, since

$$P = \sum_{q=2}^N 2^{-\lceil \log_2 q \rceil} \leq \log_2 N,$$

it will suffice to show that

$$Q = \sum_{q \leq N} \Delta_N(q)$$

is bounded below by an absolute constant. This follows from two consequences of the prime number theorem given by Rosser and Schoenfeld [14, p. 65]:

$$(3) \quad \sum_{p \leq N} \log p = N + O(N/\log N)$$

and

$$(4) \quad \sum_{p \leq N} \log p/p = \log N + O(1).$$



Using (3) and (4),

$$(5) \quad \sum_{q \leq N} \Delta_N(q) \geq \sum_{p \leq N} \Delta_N(p) \geq \sum_{p \leq N} \frac{\log p}{2p \log N} - \sum_{p \leq N} \frac{\log p}{2N \log N} = \frac{1}{2} + O\left(\frac{1}{\log N}\right).$$

The independence statement is a consequence of Theorem 1.  $\square$

Just as in the heuristic sketch, the factor generator is a subroutine in the main program, presented below. This process uses a “stopping value”  $N_0$ , which can be any convenient number.

**PROCESS R: Random factorization generator.**

If  $N \leq N_0$ , pick a random  $x$ ,  $N/2 < x \leq N$ , output the factors of  $x$ , and stop.

(†) Select a prime power  $q = p^\alpha$ ,  $2 \leq q \leq N$ , using Process F.

Let  $N' = \lfloor N/q \rfloor$ .

Call Process R recursively to choose a factored random  $y$  with  $N'/2 < y \leq N'$ .

Let  $x = y \cdot q$ .

(§) Choose a real number  $\lambda$  from the  $U(0,1)$  distribution.

If  $\lambda < \log(N/2)/\log x$ , output the factors of  $x$  and stop.

If not, return to (†).

The main result of this paper is the following theorem.

**THEOREM 3.** *Process R generates uniformly distributed random integers  $x$ ,  $N/2 < x \leq N$ , in factored form.*

*Proof.* If  $N \leq N_0$ , there is nothing to prove. Otherwise, note that for integers  $y$ ,  $\lfloor x \rfloor / 2 < y \leq \lfloor x \rfloor$  if and only if  $x/2 < y \leq x$ , and so the recursive step chooses an integer  $y$  uniformly with  $N/2q < y \leq N/q$ . Therefore, by Theorem 2,  $x$  is chosen at step (§) with probability proportional to

$$\sum_{q=p^\alpha | x} \frac{\Delta_N(q)}{\#(N/2q, N/q)} = \sum_{q=p^\alpha | x} \frac{\log p}{\log N} \cdot \frac{\#(N/2q, N/q)}{N} \cdot \frac{1}{\#(N/2q, N/q)} = \frac{\log x}{N \log N}.$$

By Theorem 1, the last part of the algorithm ensures that  $x$  is output with probability  $1/\#(N/2, N)$ .  $\square$

**5. The distribution of factor lengths.** It was stated earlier that Process F produces a factor  $q$  whose length is roughly uniformly distributed. This can be refined into the following precise statement: as  $N \rightarrow \infty$ ,  $\log q/\log N$  converges in distribution to a uniform (0,1) random variable.<sup>2</sup> This implies the following: if we define

$$F_N(x) = \Pr[q \leq x \text{ in Process F}],$$

then  $F_N(x)$  is close to  $\log x/\log N$ . Similarly, the expected values  $E[\log(N/q)]$  and  $E[\log^2(N/q)]$  are close to  $\frac{1}{2} \log N$  and  $\frac{1}{3} (\log N)^2$ , respectively. The next three lemmas give upper bounds corresponding to these approximations, which are used in the next section.

**LEMMA 1.** *If  $N > 30$ , and  $2 \leq x \leq N$ , then*

$$F_N(x) \leq \frac{\log x + 2}{\log N - 2}.$$

*Proof.* For  $N > 30$ ,

$$\sum_{p^\alpha \leq N} \log p \leq 1.04N$$

<sup>2</sup> I will not prove this as it is not needed later.

and

$$\log N - \gamma - \beta - \frac{1}{2 \log N} \leq \sum_{p \leq N} \frac{\log p}{p} \leq \log N,$$

where

$$\gamma = 0.577215 \dots$$

is Euler's constant and

$$\beta = \sum_{\alpha \geq 2} \log p / p^\alpha = 0.755366 \dots$$

(these are (3.21), (3.24) and (3.35) from Rosser and Schoenfeld [14]). Using these inequalities, plus (1) and (2),

$$2 \log N \sum_{q \leq N} \Delta_N(q) \geq \log N - \gamma - \beta - \frac{1}{2 \log N} + \sum_{\substack{p^\alpha \leq N \\ \alpha \geq 2}} \frac{\log p}{p^\alpha} - 1.04 > \log N - 2.$$

Similarly

$$2 \log N \sum_{q \leq x} \Delta_N(q) \leq \sum_{p \leq x} \frac{\log p}{p} + \beta + \sum_{q \leq x} \frac{\log p}{N} \leq \log x + 2.$$

Now apply these to the formula  $F_N(x) = \sum_{q \leq x} \Delta_N(q) / \sum_{q \leq N} \Delta_N(q)$ .  $\square$

LEMMA 2. For  $N > 30$ ,

$$E[\log(N/q)] \leq \frac{\log N}{2} \cdot \frac{\log N + 4}{\log N - 2}.$$

*Proof.* The expectation can be expressed as a Stieltjes integral:

$$\int_{2^-}^N \log(N/x) dF_N(x).$$

Using integration by parts and Lemma 1,

$$\int_{2^-}^N \log(N/x) dF_N(x) = \int_2^N \frac{F_N(x)}{x} dx \leq \int_1^N \frac{\log x + 2}{\log N - 2} \cdot \frac{dx}{x},$$

now computing the integral gives the result.  $\square$

LEMMA 3. For  $N > 30$ ,

$$E \log^2(N/q) \leq \frac{(\log N)^2}{3} \cdot \frac{\log N + 6}{\log N - 2}.$$

*Proof.* As in the last proof, the expectation is

$$\int_{2^-}^N \log^2(N/x) dF_N(x) \leq \frac{2}{\log N - 2} \int_1^N (\log N - \log x)(2 + \log x) \frac{dx}{x}. \quad \square$$

**6. The expected number of prime-power tests.** This section proves that the number of prime-power tests done by Process R on input  $N$  has expected value and standard deviation that are both  $O(\log N)$ .

For every  $N$ , define random variables as follows.  $T_N$  is the number of prime-power tests done by Process R on input  $N$ , and  $U_N$ ,  $V_N$ , and  $W_N$  count the prime-power tests done during the first call to Process F, the recursive call, and after the first return to  $(\dagger)$ , respectively.

**THEOREM 4.** *If  $N_0 > 10^6$ ,  $E(T_N) = O(\log N)$ .*

*Proof.* Let  $N > N_0$ , for otherwise the theorem is true immediately. By Theorem 2, we can choose  $C > 0$  so that  $U_N \leq C \log N$ ; we now prove by induction on  $N$  that  $T_N \leq 6C \log N$ .

Since  $T_N = U_N + V_N + W_N$ ,  $E(T_N) = E(U_N) + E(V_N) + E(W_N)$ . By the definition of  $C$  and the formula  $E(X) = E_Y(E(X|Y))$  applied to  $V_N$ , this gives

$$E(T_N) \leq C \log N + E_q(E(T_{\lfloor N/q \rfloor})) + \frac{\log 2}{\log N} E(T_N),$$

since the probability of renewal is at most  $1 - \log(N/2)/\log N$ . By induction and Lemma 2,

$$\begin{aligned} E(T_N) &\leq C \log N + 6CE(\log N/q) + \frac{\log 2}{\log N} E(T_N) \\ &\leq C \log N + 6C \frac{\log N}{2} \cdot \frac{\log N + 4}{\log N - 2} + \frac{\log 2}{\log N} E(T_N). \end{aligned}$$

This implies

$$E(T_N) \leq \frac{1}{1 - \log 2/\log N} \left( 1 + 3 \frac{\log N + 4}{\log N - 2} \right) C \log N,$$

and for  $N > 10^6$  the coefficient of  $C \log N$  is less than 6.  $\square$

The corresponding estimate for the variance is given below.

**THEOREM 5.** *If  $N_0 > 10^6$ ,  $\sigma^2(T_N) = O(\log N)^2$ .*

*Proof.* Let  $\bar{R}$  denote the process obtained by replacing the top level stopping condition  $\lambda < \log(N/2)/\log x$  by  $\lambda < 1 - \log 2/\log N$ ; the recursive call uses the unaltered Process R. We can consider both processes to be defined on the same sample space; then (extending the notation in an obvious fashion)

$$\bar{T}_N = U_N + V_N + \bar{W}_N.$$

Since  $U_N$ ,  $V_N$ , and  $\bar{W}_N$  are independent,

$$\sigma^2(\bar{T}_N) = \sigma^2(U_N) + \sigma^2(V_N) + \sigma^2(\bar{W}_N).$$

Using the formulas  $\sigma^2(X) = E(X^2) - E(X)^2$  and  $E(X) = E_Y(E(X|Y))$ ,

$$E(\bar{T}_N^2) \leq \sigma^2(U_N) + E(\bar{T}_N)^2 + E_q(E(T_{\lfloor N/q \rfloor}^2)) + \frac{\log 2}{\log N} E(\bar{T}_N^2).$$

The proof of Theorem 4 actually shows that  $E(\bar{T}_N) = O(\log N)$ ; we can therefore choose  $D > 0$  so that  $\sigma^2(U_N) + E(\bar{T}_N)^2 \leq D(\log N)^2$ . Furthermore, for any  $N$ ,  $T_N \leq \bar{T}_N$ , so

$$E(\bar{T}_N^2) \leq D(\log N)^2 + E_q(E(\bar{T}_{\lfloor N/q \rfloor}^2)) + \frac{\log 2}{\log N} E(\bar{T}_N^2).$$

An argument similar to the proof of Theorem 4, using Lemma 3, will show that  $E(\bar{T}_N^2) \leq 4D(\log N)^2$ . Since

$$\sigma^2(T_N) \leq E(T_N^2) \leq E(\bar{T}_N^2),$$

the result follows.  $\square$

**7. A single-precision time bound.** The next two sections analyze the average number of single-precision operations needed to generate a random factorization. Any serious discussion of this must answer two questions. First, the algorithm uses real numbers, which are not finite objects; how can these be simulated? Second, one might wish to use randomized prime testing; what happens when the prime tester can make mistakes?

This section addresses the real-number issue, assuming perfect prime testing; probabilistic prime testing will be treated in the next section. In what follows, a “step” means one of the basic arithmetic operations (including comparison) on single-bit numbers, or a coin flip. All questions of addressing and space requirements will be ignored.

The following result will be used repeatedly.

LEMMA 4. *Let  $T_1, T_2, \dots$  be a sequence of random variables, and let  $n$  be a positive integer-valued random variable, such that  $T_i = 0$  for every  $i > n$ . If  $E(T_i | n \geq i) \leq A$  and  $E(n) \leq B$ , then  $E(\sum_{i=1}^n T_i) \leq AB$ .*

*Proof.*

$$E\left(\sum_{i=1}^n T_i\right) = \sum_{i=1}^{\infty} E(T_i) = \sum_{i=1}^{\infty} E(T_i | n \geq i) \Pr[n \geq i] \leq A \cdot \sum_{i=1}^{\infty} \Pr[n \geq i] \leq AB. \quad \square$$

At several points in our algorithm we need to flip a coin with success probability  $\theta$ , where  $\theta$  is a fixed real number between 0 and 1. This means we compare  $\theta$  with a randomly generated real value  $\lambda$ , thus:

$$\begin{aligned} \theta &= .0101010101 \dots, \\ \lambda &= .0110010111 \dots \end{aligned}$$

A finitary procedure with the same effect simply does the comparison bit-by-bit from the left, only generating bits as they are needed. This is clearly fast; since the bits of  $\lambda$  are specified by independent coin flips, we expect to use only two of them before reaching a decision. However, it may not be convenient to generate the true bits of  $\theta$  one at a time; to avoid this difficulty, we base our procedure on approximations that might be produced by some scheme like interval arithmetic.

We need the following definition. Let  $\theta$ ,  $0 \leq \theta \leq 1$ , be a real number. A *k-bit approximation to  $\theta$*  is an integer multiple  $\theta_k$  of  $2^{-k}$  with  $|\theta_k - \theta| \leq 2^{-k}$  and  $0 < \theta_k < 1$ .

The lemma below eliminates real numbers from our algorithm; it states, in effect, that if  $\theta$  is approximable by *any* polynomial-time procedure, then a biased coin flip with success probability  $\theta$  takes constant time on the average.

LEMMA 5. *Let  $0 \leq \theta \leq 1$ , and assume that a k-bit approximation to  $\theta$  can be computed in  $f(k)$  steps, where  $f$  is a polynomial of degree  $m$  with nonnegative coefficients. Then the expected time to decide if a uniform  $(0,1)$  random variable is less than  $\theta$  is at most  $C_m f(1)$ , where  $C_m$  depends only on  $m$ .*

*Proof.* Let  $\lambda$  be the uniform  $(0,1)$  value; we can assume it is irrational, since the set of rational numbers has measure 0. Consider the following procedure: for  $k = 1, 2, 3, \dots$ , compute a  $k$ -bit approximation  $\theta_k$  to  $\theta$  and compare this to  $\lambda_k$ , the number formed from the first  $k$  bits of  $\lambda$ ; if  $\theta_k + 2^{-k} \leq \lambda_k$  or  $\lambda_k < \theta_k - 2^{-k}$ , terminate the process. The probability that no decision is reached after  $k$  steps is at most  $2^{1-k}$ , so the expected

total time is at most a constant times

$$\sum_{k=1}^{\infty} f(k)2^{1-k} = 2 \sum_{k=1}^{\infty} \sum_{j=0}^m a_j k^j 2^{-k} = 2 \sum_{j=0}^m a_j \sum_{k=1}^{\infty} k^j 2^{-k},$$

where  $a_0, \dots, a_m$  are the coefficients of  $f$ . Polyá and Szegö [12, p. 9] give the formula (valid for  $|z| < 1$ )

$$\sum_{k=1}^{\infty} k^j z^k = \frac{g_j(z)}{(1-z)^{j+1}},$$

where  $g_j$  is a polynomial with nonnegative coefficients, satisfying  $g_j(1) = j!$ . The result follows by taking  $z = 1/2$  and  $C_m = 2^{m+2} m!$ .  $\square$

LEMMA 6. *Let  $p, q, N$  be integers with  $2 \leq p \leq q \leq N$ . Then a  $k$ -bit approximation to*

$$\theta = \frac{\log p}{\log N} \cdot \frac{\#(N/2q, N/q) 2^{\lceil \log_2 q \rceil}}{N}$$

can be computed in  $O(k^3 + \log \log N)$  steps.

*Proof.* Let  $p = 2^{\alpha} \cdot \varepsilon$  and  $N = 2^{\beta} \cdot \eta$ , where  $\alpha$  and  $\beta$  are integers and  $1 \leq \varepsilon, \eta < 2$ . Then

$$(6) \quad \theta = \frac{\alpha \log 2 + \log \varepsilon}{\beta \log 2 + \log \eta} \cdot \frac{[(N+q)/2q] 2^{\lceil \log_2 q \rceil}}{N}.$$

We approximate  $\theta$  by using floating-point numbers to perform the computation implicit in the above expression;  $k + O(1)$  bits of precision suffice to get an absolute error less than  $2^{-k}$ , by the following argument. First, since  $0 \leq \theta \leq 1$ , it suffices to make the relative error in the result less than  $2^{-k}$ . Brent [3, Thm. 6.1] shows that on the interval  $1 \leq x \leq 2$ , one can compute  $\log x$  with relative error  $2^{-n}$  in  $O(n^3)$  steps. If we take  $n = k + O(1)$ , we will have enough guard bits to nullify the effect of any remaining error, since there are only a finite number of further operations. All the numbers involved have exponents that are less than  $\log_2 N$ , so the bound follows.  $\square$

LEMMA 7. *Let  $x$  and  $N$  be positive integers with  $N/2 < x \leq N$ . Then a  $k$ -bit approximation to  $\log(N/2)/\log x$  can be computed in  $O(k^3 + \log \log N)$  steps.*

*Proof.* Approximate the logarithms as indicated in the proof of Lemma 6.  $\square$

LEMMA 8. *Let  $q > 1$  be an integer. Then solving  $p^\alpha = q$  for an integer  $p$  and the largest possible integer  $\alpha$  can be done in  $O(\log q)^3 (\log \log q)^2$  steps.*

*Proof.* Let  $d = \lceil \log_2 q \rceil$ ; then necessarily  $\alpha \leq d$ . For each such value of  $\alpha$ , we solve  $X^\alpha = q$  by bisection, using 0 and  $2^{\lceil d/\alpha \rceil + 1}$  as starting values. This will find a solution or prove that none exists after  $O(d/\alpha)$  evaluations of  $f(X) = X^\alpha$ . The total time is therefore at most a constant times

$$(\log q)^3 \sum_{2 \leq \alpha \leq d} 2 \leq \alpha \leq d \frac{\log \alpha}{\alpha} = O(\log q)^3 (\log \log q)^2. \quad \square$$

The next two results assume the Extended Riemann Hypothesis (ERH), a famous conjecture of analytic number theory. The details of this hypothesis (for which see Davenport's book [4, p. 124]) are not important here; what matters is the following consequence, first proved by Miller [10].

LEMMA 9 (ERH). *To test if an integer  $p$  is prime requires  $O(\log p)^5$  operations.*

*Proof.* We write  $\nu_2(x)$  for the largest  $e$  such that  $2^e | x$ , and  $\mathbb{Z}_p^*$  for the multiplicative subgroup of integers modulo  $p$ . Then Miller's primality criterion states that  $p$  is prime

if and only if every  $a \in \mathbb{Z}p^*$  satisfies

$$(7) \quad a^{p-1} \equiv 1 \text{ and for all } k < \nu_2(p-1), a^{(p-1)/2^k} \equiv 1 \text{ implies } a^{(p-1)/2^{k+1}} \equiv \pm 1$$

(all congruences are interpreted modulo  $p$ ). If  $p$  is composite, there is a proper subgroup  $G$  of  $\mathbb{Z}p^*$  such that (7) is violated for all  $a \notin G$  [9, proof of Thm. 6]. The ERH implies that there is some number outside  $G$  that is less than  $2(\log p)^2$  [2, Thm. C]. Therefore  $p$  is prime if and only if condition (7) holds for all positive  $a \leq 2(\log p)^2$ ; the result follows.  $\square$

We now make the following changes to our algorithm: prime-power testing is done as indicated in the proofs of lemmas 8 and 9, and the real number calculations are done as indicated in Lemmas 5, 6, and 7. With these modifications we have our single-precision result.

**THEOREM 6 (ERH).** *The expected number of single-precision operations (arithmetic, comparison, coin flips) needed by Process R on input  $N$  is  $O(\log N)^6$ .*

*Proof.* Theorem 4 and inspection of the algorithm imply that none of its steps can be executed more than  $O(\log N)$  times on the average. By Lemma 4, it suffices to show that no single step of the algorithm has expected time greater than the  $O(\log N)^5$  steps sufficient to test a number less than  $N$  for primality. By Lemmas 5, 6, and 7, this is true for the real number comparisons. Everything else is easily estimated.  $\square$

The real point to this extravagant bound is that it is a polynomial in  $\log N$ . By using the prime test of Adleman, Pomerance, and Rumely [1], one can also prove an unconditional almost-polynomial time bound of  $O(\log N)^{O(\log \log \log N)}$ . However, much better estimates can be obtained by using probabilistic prime testing, as described in the next section.

**8. The use of probabilistic primality tests.** This section proves theorems analogous to the preceding results, assuming that a randomized prime test is used. First, a definition: call a factorization  $x = p_1^{e_1} \dots p_k^{e_k}$  *complete* if all the  $p_i$ 's are prime; if it uses a probabilistic prime test, Process R may output an incompletely factored number. The results in this case can be simply summarized: the distribution of completely factored numbers is still uniform, and incompletely factored numbers can be made exponentially unlikely at very little cost.

The following result is analogous to Lemma 9.

**LEMMA 10.** *To test if  $p$  is prime with bounded by  $4^{-n}$  (error only being possible when  $p$  is composite) requires  $n \cdot O(\log p)^3$  operations.*

*Proof.* Rabin [13, Thm. 1] shows that condition (7) is violated for a random  $a$  with probability at most  $1/4$ , so choose  $n$  independent random values of  $a$ .  $\square$

The prime tests referred to above have a very nice property; the decision is never wrong unless the input is composite. This is the key observation in the next proof.

**THEOREM 7.** *If the prime test used in Process F produces correct answers when the input is prime, then the distribution of completely factored outputs is uniform.*

*Proof.* Use induction on  $N$ ; if  $N < N_0$ , this is clear. Otherwise, the prime powers produced by Process F have the same relative distribution as before, since the prime tester never makes a mistake on prime input. Since every subfactorization of a complete factorization is complete, the calculation that proves Theorem 3 is still valid.  $\square$

The order-of-magnitude bound for the average number of prime power tests still holds, if the prime test used is sufficiently accurate. Since the average number of tests increases with  $N$ , a constant number of the tests (7) per prime will not suffice. Instead, we choose a bound  $\varepsilon(N)$  in advance, and make every prime test used by the algorithm have a chance of error at most  $\varepsilon(N)$ .

**THEOREM 8.** *Assume that the prime test used in Process F is correct on prime input, and has error probability at most  $\varepsilon(N)$  on composite input. If  $\varepsilon(N) < N^{-2}$ , then the number of prime-power tests done by Process R on input  $N$  has mean and standard deviation that are  $O(\log N)$ , and the probability that an incompletely factored number is produced is  $O(\varepsilon(N) \log N) = O(\log N / N^2)$ .*

*Proof.* For the time bound, it is only necessary to show that Lemma 1 still holds, say for  $N > 10^6$ . The proof of Lemma 1 amounted to a lower bound on the relative probability that  $q \leq N$  and an upper bound on the relative probability that  $q \leq x$ . The lower bound still holds, and the new upper bound is at most

$$\sum_{q \leq x} \Delta_q + \sum_{y \leq x} \varepsilon \frac{\log y \cdot \#(N/2y, N/y]}{\log N \cdot N}.$$

The second term is at most

$$\frac{1}{2 \log N} \cdot \frac{1}{N^2} \sum_{y \leq x} \left( \frac{\log y}{y} + \frac{\log y}{N} \right) \leq \frac{1}{2 \log N} \cdot \left( \frac{\log N}{N} \right)^2,$$

and this will not cause the bound to be exceeded. For the estimate relating to incorrect output, apply Lemma 4 to the random variables  $X_i$  that are 1 if the  $i$ th prime test is incorrect, and 0 otherwise, and use the inequality  $\Pr[X \geq 1] \leq E(X)$ .  $\square$

By Lemma 10, error probability less than  $N^{-2}$  can be obtained with about  $\log_2 N$  tests, each using  $O(\log N)^3$  steps. This will give a polynomial time bound analogous to Theorem 6.

**9. Experiments.** This section has two purposes: to show how the algorithm actually behaves in practice, and to discuss what modifications are necessary to implement it efficiently.

Call an arrival at (\*) in Process F a *probe*; Theorem 4 implies that Process R requires  $O(\log N)$  probes on the average. The constant implied by the “ $O$ ” symbol can be estimated by the following heuristic argument. Typically the algorithm will produce factors whose lengths are  $1/2, 1/4, 1/8, \dots$  the length of  $N$ . Presumably, then, the average number of probes is close to

$$2 \log_2 N (1 + 1/2 + 1/4 + \dots) = 4 \log_2 N,$$

since by (5), we expect Process F to use about  $2 \log_2 N$  probes on input  $N$ .

This value of  $4 \log_2 N$  is also the best bound provable as the stopping value  $N_0 \rightarrow \infty$ , and the first set of experiments were designed to see if this estimate is at all realistic when  $N_0$  is small.

To do this, I coded the algorithm verbatim in C (the “real” numbers have about 15 decimal digits of precision) with  $N_0 = 4$ . Table 1 gives statistics on the number of probes required to generate 100 random numbers for various values of  $N$ , together with the presumed mean value  $4 \log_2 N$ .

It will be seen from this table that the standard deviation tends to be a bit smaller than the mean; however, I do not even have a heuristic argument to justify this observation.

To test its feasibility for large  $N$ , I also coded the algorithm with multiprecision arithmetic on a DEC VAX 11-780, a machine that takes about 5 microseconds to multiply two 32-bit integers. When dealing with multi-word numbers, efficiency is of primary importance; this concern led to the following version of Process F.

- 1) After building the random value  $q$ , the program first checks that  $q \leq N$ , and then computes  $\Delta_N(q) 2^{\lceil \log_2 q \rceil}$  in single precision with the formula (6), as if  $q$

TABLE 1  
*Statistics on the number of probes.*

$N$	$4\log_2 N$	Average	Maximum	Standard deviation
$10^2$	26.56	20.77	60	13.95
$10^3$	39.86	35.59	218	34.31
$10^4$	53.15	56.30	367	47.49
$10^5$	66.44	71.30	329	62.24
$10^6$	79.73	72.49	472	59.58
$10^7$	93.01	74.61	346	63.43
$10^8$	106.30	111.78	491	85.30
$10^9$	119.59	120.82	453	96.78

were prime. Only if this exceeds the random value  $\lambda$  does it subject  $q$  to a prime-power test.

- 2) The first part of this test sees if for any small prime  $p$ ,

$$p|q \text{ and } p^2 \nmid q;$$

if this is true,  $q$  cannot be a prime power. This sieve procedure eliminates most  $q$  from further consideration, for the probability that a random number  $q$  survives this test for all  $p \leq B$  is about

$$\prod_{p \leq B} \left( \frac{p-1}{p} + \frac{1}{p^2} \right).$$

If we let

$$\alpha = \prod_p \left( 1 + \frac{1}{p(p-1)} \right) = 1.943596 \dots$$

then the survival probability is, by Mertens's theorem [6, p. 22], close to

$$\alpha \cdot \frac{e^{-\gamma}}{\log B}.$$

The program used  $B = 1000$ , which screens out approximately 84% of the  $q$ 's.

- 3) If  $q$  passes through the sieve, it is subjected to a "pseudo" test designed to cheaply eliminate numbers that are not prime powers. This checks that  $2^{q-1} \not\equiv 1$  and  $\gcd(2^{q-1} - 1, q) = 1$ ; if so,  $q$  can be thrown away. The average cost of this is one modular exponentiation and one gcd computation.
  - 4) Any number  $q$  that has survived so far is tested for primality, using (7) with  $a = 2, 3, 5, \dots, 29$  (the first ten primes). There is a slight advantage to small values of  $a$ , since about one-third of the multiplications done by the modular exponentiation algorithm will involve a single-word quantity.
  - 5) Only if  $q$  is not declared "prime" by the above procedure does the program try to see if it is a perfect power.
- (Various orderings of steps 1-5 were tried, and the one above seems to be the best.)

For the multiprecise implementation, a more realistic measure of the work done is the number of times  $q$  reaches the sieve. Statistics on these values are given in Table 2, from runs that generated 50 numbers each; the last column gives the average CPU time required per random factorization.



TABLE 2  
*Statistics on the number of sieve tests, and the running time in seconds.*

$N$	Average	Maximum	Standard deviation	Average time
$10^{15}$	29.66	67	17.70	1.51
$10^{30}$	65.42	225	46.36	4.66
$10^{60}$	158.92	387	95.04	20.52
$10^{120}$	250.34	805	174.54	100.08

It is worth noting that for these values of  $N$  the average running time is only slightly worse than  $O(\log N)^2$ .

Table 3 presents the mean values of four quantities related to the output values  $x$ : the number of prime factors of  $x$ , and the number of decimal digits in the largest three factors of  $x$  (from the same experiments). It will be seen that the average number of prime factors grows very slowly with  $N$ ; the observations are close to the mean values of  $\log \log N + 1.03465 \dots$  predicted by prime number theory [7, p. 346]. Finally, the average lengths of the largest three factors are roughly proportional to the length of  $N$ ; again, this is predicted by theory [7, p. 343], with constants of proportionality close to 0.62, 0.21, and 0.088, respectively.

TABLE 3  
*Statistics on the prime factors.*

$N$	Average number	Digits in largest	2nd largest	3rd largest
$10^{15}$	4.48	10.06	3.76	1.50
$10^{30}$	5.32	19.84	6.80	2.80
$10^{60}$	6.18	37.36	13.48	5.94
$10^{120}$	6.70	78.54	26.56	8.46

**Acknowledgments.** I would like to thank the following people for offering encouragement and ideas: Silvio Micali, Manuel Blum, Martin Hellman, Michael Luby, and James Demmel. I also heartily thank the seminonymous referees, whose comments improved the paper immensely.

#### REFERENCES

- [1] L. M. ADLEMAN, C. POMERANCE AND R. S. RUMELY, *On distinguishing prime numbers from composite numbers*, Ann. of Math., 117 (1983), pp. 173–206.
- [2] E. BACH, *Analytic Methods in the Analysis and Design of Number-Theoretic Algorithms*, MIT Press, Cambridge, 1985 (U.C. Berkeley Ph.D. dissertation, 1984).
- [3] R. P. BRENT, *Fast multiple-precision evaluation of elementary functions*, J. Assoc. Comput. Mach., 23 (1976), pp. 242–251.
- [4] H. DAVENPORT, *Multiplicative Number Theory*, Springer, New York, 1980.
- [5] W. FELLER, *An Introduction to Probability Theory and Its Applications (Volume I)*, John Wiley, New York, 1968.
- [6] A. E. INGHAM, *The Distribution of Prime Numbers*, Cambridge University Press, Cambridge, 1932.
- [7] D. KNUTH AND L. TRABB PARDO, *Analysis of a simple factorization algorithm*, Theoret. Comput. Sci., 3 (1976), pp. 321–348.
- [8] H. W. LENSTRA, JR., *Elliptic Curve Factorization*, Ann. of Math., 126 (1987), pp. 649–673.
- [9] M. MIGNOTTE, *Tests de primalité*, Theoret. Comput. Sci., 12 (1980), pp. 109–117.

- [10] G. L. MILLER, *Riemann's hypothesis and tests for primality*, J. Comput. System Sci., 13 (1976), pp. 300-317.
- [11] J. VON NEUMANN, *Various techniques used in connection with random digits*, J. Res. Nat. Bur. Standards (Applied Mathematics Series), 3 (1951), pp. 36-38.
- [12] G. POLYÁ AND G. SZEGÖ, *Problems and Theorems in Analysis I*, Springer, New York, 1972.
- [13] M. O. RABIN, *Probabilistic algorithm for testing primality*, J. Number Theory, 12 (1980), pp. 128-138.
- [14] J. B. ROSSER AND L. SCHOENFELD, *Approximate formulas for some functions of prime numbers*, Ill. J. Math., 6 (1962), pp. 64-94.
- [15] B. W. SCHMEISER, *Random variate generation: a survey*, in Simulation with Discrete Models: A State-of-the-Art View, T. I. Oren, C. M. Shub and P. F. Roth, eds., IEEE Press, New York, 1981.
- [16] C. P. SCHNORR AND H. W. LENSTRA, JR., *A Monte Carlo factoring algorithm with linear storage*, Math. Comp., 43 (1984), pp. 289-311.
- [17] R. SOLOVAY AND V. STRASSEN, *A fast Monte-Carlo test for primality*, SIAM J. Comput., 6 (1977), pp. 84-85.

## RSA AND RABIN FUNCTIONS: CERTAIN PARTS ARE AS HARD AS THE WHOLE\*

WERNER ALEXI<sup>†</sup>, BENNY CHOR<sup>‡</sup>, ODED GOLDRZEICH<sup>§</sup> AND CLAUD P. SCHNORR<sup>†</sup>

**Abstract.** The RSA and Rabin encryption functions  $E_N(\cdot)$  are respectively defined by raising  $x \in Z_N$  to the power  $e$  (where  $e$  is relatively prime to  $\varphi(N)$ ) and squaring modulo  $N$  (i.e.,  $E_N(x) = x^e \pmod{N}$ ,  $E_N(x) = x^2 \pmod{N}$ , respectively). We prove that for both functions, the following problems are computationally equivalent (each is probabilistic polynomial-time reducible to the other):

- (1) Given  $E_N(x)$ , find  $x$ .
- (2) Given  $E_N(x)$ , guess the least-significant bit of  $x$  with success probability  $\frac{1}{2} + 1/\text{poly}(n)$  (where  $n$  is the length of the modulus  $N$ ).

This equivalence implies that an adversary, given the RSA/Rabin ciphertext, cannot have a non-negligible advantage (over a random coin flip) in guessing the least-significant bit of the plaintext, unless he can invert RSA/factor  $N$ . The proof techniques also yield the simultaneous security of the  $\log n$  least-significant bits. Our results improve the efficiency of pseudorandom number generation and probabilistic encryption schemes based on the intractability of factoring.

**Key words.** cryptography, concrete complexity, RSA encryption, factoring integers, partial information, predicate reductions

**AMS(MOS) subject classifications.** 11A51, 11K45, 11T71, 11Y05, 11X16, 11Z50, 68Q99, 94A60

**1. Introduction.** One-way functions are the basis for modern cryptography [11] and have many applications to pseudorandomness and complexity theories [6], [29]. One-way functions are easy to evaluate, but hard to invert. Even though no proof of their existence is known (such proof would imply  $P \neq NP$ ), it is widely believed that one-way functions do exist. In particular, if factoring large numbers (a classical open problem) is hard, then the simple function of squaring modulo a composite number is one-way [22].

*Randomness and computational difficulty play dual roles.* If a function  $f$  is one-way then, given  $f(x)$ , the argument  $x$  must be "random." It cannot be the case that every bit of the argument  $x$  is easily computable from  $f(x)$ . Therefore, some of these bits are unpredictable, at least in a weak sense. A natural question is whether these bits are strongly unpredictable. That is, are there specific bits of the argument  $x$  which cannot be guessed with the slightest advantage (over a random coin flip), given  $f(x)$ .

This question was first addressed by Blum and Micali [6]. They demonstrated such a strongly unpredictable bit for the discrete exponentiation function (which is believed to be one-way). This was done by reducing the problem of inverting the discrete exponentiation function to the problem of guessing a particular bit with any non-negligible advantage.

In this paper, we deal with two functions related to the factorization problem: Rabin's function (squaring modulo a composite number) [22] and the RSA (raising to a fixed exponent modulo a composite number) [23]. Both functions are believed to

---

\* Received by the editors October 29, 1984; accepted for publication (in revised form) July 7, 1986.

<sup>†</sup> Fachbereich Mathematik, Universität Frankfurt, 600 Frankfurt/Main, West Germany.

<sup>‡</sup> Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The work of this author was supported in part by the National Science Foundation under grant MCS-8006938 and by an IBM Graduate Fellowship.

<sup>§</sup> Computer Science Department, Technion, Haifa 32000, Israel. This research was performed while the author was at the Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The author was supported in part by a Weizmann Postdoctoral Fellowship.

be one-way. We show that both functions have strongly unpredictable bits. In particular, inverting each of them is probabilistic polynomial-time reducible to guessing the least-significant bit of their argument with any non-negligible advantage.

Our results have various applications. They allow the construction of more efficient pseudorandom bit generators [6] than those previously known, based on the intractability assumption of factoring. They allow the construction of efficient probabilistic encryption schemes [15], which hide all partial information. Finally, our results imply that the RSA public-key perfectly hides all partial information about the  $\log n$  least-significant bits of the plaintext (where  $n$  is the size of the RSA modulus).

*Organization of the paper.* In § 2 we formally define the question of security for RSA least-significant bit and cover previously known results. In § 3 we review the proof of the Ben-Or, Chor and Shamir result. This investigation is the basis for our work, which is described in § 4. Section 5 extends our proof to other RSA bits, and § 6, to bits in Rabin's scheme. In § 7 we discuss applications of our results for the construction of pseudorandom bit generators and probabilistic encryption schemes. Section 8 contains concluding remarks and two open problems.

**2. Problem definition and previous results.** We begin this section by presenting notations for two number theoretic terms which will be used throughout the paper.

**DEFINITION 1.** Let  $N$  be a natural number.  $Z_N$  will denote the ring of integers modulo  $N$ , where addition and multiplication are done modulo  $N$ .

It would be convenient to view the elements of  $Z_N$  as points on a circle (see Fig. 1).

*Convention.* Throughout the paper,  $n = \lceil \log_2 N \rceil$  will denote the length of the modulus  $N$ . All algorithms discussed in this paper have inputs of length  $O(n)$ .

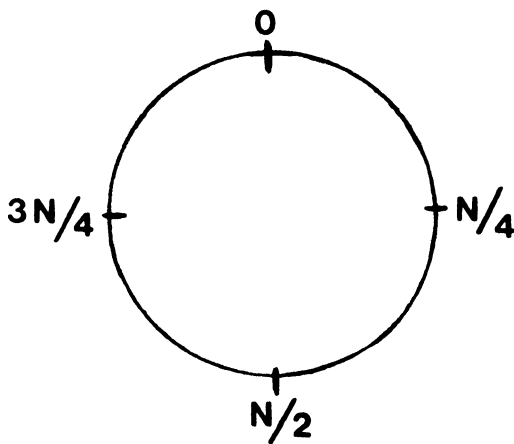


FIG. 1. Cyclic representation of  $Z_N$ .

**DEFINITION 2.** Let  $N$  be a natural number, and  $x$  an integer.  $[x]_N$  will denote the remainder of  $x$  modulo  $N$  (notice that for all  $x$ ,  $0 \leq [x]_N < N$ ).

The *RSA encryption function* is operating in the message space  $Z_N$ , where  $N = pq$  is the product of two large primes (which are kept secret). The encryption of  $x$  is  $E_N(x) = [x^e]_N$ , where  $e$  is relatively prime to  $\varphi(N) = (p-1)(q-1)$ .

We now formally define the notion of bit security for the RSA.

**DEFINITION 3.** For  $0 \leq x < N$ ,  $L_N(x)$  denotes the least-significant bit in the binary representation of  $x$ .

**DEFINITION 4.** Let  $\varepsilon(\cdot)$  be a function from integers into the interval  $[0, \frac{1}{2}]$ . Let  $\mathcal{O}_N$  be a probabilistic oracle which, given  $E_N(x)$ , outputs a guess,  $\mathcal{O}_N(E_N(x))$ , for  $L_N(x)$  (this guess might depend on the internal coin tosses of  $\mathcal{O}_N$ ). We say that  $\mathcal{O}_N$  is an  $\varepsilon(n)$ -oracle for the least-significant bit (in short,  $\varepsilon(n)$ -oracle) if the probability that  $\mathcal{O}_N$  is correct, given  $E_N(x)$  as its input, is at least  $\frac{1}{2} + \varepsilon(n)$ . The probability space is that of all  $x \in Z_N$  and all 0-1 sequences of internal coin tosses, with uniform distribution.

**DEFINITION 5.** We say that RSA least-significant bit is  $\varepsilon(n)$ -secure if there is a probabilistic polynomial time algorithm which on input  $N$ ,  $e$  (relatively prime to  $\varphi(N)$ ) and  $x \in Z_N$ , and access to an arbitrary  $\varepsilon(n)$ -oracle  $\mathcal{O}_N$ , outputs  $y$  such that  $x = E_N(y) = y^e \pmod{N}$ . (That is, the algorithm inverts  $E_N$  using any  $\varepsilon(n)$ -oracle  $\mathcal{O}_N$ .)

**DEFINITION 6.** We say that RSA least-significant bit is unpredictable if it is  $n^{-c}$ -secure for every constant  $c > 0$ .

**2.1. Previous work.** Goldwasser, Micali and Tong [17] were the first to investigate the security question of least-significant bit in RSA. They showed that the least-significant bit is as hard to determine as inverting the RSA. Furthermore, they showed that it is  $(\frac{1}{2} - 1/n)$ -secure.

In a key paper, Ben-Or, Chor and Shamir [2] showed a  $(\frac{1}{4} + 1/\text{poly}(n))$ -security result. They showed that inverting the RSA is polynomially reducible to determining the parity of messages taken from a certain small subset of the message space. To determine the parity of these messages, they performed many independent “measurements,” each consisting of querying the oracle on a pair of related points. This sampling method amplified the  $\frac{1}{4} + 1/\text{poly}(n)$  overall advantage of the oracle to “almost certainty” in determining parity for the above-mentioned subset. On the negative side, the sampling of pairs of points doubles the error of the oracle and prevents the use of less reliable oracles.

Vazirani and Vazirani [27] showed that the “error doubling” phenomena could be overcome by introducing a new oracle-sampling technique. They proved that incorporating their technique in the Ben-Or, Chor and Shamir algorithm, yields a 0.232-security result. Goldreich [13] used a better combinatorial analysis to show that the Vazirani and Vazirani algorithm yields a 0.225 result. He also pointed out some limitations of the Vazirani and Vazirani proof techniques.

All these results leave a large gap towards the desired result of proving that the least-significant bit is unpredictable (i.e.,  $1/\text{poly}(n)$  secure).

**3. A description of Ben-Or, Chor and Shamir reduction.** In this section, we present the reduction used by Ben-Or, Chor and Shamir [2]. It consists of two major parts: An algorithm which inverts the RSA using a *parity subroutine*, and a method of implementing the parity subroutine by querying an oracle for the least-significant bit.

**3.0. Definition of parity.** Let  $x$  be an integer. We define

$$\text{abs}_N(x) \stackrel{\text{def}}{=} \begin{cases} [x]_N & \text{if } [x]_N < N/2, \\ N - [x]_N & \text{otherwise.} \end{cases}$$

Pictorially,  $\text{abs}(x)$  can be viewed as the distance from  $[x]_N$  to 0 on the  $Z_N$  circle (see Fig. 2). Notice that  $\text{abs}_N(x) = \text{abs}_N(-x)$ .

The *parity* of  $x$ ,  $\text{par}_N(x)$ , is defined as the least-significant bit of  $\text{abs}_N(x)$ . For example,  $\text{par}_N(N-3) = 1$ .

**3.1. Inverting RSA using a parity subroutine.** Given an encrypted message,  $E_N(x)$ , the plaintext  $x$  is reconstructed as follows. First, two integers  $a, b \in Z_N$  are picked at

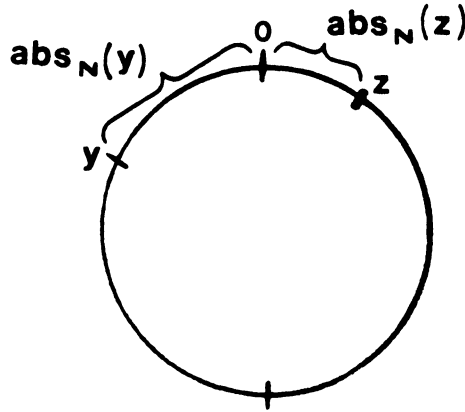


FIG. 2. The  $\text{abs}_N$  function.

random. A Brent-Kung gcd procedure [7] is applied to  $[ax]_N$  and  $[bx]_N$ . This gcd procedure uses a parity subroutine PAR which we assume, at this point, to give correct answers. Even though neither  $[ax]_N$  nor  $[bx]_N$  are explicitly known, we can manipulate them via their encryption. In particular, we can compute the encryption of any linear combination  $A[ax]_N + B[bx]_N$  when both  $A, B$  are known (since  $N$  and  $e$  are given, and  $E_N$  is a multiplicative function). When the gcd procedure terminates, we get a representation of  $\text{gcd}([ax]_N, [bx]_N) = [lx]_N$  in the form  $l$  and  $E_N(lx)$ . If  $[ax]_N$  and  $[bx]_N$  are relatively prime, then  $[lx]_N = 1$ . This fact can be detected since  $E_N(1) = 1$ . Therefore,  $x \equiv l^{-1} \pmod{N}$  can be easily computed.

1. **procedure** RSA INVERSION:
2. INPUT  $\leftarrow E_N(x)$  (and  $N, e$ )
3. **Step 1—Randomization**
4. Pick  $a, b \in Z_N$  at random.
5. **Step 2—Brent-Kung GCD** of  $[ax]_N, [bx]_N$   
 $\{z_1 = [ax]_N, z_2 = [bx]_N\}$
6.  $\alpha \leftarrow n,$
7.  $\beta \leftarrow n$
8. count  $\leftarrow 0$
9. **repeat**
10. **while** PAR  $(b, E_N(x)) = 0$  **do**  
*Comment: PAR  $(b, E_N(x))$  returns a guess for  $\text{par}_N(bx)$*   
 $b \leftarrow [b/2]_N$   
 $\{\text{gcd}([ax]_N, [bx]_N) = \text{gcd}(z_1, z_2)\}$
11.  $\beta \leftarrow \beta - 1$
12. count  $\leftarrow$  count + 1
13. **if** count  $> 6n + 3$  **then go to line 3**
14. **od**
15. **if**  $\beta \leq \alpha$  **then swap**  $(a, b), \text{swap}(\alpha, \beta)$
16. **if** PAR  $((a + b)/2, E_N(x)) = 0$   
*Comment: PAR  $((a + b)/2, E_N(x))$  returns a guess for  $\text{par}_N((ax + bx)/2)$*   
**then**  $b \leftarrow [(a + b)/2]_N$

```

17.           else  $b \leftarrow [(a - b)/2]_N$ 
                {gcd ( $[ax]_N, [bx]_N$ ) = gcd ( $z_1, z_2$ )}
18.           count  $\leftarrow$  count + 1
19.           if count  $> 6n + 3$  then go to line 3
20.       until  $b = 0$ 
;   Step 3—Inverting
21.   if  $E_N(ax) \neq \pm 1$  then go to line 3. ( $E(ax) = E(a) \cdot E(x)$ .)
22.    $x \leftarrow [\pm a^{-1}]_N$ 
23.   return  $x$ .

```

We now consider a single run of Step 2. The assertions in the braces guarantee that if the parity subroutine does not err, then the gcd of the current  $[ax]_N, [bx]_N$  is invariant. It is not hard to verify that the Brent-Kung gcd makes at most  $6n + 3$  calls to the parity subroutine.<sup>1</sup> Therefore, if the original pair  $[ax]_N, [bx]_N$  is relatively prime and the parity subroutine answered correctly on all queries, the algorithm will retrieve  $x$ . By a famous theorem of Dirichlet [19, p. 324], the probability that two random integers in the range  $[-K, K]$  are relatively prime converges to  $6/\pi^2$  as  $K$  tends to  $\infty$ .

We assume so far that the parity subroutine always returns the correct answer. As a matter of fact, the test in line 21 of the code makes sure that the algorithm never errs, even if the answers of the parity subroutine are incorrect. The variable *count* guarantees that even if the parity subroutine occasionally errs, we will not make more than  $6n + 3$  parity calls in a single gcd iteration. The probability that  $x$  will be retrieved in any single gcd attempt is

$$\frac{6}{\pi^2} \cdot \Pr(\text{all answers of PAR in this gcd attempt are correct}).$$

Thus, to invert  $E_N$  in probabilistic polynomial-time it suffices to have a “reliable” parity subroutine. In fact, it suffices to have a parity subroutine which is almost always correct on every argument  $y$  with “small”  $\text{abs}_N(y)$ . This is the case since, if  $\text{abs}_N(z_1)$  and  $\text{abs}_N(z_2)$  are “small,” then (unless the parity errs) all intermediate arguments to the parity subroutine are also “small.” More formally, we use the following definition.

**DEFINITION 7.** Let  $\varepsilon(\cdot), \delta(\cdot)$  be functions from integers into the interval  $(0, \frac{1}{2})$ . Let PAR be a parity subroutine, that on input  $d$  and  $E_N(x)$  outputs a guess for  $\text{par}_N(dx)$ . We say that PAR is  $(\varepsilon(n), \delta(n))$ -reliable if for every  $d, x \in Z_N$  with  $\text{abs}_N(dx) < \varepsilon(n)N/2$ ,

$$\text{Prob}(\text{PAR}(d, E_N(x)) \neq \text{par}_N(dx)) < \delta(n).$$

From the above discussion we derive the following lemma.

**LEMMA 1** (Ben-Or, Chor and Shamir [2]). *The RSA function  $E_N$  is invertible in  $O(\varepsilon^{-2}(n) \cdot n)$  expected number of calls to a  $(\varepsilon(n), 1/(12n + 6))$ -reliable parity subroutine.*

*Proof.* Let PAR be a  $(\varepsilon(n), 1/(12n + 6))$ -reliable parity subroutine. It suffices to give a lower bound on the probability that all PAR calls in a single gcd attempt yield correct answers. We define the following events, as functions of the random variables  $a$  and  $b$  (assuming values in  $Z_N^*$ ): Event  $S_x(a, b)$  holds if both  $\text{abs}_{S_x}(ax)$  and  $\text{abs}_{S_x}(bx)$  are smaller than  $\varepsilon(n)N/2$ . Event  $C_{i,x}(a, b)$  holds if the  $i$ th answer of PAR, on a run

<sup>1</sup> This follows from the fact that  $\alpha + \beta$  decreases by 1 in every execution of line 10, and that throughout the execution of the gcd  $|a| \leq 2^\alpha$  and  $|b| \leq 2^\beta$ . For further details see [7], [8].

of  $\gcd([ax]_N, [bx]_N)$ , is correct. Then

$$\begin{aligned}
 & \text{Prob}((\forall i)A_{i,x}(a, b)) \\
 & \geq \text{Prob}(S_x(a, b)) \cdot \text{Prob}((\forall i)A_{i,x}(a, b) | S_x(a, b)) \\
 & = \varepsilon^2(n) \cdot \left( 1 - \sum_{i=1}^{6n+3} \text{Prob}(\neg A_{i,x}(a, b) | S_x(a, b) (\forall j < i) A_{j,x}(a, b)) \right) \\
 & \geq \varepsilon^2(n) \cdot \left( 1 - (6n+3) \cdot \frac{1}{12n+6} \right) \\
 & = \varepsilon^2(n)/2. \quad \square
 \end{aligned}$$

*Remark.* In Step (2) of the RSA inversion procedure, we used a Brent-Kung gcd, which is faster than the binary gcd originally used in [2].

**3.2. Implementing the parity subroutine.** Given  $d$  and  $E_N(x)$  where  $\text{abs}_N(dx) < \varepsilon(n)N/2$ , the parity subroutine PAR determines (with overwhelming probability)  $\text{par}_N(dx)$ , by querying  $\mathcal{O}_N$ , an oracle for RSA least-significant bit, as follows. It picks a random  $r$  and asks the oracle for the least-significant bit of both  $[rx]_N$  and  $[rx+dx]_N$ , by feeding the oracle in turn with  $E_N(rx) = E_N(r)E_N(x)$  and  $E_N((r+d)x) = E_N(r+d)E_N(x)$ . The oracle's answers are processed according to the following observation. Since  $\text{abs}_N(dx)$  is small, with very high probability ( $\geq 1 - \varepsilon(n)/2$ ) no wraparound<sup>2</sup> occurs when  $[dx]_N$  is added to  $[rx]_N$ . If no wraparound occurs, the parity of  $[dx]_N$  is equal to 0 if the least-significant bits of  $[rx]_N$  and  $[rx+dx]_N$  are identical; and equal to 1 otherwise. This sampling is repeated many times; every repetition (instance) is called a *dx-measurement*.

1. **procedure PAR:**

*Comment:* PAR has access to a least-significant bit oracle  $\mathcal{O}_N$

2. INPUT  $\leftarrow d, E_N(x)$
3.      $\text{count}_0 \leftarrow 0$
4.      $\text{count}_1 \leftarrow 0$
5.     **for**  $i \leftarrow 1$  to  $m$  **do**
6.         pick  $r_i \in Z_N$  at random
7.         **if**  $\mathcal{O}_N(E_N(r_i x)) = \mathcal{O}_N(E_N(r_i x + dx))$
8.             **then**  $\text{count}_0 \leftarrow \text{count}_0 + 1$
9.             **else**  $\text{count}_1 \leftarrow \text{count}_1 + 1$
10.     **od**
11.     **if**  $\text{count}_0 > \text{count}_1$
12.         **then return** 0
13.     **else return** 1

**3.3. Discussion.** Analyzing the error probability of PAR on input  $d, E_N(x)$  reduces to analyzing the error probability of a single *dx-measurement*. Suppose that the success probability of a single *dx-measurement* can be made at least  $\frac{1}{2} + 1/\text{poly}(n)$ . Then by performing sufficiently many independent *dx-measurements*, the majority gives the correct answer for  $\text{par}_N(dx)$  with overwhelming probability.

There are two sources of error in the parity subroutine. One source is wraparound 0 in a *dx-measurement*, the other is oracle errors. Wraparounds are unlikely (i.e., they

<sup>2</sup> If  $[dx]_N = \text{abs}_N(dx)$  then *wraparound* 0 means  $[rx]_N + \text{abs}_N(dx) > N$ . If  $[dx]_N = N - \text{abs}_N(dx)$  then *wraparound* 0 means  $[rx]_N - \text{abs}_N(dx) < 0$ .



occur with probability  $\leq \varepsilon(n)/2$ , since  $\text{abs}_N(dx) \leq \varepsilon(n)N/2$ . Thus, the main source of errors in a  $dx$ -measurement is the errors of the oracle.

If no wraparound occurs, then the  $dx$ -measurement may be wrong only if the oracle errs on either end points ( $[rx]_N, [rx+dx]_N$ ). Thus the error probability of a  $dx$ -measurement is no more than *twice* the error probability of the oracle. However, there are oracles for which the error probability of a  $dx$ -measurement is *twice* the oracle error. Overcoming the *error-doubling* phenomena requires a new parity subroutine, which constitutes the core of our improvement.

**4. The main result.** In this section we prove that RSA least-significant bit is unpredictable. Working in the Ben-Or, Chor and Shamir framework, we modify the parity subroutine. For this modification we introduce two new ideas. The first idea is to avoid the error-doubling phenomena which occurred in the  $dx$ -measurement as follows. Instead of querying the oracle for the least-significant bit of both end points, we query the oracle only for the least-significant bit of one end point. The least-significant bit of the other end point is known beforehand. The second idea is a method for generating many end points with known least-significant bits. These end points are generated in a way that guarantees them “random” enough to be used as a good sample of the oracle.

**4.1. Generating  $m = \text{poly}(n)$  “random” points with known least-significant bits.** We present a method for generating  $m = \text{poly}(n)$  random points, represented as multiples of  $x$ , with the following properties:

- (1) Each point is uniformly distributed in  $Z_N$ ;
- (2) The points are pairwise independent;
- (3) The least-significant bit of each point is known, with probability  $\geq 1 - \varepsilon(n)/4$ .

We generate  $m$  points  $[r_i x]_N$  by picking two random independent elements  $k, l \in Z_N$  with uniform distribution, and computing  $[r_i x]_N = [(k + il)x]_N$  for  $1 \leq i \leq m$  (see Fig. 3). Define the random variables  $y, z \in Z_N$  by  $y = [kx]_N, z = [lx]_N$ . Each of the  $[r_i x]_N$  is uniformly distributed in  $Z_N$ . We now show that (for  $1 \leq i \neq j \leq m$ )  $[r_i x]_N$  and  $[r_j x]_N$  are two independent random variables. For every  $c_1, c_2 \in Z_N$ , the equations  $y + iz \equiv c_1 \pmod{N}$  and  $y + jz \equiv c_2 \pmod{N}$  have a unique solution in terms of  $y, z \in Z_N$ . (This is the case since all of  $N$ 's divisors are larger than  $m$ , and thus  $i - j$  has a multiplicative inverse modulo  $N$ .) Thus, for every  $c_1, c_2 \in Z_N$ ,

$$\Pr([r_i x]_N = c_1 \text{ and } [r_j x]_N = c_2) = \frac{1}{N^2} = \Pr([r_i x]_N = c_1) \cdot \Pr([r_j x]_N = c_2).$$

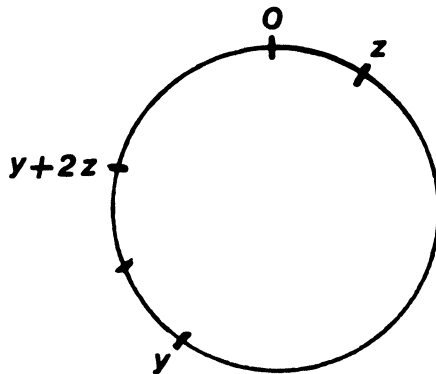


FIG. 3. The points  $y, z$  and  $[r_2 x]_N = y + 2z$ .

The least-significant bits of the  $[r_i x]_N$ 's are computed as follows: We try all possibilities for the least-significant bit of  $y$ , and for its location in one of the intervals  $[j(\varepsilon(n)N/4), (j+1)(\varepsilon(n)N/4)]$ ,  $0 \leq j < 4\varepsilon^{-1}(n)$ . We try all possibilities for the least-significant bit of  $z$ , and for its location in one of the intervals

$$\left[ j \frac{\varepsilon(n)N}{4m}, (j+1) \frac{\varepsilon(n)N}{4m} \right], 0 \leq j < 4m\varepsilon^{-1}(n).$$

There are  $(2 \cdot 4\varepsilon^{-1}(n)) \cdot (2 \cdot 4m\varepsilon^{-1}(n)) = 2^6 m \varepsilon^{-2}(n)$  possibilities altogether, and exactly one of them is correct. We will refer to that possibility as the *right alternative for  $y$  and  $z$* .

Let us now assume that we are dealing with the right alternative for  $y$  and  $z$ . Since the location of  $y$  and  $z$  are known up to  $\varepsilon(n)N/4$  and  $\varepsilon(n)N/4m$ , respectively, the integer  $w_i =_{\text{def}} y + iz$  is known up to  $\varepsilon(n)N/2$  (remember  $1 \leq i \leq m$ ). As  $[w_i]_N$  is uniformly distributed in  $Z_N$ , the probability that the integer  $w_i$  falls in an interval of length  $\varepsilon(n)N/2$  containing an integral multiple of  $N$  is exactly  $\varepsilon(n)/2$ . If  $w_i$  is not in such interval, then the integral quotient of  $w_i/N$  is determined by  $i$  and the approximate locations of  $y$  and  $z$ . This in turn, together with the least-significant bits of  $y$  and  $z$ , determines the least-significant bit of  $[w_i]_N = [r_i x]_N$ . In case the interval  $w_i$  belongs to contains an integral multiple of  $N$ , we make the (arbitrary) assumption that  $w_i$  is at the bigger part of the interval (out of the two parts determined by the integral multiple of  $N$ ).

**4.2. Using the generated  $[r_i x]_N$  in the parity subroutine.** The parity subroutine described in § 3.2 makes use of mutually independent random  $r_i$ 's, and queries the oracle for the least-significant bits of both  $[r_i x]$  and  $[r_i x + dx]_N$ . We modify it by using  $r_i$ 's generated as in § 4.1, and querying the oracle only for the least-significant bit of  $[r_i x + dx]_N$ . In the sequel we will refer to the modified parity subroutine as to PAR\*.

The generation of  $[r_i x]_N$ 's is performed once per each gcd invocation, as part of Step 1 (the randomization step) in the inversion procedure of § 3.1. The choice of  $k$  and  $l$  ( $y = [kx]_N, z = [lx]_N$ ) is independent of the choice of  $a, b$ . We run  $2^8 m \varepsilon^{-2}(n)$  copies of the gcd procedure in parallel, each with one of the possibilities for the approximate locations and least-significant bits of  $y$  and  $z$ . Each copy of the gcd procedure supplies its corresponding possibility for  $y$  and  $z$  as an auxiliary input to all calls of PAR\* that it makes. Note that the run with the right alternative for  $y$  and  $z$  has the least-significant bit of  $[r_i x]_N$  correct, for all  $1 \leq i \leq m$ , with very high probability.

**4.3. Probability analysis of the modified parity subroutine.** In this subsection we analyze the success probability of the modified parity subroutine PAR\*. We show that given an  $\varepsilon(n)$ -oracle for RSA least-significant bit,  $\mathcal{O}_N$ , and setting  $m = O(n \cdot \varepsilon^{-2}(n))$ , makes the parity subroutine PAR\* be  $(\varepsilon(n), 1/(12n+6))$ -reliable. The running time of the subroutine is polynomial in  $n$  and  $\varepsilon^{-1}(n)$ , and so is the expected running time of the entire RSA inversion procedure.

In analyzing the success probability of PAR\* on input  $d, E_N(x)$ , we assume that  $[dx]_N$  is small ( $\text{abs}_N(dx) < \varepsilon(n)N/2$ ) and it is given, as auxiliary input, the right alternative for  $y$  and  $z$ . From this point on, probabilities are taken over all choices of  $y, z$  with uniform probability distribution ( $x$  and  $d$  are considered as fixed).

Recall that on input  $d, E_N(x)$  the parity subroutine conducts  $m$   $dx$ -measurements. Each measurement "supports" either  $\text{par}_N(dx) = 0$  or  $\text{par}_N(dx) = 1$ . The subroutine returns the majority decision. For every  $1 \leq i \leq m$ , the  $i$ th individual  $dx$ -measurement consists of comparing the precomputed least-significant bit of  $[r_i x]_N$  to the answer of

the oracle for the least-significant bit of  $[r_i x + dx]_N$ . Such measurement has three potential sources of error:

- (1) The oracle errs on the least-significant bit of  $[r_i x + dx]_N$ ;
- (2) A wraparound 0 occurs when  $[dx]_N$  is added to  $[r_i x]_N$ ;
- (3) The precomputed least-significant bit of  $[r_i x]_N$  is wrong.

Note that  $[r_i x + dx]_N$  is uniformly distributed in  $Z_N$ . Therefore, a type 1 error has probability  $\frac{1}{2} - \varepsilon(n)$ . Since  $\text{abs}_N(dx) < \varepsilon(n)N/2$ , a type 2 error has probability at most  $\varepsilon(n)/2$ . A type 3 error may occur only if  $\text{abs}_N(r_i x) < \varepsilon(n)N/2$ . A more careful look at the way the least-significant bit of  $[r_i x]_N$  is determined in these “fuzzy” cases show that a type 3 error has probability at most  $\varepsilon(n)/4$ . (This follows from the assignment of correct least-significant bit values to points in the larger part of the interval.)

Thus, the overall error probability in a single  $dx$ -measurement is bounded above by  $\frac{1}{2} - \varepsilon(n)/4$ . Define the random variable

$$\zeta_i = \begin{cases} 1 & \text{if the } i\text{th } dx\text{-measurement is wrong,} \\ 0 & \text{if the } i\text{th } dx\text{-measurement is correct.} \end{cases}$$

Clearly,  $\text{Exp}(\zeta_i) = \text{Pr}(\zeta_i = 1) < \frac{1}{2} - \varepsilon(n)/4$  and  $\text{Var}(\zeta_i) < \frac{1}{4}$ . Since  $\text{Exp}(\zeta_i) < \frac{1}{2} - \varepsilon(n)/4$ , we get

$$\text{Pr}\left(\frac{1}{m} \sum_{i=1}^m \zeta_i \geq \frac{1}{2}\right) \leq \text{Pr}\left(\left|\frac{1}{m} \sum_{i=1}^m \zeta_i - \text{Exp}(\zeta_i)\right| \geq \frac{\varepsilon(n)}{4}\right).$$

Applying Chebyshev’s inequality (see [12, p. 219]) we get

$$\text{Pr}\left(\left|\frac{1}{m} \sum_{i=1}^m \zeta_i - \text{Exp}(\zeta_i)\right| \geq \frac{\varepsilon(n)}{4}\right) \leq \frac{\text{Var}\left(\frac{1}{m} \sum_{i=1}^m \zeta_i\right)}{(\varepsilon(n)/4)^2}.$$

Since (for  $1 \leq i \neq j \leq m$ )  $[r_i x]_N$  and  $[r_j x]_N$  are two independent random variables,  $\zeta_i$  and  $\zeta_j$  are also independent random variables with identical distribution. (Whenever the same function is applied to two independent random variables, the two results are independent random variables.) Let  $\bar{\zeta}_i = \zeta_i - \text{Exp}(\zeta_i)$ . By pairwise independence  $\text{Exp}(\bar{\zeta}_i \cdot \bar{\zeta}_j) = \text{Exp}(\bar{\zeta}_i) \cdot \text{Exp}(\bar{\zeta}_j)$ . Hence,

$$\begin{aligned} \text{Var}\left(\frac{1}{m} \sum_{i=1}^m \zeta_i\right) &= \frac{1}{m^2} \sum_{i=1}^m \sum_{j=1}^m \text{Exp}(\bar{\zeta}_i \cdot \bar{\zeta}_j) \\ &= \frac{1}{m^2} \left( \sum_{i=1}^m \text{Exp}(\bar{\zeta}_i^2) + \sum_{1 \leq i \neq j \leq m} \text{Exp}(\bar{\zeta}_i) \text{Exp}(\bar{\zeta}_j) \right) \\ &= \frac{1}{m^2} \cdot m \cdot \text{Exp}(\bar{\zeta}_1^2) \\ &< \frac{1}{4m}. \end{aligned}$$

Thus,  $\text{Pr}(1/m \sum_{i=1}^m \zeta_i \geq \frac{1}{2}) < 4/m\varepsilon^2(n)$ . The probability that  $1/m \sum_{i=1}^m \zeta_i \geq \frac{1}{2}$  is exactly the error probability of the parity subroutine  $\text{PAR}^*$  on a single input  $d, E_N(x)$ . To summarize, we have proved

**LEMMA 2.** *Let  $d, x \in Z_N$  and suppose that  $\text{abs}_N(dx) < \varepsilon(n)N/2$ . Let  $m \stackrel{\text{def}}{=} 64n \cdot \varepsilon^{-2}(n)$  be the number of measurements done by  $\text{PAR}^*$ . On input  $d, E_N(x)$ , the right alternative for  $y$  and  $z$ , and access to an  $\varepsilon(n)$ -oracle for RSA least-significant bit,  $\mathcal{O}_N$ , the probability that the parity subroutine  $\text{PAR}^*$  outputs  $\text{par}_N(dx)$  is at least  $1 - 1/(12n + 6)$ . The probability space is that of all choices of  $y, z \in Z_N$  and all internal coin tosses of  $\mathcal{O}_N$ . In other words, with the right alternative for  $y$  and  $z$  as an auxiliary input, the parity subroutine  $\text{PAR}^*$  is  $(\varepsilon(n), 1/(12n + 6))$ -reliable.*

**4.4. Main theorem.** Combining Lemmas 1 and 2, we get the following theorem.

**THEOREM 1.** *RSA least-significant bit is unpredictable.*

*Proof.* In Lemma 2, we have analyzed the success probability of PAR\*, assuming that it is given the right alternative for  $y$  and  $z$  as an auxiliary input. When executing the RSA inversion procedure, only one of the copies of PAR\* satisfies this condition, but this is good enough.

We conclude the proof by calculating the overall expected running time of the RSA inversion algorithm, given an  $\epsilon(n)$ -oracle for the least-significant bit. We count elementary  $Z_N$  operations (addition, multiplication, division), RSA encryptions, and oracle calls at unit cost. The expected number of times that Step 1 of the RSA inversion procedure is repeated equals  $O(\epsilon^{-2}(n))$ . For each execution of Step 1,  $O(m\epsilon^{-2}(n))$  copies of the gcd procedure are invoked. Each copy makes  $O(n)$  calls to PAR\*. The parity subroutine, in turn, makes  $O(m)$  operations. Multiplying these terms and substituting  $m = 64n\epsilon^{-2}(n)$ , the overall expected run time is

$$O(\epsilon^{-4}(n) \cdot m^2 n) = O(\epsilon^{-8}(n) \cdot n^3). \quad \square$$

**5. Extensions.** By reductions due to Ben-Or, Chor and Shamir [2], we get

**COROLLARY** (to Theorem 1).

(a) *Let  $I \subset [0, N - 1]$  be an interval of length  $N/2$ . The  $I$ -bit of  $x$  is the characteristic function of  $I$  (i.e., 1 if  $x \in I$  and 0 otherwise). This bit is unpredictable.*

(b) *The  $k$ th least-significant bit is  $(1/4 + (1/2^{n-k}) + (1/2^k) + 1/\text{poly}(n))$ -secure. At least half of these bits are  $(1/6 + (1/2^{n-k}) + (1/2^k) + 1/\text{poly}(n))$ -secure.*

**5.1. Simultaneous security.**

**DEFINITION.** We say that the  $j$  least-significant bits are *simultaneously secure* if inverting  $E_N$  is polynomial-time reducible to distinguishing, given  $E_N(x)$ , between the string of  $j$  least-significant bits of  $x$  and a randomly selected  $j$ -bit string.

We have defined the notion of simultaneous security in terms of an indistinguishability test. It is also possible to define simultaneous security in terms of an unpredictability test: Given  $E_N(x)$  and the  $j - 1$  least-significant bits of  $x$ , the  $j$ th least-significant bit of  $x$  is still  $1/\text{poly}(n)$  secure. Yao [29] has shown that the indistinguishability test is equivalent to the unpredictability test. (Although Yao's proof was given in a different setting, it still applies here.)

Our proof technique easily extends to show that  $\log n$  least-significant bits pass the unpredictability test.

**THEOREM 2.** *Let  $j \stackrel{\text{def}}{=} O(\log n)$ .*

(a) *The  $j$ th least-significant bit in the binary expansion of the plaintext is  $1/\text{poly}(n)$  secure.*

(b) *The  $j$  least-significant bits of the plaintext are simultaneously secure.*

*Proof.* (a) First note that when generating  $y$  and  $z$ , it is feasible to guess not only their 1st least-significant bit, but all  $j$  least-significant bits of  $y$  and  $z$ . The overhead for trying all possibilities is  $2^{2j}$ , which is polynomial in  $n$ . Together with the locations of  $y$  and  $z$ , these bits will determine (with high probability) all  $j$  least-significant bits of each  $[r_i x]_N$ . Also, with probability about  $2^{-2j}$ , the gcd of  $[ax]_N$  and  $[bx]_N$  is  $2^{j-1}$  (instead of 1), which we will assume is the case. This way all  $[dx]_N$ 's in the gcd calculation will have zeros in all  $j - 1$  least-significant bits. Finally, we replace all references to the least-significant bit in the inverting algorithm, by references to the  $j$ th least-significant bit. This can be done since we now have access to an oracle for the  $j$ th least-significant bit. The reader may find it convenient to view this process as taking the gcd of  $[ax/2^{j-1}]_N$  and  $[bx/2^{j-1}]_N$ . (This method of transforming certain inverting algorithms which use an oracle for the first least-significant bit into inverting

algorithms which use an oracle for the  $j$ th least-significant bit originates from Vazirani and Vazirani [27].)

(b) Going through the proof of part (a), notice that when querying the oracle about the  $j$ th least-significant bit of  $[r_i x + dx]_N$  we can give it the  $j-1$  previous bits of  $[r_i x + dx]_N$ . This is the case since, if no wraparound occurs, these  $j-1$  bits are the same as the  $j-1$  least-significant bits of  $[r_i x]_N$ , which we know.  $\square$

*Remark.* Vazirani and Vazirani [28] had previously shown that certain inverting algorithms which use an  $\varepsilon(n)$ -oracle for RSA least-significant bit, can be transformed into inverting algorithms which use an  $\varepsilon(n)$ -oracle for predicting  $x_j$  (given  $x_{j-1}, \dots, x_1$ ). It turns out that the inverting algorithm of § 4 falls into the above category [28]; this yields an alternative (but much harder) way of proving Theorem 2(b).

**5.2. Multi-prime moduli with partial factorization.** The results about bit security for the RSA function were described with respect to composite numbers  $N$  which are the product of two large primes. However, the same proofs hold for the case of multi-prime composite  $N = p_1 p_2 \cdots p_k$ , where the exponent  $e$  is relatively prime to  $\varphi(N)$ . In fact, exactly the same proofs hold also in the case that partial factorization of the modulus is given. Namely, given  $N$ ,  $e$  and some of the  $p_i$ 's, the following tasks are computationally equivalent:

- (1) Given  $E_N(x)$ , find  $x$ ;
- (2) Given  $E_N(x)$ , guess the least-significant bit of  $x$  with success probability  $\frac{1}{2} + 1/\text{poly}(n)$ .

In this context, one may wonder whether RSA remains hard to invert given partial factorization of its modulus. Using the Chinese Remainder Theorem it is not hard to show that inverting  $E_M$  ( $M = p_1 p_2$ ) is equivalent to inverting  $E_N$  ( $N = p_1 p_2 p_3 \cdots p_k$  and both exponents in  $E_M$  and  $E_N$  are the same) when all primes but  $p_1, p_2$  are known. For details see [8].

**6. Bits equivalent to factoring in Rabin's encryption function.** The Rabin encryption function is operating on the message space  $Z_N$ , where  $N = pq$  is the product of two large primes (which are kept secret). The encryption of  $x$  is  $E_N(x) = [x^2]_N$ . The ciphertext space is  $Q_N \stackrel{\text{def}}{=} \{y \mid \exists x: y \equiv x^2 \pmod{N}\}$ . Rabin [22] has shown that extracting square roots ("inverting  $E_N$ ") is polynomially equivalent to factoring.

**6.1. Previous results.** The function  $E_N$  defined above is 4 to 1 rather than being 1 to 1 (as is the case in the RSA). Blum [3] has pointed out the cryptographic importance of the fact that for  $p \equiv q \equiv 3 \pmod{4}$ ,  $E_N$  induces a permutation over  $Q_N$ . Composite numbers of this form will be called *Blum integers*.

Goldwasser, Micali and Tong [17] have presented a predicate the evaluation of which is as hard as factoring. Specifically, they showed that if  $p \equiv 3 \pmod{4}$  and  $p \equiv q \pmod{8}$  then factoring  $N$  is polynomially reducible to guessing their predicate with success probability  $1 - (1/n)$ .

Ben-Or, Chor and Shamir [2] considered the same predicate. Using a modification of their RSA techniques, they showed  $\frac{1}{4} + 1/\text{poly}(n)$  security also for this predicate. Their modification requires that  $N$  be a Blum integer and furthermore that there exist a small odd number  $h$  ( $h = \text{poly}(n)$ ) with  $(h/N) = -1$ . The correctness proof makes use of nonelementary number theory.

**6.2. Our result.** Using the techniques of § 4, we show that the least-significant bit in a variant of Rabin's encryption function is also unpredictable. Our proof uses only elementary number theory, and holds for all Blum integers.

Throughout this section we make use of the *Jacobi symbol*. Let us review the definition and some properties of the Jacobi symbol. Let  $p$  be an odd prime number, and  $h$  an integer relatively prime to  $p$ . The *Legendre symbol*  $(h/p)$  is defined to be 1 if  $h$  is a quadratic residue modulo  $p$ , and  $-1$  otherwise. For  $N = pq$ , a product of two odd primes, and  $h$  relatively prime to  $N$ , the *Jacobi symbol*  $(h/N)$  is defined to be  $(h/p) \cdot (h/q)$ . Even though the definition of the Jacobi symbol uses the factorization of  $N$ , it can be easily computed even if  $N$ 's factorization is not given. Other facts which are used in this section are:  $(h \cdot h'/N) = (h/N) \cdot (h'/N)$ , and for a Blum integer  $N$ ,  $(-1/N) = 1$ . For further details, see [21, Chap. 3].

Let  $N$  be a Blum integer. Define

$$S_N \stackrel{\text{def}}{=} \left\{ x \mid 0 \leq x < \frac{N}{2} \right\},$$

$$M_N \stackrel{\text{def}}{=} \left\{ x \mid 0 \leq x < \frac{N}{2} \wedge \left( \frac{x}{N} \right) = 1 \right\}.$$

Redefine  $E_N$  for  $x \in M_N$  as

$$E_N(x) = \begin{cases} [x^2]_N & \text{if } [x^2]_N < \frac{N}{2}, \\ [N - x^2]_N & \text{otherwise.} \end{cases}$$

This makes  $E_N$  a 1-1 mapping from  $M_N$  onto itself. The intractability result of Rabin still holds. That is, factoring  $N$  is polynomially reducible to inverting  $E_N$ . Let  $L_N(x)$  denote the least-significant bit of  $x$ .

The security of the least-significant bit of this function is now defined in a manner analogous to the RSA case. We would like to use the same techniques to demonstrate that the least-significant bit of the modified Rabin function is unpredictable. The difficulty is that the queries to the oracle may not be of the right form. Namely, we would like to feed the oracle with  $E_N(r_i x + dx)$  and get the least-significant bit of  $[r_i x + dx]_N$ , but it might happen that  $[r_i x + dx]_N \notin M_N$  and then the oracle's answer does not correspond to  $[r_i x + dx]_N$  (but rather to the square root of  $[(r_i x + dx)^2]_N$  which resides in  $M_N$ ). This ( $[r_i x + dx]_N \notin M_N$ ) may happen if either  $[r_i x + dx]_N \notin S_N$  or  $((r_i x + dx)/N) = -1$ . Both cases are easy to detect with very high probability. When any of them occur, we discard this  $dx$ -measurement. We will show that we only discard about  $\frac{3}{4}$  of the  $dx$ -measurements, and the remaining points constitute a large enough sample to retain the high reliability of the parity subroutine. A more elaborate exposition follows.

For technical reasons, we slightly change the definition of "small" here. In this section,  $h$  is small means  $\text{abs}_N(h) < \varepsilon(n)N/8$  (instead of  $\text{abs}_N(h) < \varepsilon(n)N/2$  as in § 4). This will restrict all  $[dx]_N$ 's in the gcd calculation to have  $\text{abs}_N(dx) < \varepsilon(n)N/8$ . Doing this, the probability that a wraparound of either 0 or  $N/2$  occurs when  $[dx]_N$  is added to  $[r_i x]_N$  is no greater than  $\varepsilon(n)/4$ . Similarly, the partition of  $Z_N$  (for both  $y$  and  $z$ ) is refined by a factor of 4.

Given the original encryption  $E_N(x)$ , pick  $y = kx$  and  $z = lx$ , two random multiples of  $x$ . By exhausting all possibilities, the approximate magnitude in  $Z_N$  of  $y$  and  $z$ , and their least-significant bits are known. Let  $[r_i x]_N = [y + iz]_N$  as before. If  $[r_i x]_N$  is not in an  $\varepsilon(n)N/8$  interval around either 0 or  $N/2$ , then we can determine whether  $[r_i x]_N \in S_N$ , and compute the least-significant bit of  $[r_i x]_N$  as before. In the "fuzzy" cases, where  $[r_i x]_N$  is in an  $\varepsilon(n)N/8$  interval around either 0 or  $N/2$ , we determine its least-significant bit assuming that  $[r_i x]_N \in S_N$ .

It remains to determine the parity of  $[dx]_N$  by comparing the known least-significant bit of  $[r_i x]_N$  with the least-significant bit of  $[r_i x + dx]_N$ . We consider the following three cases:

- (1) If  $[r_i x]_N \notin S_N$  (according to  $y, z$  locations), then we ignore this  $dx$ -measurement;
- (2) If  $((r_i + d)/N) = -1$ , then we ignore this  $dx$ -measurement;
- (3) If  $[r_i x]_N \in S_N$  (according to  $y, z$  locations) and  $((r_i + d)/N) = 1$ , then we feed the oracle  $\mathcal{O}_N$  with  $E_N(r_i x + dx)$ , and take its answer as our guess for the least-significant bit of  $[r_i x + dx]_N$ .

In the analysis of this procedure, we assume that we are dealing with the right alternative for  $y$  and  $z$ . With high probability ( $\geq 1 - (\varepsilon(n)/8$ ) we correctly determine whether  $[r_i x]_N \in S_N$ .

We first estimate the probability that we ignore the  $dx$ -measurement. Since the cardinality of  $M_N$  is  $|Z_N^*|/4$ , and our only error is in testing membership in  $S_N$ , we end up in Case (3) with probability  $\geq \frac{1}{4} - \varepsilon(n)/8$ .

We now estimate the error probability given that we are in Case (3). It is easier to estimate the error given that  $[r_i x]_N \in S_N$  and  $((r_i + d)/N) = 1$  (i.e., given that we should have been in Case (3)). These two conditional probabilities are very close, as we will see below. Let  $\mathcal{A}$  denote the event that we produce an incorrect value for the  $i$ th  $dx$ -measurement,  $\mathcal{B}$  denotes the event that we are in Case (3), and  $\mathcal{C}$  denotes the event  $[r_i x]_N \in S_N$  and  $((r_i + d)/N) = 1$ .

$$\begin{aligned} \Pr(\mathcal{A} | \mathcal{B}) &= \Pr(\mathcal{A} \cap \mathcal{C} | \mathcal{B}) + \Pr(\mathcal{A} \cap \bar{\mathcal{C}} | \mathcal{B}) \\ &\leq \Pr(\mathcal{A} | \mathcal{C}) \cdot \frac{\Pr(\mathcal{C})}{\Pr(\mathcal{B})} + \Pr(\bar{\mathcal{C}} | \mathcal{B}). \end{aligned}$$

The reader can easily verify that

$$\begin{aligned} \Pr(\bar{\mathcal{C}} | \mathcal{B}) &< \frac{\varepsilon(n)}{4}, \\ \Pr(\mathcal{C}) &\leq \frac{1}{4} + \frac{\varepsilon(n)}{8}, \\ \Pr(\mathcal{B}) &\geq \frac{1}{4} - \frac{\varepsilon(n)}{8}. \end{aligned}$$

The error we would have made when  $[r_i x]_N \in S_N$  and  $((r_i + d)/N) = 1$  stems from two sources.<sup>3</sup> The oracle's error and the possibility that  $[r_i x + dx]_N \notin S_N$  (although  $[r_i x]_N \in S_N$ ). The first conditional probability is bounded above by  $\frac{1}{2} - \varepsilon(n)$  and the second by  $\varepsilon(n)/8$ . Thus,  $\Pr(\mathcal{A} | \mathcal{C}) < \frac{1}{2} - \varepsilon(n) + \varepsilon(n)/8$ . Using the calculations above, we get

$$\begin{aligned} \Pr(\mathcal{A} | \mathcal{B}) &\leq \left( \frac{1}{2} - \frac{7\varepsilon(n)}{8} \right) \cdot \frac{\frac{1}{4} + \varepsilon(n)/8}{\frac{1}{4} - \varepsilon(n)/8} + \frac{\varepsilon(n)}{4} \\ &< \frac{1}{2} - \frac{\varepsilon(n)}{8}. \end{aligned}$$

Define the random variable

$$\zeta_i = \begin{cases} \frac{1}{2} & \text{if the } i\text{th } dx\text{-measurement is ignored,} \\ 1 & \text{if the } i\text{th } dx\text{-measurement is wrong,} \\ 0 & \text{if the } i\text{th } dx\text{-measurement is correct.} \end{cases}$$

<sup>3</sup> In case  $[r_i x]_N \in S_n$  we have determined correctly the least-significant bit of  $[r_i x]_N \in S_N$ . This follows by the manner in which we determine the least-significant bit of  $[r_i x]_N$ .

The reader can easily verify that  $\text{Exp}(\zeta_i) = \Pr(\zeta_i = 1) < \frac{1}{2} - \varepsilon(n)/64$  and  $\text{Var}(\zeta_i) < \frac{1}{4}$ . The rest of the analysis is similar to the analysis presented in § 4 (as also here the probability that  $(1/m) \sum_{i=1}^m \zeta_i \cong \frac{1}{2}$  is exactly the error probability of the parity subroutine). This implies

**THEOREM 3.** *The least-significant bit for the modified Rabin encryption function is unpredictable. (That is, inverting  $E_N(\cdot)$  is probabilistic polynomial-time reducible to the following: Given  $E_N(x)$  (for  $x \in M_N$ ), guess the least-significant bit of  $x$  with success probability  $\frac{1}{2} + 1/\text{poly}(n)$ .)*

**COROLLARY (to Theorem 3).** *Factoring a Blum integer,  $N$ , is polynomially reducible to guessing  $L_N(x)$  with success probability  $\frac{1}{2} + 1/\text{poly}(n)$  when given  $E_N(x)$ , for  $x \in M_N$ .*

The proofs from the previous section about simultaneous security of the  $\log n$  least significant bits hold here just as well. The extension of the result to multi-prime moduli is possible, but much harder. For details see [10].

**7. Applications.** In this section we present applications of our result to the construction of pseudorandom bit generators and probabilistic encryption schemes.

**7.1. Construction of pseudorandom bit generators.** A pseudorandom bit generator is a device that “expands randomness.” Given a truly random bit string  $s$  (the seed), the generator expands the seed into a longer pseudorandom sequence. The question of “how random” this pseudorandom sequence is depends on the definition of randomness we use. A strong requirement is that the expanded sequence will pass all polynomial time statistical tests. Namely, given a pseudorandom and a truly random sequence of equal length, no probabilistic polynomial time algorithm can tell which is which with success probability greater than  $\frac{1}{2}$  (this definition was proposed by Yao [29], who also showed it is equivalent to another natural definition—unpredictability [6]).

Blum and Micali [6] presented a general scheme for constructing such *strong pseudorandom* generators. Let  $g: M \rightarrow M$  be a 1-1 one-way function, and  $B(x)$  be an unpredictable predicate for  $g$ . Starting with a random  $s \in M$ , the sequence obtained by iterating  $g$  and outputting the bit  $b_i = B(g^i(s))$  for each iteration is strongly pseudorandom. Using their unpredictability result for the “*half<sub>p</sub>* bit” in discrete exponentiation modulo a prime  $p$ , Blum and Micali gave a concrete implementation of the scheme, based on the intractability assumption of computing discrete logarithm. More generally, if  $B_1(x), \dots, B_k(x)$  are simultaneously secure bits for  $g$ , then the sequence obtained by iterating  $g$ , and outputting the string  $\langle b_{i,1} \dots b_{i,k} \rangle = \langle B_1(g^i(s)) \dots B_k(g^i(s)) \rangle$  for each iteration, is strongly pseudorandom. Long and Wigderson [20] have shown that the discrete exponentiation function has  $\log \log p$  simultaneous secure bits.<sup>4</sup> Their result implies a pseudorandom bit generator which produces  $\log \log p$  bits per each iteration of the discrete exponentiation.

Using our results, we get an efficient implementation of strong pseudorandom generators, based on the intractability assumption of factoring. The modified Rabin function  $E_N$  is iteratively applied to the random seed  $s \in M_N$ . In the  $i$ th iteration, the generator outputs the  $\log n$  least-significant bits of  $E_N^i(s) = \pm s^{2^i} \bmod N$ . Thus it outputs  $\log n$  pseudorandom bits at the cost of one squaring and one subtraction modulo  $N$ , and is substantially faster than the discrete exponentiation generator. Previous strong pseudorandom generators based on factoring ([17], [2], [27]) required the use of the exclusive-or construction of Yao [29] and were less efficient.

Another efficient pseudorandom generator was previously constructed by Blum, Blum and Shub [4]. Their generator output one pseudorandom bit per one modular

<sup>4</sup> Kaliski [18] has recently simplified and generalized the argument.



multiplication. Blum, Blum and Shub proved that their generator is a strong pseudorandom generator if the problem of deciding quadratic residuity modulo a composite number is intractable. Vazirani and Vazirani [28] have pointed out that, using our techniques, the Blum, Blum and Shub generator is strong also with respect to the problem of factoring Blum integers.

**7.2. Construction of probabilistic public-key encryption schemes.** A probabilistic encryption scheme is said to leak no partial information if the following holds: *Whatever is efficiently computable about the plaintext given the ciphertext, is also efficiently computable without the ciphertext* [15]. Goldwasser and Micali presented a general scheme for constructing public-key probabilistic encryption schemes which leak no partial information, using a “secure trap-door predicate” [15]. A *secure trap-door predicate* is a predicate that is easy to evaluate given some “trap-door” information, but infeasible to guess with the slightest advantage without the “trap-door” information. Goldwasser and Micali also gave a concrete implementation of their scheme, under the intractability assumption of deciding quadratic residuity modulo a composite number. A drawback of their implementation is that it expands each plaintext bit into a ciphertext block (of length equal to that of the composite modulus).

Using our results, we get an implementation of a probabilistic public-key encryption scheme that leaks no partial information, based on the intractability assumption of factorization. This implementation is more efficient than the one in [14], which is also based on factoring. However, our implementation still suffers from a large bandwidth expansion.

Recently, Blum and Goldwasser [5] used our result to introduce a new implementation of probabilistic encryption, equivalent to factoring, in which the plaintext is only expanded by a *constant factor*. Blum and Goldwasser’s scheme is approximately as efficient as the RSA while provably leaking no partial information, provided that factoring is intractable.

**8. Concluding remarks and open problems.** Standard sampling techniques draw mutually independent elements from a large space. We employed a strategy of getting elements with limited mutual independence (only pairwise independence). This strategy allows more control on properties of the chosen elements.

Trading off statistical independence for control turned out to be fruitful in our context. We believe that such trade-off may be useful in other contexts as well.

We conclude by presenting two open problems:

- (1) In § 5 (6), we have shown simultaneous security results of  $O(\log n)$  bits in RSA (Rabin) encryption function. Extending the result beyond  $O(\log n)$  bits is of major theoretical and practical importance. In particular, if more bits are shown to be simultaneously secure, then the efficiency of the resulting pseudorandom generator will be greatly improved.
- (2) Another interesting question is that of investigating the bit security of the internal RSA bits—are they also  $1/\text{poly}(n)$  secure?

**Acknowledgments.** We would like to thank Michael Ben-Or, Shafi Goldwasser, Silvio Micali, Ron Rivest and Adi Shamir for very helpful discussions and useful ideas.

#### REFERENCES

- [1] W. ALEXI, B. CHOR, O. GOLDREICH AND C. P. SCHNORR, *RSA/Rabin Bits Are  $\frac{1}{2} + 1/\text{poly}(\log N)$  Secure*, Proc. 25th IEEE Symp. on Foundations of Computer Science, 1984, pp. 449–457.

- [2] M. BEN-OR, B. CHOR AND A. SHAMIR, *On the Cryptographic Security of Single RSA Bits*, Proc. 15th ACM Symp. on Theory of Computation, April 1983, pp. 421–430.
- [3] M. BLUM, *Coin Flipping by Telephone*, IEEE Spring COMCON, 1982.
- [4] L. BLUM, M. BLUM AND M. SHUB, *A simple unpredictable pseudo-random number generator*, this Journal, 15 (1986), pp. 364–383.
- [5] M. BLUM AND S. GOLDWASSER, *An efficient probabilistic encryption scheme which hides all partial information*, in Advances in Cryptology: Proceedings of CRYPTO84, G. R. Blakely and D. Chaum, eds., Springer-Verlag, Berlin, New York, 1985, pp. 288–299.
- [6] M. BLUM AND S. MICALI, *How to generate cryptographically strong sequences of pseudo-random bits*, this Journal, 13 (1984), pp. 850–864.
- [7] R. P. BRENT AND H. T. KUNG, *Systolic VLSI arrays for linear time gcd computation*, VLSI 83, IFIP, F. Anceau and E. J. Aas, eds., Elsevier Science Publishers, 1983, pp. 145–154.
- [8] B. CHOR, *Two Issues in Public Key Cryptography*, MIT Press, Cambridge, MA, 1986.
- [9] B. CHOR AND O. GOLDREICH, *RSA/Rabin least significant bits are  $\frac{1}{2} + 1/\text{poly}(\log N)$  secure*, in Advances in Cryptology: Proceedings of CRYPTO84, G. R. Blakely and D. Chaum, eds., Springer-Verlag, Berlin, New York, 1985, pp. 303–313. (Also available as Technical Memo TM-260, Laboratory for Computer Science, Massachusetts Inst. of Technology, May 1984.)
- [10] B. CHOR, O. GOLDREICH AND S. GOLDWASSER, *The bit security of modular squaring given a partial factorization of the modulus*, in Advances in Cryptology—Proceedings of CRYPTO85, H. C. Williams, ed., Springer-Verlag, Berlin, 1986, pp. 448–457.
- [11] W. DIFFIE AND M. E. HELLMAN, *New Directions in Cryptography*, IEEE Trans. Inform. Theory, Vol. IT-22 (1976), pp. 644–654.
- [12] W. FELLER, *An Introduction to Probability Theory and its Applications*, Vol. I, John Wiley, New York, 1962.
- [13] O. GOLDREICH, *On the number of close-and-equal pairs of bits in a string (with implications on the security of RSA's L.s.b.)*, in Advances in Cryptology: Proceedings of EuroCrypt84, T. Beth et al., eds., Springer-Verlag, Berlin, New York, 1985, pp. 127–141. (Also available as Technical Memo TM-256, Laboratory for Computer Science, MIT, March 1984.)
- [14] S. GOLDWASSER, *Probabilistic encryption: Theory and applications*, Ph.D. thesis, Univ. of California, Berkeley, 1984.
- [15] S. GOLDWASSER AND S. MICALI, *Probabilistic encryption*, J. Comp. System Sci., 28 (1984), pp. 270–299.
- [17] S. GOLDWASSER, S. MICALI AND P. TONG, *Why and How to Establish a Private Code on a Public Network*, Proc. 23rd IEEE Symp. on Foundations of Computer Science, November 1982, pp. 134–144.
- [18] B. S. KALISKI, *A pseudo-random bit generator based on elliptic logarithms*, M.Sc. thesis, MIT, Cambridge, MA, 1987.
- [19] D. E. KNUTH, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*, 2nd edition, Addison-Wesley, Reading, MA, 1981.
- [20] D. L. LONG AND A. WIGDERSON, *How discrete is discrete log?*, Proc. 15th ACM Symp. on Theory of Computation, April 1983, pp. 413–420. (A better version is available from the authors.)
- [21] I. NIVEN AND H. S. ZUCKERMAN, *An Introduction to the Theory of Numbers*, John Wiley, New York, 1980.
- [22] M. O. RABIN, *Digital signatures and public key functions as intractable as factorization*, Technical Memo TM-212, Laboratory for Computer Science, Massachusetts Inst. of Technology, 1979.
- [23] R. L. RIVEST, A. SHAMIR AND L. ADLEMAN, *A method for obtaining digital signature and public key cryptosystems*, Comm. ACM, 21 (1978), pp. 120–126.
- [24] C. P. SCHNORR AND W. ALEXI, *RSA bits are  $0.5 + \epsilon$  secure*, in Advances in Cryptology: Proceedings of EuroCrypt84, T. Beth et al., eds., Springer-Verlag, Berlin, New York, 1985, pp. 113–126.
- [25] A. SHAMIR, *On the generation of cryptographically strong pseudo-random number sequences*, ACM Trans. Comput. Systems, 1 (1983), pp. 38–44.
- [27] U. V. VAZIRANI AND V. V. VAZIRANI, *RSA bits are  $.732 + \epsilon$  secure*, in Advances in Cryptology: Proceedings of CRYPTO83, D. Chaum, ed., Plenum Press, New York, 1984, pp. 369–375.
- [28] ———, *Efficient and secure pseudo-random number generation*, Proc. 25th IEEE Symp. on Foundations of Computer Science, 1984, pp. 458–463.
- [29] A. C. YAO, *Theory and applications of trapdoor functions*, Proc. 23rd IEEE Symp. on Foundations of Computer Science, 1982, pp. 80–91.

## PRIVACY AMPLIFICATION BY PUBLIC DISCUSSION\*

CHARLES H. BENNETT†, GILLES BRASSARD‡ AND JEAN-MARC ROBERT§

**Abstract.** In this paper, we investigate how the use of a channel with perfect authenticity but no privacy can be used to repair the defects of a channel with imperfect privacy but no authenticity. More precisely, let us assume that Alice and Bob wish to agree on a secret random bit string, and have at their disposal an imperfect private channel and a perfect public channel. The private channel is imperfect in various ways: transmission errors can occur, and partial information can leak to an eavesdropper, Eve, who also has the power to suppress, inject, and modify transmissions arbitrarily. On the other hand, the public channel transmits information accurately, and these transmissions cannot be modified or suppressed by Eve, but their entire contents becomes known to her. We consider the situation in which a random bit string  $x$  has already been transmitted from Alice to Bob over the private channel, and we describe interactive public channel protocols that allow them, with high probability: (1) to assess the extent to which the private channel transmission has been corrupted by tampering and channel noise; and (2) if this corruption is not too severe, to repair Bob's partial ignorance of the transmitted string and Eve's partial knowledge of it by distilling from the transmitted and received versions of the string another string, in general shorter than  $x$ , upon which Alice and Bob have perfect information, while Eve has nearly no information (or in some cases exactly none), except for its length. These protocols remain secure against unlimited computing power.

**Key words.** cryptography, error-correcting codes, information theory, key exchange, privacy, randomness, universal hashing,  $t$ -resilient functions, wiretap channel

**AMS(MOS) subject classifications.** 94A60, 94A40

**1. Introduction.** Alice and Bob wish to agree on a secret random bit string. In order to achieve this goal, they have at their disposal an imperfect private channel and an authenticated public channel. The private channel is imperfect in various ways: transmission errors can occur, and partial information can leak to Eve, the eavesdropper, who also can modify the transmissions arbitrarily, as explained below. The only limitation we impose on Eve is the knowledge by Alice and Bob of an upper bound on the amount of partial information that can leak to her when she eavesdrops on a private channel transmission.

We allow Eve to tamper arbitrarily with the private channel transmissions. For instance, she can suppress the transmission of selected bits, perhaps to replace them with bits of her choice, to inject new bits, to toggle transmitted bits or to jumble them around. We allow her to introduce as much malicious noise as she wishes. In this paper, we granted Eve unlimited tampering power even though probably no real channel performs quite this badly, so that our results will hold true in any circumstance.

The quantum channel of [BB1], [BB2] is a prime example of an imperfect private channel, and this paper effectively allows the removal of its previous defects. Indeed,

---

\* Received by the editors October 28, 1985; accepted for publication (in revised form) March 9, 1987. Part of this work was presented at CRYPTO 85 under the title, "How to Reduce your Enemy's Information."

† IBM T. J. Watson Research Laboratory, Yorktown Heights, New York 10598.

‡ Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, C.P. 6128, Succ. "A" Montréal, Québec, Canada H3C 3J7. The work of this author was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant A4107 and National Science Foundation grant MCS-8204506. Part of this research was conducted while the author was at the University of California, Berkeley, California 94720.

§ School of Computer Science, McGill University, 805 Sherbrooke St. West, Montréal, Québec, Canada H3A 2K6. This work was partially supported by Natural Sciences and Engineering Research Council of Canada grant A4107. This research was conducted while this author was at the Université de Montréal, Montréal, Québec, Canada H3C 3J7.

the quantum channel allows an eavesdropper to attempt reading a few bits of the transmission with a reasonable probability of not perturbing it, and hence of escaping detection. It also allows several types of blind tampering, such as toggling a selected bit (even if unable to read it) by passing the corresponding faint light pulse through an appropriate sugar solution. More classically, Diffie and Hellman's public-key distribution scheme [DH] can be thought of as an imperfect private "channel." Indeed, it efficiently leaks partial information on the (not-so-random) string exchanged, even if the discrete logarithm is hard to compute, because it is easy for an eavesdropper to determine whether the resulting secret is a quadratic residue or not.

On the other hand, the public channel transmits information accurately (possibly because it is supplemented by a classic error-correcting code [vL], [MS]), and these transmissions cannot be modified or suppressed by Eve, but their entire contents becomes known to her. Newspapers are an example of a secure public channel on which eavesdropping is easy but tampering nearly impossible. Such inherently authentic public channels are commonly suggested for disclosing public keys in a public-key cryptography/digital signature system such as [RSA]. If message authenticity is not thus enforced by the physical properties of the channel, it can be provided by an unconditionally secure authentication scheme such as that of [WC]. In this latter case, a small number of shared secret random bits must be known initially between Alice and Bob, and some of these are used up in the process of authentication; thus the net effect of our protocol in this case can be viewed as key expansion rather than key distribution. Computationally secure authentication [Br], [GGM] can also be used if protection against unlimited computing power is not sought. We shall assume throughout that Alice and Bob did not share initially any secret information, except perhaps for what is needed to implement this public channel authentication feature.

In this paper, we assume that some random bit string has already been transmitted from Alice to Bob over the private channel. We investigate authenticated public channel protocols that, with high probability, detect tampering and transmission errors. Subsequent protocols transform both strings in such a way as to eliminate most, and in some cases all, of Eve's information on the resulting string, except for its length. These public channel protocols remain secure against unlimited computing power, so that the entire exchange is as secure as the initial private channel transmission. It should be noted that excessive tampering on the private channel can result in suppressing communications between Alice and Bob, but it cannot fool them into thinking that they share a secret random string when in fact their strings are different or otherwise compromised.

Let us make our setting mathematically precise. Alice and Bob initially share no secret information. Alice chooses a random string  $x$  of length  $N$  and transmits it over the private channel. Independently, Eve chooses an eavesdropping function  $e: \{0, 1\}^N \rightarrow \{0, 1\}^K$ , where  $K < N$ , and a tampering function  $t: \{0, 1\}^N \times \{0, 1\}^\omega \rightarrow \{0, 1\}^N$ . Alice and Bob know  $K$  but otherwise nothing about  $e$  or  $t$ . When Alice transmits  $x$ , Eve learns the value  $e(x)$  and forwards the potentially corrupted value  $y = t(x, R)$  to Bob, where  $R$  is a random string representing channel noise. Notice that Eve does not herself learn the value of  $t(x, R)$  nor can she influence the random string  $R$  (although she may choose a function  $t$  that does not take  $R$  into consideration). Her information about  $x$  from the private channel transmission consists *only* of knowing  $e$  and  $e(x)$ ; her information about  $y$  consists only of knowing  $e$ ,  $e(x)$ , and knowing the function  $t$  that was applied to  $x$  in order to obtain  $y$ . In this very hostile context, we show that Alice and Bob can *publicly* agree on a protocol that will allow them to ascertain whether  $x = y$  (with an exponentially small error probability) and, if this is so, to end up with

a shorter shared string  $z$  on which Eve has nearly no information or, in some cases, no information at all. If  $x \neq y$ , they can detect this with high probability and, if the differences are not too great, continue with the protocol.

To summarize, we investigate how the use of a channel with perfect authenticity but no privacy can be used to repair the defects of a channel with imperfect privacy but no authenticity. In § 2, we explain why classic error-correcting codes are inappropriate in this context. In § 3, we investigate how transmission errors and tampering can be detected with high probability, and sometimes corrected, at the cost of leaking some information to Eve. In § 4, we investigate how Alice and Bob can subsequently reduce arbitrarily Eve's information at the cost of reducing slightly the length of their shared random string, assuming they have an a priori upper bound on the amount of information that Eve collected on the private channel. In § 5, we investigate the possibility of depriving Eve entirely of any information on the final shared random string at the cost of reducing its length more substantially.

Before we get started, let us give the following definition and some notation: if  $i < j$ , a function  $f: \{0, 1\}^j \rightarrow \{0, 1\}^i$  is *equitable* if  $\#\{x | f(x) = a\} = 2^{j-i}$  for every binary string  $a$  of length  $i$ . If  $x$  and  $y$  are bit strings of equal length,  $x \oplus y$  denotes their bit-by-bit exclusive-or. Finally, if  $x$  is a length  $N$  bit string and if  $0 \leq K \leq N$ ,  $x \bmod 2^K$  denotes the length  $K$  bit string consisting of the rightmost  $K$  bits of  $x$ , and  $x \operatorname{div} 2^K$  denotes the length  $N - K$  bit string obtained from  $x$  by deleting its rightmost  $K$  bits. We shall herein assume that the reader is familiar with the notions of information theory [G], [Mc], [S], universal hashing [CW], [WC], error-correcting codes [vL], [MS], and the theory of finite fields [Be].

**2. The inadequacy of classical error-correcting codes.** Let us recall that the imperfect private channel considered here is susceptible not only to random transmission errors, but also to any amount of controlled and malicious tampering. This tampering capability does not directly give Eve any additional information on  $x$ . It could, however, give her indirect information, because it may force Alice and Bob to subsequently discuss the situation over the public channel, as explained in § 3. The classic theory of error-correcting codes [vL], [MS] is not quite adequate for our purposes because it is based on the assumptions that few errors are more likely to occur than many, and that errors are not maliciously set by an opponent.

For instance, let  $x$  and  $y$  be Alice and Bob's strings, respectively, and let  $N$  be their length. Eve's tampering ability enables her to actually select  $x \oplus y$ , barring actual transmission errors. This is clearly intolerable if error detection is attempted through a linear error-correcting code [MS]. Indeed, let  $x$  be the private channel transmitted code word corresponding to Alice's chosen random string. Let  $z$  be any code word chosen by Eve. If she perturbs the private channel transmission so that Bob receives  $y = x \oplus z$ , it will not be possible for him to detect tampering. Notice that Eve can achieve this without gaining any knowledge of the contents of the original transmission  $x$ .

Nonlinear error-correcting codes are not susceptible to the above threat, but they fall to an even simpler one because Eve is also capable of replacing Alice's bits by bits of her choice. If Alice sends some code word over the private channel, it suffices for Eve to suppress the original communications entirely, and inject any other code word of her choice instead. This simple-minded attack can be hindered by the post facto application of an error-correcting code, as discussed in § 3.2.

Classic error detection is therefore impossible in our context if Eve knows in advance of the private channel transmission which code is to be used. On the other

hand, we assumed that Alice and Bob did not initially share any secret, except perhaps to implement the authenticated public channel. A solution is that Alice randomly chooses an error-correcting code, produces the code word  $x$  corresponding to her chosen random string, and sends  $x$  to Bob through the private channel. She then waits for Bob to use the authenticated public channel in order to acknowledge receipt of the private transmission. Only at this point does Alice use the public channel to send Bob a description of the error-correcting code. This allows Bob to recover the original string and to check for errors and tampering. This randomization approach allows Alice to reveal sensitive information to Bob, hence to Eve, only after it is already too late for her to efficiently alter the private channel transmission in an undetectable way.

Although such use of randomness is our main tool in this paper, it remains true that classic linear error-correcting codes are not appropriate because, even randomly chosen, they still assume few errors to be more likely than many. For instance, let us assume that Alice uses a Hamming code of dimension  $[N, K]$  and that the random part of the protocol is the order in which she sends the code-word bits. It is no longer possible for Eve to toggle selected bits and to be certain to escape detection because she does not know which bits to toggle. However, there are exactly  $\binom{N}{2}/3$  code words of Hamming weight 3 [MS], whereas there are  $\binom{N}{3}$  length  $N$  bit strings of weight 3. Therefore, Eve can toggle 3 random bits and escape detection with probability

$$\left( \binom{N}{2} / 3 \right) / \binom{N}{3} = (N-2)^{-1}.$$

Using such a protocol, Alice and Bob could only achieve a very high probability of not being fooled, say  $1 - 2^{-50}$ , at the cost of exchanging unreasonably long strings.

It is instructive to compare our setting with the problem solved by the wiretap channel of Wyner [W] which achieves similar results in a more classically information-theoretic setting. In Wyner's setting, Alice encodes information by a channel code of her choice. The output of her encoder is fed into two classic (discrete, memoryless) communication channels: the *main channel*, leading to the intended receiver Bob, and the *wiretap channel*, of lesser capacity than the main channel, leading to the eavesdropper. All participants know the channel code and the statistical properties of the two channels. Under these conditions, Wyner showed that by appropriate choice of the channel code, Alice can exploit the difference in capacity between the two channels communicate reliably with Bob while maintaining almost perfect secrecy from the eavesdropper.

In our setting, the users have an additional resource: the authenticated public channel. This allows them to cope with a more powerful eavesdropper. Our eavesdropper is more powerful in two ways, either of which would be fatal in Wyner's setting: she can tamper with Alice's communications as well as listen to them, and she eavesdrops by evaluating an arbitrary  $N$ -bit to  $K$ -bit function of her choice, unknown to Alice and Bob.

In § 3.1, we describe error-detection schemes such that the probability of undetected tampering and transmission errors is independent of the number and position of altered bits. Moreover, this probability can be exponentially small in the length of the strings transmitted. Although never fully appropriate for the detection of tampering, classic error-correcting codes remain interesting in order to correct actual transmission errors over the private channel, as we investigate in § 3.2.

**3. Detection and correction of transmission errors and tampering.** Let  $x$  be some random bit string selected by Alice. Assume she transmits it directly through the imperfect private channel, and let  $y$  be the string thus received by Bob. Let  $N$  be the

length of both strings. The public channel protocols described in § 3.1 allow Alice and Bob to detect whenever  $x \neq y$  with an arbitrarily small error probability, independently of how  $y$  differs for  $x$ . Should  $y$  be found to differ from  $x$ , the protocol of § 3.2 can be used to reconcile them with high probability. The reconciliation protocol can also be used *preventively*, before using an error-detection protocol from § 3.1, if  $y$  is expected to be different from  $x$  merely due to normal transmission errors. The fact that these protocols leak information to Eve about  $x$  is considered in § 4.

**3.1. Error detection.** A very simple but impractical way of testing whether  $x = y$  is for Alice to choose a random function  $f: \{0, 1\}^N \rightarrow \{0, 1\}^K$ , where  $K$  is a security parameter. After the private channel transmission is completed, she sends  $f(x)$  to Bob over the public channel, together with a complete description of the function  $f$ . Should Bob find out that  $f(y) = f(x)$ , this would be considered as strong evidence that  $y = x$ , the error probability being  $2^{-K}$ , independently of the length of the strings and how they might differ. On the other hand, should  $f(y)$  be different from  $f(x)$ , Bob could report to Alice with certainty that he did not receive the correct string. Notice that the amount of information on  $x$  leaking to Eve from this protocol depends only on the security parameter  $K$ , and not on the length  $N$  of the strings (except of course for the fact that  $K < N$ ). This would not be the case if a classic error-detecting code has been used. Unfortunately, this scheme cannot be used in practice because there are  $2^{K2^N}$  different such functions, and therefore as many as  $K2^N$  bits are typically needed to merely transmit a description of the randomly chosen function.

In some cases, it may be preferable for Alice to choose randomly the function  $f$  among the set of *equitable* functions only. This can be done in theory (although not in practice when  $N$  is large) by randomly selecting a permutation  $\pi: \{0, 1\}^N \rightarrow \{0, 1\}^N$  and defining  $f(x) = \pi(x) \bmod 2^K$  for each string  $x$  of length  $N$ . Notice that this allows a (very slight) reduction of the probability of undetected transmission errors or tampering from  $2^{-K}$  to  $(2^{N-K} - 1)/(2^N - 1)$ .

Universal hashing [CW] provides an efficient way to achieve the same goal. After the private channel transmission is completed, Alice randomly chooses a function  $f: \{0, 1\}^N \rightarrow \{0, 1\}^K$  among some universal<sub>2</sub> class of functions. She then sends both  $f(x)$  and a description of  $f$  to Bob. Thanks to universal hashing, the description of  $f$  can be transmitted efficiently this time. After computing  $f(y)$ , Bob checks whether it agrees with  $f(x)$ . If it does, a basic property of universal hashing allows them to assume that  $x = y$ , their probability of error being bounded again by  $2^{-K}$ .

We refer the reader to [CW], [WC] for definition and discussion of universal hashing. Several universal<sub>2</sub> and strongly universal<sub>2</sub> classes are described there. Let us only stress here that they are entirely reasonable in practice. For instance, it suffices to send about  $2N$  bits over the public channel to describe a function  $f: \{0, 1\}^N \rightarrow \{0, 1\}^K$  randomly selected among  $H_1$  [CW] or  $P$  (§ 4), once the private channel transmission is completed. Compare this with the unreasonable  $K2^N$  bits needed to describe a random function! Moreover, the computation of  $f(x)$  is very efficient. Therefore, universal<sub>2</sub> hashing provides all the advantages of truly random functions, but none of the inconveniences.

As we mentioned in § 2, it is crucial that the actual verification function be transmitted to Bob only after Alice has received confirmation through the public channel that the private channel transmission is completed. This deprives Eve of any strategy that would reduce her chances of being detected. For instance, if she knew in advance that the function  $f(x)$  simply returns the last  $K$  bits of  $x$ , she could arbitrarily tamper with the other  $N - K$  bits without fear of detection.

It is interesting to compare our use of universal hashing with the classic use of this technique for message authentication [WC]. The use of hashing for authentication depends on randomly choosing a hash function and keeping it secret at least until after the message to be authenticated has been received. In our protocol for error detection, the hash function cannot be kept secret, but it must be chosen randomly *after* the message has been transmitted.

An alternative to universal hashing comes to mind: polyrandom collections [GGM]. However, universal hashing is more appropriate in this context because it offers security against unlimited computing power. It does not rely on unproved assumptions, and it can be computed more efficiently.

**3.2. Reconciliation of the strings.** Whether  $f: \{0, 1\}^N \rightarrow \{0, 1\}^K$  is chosen as a purely random function or within some universal<sub>2</sub> class of functions, what should Alice and Bob do if they find that  $f(x)$  differs from  $f(y)$ ? Whether anything can be done to recover from this situation depends on how much  $x$  differs from  $y$ . If the Hamming distance between  $x$  and  $y$  is relatively small, due to limited channel noise and/or limited tampering, we show here how Alice and Bob can find and correct the errors in  $y$ . Using subsequent protocols from § 4, Alice and Bob can then distill from  $x$  and  $y$  a shorter string on which Eve has nearly (and in some cases exactly) no information. On the other hand, if the distance between  $x$  and  $y$  is large, Alice and Bob will be able to discover this fact, but they will have no recourse but to discard  $x$  and  $y$  and begin the protocol all over again by transmitting a new random string through the private channel. As mentioned in the introduction, Eve can effectively prevent Alice and Bob from agreeing on a secret bit string by interfering strongly with every private channel transmission, but she cannot (except with low probability) fool them into thinking they have succeeded when in fact they have not.

Reconciliation is particularly easy if it is suspected that only one or two errors have occurred. Then Bob can try computing  $f(z)$  on all strings differing from  $y$  by only one or two bits, in the hope of finding a match for  $f(x)$  and thus a likely candidate for  $x$ . We call this approach *bit twiddling*.

In the presence of more than a very few errors, bit twiddling becomes too time-consuming, but it is still practical to find and remove the errors by a post facto application of error-correcting codes, as described below. If many errors are expected even when no tampering occurs, the error detection protocol of § 3.1 should be deferred until after most or all of the errors have thus been found and corrected.

Many traditional error-correcting codes, such as Hamming codes, can be written in a *systematic* format, in which each code word consists of the original source word followed by a string of check bits. Given a source word  $x$ , the encoder thus generates the concatenation  $xC(x)$ , where  $C(x)$  is some check string depending on  $x$ , and sends this longer string into the channel. At the receiving end, the redundancy in the code is used (if one is lucky) to recover the original  $x$  despite the potential corruption of both  $x$  and  $C(x)$  by channel noise. Systematic codes have no special advantage over unsystematic ones (in which source and check information are mixed) for most ordinary error-correcting applications, but they are useful in the present setting because they allow the calculation of the check information  $C(x)$  to be performed post facto, after the uncoded source data  $x$  has been sent through the private channel. They also allow sending  $x$  and  $C(x)$  on different channels.

In § 2, it was pointed out that error-correcting codes used in the traditional non-post facto manner cannot defend against malicious tampering, because Eve, knowing the code, can escape detection by deliberately mutating one channel code word into



another. For example, in the case of a systematic code, if  $xC(x)$  was sent by Alice on the channel, Eve could substitute  $zC(z)$ , even without learning anything about  $x$ , and Bob would be none the wiser. This threat can be avoided by applying the code in a post facto manner, and by taking two further precautions:

(i) The check string  $C(x)$  generated by the code is not transmitted through the private channel, where Eve could alter it, but through the public channel, where she can only listen to it.

(ii) Before application of the error-correcting code, the strings  $x$  and  $y$  to be reconciled are subjected to a preliminary *uniformization transformation*, consisting of random permutation and complementation. The purpose of this transformation is to make the difference between  $x$  and  $y$  behave as if  $y$  were the result of sending  $x$  through a memoryless binary symmetric channel. To achieve this preliminary goal, Alice randomly chooses a permutation  $\sigma: \{1, 2, \dots, N\} \rightarrow \{1, 2, \dots, N\}$  and a length  $N$  bit string  $w$ . She then transmits both  $w$  and a description of  $\sigma$  to Bob over the public channel. Finally, Alice and Bob transform their strings  $x$  and  $y$ , respectively, by first shuffling the bits according to  $\sigma$ , then taking the exclusive-or of the permuted string with  $w$ . It is clear that this leaves the number of errors unchanged, but redistributes them to random places not under the control of Eve, who is thus prevented from exploiting knowledge of the error-correcting code  $C$  to introduce a pattern of errors against which the code would perform less well than average. It is also clear that the uniformization transformation releases no new information to Eve about  $x$  and  $y$ .

To summarize, in order to use a systematic error-correcting code  $C$  for reconciliation, Alice and Bob first uniformize their strings  $x$  and  $y$ , thus producing  $x_0$  and  $y_0$ . Alice then applies  $C$  to  $x_0$  and transmits the result  $C(x_0)$  to Bob over the public channel, thereby giving away at most  $|C(x_0)|$  bits of information to Eve about  $x_0$ . Bob uses  $C(x_0)$  to correct the errors in  $y_0$ , thereby recovering  $x_0$  if the code  $C$  is sufficient to correct the errors that have occurred between  $x_0$  and  $y_0$ . If  $C(x_0)$  is significantly shorter than  $x_0$ , there is still some information about  $x_0$  that Eve does not know, and the methods of § 4 can be used to nearly obliterate the information known to her, by allowing Alice and Bob to derive from  $x_0$  a shorter string, of length approximately  $|x_0| - |C(x_0)|$ , about which Eve has less than one bit of information.

It remains to be decided what kind of systematic error-correcting code to use. If  $\varepsilon$ , the density of errors, is not too great to begin with, a simple block code (such as a Hamming code if we expect at most two errors per block) would suffice to reduce the number of errors to the point where they can be found and corrected by bit twiddling. If  $\varepsilon$  is greater (say 0.01 or more), a systematic convolutional code [G] would be better, since these tree codes, unlike block codes, can achieve exponential error reduction without exponential decoding effort, while still transmitting data at a rate at least half the theoretical channel capacity. As emphasized above, whatever error-correcting code is used should be used in a post facto manner, being applied to the uniformized version of Alice's data, and the resulting check information  $C(x_0)$  being sent to Bob over the noiseless, tamper-proof public channel.

In the traditional non-post facto situation, where both  $x$  and  $C(x)$  are sent through a binary symmetric channel with error probability  $\varepsilon$ , the capacity  $|x|/(|x| + |C(x)|)$  achievable by convolutional codes with polynomial expected decoding effort is denoted  $R_{\text{comp}}$ , and has the value  $1 - \log^1(1 + 2\sqrt{\varepsilon(1-\varepsilon)})$ , which ranges between 0.5 and 1.0 times the theoretical capacity  $1 - \mathbf{H}(\varepsilon)$ , where  $\mathbf{H}(\varepsilon) = \varepsilon \log 1/\varepsilon + (1-\varepsilon) \log 1/(1-\varepsilon)$  is the *entropy function* [G]. In the present situation, the original data is transmitted

<sup>1</sup> Unless otherwise stated, all logarithms in this paper are to the base 2.

with error probability  $\epsilon$ , but the check information is transmitted noiselessly; and capacity should thus be defined differently, as the maximum of  $(|x_0| - |C(x_0)|) / |x_0|$ , taken over all possible codes  $C$ , since this is the fraction of the original information in  $x_0$  that remains secret after reconciliation. Nevertheless, in the post facto situation, Shannon showed that the theoretical capacity is still given by  $1 - H(\epsilon)$ . By arguments parallel to the derivation of  $R_{\text{comp}}$  [G], it can be shown that the effective capacity achievable by convolutional coding is still given by

$$R_{\text{comp}} = 1 - \log(1 + 2\sqrt{\epsilon(1 - \epsilon)}).$$

In practice, the parameter  $\epsilon$  would not need to be known beforehand, since it can be determined interactively, by having Alice first compute a generous amount of check information, but then release only as much of this as Bob finds he needs for efficient decoding. A table comparing  $R_{\text{comp}}$  with Shannon's theoretical capacity  $c$  for some values of  $\epsilon$  follows.

$\epsilon$	$R_{\text{comp}}$	$c$
0.001	0.9116	0.9886
0.01	0.7382	0.9192
0.03	0.5765	0.8056
0.05	0.4781	0.7136
0.10	0.3219	0.5310
0.25	0.1000	0.1887
0.40	0.0146	0.0290

**4. Reduction of the eavesdropper's information.** Assuming that Alice and Bob agree on their strings as a result of one of the protocols discussed above, Eve has two different sources of information on that string: partial eavesdropping on the private channel, as the original random bit string was being transmitted, and complete eavesdropping on the public channel, as the agreement protocol was being carried out.

We now investigate how to reduce Eve's information arbitrarily close to zero at the cost of slightly shrinking the random bit string shared between Alice and Bob. In § 4.1, we assume that no eavesdropping on the private channel has occurred, but that transmission errors were possible. We also assume that the number of errors, if any, is small enough to be handled by bit twiddling (this assumption is removed in § 4.3). In § 4.2, we assume, on the contrary, that the eavesdropper has acquired partial information on the private channel transmission, but that tampering and transmission errors have not occurred. We finally consider in § 4.3 the more realistic case where both eavesdropping and arbitrary tampering on the private channel are possible, so the eavesdropper may gain information both directly from eavesdropping on the private transmission and indirectly by listening to the public channel reconciliation and error-detection protocols of § 3. All these protocols are secure against an eavesdropper with unlimited computing power.

**4.1. Eliminating the public channel eavesdropper's information.** Let us assume for the moment that Eve is unable to eavesdrop on the private channel but that transmission errors may have occurred. Assuming the number of errors is small enough to be handled by bit twiddling, we now show that Alice and Bob can agree on a secret random string on which Eve has *no* information, except for its length. In a companion paper [BBR], we investigate how to handle efficiently an error rate that would make bit twiddling ineffective. The interactive public discussion protocol discussed there allows Alice and

Bob to agree with high probability on a (shorter) secret random string on which Eve still has no information. Alternatively, § 4.3 of the present paper uses the post facto systematic convolutional codes of § 3.2 in order to handle normal transmission errors, but at the cost of leaking to Eve an arbitrarily small fraction of one bit of information about the final random bit string.

Let  $x$  be the random string sent from Alice to Bob over the imperfect private channel. Let  $y$  be the string received by Bob. An error detection protocol of § 3.1 is first applied to make sure, with high probability, that the random strings of Alice and Bob are identical. If not, bit twiddling is attempted by Bob to transform  $y$  into  $x$ . If bit twiddling fails, either repeat the whole process (Alice sends a fresh new random string to Bob, etc.), use the interactive protocol of [BBR], or refer to § 4.3. At this point, assume that Alice and Bob agree with high probability on some length  $N$  string  $x$ , but that Eve has gained information on this string by listening to the public channel error-detection protocol. We wish to eradicate this information of hers.

Let  $f: \{0, 1\}^N \rightarrow \{0, 1\}^K$  be the function used in the error-detection protocol. Eve knows the  $K$  bit value of  $f(x)$ , together with the function  $f$  itself. Although this may not give her any physical bits of  $x$ , she has  $K$  bits of information on  $x$  in the sense of Shannon's information theory [S], assuming that  $f$  is equitable. If  $f$  is not equitable, Eve's expected information is less than  $K$  bits, although she could know more occasionally. Her information can be characterized by the set  $C = \{z \in \{0, 1\}^N \mid f(z) = f(x)\}$  of possible candidates for  $x$ . From Eve's point of view, each element of  $C$  is equally likely to be the string  $x$  currently shared between Alice and Bob. Notice that Alice and Bob also have complete knowledge on the set  $C$ .

In order to eliminate Eve's information, Alice and Bob publicly agree on a function  $g: \{0, 1\}^N \rightarrow \{0, 1\}^R$ , for some integer  $R \leq N - K$ , such that knowledge of the set  $C$  yields no information on  $g(x)$ , the final string on which Alice and Bob agree. In other words, the purpose of this function  $g$  is to shrink the string  $x$  by at least  $K$  bits, in order to compensate for the  $K$  bits of information that Eve knows on  $x$ . It is clear that at least  $K$  bits of  $x$  must be sacrificed to privacy, but are  $K$  bits enough in general? The proper choices of  $R$  and of this information-reduction function  $g$  depend on which error-detection function  $f$  was chosen from § 3.1.

**4.1.1. Eliminating the information given by a truly random error-detection function.** Assume the error-detection function  $f$  was chosen randomly from among all functions from  $\{0, 1\}^N$  to  $\{0, 1\}^K$ . This error-detection protocol is of no practical interest, because it would require  $K2^N$  bits to merely transmit function  $f$ . It is nonetheless instructive to figure out how the information given on  $x$  by  $f(x)$  can be eliminated in this context. Indeed, this provides a nice intuitive insight into the realm of information reduction. Moreover, it is interesting to compare what can be achieved in the truly random case with the practical world (§ 4.1.2). By analogy, recall Shannon's result that any channel can come arbitrarily close to achieving its theoretical transmission capacity through the use of random codes that cannot be implemented in practice [S], and that no practical codes known thus far can perform nearly as well. As we will see, such is not the case here. For this reason, we only state the main results pertaining to truly random functions and we refer the reader to [R] for the somewhat tedious proofs.

Recall that Eve's information about  $x$  is characterized by the set  $C$  of possible candidates, and that this set is also known for Alice and Bob. Let  $\#C$  denote the number of elements in  $C$ . Let  $y$  be the index of  $x$  in  $C$  when  $C$  is ordered in lexicographic order. Then  $y$  is available to both Alice and Bob, whereas it is just as likely to take any value between 1 and  $\#C$  as far as Eve is concerned. If  $\#C$  is large enough, such

a  $y$  can thus be used as the resulting shared secret. The ideal situation occurs when the function  $f$  is equitable. In this case, we always have  $\#C = 2^{N-K}$  and thus  $y$  is a uniformly distributed random bit string of length  $N - K$  on which Eve has no information whatsoever save its length. This idea is captured in the following theorem:

**THEOREM 1.** *Let  $N$  be the length of the originally transmitted bit string and let  $K < N$  be the safety parameter used for error detection. Let  $\pi : \{0, 1\}^N \rightarrow \{0, 1\}^N$  be a randomly chosen permutation. Let  $f : \{0, 1\}^N \rightarrow \{0, 1\}^K$  and  $g : \{0, 1\}^N \rightarrow \{0, 1\}^{N-K}$  be equitable functions defined by  $f(x) = \pi(x) \bmod 2^K$  and  $g(x) = \pi(x) \operatorname{div} 2^K$ . Then, knowledge of  $\pi$  (hence of  $f$  and  $g$ ) and  $f(x)$  yields no information on  $g(x)$ , except that it is of length  $N - K$ .*

From an information theoretic point of view, one might wonder if it is necessary that this information-reduction function  $g$  be custom made for the particular error-detection function  $f$  being used. At least two ideas come to mind: What happens if  $g$  itself is chosen randomly, independently of  $f$ ? and Could  $g$  be known to Eve even before the private channel transmission takes place? Although these ideas do not allow wiping out Eve’s information with certainty, they come close.

Let us first consider the case when the information-reduction function  $g$  is randomly chosen among all functions  $\{0, 1\}^N \rightarrow \{0, 1\}^R$ , where  $R \leq N - K$  is the desired length of the final string. The intuitive hope is that a random  $g$  is very likely to map the elements of  $C$  nearly uniformly onto  $\{0, 1\}^R$ , thus releasing very little information on the value of  $g(x)$  from knowledge of  $C$  and  $g$  alone.

In order to state the theorems precisely, we need to introduce some information-theoretic formalism that will be used throughout § 4. Let  $N$  and  $K$  be as in Theorem 1. Let  $S$  be any nonnegative integer smaller than  $N - K$ . Let  $R = N - K - S$ . Let  $\mathbf{X}$ ,  $\mathbf{F}$  and  $\mathbf{G}$  be three independent uniformly distributed random variables ranging over  $\{0, 1\}^N$ ,  $\{f|f : \{0, 1\}^N \rightarrow \{0, 1\}^K\}$  and  $\{g|g : \{0, 1\}^N \rightarrow \{0, 1\}^R\}$ , respectively. Define new dependent random variables  $\mathbf{Y}$  and  $\mathbf{Z}$  ranging over  $\{0, 1\}^K$  and  $\{0, 1\}^R$ , respectively, by setting  $y = f(x)$  and  $z = g(x)$ .

The *expected amount of (Shannon) information* given on  $g(x)$  by  $y = f(x)$ ,  $f$  and  $g$  is defined by following formula:

$$\mathbf{I}(\mathbf{Z}; \mathbf{YFG}) = \sum_z \sum_y \sum_f \sum_g \operatorname{prob} [z, y, f, g] \log \frac{\operatorname{prob} [z|yfg]}{\operatorname{prob} [z]}.$$

Notice that “expected” means “averaged over all possible choices for  $f$ ,  $g$  and  $x$ .”

**THEOREM 2.** *Let  $N$ ,  $K$ ,  $S$ ,  $R$ ,  $\mathbf{F}$ ,  $\mathbf{G}$ ,  $\mathbf{Y}$  and  $\mathbf{Z}$  be as above, then*

$$\mathbf{I}(\mathbf{Z}; \mathbf{YFG}) \leq \log (1 + 2^{-S}) < 2^{-S} / \ln 2.$$

*Furthermore, this bound is fairly tight because*

$$\mathbf{I}(\mathbf{Z}; \mathbf{YFG}) \cong \left( \frac{1}{2} - \frac{1}{2^r} \right) \frac{2^{-S}}{\ln 2} - \frac{2}{2^N - 1}$$

*whenever  $3 \leq S \leq N - K - 2$ .*

Intuitively, this says that if  $f$  and  $g$  are randomly chosen functions from  $\{0, 1\}^N$  to  $\{0, 1\}^K$  and to  $\{0, 1\}^R$ , respectively, and if  $x$  is randomly chosen among all bit strings of length  $N$ , then the expected amount of Shannon information given on  $g(x)$  by  $f(x)$ ,  $f$  and  $g$  is less than  $2^{-S} / \ln 2$  bits.

As for the second idea, it turns out that a comparable level of information reduction can be achieved through the use of any ad hoc equitable function. In particular, it is enough to simply chop off any  $K + S$  physical bits of  $x$  in order to reduce the expected eavesdropper’s information below  $2^{-S} / \ln 2$  bits. This remains true even if the reduction function is chosen a priori and known to the eavesdropper before the private channel transmission.

**THEOREM 3.** *Let  $N$ ,  $R$  and  $S$  be as above. Let  $g: \{0, 1\}^N \rightarrow \{0, 1\}^R$  be any fixed equitable function, then the expected amount of information given on  $g(x)$  by  $f(x)$ ,  $f$  and  $g$  is less than  $2^{-S}/\ln 2$  bits.*

**4.1.2. Eliminating the information given by universal hashing error detection.** Let us now assume that a practical error-detection protocol was used from those proposed in § 3.1: the function  $f: \{0, 1\}^N \rightarrow \{0, 1\}^K$  was randomly chosen among some universal<sub>2</sub> class of hash functions. Rather than developing a general theory of information elimination in this context, let us design an ad hoc technique for a specific universal<sub>2</sub> class mentioned in [WC], which we call  $P$ . We assume here that the reader is familiar with Galois field theory [Be].

Consider  $a, b \in \text{GF}(2^N)$  such that  $a \neq 0$ . The degree one polynomial  $q_{a,b}(x) = ax + b$ , arithmetic being done in  $\text{GF}(2^N)$ , defines a permutation of  $\text{GF}(2^N)$ . If we let  $\sigma: \text{GF}(2^N) \rightarrow \{0, 1\}^N$  stand for the natural one-to-one correspondence, this induces a permutation  $\pi_{a,b}(x): \{0, 1\}^N \rightarrow \{0, 1\}^N$  defined by  $\pi_{a,b}(x) = \sigma(q_{a,b}(\sigma^{-1}(x)))$ . Therefore, for any fixed  $K \leq N$ , the function  $h_{a,b}(x): \{0, 1\}^N \rightarrow \{0, 1\}^N$  defined by  $h_{a,b}(x) = \pi_{a,b}(x) \bmod 2^K$  is equitable. Define the class  $P = \{h_{a,b} \mid a, b \in \text{GF}(2^N), a \neq 0\}$ . It is elementary to prove that  $P$  forms a universal<sub>2</sub> class of hash functions, so that it can be used for the error-detection protocol of § 3.1. (In fact, the class becomes *strongly* universal<sub>2</sub> if we allow  $a = 0$ , but this is to be avoided here because  $h_{0,b}$  is not equitable.)

**THEOREM 4.** *Let  $a$  and  $b$  be any elements of  $\text{GF}(2^N)$  such that  $a \neq 0$ . Let  $x$  be a random string of length  $N$ . Then knowledge of  $a$ ,  $b$  and  $h_{a,b}(x)$  gives no information on the string defined as  $g_{a,b}(x) = \pi_{a,b}(x) \bmod 2^K$ , except that it is of length  $N - K$ .*

*Proof.* This is an immediate consequence of the fact that  $\pi_{a,b}$  is a permutation of  $\{0, 1\}^N$ : knowledge of the last  $K$  bits of  $\pi_{a,b}(x)$  gives no information on its first  $N - K$  bits.  $\square$

In conclusion, use of the universal<sub>2</sub> class  $P$  allows Alice and Bob to verify whether their strings are identical, with an error probability of at most  $2^{-K}$ . If they turn out to be the same (or if they differ little enough that bit twiddling is applicable), they can be transformed into new strings that are only  $K$  bits shorter, on which Eve has no information at all. This is clearly optimal.

It is worth mentioning that the conceptually simpler universal<sub>2</sub> class  $H_1$  of [CW] can be used instead of  $P$  for error detection. This still allows subsequently a good, efficient information-reduction scheme, but wiping out with certainty Eve's information does not seem to be feasible. This is due to the fact that the functions in  $H_1$  are not equitable. Using this class, it is nonetheless always possible to reduce the eavesdropper's expected information below 2 bits, and even below any threshold  $\delta > 0$  by choosing  $N$  large enough. For more details, please consult [R].

The following section investigates the situation in which eavesdropping has occurred, but tampering and transmission errors are not a concern for Alice and Bob.

**4.2. Reducing the private channel eavesdropper's information.** Let us now assume that partial eavesdropping may have occurred on the private channel. Recall that eavesdropping consists of Eve selecting a function  $e: \{0, 1\}^N \rightarrow \{0, 1\}^K$  of her choice, whose value  $e(x)$  she learns when  $x$  is transmitted over the private channel. Alice and Bob know  $K$  but otherwise have no information on which function  $e$  was chosen by Eve. If Eve chooses an equitable function,  $e(x)$  gives her  $K$  bits of information on  $x$ . Otherwise, her expected amount of information is smaller, but she could occasionally get more. In this section, we assume that transmission errors and tampering are not a worry for Alice and Bob, so that an error-detection protocol of § 3.1 is not carried out. This assumption is removed in § 4.3.

To summarize, let  $x$  be the length  $N$  bit string common to Alice and Bob, and let  $e(x)$  be the  $K$  bits of information known by Eve about  $x$ . Alice and Bob wish to publicly agree on some function  $g: \{0, 1\}^N \rightarrow \{0, 1\}^R$ , for some  $R \leq N - K$ , such that knowledge of  $e$ ,  $e(x)$  and  $g$  leaves Eve with an arbitrary small fraction of one bit of information about  $g(x)$ .

A similar question was addressed by Ozarow and Wyner [OW]. However, their solution is nonconstructive (based on random coding), it assumes that the eavesdropper can only read physical bits from the private channel, and it does not reduce her information below one bit. On the other hand, their setting is not restricted to the exchange of a random string, as they wish Alice to be able to safely transmit a message of her choice to Bob. Moreover, they do not need to use an authenticated public channel after the private transmission. Their results should also be compared with our information-elimination protocol from § 5.

Back to the analogy with Wyner's *original* wiretap channel [W], the requirement that the eavesdropping function be compressive is analogous to Wyner's requirement that the wiretap channel have less capacity than the main channel. One might hope to generalize our setting to cover eavesdropping through an arbitrary channel of capacity  $K/N$ , selected by Eve but unknown to Alice and Bob. However, this generalization would weaken our results rather than strengthen them. For example, Eve could satisfy the  $K/N$  capacity bound by asking for all  $N$  bits of Alice's message  $K/N$  of the time, while asking for none of it the rest of the time.

Notice that the effect of eavesdropping over the private channel is very similar to that of eavesdropping over the public channel described in § 4.1 in that the information gained by Eve can be characterized by a set  $E = \{z \in \{0, 1\}^N \mid e(z) = e(x)\}$  of possible candidates for  $x$ . However, there is a fundamental difference: contrary to the previously discussed set  $C$ , it is not the case that Alice and Bob have complete knowledge of  $E$ . For this reason, it is not possible for them, in general, to eliminate Eve's information with certainty.

**THEOREM 5.** *If Eve is free to choose her function  $e: \{0, 1\}^N \rightarrow \{0, 1\}^K$  without any constraints, there is always a chance that she will gain complete information on  $g(x)$ , no matter how Alice and Bob choose the function  $g: \{0, 1\}^N \rightarrow \{0, 1\}^R$ .*

*Proof.* The idea is to make  $e$  very nonequitable, so as to have a small chance of learning  $x$  exactly from  $e(x)$ . For example, Eve could choose:

$$e(x) = \begin{cases} x \bmod 2^K & \text{if } x \operatorname{div} 2^K = 0^{N-K}, \\ 0^K & \text{otherwise.} \end{cases}$$

With probability  $(2^K - 1)/2^N$ ,  $e(x)$  yields complete information on  $x$ ; hence on  $g(x)$ .  $\square$

The eavesdropping function described above is for gamblers only: with probability greater than  $1 - 2^{K-N}$ ,  $e(x)$  gives almost no information on  $x$ . In fact, the expected information on  $x$  given by  $e(x)$  is less than  $(N + 1/\ln 2)2^{K-N}$  bits. For instance, if Eve is allowed to observe the result of a 50-bit function applied to a 56-bit DES key [NBS], her expected information would be less than one bit if she had used the above very nonequitable eavesdropping function. Clearly, it would be more prudent for Eve to select some equitable function, so as to maximize her expected amount of information on  $x$ . Even then, an analogue to Theorem 5 can be given:

**THEOREM 6.** *No matter how Alice and Bob choose their function  $g: \{0, 1\}^N \rightarrow \{0, 1\}^R$ , for any  $R > 0$ , there is always an equitable function  $e: \{0, 1\}^N \rightarrow \{0, 1\}^K$ , for any  $K > 0$ , such that knowledge of  $e$ ,  $g$  and  $e(x)$  yields some information on  $g(x)$ .*

*Proof.* Should Alice and Bob choose a nonequitable function  $g$ , Eve would have some information on  $g(x)$  from the mere knowledge that  $x$  is truly random, without

even looking at  $e(x)$ . On the other hand, assuming that  $g$  is equitable,  $e(x)$  could give as much as  $\min(K, R)$  bits of information on  $g(x)$ . This is accomplished if one of the two functions is an equitable refinement of the other. Of course, Eve cannot select  $e$  so that this will happen, because she does not yet know  $g$  when she has to choose  $e$ . However, Alice and Bob cannot prevent this coincidence from happening, or even detect it, because they never get to know the function  $e$ .  $\square$

The above two theorems show that the best Alice and Bob can hope for is to reduce Eve's information arbitrarily close to zero. There can be no analogue to Theorems 1 and 4. Nonetheless, if we restrict even further Eve's choice of  $e$ , so that she can only read a selection of *physical* bits of  $x$ , it becomes possible again for Alice and Bob to eliminate her information entirely, as discussed in § 5.

As usual, we consider two approaches for the reduction of Eve's information: one based on truly random functions and one based on universal hashing techniques. Section 4.2.1 is only of theoretical interest, and the proofs can be found in [R]. Here again, it is nice to find out that the practical schemes of § 4.2.2 are just as efficient for information reduction as the unrealistic random scheme.

**4.2.1. A random information-reduction approach.** Recall that knowledge of  $e: \{0, 1\}^N \rightarrow \{0, 1\}^K$  allows Eve to restrict the set of her possible candidates for  $x$  to  $E = \{z \in \{0, 1\}^N \mid e(z) = e(x)\}$ , where  $\#E = 2^{N-K}$  if  $e$  is equitable.

LEMMA 7. *Let  $E$  be a nonempty set of equally likely candidates for  $x$ , and let  $R$  be an integer. Let  $g: \{0, 1\}^N \rightarrow \{0, 1\}^R$  be a randomly chosen function. Then, knowledge of  $E$  and  $g$  yields less than an expected  $\log(1 + 2^R / \#E)$  bits of information on  $g(x)$ . Here, the result holds for any specific  $E$  and the average is only over all choices for  $g$ .*

THEOREM 8. *Let  $e: \{0, 1\}^N \rightarrow \{0, 1\}^K$  be any function, let  $S < N - K$  be a safety parameter, and let  $R = N - K - S$ . If  $g: \{0, 1\}^N \rightarrow \{0, 1\}^R$  is chosen randomly, the expected amount of information on  $g(x)$  given by knowledge of  $e$ ,  $g$  and  $e(x)$  is less than  $\log(1 + 2^{-S})$  bits, hence less than  $2^{-S} / \ln 2$  bits.*

**4.2.2. A universal hashing information-reduction approach.** Contrary to the error detection protocols of § 3, it is no longer sufficient to consider  $\text{universal}_2$  classes: here, we need *strongly universal* $_2$  classes [WC].

LEMMA 9. *Let  $E$  and  $R$  be as in Lemma 7. Let  $H$  be a publicly known strongly  $\text{universal}_2$  class of hash functions from  $\{0, 1\}^N$  to  $\{0, 1\}^R$ . Let  $g$  be a function chosen randomly within  $H$ . Then, knowledge of  $E$  and  $g$  yields less than an expected  $\log(1 + 2^R / \#E)$  bits of information on  $g(x)$ . Again, the average is only over all choices of  $g$ , not over  $E$  or  $H$ .*

*Proof.* Let us first recall some notation from universal hashing: if  $x \in \{0, 1\}^N$ ,  $z \in \{0, 1\}^R$  and  $g \in H$ , then

$$\Delta_z^{x,g} = \begin{cases} 1 & \text{if } g(x) = z, \\ 0 & \text{otherwise.} \end{cases}$$

If  $E \subseteq \{0, 1\}^N$ , then

$$\Delta_z^{E,g} = \sum_{x \in E} \Delta_z^{x,g} = \#\{x \in E \mid g(x) = z\}.$$

Similarly, if  $F \subseteq H$ , then

$$\Delta_z^{x,F} = \sum_{g \in F} \Delta_z^{x,g} = \#\{g \in F \mid g(x) = z\}.$$

By definition of strong  $\text{universal}_2$ ,

$$\#\{g \in H \mid g(x) = z \text{ and } g(x') = z'\} = \#H / 2^{2R}$$

for any  $x, x' \in \{0, 1\}^N$  and  $z, z' \in \{0, 1\}^R$ , provided  $x \neq x'$ .

An immediate consequence of this definition is that

$$\begin{aligned}\Delta_z^{x,H} &= \#\{g \in H \mid g(x) = z\} \\ &= \sum_{z' \in \{0,1\}^R} \#\{g \in H \mid g(x) = z \text{ and } g(x') = z'\} \\ &= 2^R \#H / 2^{2R} = \#H / 2^R,\end{aligned}$$

where  $x'$  is chosen as any string different from  $x$  in  $\{0, 1\}^N$ . Therefore,

$$\sum_{g \in H} \Delta_z^{E,g} = \sum_{x \in E} \Delta_z^{x,H} = \#E \#H / 2^R.$$

Similarly,

$$\begin{aligned}\sum_{g \in H} (\Delta_z^{E,g})^2 &= \sum_{g \in H} \left( \sum_{x \in E} \Delta_z^{x,g} \right)^2 \\ &= \sum_{g \in H} \sum_{x \in E} \sum_{x' \in E} \Delta_z^{x,g} \Delta_z^{x',g} \\ &= \sum_{x \in E} \sum_{x' \in E} \#\{g \in H \mid g(x) = z \text{ and } g(x') = z\} \\ &= \#E [\#H / 2^R + (\#E - 1) \#H / 2^{2R}]\end{aligned}$$

(from splitting the cases  $x' = x$  and  $x' \neq x$ )

$$= \frac{\#E \#H}{2^R} \left[ 1 + \frac{\#E - 1}{2^R} \right].$$

Let us now come back to Eve's set  $E \subseteq \{0, 1\}^N$  of equally likely candidates for  $x$ . In this proof, we consider two independent random variables  $\mathbf{X}$  and  $\mathbf{G}$  ranging over  $\{0, 1\}^N$  and the set of functions from  $\{0, 1\}^N$  to  $\{0, 1\}^R$ , respectively, with the probability distributions

$$\text{prob} [\mathbf{X} = x] = \begin{cases} 1/\#E & \text{if } x \in E, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\text{prob} [\mathbf{G} = g] = \begin{cases} 1/\#H & \text{if } g \in H, \\ 0 & \text{otherwise.} \end{cases}$$

Consider also the dependent random variable  $\mathbf{Z}$  ranging over  $\{0, 1\}^R$  corresponding to the equation  $z = g(x)$ . We wish to find an upper bound on  $\mathbf{I}(\mathbf{Z}; \mathbf{G})$ , the expected information on  $g(x)$  given from the knowledge of  $g$  and of the fact that  $x \in E$ . For this, we need to compute a few conditional, joint and marginal probabilities: given  $z \in \{0, 1\}^R$  and  $g \in H$ ,

$$\begin{aligned}\text{prob} [\mathbf{Z} = z \mid \mathbf{G} = g] &= \Delta_z^{E,g} / \#E, \\ \text{prob} [\mathbf{Z} = z, \mathbf{G} = g] &= \text{prob} [\mathbf{G} = g] \text{prob} [\mathbf{Z} = z \mid \mathbf{G} = g] \\ &= \Delta_z^{E,g} / \#E \#H,\end{aligned}$$



and

$$\begin{aligned} \text{prob} [\mathbf{Z} = z] &= \sum_{g \in H} \text{prob} [\mathbf{Z} = z, \mathbf{G} = g] \\ &= \frac{1}{\# E \# H} \sum_{g \in H} \Delta_z^{E,g} = 2^{-R}. \end{aligned}$$

By information-theoretic definitions, we therefore obtain:

$$\begin{aligned} \mathbf{I}(\mathbf{Z}; \mathbf{G}) &= \sum_{z \in \{0,1\}^R} \sum_{g \in H} \text{prob} [\mathbf{Z} = z, \mathbf{G} = g] \log \frac{\text{prob} [\mathbf{Z} = z | \mathbf{G} = g]}{\text{prob} [\mathbf{Z} = z]} \\ &= \sum_z \sum_g \frac{\Delta_z^{E,g}}{\# E \# H} \log \frac{\Delta_z^{E,g} 2^R}{\# E} \\ &= 2^{-R} \sum_z \sum_g \Delta_z^{E,g} \frac{2^R}{\# E \# H} \log \frac{\Delta_z^{E,g} 2^R}{\# E} \\ &\leq 2^{-R} \sum_z \log \sum_g (\Delta_z^{E,g} 2^R / \# E)^2 / \# H \end{aligned}$$

(by Jensen's lemma [Mc], because  $\sum_g \Delta_z^{E,g} 2^R / \# E \# H = 1$ )

$$\begin{aligned} &= 2^{-R} \sum_z \log \left( \frac{2^{2R}}{(\# E)^2 \# H} \sum_g (\Delta_z^{E,g})^2 \right) \\ &= 2^{-R} \sum_z \log \left( \frac{2^R}{\# E} + \frac{\# E - 1}{\# E} \right) \\ &< \log (1 + 2^R / \# E). \quad \square \end{aligned}$$

**THEOREM 10.** *Let  $e$ ,  $S$  and  $R$  be as in Theorem 8, and let  $H$  and  $g$  be as in Lemma 9. The expected amount of information on  $g(x)$  given by knowledge of  $e$ ,  $g$  and  $e(x)$  is less than  $\log (1 + 2^{-S})$  bits, hence less than  $2^{-S} / \ln 2$  bits. Notice that this is true for every  $e$ , despite the fact that Eve already knows the class  $H$ , but of course not the specific function  $g$ , when she gets to choose her function  $e$ .*

*Proof.* This is an immediate consequence of Lemma 9 if  $e$  is equitable. Indeed, the eavesdropper's set  $E$  is then always reduced to exactly  $2^{N-K}$  equally likely candidates for  $x$ . The expected information on  $g(x)$  given by knowledge of  $E$  and  $g$  is thus less than  $\log (1 + 2^R / 2^{N-K}) = \log (1 + 2^{-S}) < 2^{-S} / \ln 2$  bits.

If  $e$  is not equitable, the theorem still follows from Lemma 9, but through the use of Jensen's lemma [Mc]. For any  $x \in \{0, 1\}^N$ , let  $E_x = \{y \in \{0, 1\}^N | e(y) = e(x)\}$ . Since each  $x \in \{0, 1\}^N$  is equally likely a priori, Lemma 9 tells us that the expected information on  $g(x)$  given by knowledge of  $g$ ,  $e$  and  $e(x)$  is less than

$$\sum_{x \in \{0,1\}^N} 2^{-N} \log (1 + 2^R / \# E_x) \leq \log \sum_x (2^{-N} + 2^{R-N} / \# E_x)$$

(by Jensen's lemma, because  $\sum_x 2^{-N} = 1$ )

$$\leq \log (1 + 2^{-S})$$

(because  $\sum_x 1 / \# E_x \leq 2^K$ , by an easy exercise left to the reader).  $\square$

Let us finally point out that *almost* strongly universal<sub>2</sub> classes [WC] can also be used in this information-reduction context. A similar analysis shows that if  $g$  is chosen

randomly from an almost strongly universal<sub>2</sub> class, knowledge of  $e$ ,  $g$  and  $e(x)$  yields at most  $1 + \log(1 + 2^{-(S+1)})$  bits of information to Eve about  $g(x)$ . This could be interesting when  $R$  is much smaller than  $N$ : in that case the description of a randomly chosen function within an almost strongly universal<sub>2</sub> class requires significantly fewer bits to be transmitted over the public channel [WC].  $\square$

**4.3. Putting the concepts together.** If Eve obtained some information on the string  $x$  as it was being transmitted over the private channel, and additional information by listening to the public channel messages exchanged during the error-correction and error-detection protocols of §§ 3.2 and 3.1, it may still be possible for Alice and Bob to efficiently agree on a random string on which Eve has nearly no information. The key idea is that if Eve obtained an expectation of  $K$  bits from the private channel transmission, and an expectation of  $L$  more bits from eavesdropping on the subsequent public discussion, then she can expect at most  $K + L$  bits from both sources combined. This is formalized in the following easily proven lemma.

LEMMA 11. *Let  $e: \{0, 1\}^N \rightarrow \{0, 1\}^K$  and  $f: \{0, 1\}^N \rightarrow \{0, 1\}^L$  be any two functions. Let  $x$  be a random string of length  $N$ . The expected information on  $x$  given by knowledge of  $e$ ,  $e(x)$ ,  $f$  and  $f(x)$  is at most  $K + L$  bits.*

Therefore, an obvious adaptation of the protocol implied by Theorem 10 allows for the efficient reduction of Eve's information to less than  $2^{-S}/\ln 2$  bits for any  $S < N - K - L$ , at the cost of ending up with a random string of length  $N - K - L - S$ . Notice that ad hoc information-elimination schemes, such as those in Theorems 1 and 4, offer no advantages in this context (unless an information-elimination scheme from § 5 is used initially). Therefore, the choice of a universal<sub>2</sub> class in the error-detection part of the protocol does not have to be motivated by the existence of a subsequent information-elimination scheme.

**5. Elimination of the eavesdropper's information.** The protocols of § 4.2 should be sufficient for most applications, despite the fact that Eve retains an arbitrarily small fraction of one bit of information on the resulting shared random string. Although we were able to eliminate her information entirely in Theorems 1 and 4, the techniques used could only be applied because Alice and Bob had complete knowledge of Eve's information. As shown in Theorems 5 and 6, this cannot be extended whenever Eve is allowed to access a limited amount of information *of her choice* from the private channel transmission.

In this section, we investigate a protocol by which Alice and Bob can nonetheless wipe out Eve's information, assuming that she obtained a maximum of  $K$  *physical* bits of her choice from the private channel transmission, as opposed to the more general  $K$  bits of information in Shannon's sense discussed in § 4. Although the value of  $K$  is known to Alice and Bob, they do not know, of course, which particular bits of their string are compromised. This protocol is expensive in the sense that the resulting string is generally substantially shorter than those resulting from the protocols of § 4; however, this is the unavoidable price to pay in order to make sure that Eve is left with no information at all.

An error-detection protocol could be applied, if desired, after Eve's information has been eliminated, still leaving her with no information if the universal<sub>2</sub> class  $P$  of Theorem 4 is used. Bit twiddling on the initial strings is also possible afterwards in order to reconcile the final strings, if there were only one or two transmission errors. Unfortunately, the more sophisticated protocol of § 3.2 would transform Eve's knowledge from physical bits to information in Shannon's sense, so that the elimination

protocols described below could no longer be applied. We do not know how to efficiently eliminate Eve's information and reconcile Alice and Bob's strings if several transmission errors occurred.

**5.1. The notion of  $(N, J, K)$ -functions.** In order to eliminate Eve's information, we introduce the following definition: for any integers  $N, J$  and  $K$  such that  $N \geq J + K$ ,  $J > 0$  and  $K > 0$ , a function  $f: \{0, 1\}^N \rightarrow \{0, 1\}^J$  is  $(N, J, K)$  if, no matter how one fixes any  $K$  of its input bits, each of the  $2^J$  output bits can be produced in exactly  $2^{N-J-K}$  different ways by varying the remaining  $N - K$  input bits. Intuitively, an  $(N, J, K)$ -function compresses an  $N$  bit string into a  $J$  bit string in such a way that knowledge of any  $K$  of the input bits gives no information on the output. This is equivalent to the notion of  $t$ -resilient functions independently introduced by [CGHFRS].

Given such a function, Alice and Bob can apply it to their respective strings, thus producing a new (shorter) string on which Eve has no information. Notice that this still holds even if she already knows which function will be used by Alice and Bob in advance of her deciding which  $K$  bits to read from the private channel. Therefore, the subsequent public transmission between Alice and Bob is not necessary in this case. By analogy with [OW], these  $(N, J, K)$ -functions are not restricted to the exchange of random strings. If Alice wished to communicate some *specific*  $J$  bit string  $y$  to Bob, she could send over the private channel some randomly chosen  $N$  bit string  $x$  such that  $f(x) = y$ . This would allow Bob to obtain  $y$  unambiguously, assuming no transmission errors occurred, whereas Eve would gain no information on  $y$  from eavesdropping over any  $K$  bits of  $x$ . One (nonconstructive) protocol in [OW] achieves something similar with  $N = J + K$ , which is better than what we get here, but it yields slightly more than one bit of information to Eve about  $y$ .

*Example.* The function  $f(u, v, w, x, y, z) = (u \oplus v \oplus w \oplus x, w \oplus x \oplus y \oplus z)$  is  $(6, 2, 3)$ : knowledge of any 3 of the input bits yields no information at all on the output.

The case  $J = N - K$  is the best possible because there is obviously no hope of producing a completely secret string of length  $N - K + 1$  if Eve knows  $K$  of the  $N$  original bits. A function  $f$  that is  $(N, N - K, K)$  is said to be  $(N, K)$ . The following theorem shows how to build  $(N, K)$ -functions whenever they exist.

**THEOREM 12.** (1) For any  $N > 1$ , there are  $(N, 1)$  and  $(N, N - 1)$ -functions.

(2) For any  $N > 3$ , there are no  $(N, K)$ -functions whenever  $1 < K < N - 1$ .

*Proof.* (1) Produce the  $i$ th output bit as the exclusive-or of the  $i$ th and the  $(i + 1)$ st input bits to get an  $(N, 1)$ -function; and produce the only output bit as the exclusive-or of all  $N$  input bits to get an  $(N, N - 1)$ -function.

(2) Assume for a contradiction that some  $f: \{0, 1\}^N \rightarrow \{0, 1\}^{N-K}$  is  $(N, K)$  for  $N > 3$  and  $1 < K < N - 1$ . By definition  $f(x_1) \neq f(x_2)$  for any two distinct strings  $x_1$  and  $x_2$  that have at least  $K$  bits in common. Let  $X = \{x \in \{0, 1\}^N \mid x \neq 0^N \text{ and } x \bmod 2^K = 0^K\}$ , the set of nonzero  $N$  bit strings ending with  $K$  zeros. Notice that  $f$  must be one-to-one on  $X$  because any two strings in  $X$  have their last  $K$  bits in common. Now, consider the strings  $u = 1^{N-K+1}0^{K-1}$  and  $v = 1^{N-K}0^{K-1}1$ . Both  $u$  and  $v$  have at least  $K$  bits in common with each string of  $X$ . Therefore,  $f(u) \notin f[X]$  and  $f(v) \notin f[X]$ . Since  $f[X]$  spans all of  $\{0, 1\}^{N-K}$  but one string, we must have  $f(u) = f(v)$ . This is a contradiction because  $u$  and  $v$  have  $(K - 2) + (N - K) = N - 2 \geq K$  bits in common.  $\square$

Moreover, there exist only two distinct  $(N, N - 1)$ -functions for each  $N > 1$ : the one given in the proof of Theorem 12 and its complement:

**THEOREM 13.** The only  $(N, N - 1)$  functions are  $f(x_1, x_2, \dots, x_N) = x_1 \oplus x_2 \oplus \dots \oplus x_N$  and its complement.

*Proof.* This easy proof by induction on  $N$  is left to the reader.  $\square$

**5.2. How to build  $(N, J, K)$ -functions.** We wish to answer the following question: given  $N$  and  $K$ , what is the maximum value for  $J$  such that an  $(N, J, K)$ -function exists? Alternatively, given  $N$  and  $J$ , find the maximum value for  $K$ . In other words, what is the longest secret random string on which Alice and Bob can agree if they start from a random string of length  $N$ , of which  $K$  bits are compromised? Theorem 12 showed that  $J$  must be strictly smaller than  $N - K$  unless  $K = 1$  or  $K = N - 1$ .

We were unable to answer the above question in its full generality. For this reason, we restrict our attention to the special class of  $(N, J, K)$ -functions for which every output bit is produced as the exclusive-or of some of the input bits. Such functions are referred to as xor- $(N, J, K)$ -functions. We conjecture that these functions are as efficient as possible, in the sense that if no xor- $(N, J, K)$ -functions exist for given values of  $N, J$  and  $K$ , then no general  $(N, J, K)$ -functions exist either. This *Xor-Conjecture* is proved in [CGHFRS] for the case  $J = 2$ .

The following characterization, known as the *Xor-Lemma*, allows us to establish an equivalence between xor- $(N, J, K)$ -functions and binary linear codes [vL].

LEMMA 14 (independently discovered by [CGHFRS]). *Let  $M$  be a  $J \times N$  Boolean matrix. Let  $f: \{0, 1\}^N \rightarrow \{0, 1\}^J$  be the function represented by  $M$  in the natural way (i.e.,  $f(x) = xM'$ , all operations being performed modulo 2). The function  $f$  is  $(N, J, K)$  if and only if the exclusive-or of any set of rows of  $M$  contains at least  $K + 1$  ones.*

*Proof.* One direction is obvious: if the exclusive-or of some subset of the rows contains  $K$  ones or less, it is sufficient to know the value of these  $K$  input bits to infer the exclusive-or of the corresponding output bits.

Conversely, assume that the exclusive-or of any set of rows of  $M$  contains at least  $K + 1$  ones. We have to show that, no matter how many  $K$  input bits are fixed, each of the  $2^J$  output strings can be obtained in exactly  $2^{N-J-K}$  different ways by varying the other  $N - K$  input bits. Without loss of generality, let us assume that the first  $K$  input bits are fixed, say to some  $u \in \{0, 1\}^K$ . Let  $M_1$  and  $M_2$  stand for the first  $K$  and the last  $N - K$  columns of  $M$ , respectively.

The key observation is that  $M_2$  has full rank, because if the exclusive-or of some rows of  $M_2$  were zero, the exclusive-or of the same rows of  $M$  would contain at most  $K$  ones. A classic result of linear algebra applies to conclude that there exists a nonsingular  $(N - K) \times (N - K)$  matrix  $F$  such that  $M_2 = RF$ , where  $R$  is the  $J \times (N - K)$  matrix such that  $R_{ij} = 1$  for  $1 \leq i \leq J$  and  $R_{ij} = 0$  otherwise [ND].

Now, consider any  $y \in \{0, 1\}^J$ . Let  $z$  be any string in  $\{0, 1\}^{N-K-J}$ . Let  $v = (y \oplus uM_1', z)(F')^{-1}$ . Let  $x = (u, v)$ . We have  $f(x) = xM' = uM_1' \oplus vM_2' = uM_1' \oplus [(y \oplus uM_1', z)(F')^{-1}F'R'] = y$ . Furthermore, it is clear that different values for  $z$  give in this way different values for  $v$ , hence for  $x$ . Therefore for any  $y \in \{0, 1\}^J$ , there are at least  $2^{N-K-J}$  different  $x \in \{0, 1\}^N$  such that the first  $K$  bits of  $x$  are  $u$  and  $f(x) = y$ . By a pigeonhole argument,  $2^{N-K-J}$  is the exact number of such  $x$ 's. This is the required condition for  $f$  to be  $(N, J, K)$ .  $\square$

THEOREM 15 (independently discovered by [CGHFRS]). *For given values of  $N, J$  and  $K$ , there exists an xor- $(N, J, K)$ -function if and only if there exists an  $[N, J]$  binary linear code with minimum distance at least  $K + 1$  between any two code words.*

*Proof.* This is an immediate consequence of Lemma 14, if one makes the correspondence between the matrix  $M$  used to represent the  $(N, J, K)$ -function and the generator matrix  $G$  of the binary linear code.  $\square$

Consequently, our problem is equivalent to a classic problem of algebraic coding theory. Unfortunately, no efficient algorithms, much less closed-formed formulae, are known to determine the largest possible minimum code-word distance among all  $[N, J]$  binary linear codes. There are, however, several well-known lower and upper bounds

on this value  $[vL]$ ,  $[HS]$ ,  $[MS]$ , and these bounds apply just as well to our problem.

For instance, Hamming codes  $[MS]$  tell us that  $\text{xor}-(2^L-1, 2^L-L-1, 2)$ -functions exist for every  $L \geq 2$ . Conversely, Hamming's upper bound  $[MS]$  shows that no  $\text{xor}-(2^L-1, 2^L-L, 2)$ -functions can exist because

$$2^{(2^L-1)-(2^L-L-1)} = \sum_{i=0}^1 \binom{2^L-1}{i}.$$

Notice that elimination of Eve's information in this case ( $K=2$ ) costs  $L-2-S$  additional bits than if we had been satisfied to reduce her information below  $2^{-S}/\ln 2$  bits, as in § 4.2.

Similarly, Griesmer's upper bound and the simplex code  $[MS]$  allow for the building of  $\text{xor}-(2^L-1, L, 2^{L-1}-1)$ -functions for any  $L \geq 2$ , whereas neither  $\text{xor}-(2^L-1, L, 2^{L-1})$ -functions nor  $\text{xor}-(2^L-1, L+1, 2^{L-1}-1)$ -functions can exist. Moreover, Varsharmov-Gilbert's lower bound together with McEliece's upper bound  $[vL]$ ,  $[MS]$  allow for the construction of  $\text{xor}-(N, J, K)$ -functions such that  $J$  is at least half the optimal ( $\text{xor}$ ) value, as long as  $K/N < 0.3$  and  $N$  is large enough. Also, the *zigzag* of  $[BCR]$  yields an  $\text{xor}-(3^L, 2^L, 2^L-1)$ -function, for every positive integer  $L$ .

Finally, the general question of  $(N, J, K)$ -functions is solved completely when  $K > \frac{2}{3}N-1$ , thanks to the proof of the Xor-Conjecture for  $J=2$   $[CGHFRS]$ . In this case, it is easy to see that no  $\text{xor}-(N, 2, K)$ -functions can exist, and therefore no general  $(N, 2, K)$ -functions can exist either. Since the exclusive-or of all the input bits is  $(N, 1, K)$ , we conclude that 1 is the maximum possible value for  $J$  when  $\frac{2}{3}N-1 < K < N$ . On the other hand, there is always an  $(N, 2, \lfloor \frac{2}{3}N-1 \rfloor)$ -function. We encourage the reader to consult  $[CGHFRS]$  for additional results on  $(N, J, K)$ - (*alias*  $t$ -resilient) functions.

**6. Conclusions.** If no eavesdropping occurred over the private channel, it is possible for Alice and Bob to publicly verify that no transmission errors or tampering occurred either, with a  $2^{-K}$  error probability, and end up with an entirely secret final string that is only  $K$  bits shorter than the original private transmission. This is optimal.

If partial eavesdropping occurred over the private channel, leaking up to  $K$  bits of information to Eve, in Shannon's sense, it is still possible for Alice and Bob to publicly verify that no transmission errors or tampering occurred, with a  $2^{-L}$  error probability, and end up with a final string that is  $K+L+S$  bits shorter than the original private transmission, on which Eve has less than  $2^{-S}/\ln 2$  bits of information on the average. Moreover, discrepancies between the transmitted and received versions of the private channel transmission, whether they are due to channel noise or tampering, can be handled at the cost of a further reduction in the length of the final shared secret string. If the discrepancies are too numerous, no final shared secret string can be constructed, but Alice and Bob will detect this condition with very high probability, and will not be misled into constructing a string that is neither shared nor secret.

Finally, if partial eavesdropping over the private channel is restricted to  $K$  physical bits secretly chosen by Eve, it becomes possible again for Alice and Bob to verify with high probability that no errors or tampering occurred, and to end up with a new string on which Eve has no information whatsoever. However, the new string will be substantially shorter than if Alice and Bob had tolerated knowledge by Eve of an arbitrarily small fraction of one bit of information. This remains possible even if a small number of transmission errors occurred, but we do not know how to eliminate Eve's information and reconcile the strings efficiently in the presence of severe transmission errors.

## REFERENCES

- [BB1] C. H. BENNETT AND G. BRASSARD, *Quantum cryptography and its application to provably secure key expansion, public-key distribution and coin-tossing*, Proc. IEEE International Conference on Computers, Systems and Signal Processing, Bangalore, India, December 1984, pp. 175-179.
- [BB2] ———, *An update on quantum cryptography*, in *Advances in Cryptology: Proc. CRYPTO 84*, G. R. Blakley and D. Chaum, eds., Lecture Notes in Computer Science 196, Springer-Verlag, Berlin, 1985, pp. 475-480.
- [BBR] C. H. BENNETT, G. BRASSARD AND J.-M. ROBERT, *A perfect secrecy interactive reconciliation protocol*, in preparation; some of the results can be found in *How to reduce your enemy's information*, in *Advances in Cryptology: Proc. CRYPTO 85*, H. C. Williams, ed., Lecture Notes in Computer Science 218, Springer-Verlag, Berlin, 1986, pp. 468-476.
- [Be] E. R. BERLEKAMP, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.
- [Br] G. BRASSARD, *On computationally secure authentication tags requiring short secret shared keys*, in *Advances in Cryptology: Proc. CRYPTO 82*, D. Chaum, R. L. Rivest and A. T. Sherman, eds., Plenum, New York, 1983, pp. 267-275.
- [BCR] G. BRASSARD, C. CRÉPEAU AND J.-M. ROBERT, *Information theoretic reductions among disclosure problems*, Proc. 27th IEEE Symposium on Foundations of Computer Science, Toronto, Ontario, 1986, pp. 168-173.
- [CW] J. L. CARTER AND M. N. WEGMAN, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143-154.
- [CGHFRS] B. CHOR, O. GOLDBREICH, J. HASTAD, J. FREIDMANN, S. RUDICH AND R. SMOLENSKY, *The bit extraction problem or  $t$ -resilient functions*, Proc. 26th IEEE Symposium on Foundations of Computer Science, Portland, Oregon, 1985, pp. 396-407.
- [DH] W. DIFFIE AND M. E. HELLMAN, *New directions in cryptography*, IEEE Trans. Inform. Theory, IT-22 (1976), pp. 644-654.
- [G] R. G. GALLAGER, *Information Theory and Reliable Communications*, John Wiley, New York, 1968.
- [GGM] O. GOLDBREICH, S. GOLDWASSER AND S. MICALI, *How to construct random functions*, Proc. 25th IEEE Symposium on Foundations of Computer Science, Singer Island, FL, 1984, pp. 464-479.
- [HS] H. J. HELGERT AND R. D. STINOFF, *Minimum distance bounds for binary linear codes*, IEEE Trans. Inform. Theory, IT-19 (1973), pp. 344-356.
- [vL] J. H. VAN LINT, *Introduction to Coding Theory*, Graduate Text in Mathematics 86, Springer-Verlag, New York, 1982.
- [MS] F. J. MACWILLIAMS AND N. J. A. SLOANE, *The Theory of Error-Correcting Codes*, North-Holland, New York, 1977.
- [Mc] R. J. MCELIECE, *The theory of information and coding*, in *Encyclopedia of Mathematics and its Applications*, Vol. 3, Addison-Wesley, Reading, MA, 1977.
- [NBS] NATIONAL BUREAU OF STANDARDS, *Data Encryption Standard*, FIPS PUB 46, Washington, DC, 1977.
- [ND] B. NOBLE AND J. W. DANIEL, *Applied Linear Algebra*, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1977.
- [OW] L. H. OZAROW AND A. D. WYNER, *Wire-tap channel II*, Bell System Tech. J., 63 (1984), pp. 2135-2157.
- [RSA] R. L. RIVEST, A. SHAMIR AND L. ADLEMAN, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. ACM, 21 (1978), pp. 120-126.
- [R] J.-M. ROBERT, *Détection et correction d'erreur en cryptographie*, Masters thesis, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal, Québec, Canada, 1985.
- [S] C. SHANNON, *The mathematical theory of communication*, Bell System Tech. J., 27 (1948), pp. 379-423, 623-656.
- [WC] M. N. WEGMAN AND J. L. CARTER, *New hash functions and their use in authentication and set equality*, J. Comput. System Sci., 22 (1981), pp. 265-279.
- [W] A. D. WYNER, *The wire-tap channel*, Bell System Tech. J., 54 (1975), pp. 1355-1387.

## UNBIASED BITS FROM SOURCES OF WEAK RANDOMNESS AND PROBABILISTIC COMMUNICATION COMPLEXITY\*

BENNY CHOR<sup>†</sup> AND ODED GOLDREICH<sup>‡</sup>

**Abstract.** A new model for weak random physical sources is presented. The new model strictly generalizes previous models (e.g., the Santha and Vazirani model [27]). The sources considered output strings according to probability distributions in which *no single string is too probable*.

The new model provides a fruitful viewpoint on problems studied previously such as:

- *Extracting almost-perfect bits from sources of weak randomness.* The question of possibility as well as the question of efficiency of such extraction schemes are addressed.
- *Probabilistic communication complexity.* It is shown that most functions have linear communication complexity in a very strong probabilistic sense.
- *Robustness of BPP* with respect to sources of weak randomness (generalizing a result of Vazirani and Vazirani [32], [33]).

**Key words.** randomness, physical sources, discrete probability distributions, communication complexity, randomized complexity classes

**1. Introduction.** The notion of randomness is central to the theory of computation. Thus, the question of whether and how randomness can be implemented in a computer is of major importance. Our intention is not to address the metaphysical aspect of the above question. Rather we assume that there are physical phenomena which appear to be “somewhat random,” and study the consequences of such an assumption.

In reality, there is a variety of physical sources, the output of which appears to be unpredictable in some sense (e.g., noise diodes, Geiger counters, etc.). However, these sources do not seem to be perfect (i.e., they do not output a uniform distribution). This phenomenon is amplified when trying to convert the analogue signal to a digital one, and in particular when sampling the physical source very frequently.

The main contribution of this paper is in presenting a general model for sources of weak randomness. This model not only generalizes previous models, but is also very convenient to manipulate and analyze. The new model provides a new viewpoint on several problems studied previously, and enables us to obtain interesting new results:

- *Extracting almost-perfect bits from sources of weak randomness.* It is shown that almost all functions can be used for extracting many “almost-unbiased” bits from two independent sources of “weak” randomness. An explicit function which performs almost as well is also presented. These results yield an extraction scheme which is efficient both in terms of output entropy and computational complexity.
- *Probabilistic communication complexity.* It is shown that most Boolean functions have linear communication complexity in a very strong probabilistic sense. This resolves an open problem of Yao [35].
- *Robustness of BPP* with respect to sources of weak randomness. It is shown that any probabilistic polynomial-time algorithm can be modified so that it works with bits supplied by a *single* source of weak randomness.

---

\* Received by the editors September 12, 1986; accepted for publication (in revised form) May 30, 1987.

<sup>†</sup> Massachusetts Institute of Technology, Laboratory for Computer Science, Cambridge, Massachusetts 02139. The research of this author was supported in part by an IBM graduate fellowship and a Bantrell postdoctoral fellowship. Present address, Department of Computer Science, Technion, Haifa 32000, Israel.

<sup>‡</sup> Massachusetts Institute of Technology, Laboratory for Computer Science, Cambridge, Massachusetts 02139. The research of this author was supported in part by a Weizmann Postdoctoral Fellowship. Present address, Computer Science Department, Technion, Haifa 32000, Israel.

**1.1. Previous models.** Previous works on extracting unbiased bits from nonperfect sources have implicitly or explicitly proposed models of “weak randomness.” Von Neumann’s classic algorithm [18] deals with sequences of bits generated by independent tosses of a single coin with fixed bias. This model is totally memoryless. Blum [6] models physical sources as finite state Markov chains (with unknown transition probabilities). In this model, it is possible to describe a dependency of the next bit (output by the source) on the previous  $c$  bits (for any fixed  $c$ ).

Santha and Vazirani [27] have further relaxed the restrictions on the physical source. Their model, hereafter referred to as the *SV-model*, is the starting point for our investigations. In the SV-model each bit in the output sequence is “slightly random” in the sense that it is 0 with probability at least  $\delta$  and 1 with probability at least  $\delta$ , where  $\delta \leq \frac{1}{2}$  is a constant. This allows us to model a probabilistic dependency of the next bit (output by the source) on all previous bits. However, no bit of the output may be totally determined by the previous bits. It follows that in the SV-model, every bit sequence is output with some positive probability. This restriction could be violated by some “random” physical sources, which are constrained in a way that prevents certain bit sequences.

**1.2. The new model.** We introduce and study a general model for physical sources, hereafter referred to as the *model of Probability-Bounded sources (PRB-sources)*. Loosely speaking, the probability that a PRB-source will output a particular *string* is bounded above by some parameter. This allows the source to be very imperfect, still it may not concentrate its probability mass on too few strings.

The PRB-model is formalized using two constants  $l$  (length parameter) and  $b$  (probability bound). A physical source  $S$  is a device which outputs an infinite sequence of bits. We say that  $S$  is an  $(l, b)$ -source if for every prefix  $\alpha$  of the output sequence, and every  $l$ -bit string  $\beta$ , the conditional probability that the next  $l$  bits output by  $S$  equal  $\beta$  is at most  $2^{-b}$  (i.e.,  $\Pr(\beta | \alpha) \leq 2^{-b}$ ).

The PRB-model is a strict generalization of the SV-model. To see the inclusion, note that any SV-source with parameter  $\delta$  is a  $(1, \log_2(1 - \delta)^{-1})$ -source. To see that the inclusion is proper, consider the  $(2, 1)$ -source which outputs 11 with probability  $\frac{1}{2}$  and 10 with probability  $\frac{1}{2}$ . Clearly, this source is not an SV-source. Thus, *all positive results* (with respect to the PRB-model) *presented in this paper apply also to the SV-model*.

**1.3. Extracting unbiased bits from sources of weak randomness.** Algorithms for extracting unbiased bits from nonperfect sources depend on the underlying source model. Von Neumann’s algorithm [18] for generating a sequence of unbiased bits by using a coin with fixed bias, is a well-known classic:

- (1) Toss the biased coin twice. Denote the outcome by  $\sigma\tau \in \{HH, HT, TH, TT\}$ .
- (2) If  $\sigma = \tau$  then goto step (1). (Nothing is output.)
- (3) If  $\sigma\tau = HT$  output 0; If  $\sigma\tau = TH$  output 1; Goto step (1).

Elias [11] improved upon von Neumann’s algorithm, showing how to nearly achieve the entropy of the source. He also considered special types of visible finite Markov chains. Elias’s algorithm produces perfect bits from such sources.

Blum [6] has considered extracting (perfect) unbiased bits from general finite Markov chains with *unknown* structure and transition probabilities. He gave algorithms which work in linear expected time. Using Elias’s techniques [11], the extracted bits reach the entropy of the source in the limit.

It seems that as far as extracting *perfect* unbiased bits is concerned, Blum’s schemes are optimal. However, as pointed out by Santha and Vazirani [27], for practical purposes we may lower the standards and settle for “almost” unbiased bits. Having this goal



in mind, they further relaxed the restrictions on the physical source and introduced the SV-model (see § 1.1). Santha and Vazirani showed that a single SV-source cannot be used to extract almost unbiased bits, while sufficiently many independent SV-sources can be used for this purpose. Vazirani [29] showed that by applying inner-product mod 2 to strings of length  $C_\delta \cdot \log_2 \varepsilon^{-1}$  output by two independent SV-sources, a bit with bias  $\leq \frac{1}{2} + \varepsilon$  is produced.

Summarizing the results in [27] and [29], we conclude that the SV-model presents a sufficient condition for the extraction of almost unbiased bits from two independent physical sources. We substantially relax this condition.

In this paper we show that almost all functions can be used to extract many independent unbiased bits from the output of any two independent  $(l, b)$ -sources. To be more specific, let  $m = \frac{1}{3}(b - 3 - \log l) > 0$ , and consider extraction functions from  $l+l$  bits to  $m$  bits. The  $m$  extracted bits are *almost unbiased and independent* in the sense that each  $m$ -bit string appears with probability at least  $(1 - 1/2^m) \cdot 2^{-m}$  and at most  $(1 + 1/2^m) \cdot 2^{-m}$ . This is achieved by a  $1 - 2^{-2^b}$  fraction of all functions from  $2l$ -bit strings to  $m$ -bit strings. Notice that the number of bits we extract from the two sources is within a constant factor ( $\approx 1/6$ ) of the information theoretic bound, a feature not achieved in previous works [27], [29].

We also prove that, for all  $b_1 + b_2 \geq l + 2 + 2 \log_2 \varepsilon^{-1}$ , all functions corresponding to  $2^l$ -by- $2^l$  Hadamard matrices can be used to extract a single bit with bias  $\leq \frac{1}{2} + \varepsilon$  from any two independent PRB-sources which are  $(l, b_1)$ - and  $(l, b_2)$ -distributed, respectively.

A new result contained in this paper, resolves a problem left open in the preliminary version of this work [9]: *an extraction scheme which is efficient both in terms of information rate and computation complexity*. The core of the new method is the discrete logarithm function, and its analysis is based on the method of trigonometric sums. Recently, this open problem was resolved independently by Vazirani [31]. His solution is simpler than ours.

**1.4. Probabilistic communication complexity.** Vazirani pointed out that “good” bit-extraction functions have high communication complexity [29]. We establish further connections between the two notions. We show that functions which can be used for extracting an almost unbiased bit from two probability-bounded sources have *linear* communication complexity in a very strong sense. It follows that almost all functions, and in particular all functions corresponding to Hadamard matrices, have linear communication complexity. This resolves Yao’s open problem [35] regarding the probabilistic communication complexity of random functions and of the set intersection function. (Related lower bounds on the communication complexity of random functions were presented independently by Alon, Frankl and Rödl [5] and by Orlitsky and El-Gamal [20]. Our linear  $(\Omega(n))$  lower bound on the inner product modulo 2 function, improves over Vazirani’s  $\Omega(n/\log n)$  bound presented in [29].)

Another contribution in the field of communication complexity is the presentation of definitions and results for the case that the inputs are taken from probability-bounded distributions (i.e., distributions in which no string is too likely). This contribution is in the spirit of Vazirani’s suggestion to analyze the communication complexity with respect to inputs chosen by an SV-model [29]. However, we feel that probability-bounded distributions are more natural in the context of communication complexity. We consider *randomized* protocols where the objective is to guess the value of the function with *average* success probability exceeding  $\frac{1}{2} + \varepsilon$ . Both the average length of a run and the average success probability are taken with respect to the “best” (for the protocol) probability-bounded distribution. We show that, even with respect to such

protocols and distributions, the *average* communication complexity of almost all functions is linear in the probability bound  $b$  (where no input appears with probability greater than  $2^{-b}$ ).

**1.5. On the robustness of BPP.** The class  $R$  [1] and its symmetric version BPP [13] consist of problems which can be solved with high probability in polynomial time. The probability is taken over the tosses of an unbiased coin. Umesh Vazirani raised the question whether BPP problems can be efficiently solved if a (single) SV-source is producing the coin tosses. Recently, Vazirani and Vazirani have answered this question affirmatively [32], [33]. In this paper, we generalize their result by showing that BPP problems can be efficiently solved if a (single) PRB-source is producing the coin tosses. The underlying principles of our proof originate from Vazirani and Vazirani [32], [33] and [30].

The main idea of the proof is that while a single PRB-source is useless for producing a *single* unbiased bit, it can nevertheless be used for producing polynomially many bits, most of which are unbiased. Our key observation is that *any* function which extracts almost unbiased bits from any two independent PRB-sources, can be used for this purpose. Thus, our contribution in explicitly reducing the problem of “the robustness of BPP” to the problem of “extracting almost unbiased bits from two independent sources.”

**1.6. Organization.** In § 2, we present our basic definitions and results concerning the extraction of unbiased bits from sources of weak randomness. These results are the basis for the rest of the paper. Section 2.1 consists of definitions. In § 2.2, we present impossibility results. In § 2.3, we introduce the notion of flat distributions and demonstrate its importance. In § 2.4, we show that almost all functions extract unbiased bits from any two independent PRB-sources, and in § 2.5 we show that functions corresponding to Hadamard matrices also perform well.

Each of the next three sections is based on § 2 only, and can be read independently of the others. In § 3, we further study the problem of extracting unbiased bits from probability-bounded sources. In § 3.1, we analyze extraction schemes with respect to two efficiency measures: rate and computation complexity. In § 3.2, we present and analyze the “discrete logarithm” extraction scheme. In § 3.3, we consider extraction from slightly dependent sources. In § 3.4, we consider various extensions of our model and results.

In § 4, we present results concerning probabilistic communication complexity. In § 4.1, we present old and new definitions of probabilistic communication complexity. In § 4.2, we prove linear lower bounds on the communication complexity of functions, and in § 4.3 we present almost-matching upper bounds. In § 4.4, we suggest and investigate a robust notion of communication complexity.

In § 5, we deal with the robustness of BPP with respect to probability-bounded sources.

**2. Extracting unbiased bits—Part I.** In this section we present our basic definitions and results concerning the extraction of unbiased bits from sources of weak randomness. These results will be the basis for our more advanced study of the efficiency of extraction schemes, as well as our results concerning communication complexity and the robustness of BPP. In § 2.1, we define probability bounded sources (distributions) and robust extraction schemes. In § 2.2, we present impossibility results which will be used later to demonstrate the optimality of our positive results. In § 2.3, we introduce the notion of flat distributions and demonstrate its importance. In § 2.4, we use a counting

argument to prove the existence of good extraction schemes. In § 2.5, we show that functions corresponding to Hadamard matrices constitute good extraction schemes.

**2.1. Definitions.** The first two definitions are used to characterize the PRB-sources.

**DEFINITION 1.** Let  $l$  be a positive integer, and  $b > 0$  a real number. Let  $X$  be a random variable assuming values in  $\{0, 1\}^l$ . We say that  $X$  is  $(l, b)$ -distributed if for every  $\alpha \in \{0, 1\}^l$ , the probability that  $X = \alpha$  is  $\leq 2^{-b}$ .

**DEFINITION 2.** Let  $X_1, X_2, \dots, X_t$  be a sequence of random variables, each assuming values in  $\{0, 1\}^l$ . The random variable  $X_t$  is  $(l, b)$ -distributed given  $X_1, \dots, X_{t-1}$  if for every  $\alpha \in \{0, 1\}^{(t-1) \cdot l}$  and  $\beta \in \{0, 1\}^l$ ,  $\Pr(X_t = \beta \mid X_1 \cdots X_{t-1} = \alpha) \leq 2^{-b}$ .

An  $(l, b)$ -source is an infinite sequence of random variables  $X_1, X_2, X_3 \cdots$  each assuming values in  $\{0, 1\}^l$  such that for every  $t$ , the random variable  $X_t$  is  $(l, b)$ -distributed given the values of the variables  $X_1$  through  $X_{t-1}$ .<sup>1</sup> Unless otherwise stated, all distributions are conditioned on the entire past.

The next definitions will be used in evaluating the quality of the extracted bits.

**DEFINITION 3.** Let  $Z$  be a random variable assuming values in  $\{0, 1\}^m$ .  $Z$  is said to be  $\epsilon$ -robust if for every  $\alpha \in \{0, 1\}^m$ ,

$$(1 - \epsilon) \cdot 2^{-m} \leq \Pr(Z = \alpha) \leq (1 + \epsilon) \cdot 2^{-m}.$$

**DEFINITION 4.** Let  $X_1, X_2, \dots, X_s$  be  $s$  independent random variables, each assuming values in  $\{0, 1\}^l$ . A function  $f: \{0, 1\}^{s \cdot l} \rightarrow \{0, 1\}^m$  is said to be  $\epsilon$ -robust on  $X_1, X_2, \dots, X_s$  if the random variable  $Z \stackrel{\text{def}}{=} f(X_1, X_2, \dots, X_s)$  is  $\epsilon$ -robust.

A function  $f: \{0, 1\}^{s \cdot l} \rightarrow \{0, 1\}^m$  is said to be  $\epsilon$ -robust with respect to properties  $P_1, P_2, \dots, P_s$  if  $f$  is  $\epsilon$ -robust on every  $s$  independent random variable  $X_1, X_2, \dots, X_s$ , satisfying  $P_1, P_2, \dots, P_s$ , respectively.

**2.2. Impossibility results.** It is no surprise that one probability-bounded source cannot be used to generate unbiased bits, since probability-bounded sources include SV-sources for which an impossibility result was shown [27]. Yet, a stronger impossibility result holds for our model.

**THEOREM 1.** Let  $k \geq 1$  be an integer, and  $f: \{0, 1\}^{k \cdot l} \rightarrow \{0, 1\}$  be a Boolean function. Then there exists a  $\sigma \in \{0, 1\}$  and a sequence of  $k$  random variables  $X_1, X_2, \dots, X_k$ , each  $(l, l-1)$ -distributed given the previous ones, such that  $f(X_1 X_2 \cdots X_k)$  is identically  $\sigma$ .

*Proof.* The proof is by induction on  $k$ . For the base case  $k = 1$ , assume without loss of generality, that  $f$  attains the value 1 on at least half the inputs. Setting  $X_1$ 's probability distribution to be uniform on these inputs and 0 otherwise,  $f(X_1)$  is identically 1. By the induction hypothesis, for every  $\alpha \in \{0, 1\}^l$ , there is a  $\sigma \in \{0, 1\}$  such that the function  $f_\alpha(X_2, \dots, X_k) = f(\alpha, X_2, \dots, X_k)$  can be made identically  $\sigma$ . Without loss of generality, for at least half the  $\alpha$ 's,  $f_\alpha$  can be made identically 1. When we set  $X_1$ 's probability distribution to be uniform on these  $\alpha$ 's and 0 elsewhere, the theorem follows.  $\square$

While a single source cannot be used at all, there is a lower bound on the robustness of functions applied to the output of two probability-bounded sources. We start with a combinatorial lemma.

**LEMMA 2.** Let  $M$  be an  $L \times N$  Boolean matrix. Then there exists a  $\sigma \in \{0, 1\}$  and an  $L/16 \times N/2$  submatrix of  $M$  containing at least  $\frac{1}{2} \cdot (L/16)(N/2 + \sqrt{N/2})$   $\sigma$ -entries.

*Proof.* Without loss of generality, at least half the rows contain at least half 1's. We restrict ourselves to these  $L/2$  rows. Fix any such row, pick  $N/2$  columns at random, and let  $P$  denote the probability that at least  $T = \frac{1}{2}(N/2 + \sqrt{N/2})$  of the

<sup>1</sup> This definition is somewhat less restrictive than the one sketched in the Introduction.

corresponding entries are 1's. Clearly,  $P$  is minimized when each of these rows contains exactly  $N/2$  ones. In that case  $P$  is the tail of a hypergeometric distribution, and by Uhlmann (see [16, Chap. 6, § 5, p. 151]) is bounded below by the corresponding tail of the binomial distribution. That is,

$$P \geq 2^{-N/2} \sum_{i=T}^{N/2} \binom{N/2}{i}.$$

This last expression can be approximated by the normal distribution, and in particular is bounded below by  $1 - \Phi(1) \geq 1 - 0.8413 > \frac{1}{8}$ . By standard probabilistic arguments, this implies that there is a choice of  $N/2$  columns and  $\frac{1}{8} \cdot L/2 = L/16$  rows which have the desired proportion of 1's.  $\square$

**THEOREM 3.** *Let  $b \leq l - 1$ , and  $f: \{0, 1\}^{2^l} \mapsto \{0, 1\}$  be an arbitrary Boolean function. Then there exist a  $\sigma \in \{0, 1\}$  and two independent random variables  $X$  and  $Y$ , such that  $X$  is  $(l, l - 4)$ -distributed  $Y$  is  $(l, b)$ -distributed, and  $\Pr(f(X, Y) = \sigma) > \frac{1}{2}(1 + 2^{-b/2})$ .*

*Proof.* View  $f$  as a  $2^l$ -by- $2^l$  Boolean matrix, with the  $(i, j)$ th entry specifying  $f(i, j)$ . Let  $L = 2^l$  and  $N = 2^{b+1}$ . Applying Lemma 2 to an arbitrary  $L$ -by- $N$  submatrix  $S$ , there exist a  $\sigma$  and a  $2^{l-4}$ -by- $2^b$  submatrix  $S'$  of  $S$  with a fraction  $\frac{1}{2}(1 + 2^{-b/2})$  of  $\sigma$ -entries. Making  $X$  flat on the rows of  $S'$ , and  $Y$  flat on its columns, we get the desired result.  $\square$

The above argument was based on estimating the probability that the number of ones in randomly selected columns is at least one standard deviation away from the mean. One can consider the probability that this number is several standard deviations away from the mean. This yields a bigger bias but fewer rows, and thus a more concentrated  $X$ . Thus, for every constant  $\nu$  and sufficiently large  $b$ , there exists  $\sigma \in \{0, 1\}$  such that  $\Pr(f(X, Y) = \sigma) > \frac{1}{2}(1 + 2^{-(b-\nu)/2})$ .

When  $b$  is very small, the situation is even worse.

**THEOREM 4.** *Let  $b \leq \log_2(1 - \log_2 l) - 1$ , and  $f: \{0, 1\}^{2^l} \mapsto \{0, 1\}$  be an arbitrary Boolean function. Then there exist a  $\sigma \in \{0, 1\}$  and two independent random  $(l, b)$ -distributed variables  $X$  and  $Y$ , such that  $\Pr(f(X, Y) = \sigma) = 1$ .*

The statement here is actually a (slightly weaker) version of a known Ramsey type theorem that every  $n \times n$  Boolean matrix contains a monochromatic  $t \times t$  submatrix, where  $t \approx \log_2 n - \log_2 \log_2 n$ .

*Proof.* Consider (arbitrarily) the first  $r \stackrel{\text{def}}{=} 2^{b+1}$  columns in the  $2^l \times 2^l$  matrix of  $f$ . This defines a  $2^l \times r$  submatrix. There is an  $r$ -bit string which occurs in at least  $2^l/2^r$  rows of the submatrix. Pick these rows. Let  $\sigma \in \{0, 1\}$  be a bit which occurs  $t \geq r/2$  times in each of these rows. Picking the  $t$  columns containing  $\sigma$ , we get a  $2^{l-r} \times t$  submatrix with identical entries  $\sigma$ . As  $2^{l-r} > 2^b$  and  $t \geq 2^b$ , this submatrix corresponds to a pair of  $(l, b)$ -distributed variables.  $\square$

**2.3. Flat distributions.** In this section we introduce the notion of flat distributions. The importance of this notion stems from two facts. First, as we will shortly show, the worst behaviour of extraction functions occurs on flat distributions. Second, as demonstrated through the paper, flat distributions are very easy to deal with.

**DEFINITION 5.** Let  $X$  be a random variable assuming values in  $\{0, 1\}^l$ , and  $S \subset \{0, 1\}^l$ . We say that  $X$  is *equiprobable on  $S$*  if for every  $\alpha, \beta \in S$ ,

$$\Pr(X = \alpha) = \Pr(X = \beta).$$

We say that  $X$  is *flat on  $S$*  if  $X$  is equiprobable on  $S$  and for  $\alpha \notin S$ ,  $\Pr(X = \alpha) = 0$ . We say that  $X$  is  *$(l, b)$ -flat* if  $X$  is  $(l, b)$ -distributed and there exists some  $S$  such that  $X$  is flat on  $S$ .

For simplicity, we assume throughout this section that  $2^b, 2^{b_1}$  and  $2^{b_2}$  are integers. Flat distributions are interesting because the “worst-case behaviour” of a function occurs on them. Namely,

LEMMA 5. For every function  $f: \{0, 1\}^{2l} \mapsto \{0, 1\}^m$  and every  $\alpha \in \{0, 1\}^m$

$$\sup_{\substack{X, Y \text{ are independent} \\ X \text{ is } (l, b_1)\text{-distributed} \\ Y \text{ is } (l, b_2)\text{-distributed}}} \{\Pr(f(X, Y) = \alpha)\} = \max_{\substack{X, Y \text{ are independent} \\ X \text{ is } (l, b_1)\text{-flat} \\ Y \text{ is } (l, b_2)\text{-flat}}} \{\Pr(f(X, Y) = \alpha)\}$$

and

$$\inf_{\substack{X, Y \text{ are independent} \\ X \text{ is } (l, b_1)\text{-distributed} \\ Y \text{ is } (l, b_2)\text{-distributed}}} \{\Pr(f(X, Y) = \alpha)\} = \min_{\substack{X, Y \text{ are independent} \\ X \text{ is } (l, b_1)\text{-flat} \\ Y \text{ is } (l, b_2)\text{-flat}}} \{\Pr(f(X, Y) = \alpha)\}.$$

*Proof.* Denote  $p_i = \Pr(X = i)$  and  $q_j = \Pr(Y = j)$ . Let  $f_\alpha(i, j) = 1$  if  $f(i, j) = \alpha$  and 0 otherwise. Then

$$\begin{aligned} P_\alpha(X, Y) &\stackrel{\text{def}}{=} \Pr(f(X, Y) = \alpha) \\ &= \sum_{i,j} \Pr(X = i, Y = j) \cdot f_\alpha(i, j) \\ &= \sum_{i,j} p_i q_j f_\alpha(i, j). \end{aligned}$$

The last equality follows from the independence of  $X$  and  $Y$ .  $P_\alpha(X, Y)$  is a function of the variables  $p_i, q_j$ , and it attains a global maximum in the range  $0 \leq p_i \leq 2^{-b_1}, 0 \leq q_j \leq 2^{-b_2}, \sum_i p_i = \sum_j q_j = 1$ . We look for a characterization of this global maximum. Fixing the probability distribution of  $X$  (i.e., fixing the  $p_i$ 's),  $P_\alpha(X, Y)$  is a linear program in the  $q_j$ 's, subject to the constraints  $0 \leq q_j \leq 2^{-b_2}$  and  $\sum_j q_j = 1$ . (For basic linear programming terminology consult [22, Chap. 2].) One can verify that every basic feasible solution has exactly  $2^{b_2}$  nonzero variables  $q_j$ , each equals  $2^{-b_2}$ . Thus, we have shown that for every fixed  $X$ , flat distributions are among the distributions which maximize/minimize the value  $P_\alpha(X, Y)$ , over all possible choices of  $(l, b)$ -distributions for  $Y$ . The same obviously holds for fixed  $Y$ . Now let  $X_0, Y_0$  be the pair of  $(l, b_1)$ -distribution and  $(l, b_2)$ -distribution where  $P_\alpha(X, Y)$  attains its maximum. Then both  $X_0$  and  $Y_0$  must be flat. Note that the characterization holds for *any* function  $f$ .  $\square$

We demonstrate the utility of Lemma 5 by using it to argue that the following Boolean function  $f: \{0, 1\}^2 \times \{0, 1\}^2 \mapsto \{0, 1\}$  (tabulated below) is  $\frac{1}{2}$ -robust with respect to all pairs of independent  $(2, 1)$ -distributed variables.

$X \setminus Y$	00	01	10	11
00	0	0	1	1
01	1	0	0	1
10	0	1	0	1
11	1	0	1	0

Using Lemma 5, it suffices to consider the behaviour of the function on all pairs of independent  $(2, 1)$ -flat variables. There are  $6^2 = 36$  possibilities altogether, each corresponding to a  $2 \times 2$  submatrix of this table. It is readily verified that no such

submatrix contain all 1's (or all 0's). Thus, for every pair of independent  $(2, 1)$ -distributed variables,  $X$  and  $Y$ , we have  $\frac{1}{4} \leq \Pr(f(X, Y) = 1) \leq \frac{3}{4}$ .

**2.4. A counting argument.** In this section we show that two independent probability-bounded sources can be used to get almost-unbiased bits. In fact we show that almost all functions can be used for this purpose. To this end, we use the characterization of the distributions on which the “worst-case behaviour” of any function occurs (i.e., Lemma 5), and apply a counting argument to estimate the fraction of functions which are good with respect to all flat distributions.

It is helpful to note that there is a natural correspondence between flat distributions and the set of strings on which they are concentrated. This suggests the following notation: Let  $Z$  be an arbitrary fixed  $(l, b)$ -flat variable. We write  $z \in S_Z$  if  $\Pr(Z = z) = 2^{-b}$ .

In the next lemma we consider two fixed and independent flat variables, and bound below the fraction of functions which are  $\epsilon$ -robust for these two *specific* variables.

LEMMA 6. *Let  $X$  and  $Y$  be two independent distributions, such that  $X$  is  $(l, b_1)$ -flat and  $Y$  is  $(l, b_2)$ -flat, and  $0 < \epsilon < \frac{1}{2}$ . The fraction of functions  $f: \{0, 1\}^{2l} \mapsto \{0, 1\}^m$ , which are  $\epsilon$ -robust on  $X$  and  $Y$ , is at least  $1 - 2^{m - \epsilon 2^{2b_1 + b_2 - m - 2}}$ .*

*Proof.* We say that a function  $f: \{0, 1\}^{2l} \mapsto \{0, 1\}^m$  is  $\epsilon$ -bad on the string  $\alpha$  ( $\alpha \in \{0, 1\}^m$ ) if

$$\frac{|\{(x, y) \in S_X \times S_Y : f(x, y) = \alpha\}|}{2^{b_1 + b_2}} \notin [(1 - \epsilon) \cdot 2^{-m}, (1 + \epsilon) \cdot 2^{-m}].$$

Let  $P_{\alpha, \epsilon}$  denote the fraction of functions which are  $\epsilon$ -bad on the string  $\alpha$ . To study  $P_{\alpha, \epsilon}$  we consider the probability space of the functions in  $f: \{0, 1\}^{2l} \mapsto \{0, 1\}^m$  taken with uniform distribution. For each  $i \in S_X$  and  $j \in S_Y$ , let the random variables  $\zeta_{ij}$  be defined as follows:

$$\zeta_{ij} = \begin{cases} 1 & \text{if } f(i, j) = \alpha, \\ 0 & \text{otherwise.} \end{cases}$$

Then

$$P_{\alpha, \epsilon} = \Pr \left( \frac{1}{2^{b_1 + b_2}} \left| \sum_{i \in S_X, j \in S_Y} \left( \zeta_{ij} - \frac{1}{2^m} \right) \right| \geq \frac{\epsilon}{2^m} \right).$$

Recall the Chernoff Bound [26, Chap. VII, § 4, Thm. 2]: Let  $\zeta_1, \zeta_2, \dots, \zeta_t$  be independent random variables with  $\Pr(\zeta_i = 1) = p$  and  $\Pr(\zeta_i = 0) = 1 - p$ , where  $p \leq \frac{1}{2}$ . Then for all  $0 < \delta \leq p(1 - p)$  we have

$$\Pr \left( \frac{1}{t} \cdot \left| \sum_{i=1}^t (\zeta_i - p) \right| \geq \delta \right) \leq 2 \cdot \exp \left[ - \frac{t\delta^2}{2p(1-p) \left( 1 + \frac{\delta}{2p(1-p)} \right)^2} \right].$$

Letting  $p = 2^{-m}$ ,  $t = 2^{b_1 + b_2}$  and  $\delta = 2^{-m} \cdot \epsilon$ , we get

$$P_{\alpha, \epsilon} < \exp \left[ - \frac{1}{4} \epsilon^2 2^{b_1 + b_2 - m} \right].$$

Switching to base 2 and summing over all possible  $\alpha$ 's, the probability that a function  $f: \{0, 1\}^{2l} \mapsto \{0, 1\}^m$  is  $\epsilon$ -bad on some  $\alpha \in \{0, 1\}^m$  is at most

$$\sum_{\alpha \in \{0, 1\}^m} P_{\alpha, \epsilon} < 2^{m - \epsilon^2 2^{b_1 + b_2 - m - 2}}$$

The lemma follows.  $\square$

We are now ready to prove that almost all functions work.

**THEOREM 7.** *Let  $1 \leq m \leq b$  be an integer, and  $0 < \varepsilon \leq \frac{1}{2}$ . Let  $F$  be the set of functions  $f: \{0, 1\}^{2l} \mapsto \{0, 1\}^m$ .*

(1) *Let  $G_{b,\varepsilon} \subset F$  the set of functions which are  $\varepsilon$ -robust for any two independent  $(l, b)$ -distributed random variables. If  $m + 2 \log_2 \varepsilon^{-1} \leq b - 2 - \log_2(2l + 1)$  then*

$$\frac{|G_{b,\varepsilon}|}{|F|} > 1 - 2^{-2^b}.$$

(2) *Let  $H_{b,\varepsilon} \subset F$  the set of functions which are  $\varepsilon$ -robust for any two independent random variables which are  $(l, b_1)$ -distributed and  $(l, b_2)$ -distributed, respectively, where  $b_1 + b_2 \geq 2b$ . If  $m + 2 \log_2 \varepsilon^{-1} \leq 2b - l - 5$  then*

$$\frac{|H_{b,\varepsilon}|}{|F|} > 1 - 2^{-2^l}.$$

*Proof.* For part (1), by Lemma 5,  $f \in G_{b,\varepsilon}$  if and only if it is  $\varepsilon$ -robust for every two independent  $(l, b)$ -flat random variables. By Lemma 6, the fraction of functions in  $F$  which fail on a particular pair of independent  $(l, b)$ -flat variables is double-exponentially vanishing ( $< 2^{m - \varepsilon^2 2^{2b - m - 2}}$ ). Evidently, the fraction of functions which could fail on some pair of independent  $(l, b)$ -flat variables, is at most the number of pairs of  $(l, b)$ -flat variables times the above fraction. Let  $N_b$  denote the number of  $(l, b)$ -flat variables. Clearly

$$N_b = \binom{2^l}{2^b} < \frac{2^{l \cdot 2^b}}{2^b}.$$

Thus,

$$\frac{|G_{b,\varepsilon}|}{|F|} \geq 1 - N_b^2 \cdot 2^{m - \varepsilon^2 2^{2b - m - 2}} > 1 - 2^{2^b \cdot (2l - \varepsilon^2 2^{2b - m - 2})}.$$

Since  $m + 2 \log_2 \varepsilon^{-1} \leq b - 2 - \log_2(2l + 1)$ , we get  $2l - \varepsilon^2 2^{b - m - 2} \leq -1$ , and (1) follows.

Part (2): For every fixed  $b_1$  and  $b_2$ , the fraction of functions in  $F$  which fail on a particular pair of independent variables which are  $(l, b_1)$ -flat and  $(l, b_2)$ -flat resp., is  $< 2^{m - \varepsilon^2 2^{b_1 + b_2 - m - 2}}$ . We multiply this fraction by  $(2^{2^l})^2 = 2^{2^{l+1}}$ , which is an obvious upper bound on the number of possible pairs of flat variables. We get

$$\frac{|H_{b,\varepsilon}|}{|F|} \geq 1 - 2^{2^{l+1}} \cdot 2^{m - \varepsilon^2 2^{2b - m - 2}} > 1 - 2^{2^{l+1} + m - \varepsilon^2 2^{2b - m - 2}}.$$

Since  $m + 2 \log_2 \varepsilon^{-1} \leq 2b - l - 5$ , we get  $2^{l+1} + m - \varepsilon^2 2^{2b - m - 2} \leq -2^l$ , and (2) follows.  $\square$

There is a trade-off between  $m$ , the number of extracted bits, and  $\varepsilon$ , the robustness of these bits. Some cases of special interest are listed below:

(1) Setting  $m = b - 4 - \log_2(2l + 1)$  and  $\varepsilon = \frac{1}{2}$ , we convert two independent  $(l, b)$ -sources to a single  $(m, m - 1)$ -source. Intuitively, this conversion is very efficient in terms of rate: even if the entropy of the input sources is  $b$  units per each block, we extract a block of  $\approx b$  bits with entropy  $\approx b$ .

(2) Setting  $m = (b - 2 - \log_2(2l + 1))/3$  and  $\varepsilon = 2^{-m}$ , we see that most functions can be used to extract many high-quality bits per each block of the two independent  $(l, b)$ -sources.

(3) Setting  $m = 1$  and  $\varepsilon = 2^{-(b - 3 - \log_2(2l + 1))/2}$ , we see that all but a  $2^{-2^b}$  fraction of the Boolean functions are  $\varepsilon$ -robust with respect to two independent  $(l, b)$ -sources. This bias is almost optimal: Theorem 3 states that no Boolean function can be

$2^{-(b-O(1))/2}$ -robust with respect to such sources. Theorem 4 asserts that “nothing can be extracted” if  $b < \log_2(l - \log_2 l) - 1$ .

(4) Setting  $m = 1$  and  $\epsilon = 2^{-(2b-6-l)/2}$ , we see that all but a  $2^{-2^l}$  fraction of the Boolean functions are  $\epsilon$ -robust for any pair of  $(l, b_1), (l, b_2)$  sources satisfying  $b_1 + b_2 = 2b$ .

**2.5. Hadamard matrices.** In § 2.3 we showed that the bias of a Boolean function  $f: \{0, 1\}^{2^l} \mapsto \{0, 1\}$  with respect to two independent  $(l, b)$ -sources, can be estimated by considering flat distributions only. Viewing  $f$  as a  $2^l \times 2^l$  matrix of  $\pm 1$ , this corresponds to taking all  $2^b \times 2^b$  submatrices of  $f$  (not necessarily consecutive), and estimating the maximum submatrix elements’ sum. While in the previous section we showed that most functions have a small submatrix sum, this section deals with a specific class of functions, whose matrices are Hadamard matrices.

A *Hadamard matrix* is a  $\pm 1$  matrix in which every two distinct rows (columns) are orthogonal (see [14, Chap. 14] and [17, Chap 2, § 3]). Hadamard matrices are a subject of rich literature. In particular it is well known that submatrices of any Hadamard matrix are “balanced.” In order to make the paper self-contained, we present a proof of this fact, following Erdős and Spencer [12, p. 88].

LEMMA 8 (J. H. Lindsey). *Let  $H = (h_{i,j})$  be a  $t \times t$  Hadamard matrix. Then the sum of elements in every  $r \times s$  submatrix of  $H$  is at most  $\sqrt{s} \cdot r \cdot t$ .*

Lindsey’s lemma, as it appears in [12, p. 88], is only a special case of Lemma 8 (although the proof generalizes easily). The lemma as it appears here, with a slightly different proof, appears in [4].

*Proof.* Since orthogonality is preserved under any row and column permutation, it suffices to consider  $|\sum_{i=1}^r \sum_{j=1}^s h_{i,j}|$ , the sum of elements in the leftmost/uppermost  $r \times s$  submatrix. Let  $\vec{h}_i$  denote the  $i$ th row of  $H$ , and

$$\vec{I} \stackrel{\text{def}}{=} (\overbrace{1, 1, \dots, 1}^{s \text{ ones}}, \overbrace{0, 0, \dots, 0}^{t-s \text{ zeros}}).$$

Then by the Cauchy-Schwartz inequality

$$\begin{aligned} \left| \sum_{i=1}^r \sum_{j=1}^s h_{i,j} \right| &= \left| \sum_{i=1}^r \vec{h}_i \cdot \vec{I} \right| \\ &\leq \left\| \sum_{i=1}^r \vec{h}_i \right\|_2 \cdot \|\vec{I}\|_2 \\ &= \left\| \sum_{i=1}^r \vec{h}_i \right\|_2 \cdot \sqrt{s}. \end{aligned}$$

Since the  $\vec{h}_i$  are orthogonal,

$$\left\| \sum_{i=1}^r \vec{h}_i \right\|_2 = \sqrt{\sum_{i=1}^r \|\vec{h}_i\|_2^2} = \sqrt{r \cdot t}$$

and the bound on  $|\sum_{i=1}^r \sum_{j=1}^s h_{i,j}|$  follows.  $\square$

THEOREM 9. *Let  $M$  be an  $2^l \times 2^l$  Hadamard matrix corresponding to the Boolean function  $f$  (i.e.,  $f(i, j) = \frac{1}{2}(1 + M_{i,j})$ ). Suppose  $b_1 + b_2 \geq l + 2 + 2 \log_2 \epsilon^{-1}$ , where  $\epsilon < 1$ . Then the function  $f$  is  $\epsilon$ -robust with respect to any pair of independent random variables  $X, Y$  which are  $(l, b_1)$ -distributed and  $(l, b_2)$ -distributed, respectively.*

*Proof.* By Lemma 5, it suffices to show that every  $2^{b_1} \times 2^{b_2}$  submatrix has a relatively small elements’ sum. Substituting in Lemma 8,  $r = 2^{b_1}$ ,  $s = 2^{b_2}$  and  $t = 2^l$ , the submatrix sum is at most  $(\epsilon/2) \cdot 2^{b_1+b_2}$ .  $\square$



Subsequently Noga Alon proved a more general statement (without using Lemma 5) [3].

*Remarks.* (1) The case where  $b_2 = l - 1$  will be useful in § 5. We get that any Hadamard matrix is  $2^{-(b-3)/2}$ -robust with respect to any pair of independent random variables which are  $(l, l - 1)$ -distributed and  $(l, b)$ -distributed, respectively.

(2) Inner-product modulo 2 corresponds to a special form of Hadamard matrices, known as Sylvester matrices. This provides an alternative proof for Vazirani’s Theorem [29], for the case  $\delta > 1 - \sqrt{1/2} \approx 0.293$  (but not for smaller  $\delta$ ’s).

For inner-product modulo 2, Theorem 9 cannot be significantly improved (with respect to probability bounded sources).

**PROPOSITION 10.** *Let  $b_1 + b_2 \leq l - 4 + 2 \log_2 \varepsilon^{-1}$ , where  $\varepsilon < 1$ . Then the inner-product modulo 2 function is not  $\varepsilon$ -robust on some pair of independent,  $(l, b_1)$ -distributed and  $(l, b_2)$ -distributed variables.*

*Proof.* First, consider the case where  $b_1 + b_2 \leq l$ . Picking  $X$  to be flat on strings of the form  $0^{l-b_1}\{0, 1\}^{b_1}$  and  $Y$  to be flat on  $\{0, 1\}^{b_2}0^{l-b_2}$  the inner product of  $X$  and  $Y$  is identically 0. For the case  $b_1 + b_2 > l$ , repeating exactly the same construction does not yield the desired bias. However, we can modify it using Theorem 3. Let  $\Delta \stackrel{\text{def}}{=} b_1 + b_2 - l$ . Consider the following family of  $(l, b_1)$ -distributed variables  $\mathcal{X}$ . Each variable  $X \in \mathcal{X}$  is the concatenation of three independent variables  $X_1, X_2, X_3$ , where  $X_1$  is uniformly distributed over  $\{0, 1\}^{b_1 - \Delta - 4}$ ,  $X_2$  is  $(\Delta + 8, \Delta + 4)$ -distributed, and  $X_3$  has  $l - b_1 - 4$  bits which are identically 0. Similarly,  $Y = Y_1 Y_2 Y_3 \in \mathcal{Y}$  satisfies  $Y_1$  is identically  $0^{l-b_2-4}$ ,  $Y_2$  is  $(\Delta + 8, \Delta + 4)$ -distributed, and  $Y_3$  is uniformly distributed over  $\{0, 1\}^{b_2 - \Delta - 4}$ . For every pair  $X \in \mathcal{X}$  and  $Y \in \mathcal{Y}$ , the inner product of  $X$  and  $Y$  equals the inner product of  $X_2$  and  $Y_2$ . Since both  $X_2$  and  $Y_2$  could be any  $(\delta + 8, \delta + 4)$ -distributed variables, by Theorem 3 their inner product may have bias  $> \frac{1}{2}(1 + 2^{-(\delta+4)/2}) = \frac{1}{2}(1 + 2^{(l-b_1-b_2-4)/2})$ . Thus, for  $\varepsilon \leq 2^{(l-b_1-b_2-4)/2}$ , the function is not  $\varepsilon$ -robust.  $\square$

The last proposition demonstrates an inherent limitation of the inner-product function with respect to  $(l, b)$ -distributions when  $b \leq l/2$ . This limitation need not be shared by all Hadamard matrices. In fact, a simple construction, known as *the Paley Graph*, is conjectured in the combinatorial folklore to have a stronger imbalance (small submatrix sum) property.

Let  $p$  be a prime, and  $\binom{i}{p}$  be the Legendre symbol of the residue  $i \pmod p$ . The matrix  $M$  with  $M_{i,j} = \binom{i-j}{p}$  is “almost” Hadamard [17, p. 47], as for any  $0 \leq a < b \leq p - 1$ ,

$$\sum_{i=0}^{p-1} \binom{i-a}{p} \cdot \binom{i-b}{p} = -1.$$

Thus, with minor modifications, Theorem 9 applies also to the matrix  $M$ .

**CONJECTURE.** For any constant  $0 < \mu < 1$ , there exists a constant  $1.5\mu < c_\mu < 2\mu$  such that every  $p^\mu \times p^\mu$  submatrix of  $M$  has elements’ sum at most  $p^{c_\mu}$ , for large enough  $p$ .

*Remark.* By Theorem 3, the constant  $c_\mu$  must satisfy  $c_\mu > 1.5\mu$ .

**COROLLARY 11.** *Let  $f(i, j) = \frac{1}{2} \cdot (1 + \binom{i-j}{p})$ . Under the Paley Graph conjecture, the function  $f$  is  $p^{c_\mu - 2\mu}$ -robust with respect to any pair of independent  $(\log_2 p, \mu \cdot \log_2 p)$ -distributed random variables.*

**3. Extracting unbiased bits—Part II.** In this section we further investigate the problem of extracting unbiased bits from probability-bounded sources. In § 3.1, we introduce two efficiency measures: rate and computation complexity and consider extraction schemes, arising from our results, with respect to these measures. In § 3.2, we present and analyze the discrete logarithm extraction scheme (this result did not

appear in the preliminary version of this paper [9]). In § 3.3, we consider extraction from slightly dependent sources. In § 3.4, we further extend the probability-bounded model: we consider sources with lower bound on entropy, and sources with varying block length and probability bound.

**3.1. Efficiency considerations.** Before going any further, let us discuss the significance of the results presented in § 2. A first consequence is that it is possible to generate a sequence of almost-unbiased and independent bits from the output of two independent probability-bounded sources. Once the question of possibility is resolved, we are interested in the efficiency of the extraction schemes. We consider two measures—*rate* and *computational complexity*—both with respect to the desired robustness.

**3.1.1. Efficiency measures.** In § 2, we have considered functions which operate on corresponding  $l$ -bit blocks of two  $(l, b)$ -sources. In order to improve the robustness, we now consider deterministic algorithms (families of functions) which may use several blocks from each  $(l, b)$ -source at a time. Given a “robustness parameter”  $n$ , the algorithm will output bits with bias in the range  $\frac{1}{2} \pm \varepsilon(n)$ . The number of blocks used by the algorithm will typically increase with  $n$ . Of special interest is the case where  $\varepsilon^{-1}(n)$  grows faster than any polynomial. In this case the extracted bits are as good as perfect bits for all “poly( $n$ )-time purposes.”

In the following definition of an extraction scheme,  $\varepsilon(n)$  denotes the robustness of the scheme,  $s(n)$  denotes the number of bits taken from each source,  $e(n)$  denotes the number of extracted bits, and  $c$  denotes the number of  $(l, b)$ -sources used.

**DEFINITION 6.** Let  $\varepsilon$  be a function from integers to the interval  $(0, 1)$ , and  $s, e$  be functions from integers to integers. Let  $c, l$  be integers and  $b$  ( $0 \leq b \leq l$ ) be real. An  $(\varepsilon(\cdot), s(\cdot), e(\cdot), l, b, c)$ -*extraction scheme* is a family of functions  $\{f_n\}$  such that for every  $n$  the following holds:

- (1) For every  $\alpha_1, \alpha_2, \dots, \alpha_c \in \{0, 1\}^{s(n)}$   $f_n(\alpha_1, \alpha_2, \dots, \alpha_c) \in \{0, 1\}^{e(n)}$ .
- (2) The function  $f_n: \{0, 1\}^{c \cdot s(n)} \rightarrow \{0, 1\}^{e(n)}$  is  $\varepsilon(n)$ -robust with respect to any  $c$  independent variables  $X_1, X_2, \dots, X_c$ , where each of the  $X_i$ 's is the first  $s(n)$  bits output by some  $(l, b)$ -source.

The efficiency of an algorithmic scheme should be evaluated with respect to the resources it uses. In the setting of randomness extraction schemes the resources to be considered are the amount of “randomness” in the input sources and the deterministic computation required to effect the extraction. We measure the efficiency with respect to the randomness resource by the ratio of the entropy entering the extraction scheme and the entropy leaving it.

**DEFINITION 7.** The *rate* of an  $(\varepsilon(\cdot), s(\cdot), e(\cdot), l, b, c)$ -extraction scheme is defined as

$$r(n) \stackrel{\text{def}}{=} \frac{e(n) \cdot H_O}{c \cdot s(n) \cdot H_I},$$

where  $H_O$  is the entropy of each output bit ( $H_O \approx 1 - \varepsilon^2(n)$ ), and  $H_I = b/l$  is a lower bound on the average entropy of each input bit. If there exists a constant  $r > 0$  such that for every  $n$ ,  $r(n) > r$ , then we say that the extraction scheme has *constant rate*.

For every  $n$ , the rate  $r(n)$  is the ratio of the input and output entropies to  $f_n$ . The entropy of the input is taken by the worst possible one since the extraction scheme cannot “adapt” to better sources without an explicit guarantee.

**DEFINITION 8.** The *computational complexity* of an extraction scheme,  $\{f_n\}$ , is defined as the complexity of a family of circuits  $\{C_n\}$  such that for every  $n$ , the circuit  $C_n$  implements  $f_n$ .

**3.1.2. Efficient extraction schemes.** We now present several extraction schemes, which follow immediately from the results presented in § 2, and analyze their performance with respect to the above efficiency measures. The following is a simple but important observation used in developing these schemes: for every integer  $q > 1$ , an  $(l, b)$ -source is also a  $(q \cdot l, q \cdot b)$ -source.

*A rate efficient scheme which is not computationally efficient.* One consequence of Theorem 7 is that for every  $l$  and  $b$  and every desired bias  $\varepsilon(\cdot)$ , there is a nonuniform circuit family  $\{C_n\}$  which extracts bits at constant rate from any two independent  $(l, b)$ -sources. This is obtained by letting  $s(n) = (3l/b) \log_2 \varepsilon^{-1}(n)$ ,  $e(n) = \log_2 \varepsilon^{-1}(n)$ , and using Theorem 7 with respect to two  $(s(n), (b/l) \cdot s(n))$ -sources. Since the entropy per input block is at least  $b$ , the rate is  $\approx 1/6$ . The size of  $C_n$  is  $\varepsilon^{-6l/b}(n) \cdot \log_2 \varepsilon^{-1}(n)$ .

*Computationally efficient schemes which do not have constant rate.* By Theorem 9, for every  $l$  and  $b > l/2$  and every desired bias  $\varepsilon(\cdot)$ , taking the binary inner product of  $l \cdot (b - l/2)^{-1} \log_2 \varepsilon^{-1}(n)$  bits from two independent  $(l, b)$ -sources, a single  $\varepsilon(n)$ -robust bit is extracted. While this yields an efficient algorithm, its rate is  $1/\theta(\log \varepsilon^{-1}(n))$ .

Under the number theoretic conjecture of § 2.5, efficient algorithms exist for *any*  $l$  and  $b$ . For every desired bias  $\varepsilon(\cdot)$ , let  $p > \varepsilon^{-C(l/b)}(n)$  be a prime (where  $C(\rho) > 0$  is a constant depending on  $\rho$ ). Taking  $\log_2 p$  bits from each of the two independent  $(l, b)$ -sources, and computing the Legendre symbol of their integer difference modulo  $p$ , we get an  $\varepsilon(n)$ -robust bit. The extracted bit can be computed by an algorithm running in time polynomial in  $\log_2 \varepsilon^{-1}(n)$  (and  $l/b$ ).

A direct consequence of Theorem 7 is that for every  $l$  and  $b > 5 + \log_2 l$  there exists a table of size  $2^{2l}$  which transforms two independent  $(l, b)$ -sources into one  $(m, m - 2^{-m})$ -source, where  $m = (b - 3 - \log_2 l)/3$ . In other words, we can transform two independent but very weak sources into one source which is quite good (although it is far from being “almost perfect”). For every bias  $\varepsilon(n)$ , using the inner-product function on the output of the virtual  $(m, m - 2^{-m})$ -source and a third independent  $(l, b)$ -source, we get the desired bias. We conclude that for every  $0 < b \leq l$  and  $\varepsilon(\cdot)$ , there is a fast algorithm (running in time  $O(\log_2 \varepsilon^{-1}(n))$ ) that on input  $n$  and access to three independent  $(l, b)$ -sources, generates  $\varepsilon(n)$ -robust bits.

The problem of finding an extraction scheme which combines both rate and computational efficiency was left open in our preliminary report [9]. This was true even for the SV-model. In the following section we present a solution to that problem.

**3.2. An extraction scheme efficient in both measures.** In this section we present an extraction scheme based on  $k$ th power residues modulo a prime. The scheme is efficient both in terms of information rate and computation complexity. This scheme is a generalization of the Paley graph construction, and was developed through conversations with László Babai. We begin this section by presenting the scheme. We then use results of Ajtai, Babai, Hajnal, Komlós, Pudlák, Rödl, Szemerédi and Turán [2] to show that the scheme has high robustness. To guarantee high information rate, our scheme uses large values of  $k$ , and is related to computing (partial) discrete logarithms in  $Z_p$ . We investigate the conditions under which the scheme is efficiently computable, and show that primes satisfying these conditions can be precomputed in expected polynomial time, given access to two probability-bounded sources.

**DEFINITION 9.** Let  $p$  be a prime,  $g$  a primitive element of  $Z_p$ , and  $k > 1$  an integer dividing  $p - 1$ . We define  $f_k: Z_p \times Z_p \mapsto \{0, 1, \dots, k - 1\}$  by  $f_k(x, y) = (\log_g(x - y)) \bmod k$ , where  $\log_g^t$  is the discrete logarithm of  $t$  to base  $g$  in  $Z_p$ .

*Comments.* (1) For  $\alpha \in \{0, 1, \dots, k - 1\}$ , let  $R_\alpha = \{g^{\alpha+ik}: 0 \leq i \leq (p - 1)/k\}$ . Then  $f_k(x, y) = \alpha$ , if and only if  $x - y = z$  for some  $z \in R_\alpha$ .

(2) By restricting the function to a subset of  $Z_p$ , the function  $f_k$  can be viewed as a function from  $\{0, 1\}^l \times \{0, 1\}^l$  to  $\{0, 1, \dots, k-1\}$ , for  $l = \lfloor \log_2 p \rfloor$ .

(3) The range of  $f_k$  is  $\{0, 1, \dots, k-1\}$ . Taking  $m = \lfloor \log_2 k \rfloor$ , the range of  $f_k$  can be viewed as  $\{0, 1\}^m \cup \{\perp\}$  (in case of  $\perp$ , the function is undefined). This causes at most a factor 2 loss in entropy.

To evaluate the robustness of  $f_k$ , we would like to have upper and lower bounds on  $\Pr(f(X, Y) = \alpha)$  for all pairs of independent  $(l, b_1)$ -distributed,  $(l, b_2)$ -distributed sources  $X, Y$ . By Lemma 5, it suffices to consider flat distributions. Therefore, we are interested in bounds on the number of solutions  $x - y = z$  for  $x \in A, y \in B, z \in R_\alpha$ , where  $A, B \subset Z_p$  are arbitrary subsets of size  $2^{b_1}, 2^{b_2}$ , respectively. Let  $\nu(A, B, \alpha)$  denote this number. Clearly,  $\nu(A, B, \alpha) = \nu(g^{-\alpha}A, g^{-\alpha}B, 0)$ , and thus it suffices to consider  $R_0$ , the set of  $k$ th residues modulo  $p$ .

Let  $\omega \in C$  ( $C$  denotes the complex field) be a primitive  $p$ th root of unity. Let  $\varphi_k(j) = \sum_{x \in R_0} \omega^{jx}$ , and  $\Phi_k = \max_{1 \leq j \leq p-1} |\varphi_k(j)|$ . A result of [2] relates  $\nu(A, B, \alpha)$  to the size of  $A, B$  via  $\Phi_k$ .

LEMMA 12 [2]. *Let  $A, B \subset Z_p$  be two arbitrary subsets, and  $\alpha$  an integer,  $0 \leq \alpha \leq (p-1)/k$ . Then*

$$\left| \nu(A, B, \alpha) - \frac{|A| \cdot |B|}{k} \right| \leq \Phi_k \sqrt{|A| \cdot |B|}.$$

The following bound on  $\Phi_k$  was given by László Babai (private communication).

LEMMA 13.  $\Phi_k < \sqrt{p}$ .

*Proof.* The proof uses methods of trigonometric sums over finite fields (see [28]).

We start by presenting some definitions and notation. An additive character of  $Z_p$  is a mapping  $\psi: Z_p \mapsto C$  satisfying  $\psi(a+b) = \psi(a) \cdot \psi(b)$ ; the unit character satisfies  $\psi_0(\cdot) = 0$ . A multiplicative character of  $Z_p$  is a mapping  $\chi: Z_p^* \mapsto C^*$  satisfying  $\chi(a \cdot b) = \chi(a) \cdot \chi(b)$ ; the unit character satisfies  $\chi_0(\cdot) = 1$  ( $\chi(0) \stackrel{\text{def}}{=} 0$  unless  $\chi = \chi_0$ ). A Gaussian sum  $S(\chi, \psi)$  is defined as  $\sum_{x \in Z_p} \chi(x)\psi(x)$ . It is well known [28, p. 47, Thm. 3A] that for  $\chi \neq \chi_0, \psi \neq \psi_0, |S(\chi, \psi)| = \sqrt{p}$ , while  $S(\chi_0, \psi) = 0$ .

Let  $\xi \in C$  be a primitive  $k$ th root of unity. For  $0 \leq t \leq k-1$ , define  $\chi_t: Z_p^* \mapsto C^*$  by  $\chi_t(x) = \xi^{t \log_g x}$ , then  $\chi_t$  is a multiplicative character,  $\chi_0$  is indeed the unit character and

- (1)  $x \in R_0 \Rightarrow \log_g x = ki \Rightarrow \chi_t(x) = 1,$
- (2)  $0 \neq x \notin R_0 \Rightarrow \log_g x = ki + \alpha \quad \text{for some } 1 \leq \alpha \leq k-1$   
 $\Rightarrow \sum_{t=0}^{k-1} \chi_t(x) = \sum_{t=0}^{k-1} (\xi^\alpha)^t = 0,$
- (3)  $x = 0 \Rightarrow \sum_{t=0}^{k-1} \chi_t(x) = 1.$

For  $1 \leq j \leq p-1$ , define  $\psi_j: Z_p \mapsto C$  by  $\psi_j(x) = \omega^{jx}$ , then  $\psi_j$  is an additive character  $\neq \psi_0$ . Using (1), (2) and (3), we get (for all  $j$ )

$$\begin{aligned} \sum_{t=0}^{k-1} S(\chi_t, \psi_j) &= \sum_{t=0}^{k-1} \sum_{x \in Z_p} \chi_t(x)\psi_j(x) \\ &= \sum_{x \in R_0} \omega^{jx} \sum_{t=0}^{k-1} \chi_t(x) + \sum_{0 \neq x \notin R_0} \omega^{jx} \sum_{t=0}^{k-1} \chi_t(x) + \sum_{t=0}^{k-1} \chi_t(0) \\ &= k \sum_{x \in R_0} \omega^{jx} + 1. \end{aligned}$$

Using the above-mentioned equalities for Gaussian sums, the last equality implies that for every  $1 \leq j \leq p-1$

$$\begin{aligned} |\varphi_k(j)| &= \left| \sum_{x \in \mathbb{R}_0} \omega^{jx} \right| = \frac{1}{k} \left| -1 + \sum_{t=0}^{k-1} S(\chi_t, \psi_j) \right| \\ &\leq \frac{1}{k} \left( 1 + \sum_{t=0}^{k-1} |S(\chi_t, \psi_j)| \right) \\ &= \frac{1}{k} (1 + 0 + (k-1)\sqrt{p}) \\ &< \sqrt{p}. \end{aligned} \quad \square$$

Reformulating these bounds in terms of sources and robustness, we get Theorem 14.

**THEOREM 14.** *Let  $p(n)$ ,  $k(n)$ ,  $l(n) = \lfloor \log_2 p(n) \rfloor$ ,  $f_{k(n)}$  be as above, and  $\varepsilon(n) > 0$ . Suppose  $b_1(n) + b_2(n) \geq l(n) + 1 + 2 \log_2 \varepsilon^{-1}(n) + 2 \log_2 k(n)$ . Then  $f_{k(n)}: Z_{p(n)} \times Z_{p(n)} \mapsto \{0, 1, \dots, k(n)-1\}$  is  $\varepsilon(n)$ -robust for any pair of independent,  $(l(n), b_1(n))$ -distributed,  $(l(n), b_2(n))$ -distributed sources.*

*Proof.* It suffices to consider flat sources  $X, Y$ . Let  $A \subset Z_{p(n)}$  be the set where  $\Pr(X = a) = 1/2^{b_1(n)}$  (similarly for  $B, Y, b_2(n)$ ). Then  $|A| = 2^{b_1(n)}$ ,  $|B| = 2^{b_2(n)}$ , and for every  $\alpha \in \{0, 1, \dots, k(n)-1\}$  we have

$$\Pr(f(X, Y) = \alpha) = \frac{\nu(A, B, \alpha)}{2^{b_1(n)} \cdot 2^{b_2(n)}}.$$

Combining Lemmas 12 and 13, we have

$$\frac{1}{k(n)} \left( 1 - k(n) \sqrt{\frac{p(n)}{2^{b_1(n)+b_2(n)}}} \right) < \Pr(f(X, Y) = \alpha) < \frac{1}{k(n)} \left( 1 + k(n) \sqrt{\frac{p(n)}{2^{b_1(n)+b_2(n)}}} \right).$$

Substituting  $p(n) < 2^{l(n)}$ ,  $b_1(n) + b_2(n) \geq l(n) + 1 + 2 \log_2 \varepsilon^{-1}(n) + 2 \log_2 k(n)$ , the claim follows.  $\square$

We now establish some relations between the various quantities above. We denote by  $n$  the security parameter, and parameterize by it the block length, the probability bounds, the prime  $p$ , the divisor of  $p-1$ ,  $k$  and the bias guarantee  $\varepsilon$  (that is, they will be denoted by  $l(n), b_1(n), b_2(n), p(n), k(n)$  and  $\varepsilon(n)$ , respectively). Typically,  $\varepsilon(n) = 1/n^{h(n)}$  where  $h(n) \geq 0$  is either a constant (the case of a polynomial bias) or a function tending to  $\infty$  with  $n \rightarrow \infty$  (the case of a subpolynomial bias).

The information rate of the scheme is  $\approx \log k(n)/2 \log p(n)$ . Therefore, in order to guarantee a constant rate,  $k(n)$  must be  $> p(n)^d$  for some constant  $d, 0 < d < 1$ . We will typically use  $\frac{3}{16} \leq d \leq \frac{1}{4}$ . Assuming  $b_1(n) + b_2(n) \geq 1.75l(n) + 1$  (an assumption we will later justify), and substituting  $\log_2 \varepsilon^{-1}(n) = h(n) \log_2 n$ ,  $\log_2 k(n) = d \cdot l(n)$  in the equality of the last theorem, we see that  $l(n) \geq h(n)/(3/8 - d) \log_2 n$  is a necessary condition for the scheme to produce  $\varepsilon(n) = n^{-h(n)}$ -robust bits. The case of equality in the last equation implies that  $\varepsilon^{-1}(n) \leq k(n) \leq \varepsilon^{-2}(n)$  for  $\frac{3}{16} \leq d \leq \frac{1}{4}$ , an expression which relates the size of  $k(n)$  to the robustness of bits produced by  $f_{k(n)}$ .

The computation complexity of the scheme equals the (deterministic) complexity of finding discrete logarithms modulo  $k(n)$  in the field  $Z_{p(n)}$ . We first assume that  $p(n), k(n)$  and  $g$ , a primitive element of  $Z_{p(n)}$ , are given, and analyze the run time of the scheme. We then turn to the complexity of the preprocessing stage, in which  $p(n), k(n)$  and  $g$  are produced.

Given  $p(n), k(n)$  and  $g$ , a primitive element of  $Z_{p(n)}$ , then by essentially trying all possible candidates, we can compute  $\log_g(\cdot) \bmod k(n)$  in time  $O(k(n) \log^3(p(n)))$

(see [19]). Thus, if the bias  $\epsilon(n)$  is required to be just polynomial in  $n$  ( $\epsilon(n) = n^{-O(1)}$ ), then by employing our scheme with brute-force discrete logarithm subroutine, the computational complexity is polynomial in  $n$ .

In order to generate subpolynomially biased bits  $\epsilon(n) = n^{-h(n)}$ , with  $h(n) \rightarrow \infty$ , we need more efficient ways of computing discrete logarithms (modulo  $k(n)$ ) in  $Z_{p(n)}$ . There are known algorithms with complexity  $2^{O(\sqrt{\log p(n)} \log \log p(n))}$ , and this run-time would suit our needs, but unfortunately these algorithms are randomized, so we cannot use them to (deterministically) evaluate  $f_{k(n)}$ . Instead, we look for  $p(n)$ 's with smooth  $k(n)$ , and use an algorithm of Pohlig and Hellman [24] which is sufficiently fast in such circumstances.

A positive integer  $z$  is called  $y$ -smooth if all its prime factors are  $\leq y$ . Suppose  $p(n)$  is a prime in the range  $n^{\sqrt{\log \log n}} \leq p(n) \leq n^{2\sqrt{\log \log n}}$ , so that  $k(n) \mid (p(n) - 1)$  is  $n$ -smooth. Given a primitive element  $g$  of  $Z_{p(n)}$ , the Pohlig and Hellman algorithm [24] finds  $\log_g(\cdot) \pmod{k(n)}$  in  $O(n \log^3 n)$  deterministic time.

Given  $l(n)$ , every pair of  $(l, b_1)$ -distributed,  $(l, b_2)$ -distributed sources can be viewed as  $(l(n), l(n) \cdot b_1/l)$ ,  $(l(n), l(n) \cdot b_2/l)$  sources, respectively. Using "nice" primes as above, we have the following theorem.

**THEOREM 15.** *Suppose  $p(n)$  is a prime in the range  $n^{\sqrt{\log \log n}} \leq p(n) \leq n^{2\sqrt{\log \log n}}$ , so that  $k(n) \mid p(n) - 1$ ,  $p(n)^{3/16} \leq k(n) \leq p(n)^{1/4}$ , and  $k(n)$  is  $n$ -smooth. Let  $g \in Z_{p(n)}$  be a given primitive element. Furthermore, let  $l, b_1, b_2$  satisfy  $b_1 + b_2 \geq 1.75l$ . Then for any pair of independent  $(l, b_1)$ -distributed and  $(l, b_2)$ -distributed sources, the function  $f_{k(n)}: Z_{p(n)} \times Z_{p(n)} \mapsto \{0, 1, \dots, k(n) - 1\}$  produces  $1/n^{\sqrt{\log \log n}/8}$ -robust bits, with information rate at least  $\frac{3}{32}$  and computational complexity  $O(n \log^3 n)$ .*

We now turn to the preprocessing stage, starting with the question of finding appropriate primes. Let  $\Psi(x, y)$  denote the number of positive integers not exceeding  $x$  which are  $y$ -smooth. Canfield, Erdős and Pomerance [8] proved the following theorem concerning  $\Psi(x, y)$ :

**THEOREM 16** [8]. *For  $y \geq \log^2 y$ ,  $\Psi(x, y) = xu^{-u+o(u)}$ , where  $u = \log x / \log y$ .*

In particular, for large enough  $x$ ,  $\Psi(x, y) \geq xu^{-2u}$ . Choosing  $h(n) = 2\sqrt{\log \log n}$ ,  $y = n$  and  $x = n^{h(n)}$ , we have that  $u = \log x / \log y = h(n)$ . Substituting these quantities in the last theorem, we conclude that for large enough  $n$ , the probability that a randomly chosen  $z \leq n^{h(n)}$  will be  $n$ -smooth is bounded below by  $h(n)^{-2h(n)} = 1/(2\sqrt{\log \log n})^{4\sqrt{\log \log n}} > \log^{-1/3} n$ . Obviously, the same lower bound holds for the probability that a random  $z$  has an  $n$ -smooth divisor  $d$  such that  $z^{3/16} < d < z^{1/4}$ . However, for our purposes it is not enough for  $z$  to have such divisor, but  $z + 1$  must be a prime as well. Carl Pomerance (private communication) has provided us with an estimate of the probability of this event.

**THEOREM 17.** *For a randomly chosen  $n^{\sqrt{\log \log n}} \leq z \leq n^{2\sqrt{\log \log n}}$ ,*

$$\Pr(z \text{ has an } n\text{-smooth divisor in the range } [z^{3/16}, z^{1/4}] \text{ and } z + 1 \text{ is prime}) \geq \frac{1}{\log^2 n}.$$

By choosing random integers in the above range, an appropriate prime  $p(n)$  with a large  $n$ -smooth divisor  $k(n)$  and a generator for  $Z_p$  can be found in expected polynomial time, given access to an unbiased independent coin. This is done as follows. First, we choose  $p(n)$  at random, factor  $p(n) - 1$  and look for a sufficiently large  $n$ -smooth divisor  $k(n)$ . For factoring  $p(n) - 1$ , we use Dixon's algorithm [10], that runs in expected time  $2^{O(\sqrt{\log p(n)} \log \log n)}$  (which is polynomial in  $n$ ). Next, we verify that  $p(n)$  is a prime, using Pratt's algorithm [25] (again using Dixon's algorithm as a factoring subroutine, and trying to find primitive elements by choosing elements at random). In case  $p(n)$  is indeed a prime, Pratt's algorithm yields a primitive element

of  $Z_{p(n)}$ . We now substitute the unbiased independent coin used in the preprocessing, by bits extracted from two probability-bounded sources, using any of the computationally efficient (but not necessarily rate-efficient) schemes of § 3.1.

Finally, in order to satisfy  $b_1 + b_2 > 1.75l$ , we start with *four* independent  $(l_0, b_0)$ -distributed sources (where the ratio between the constants  $b_0$  and  $l_0$  can be arbitrarily small). Using Theorem 7 and the observation that (for every integer  $q > 1$ ) an  $(l, b)$ -source is also a  $(q \cdot l, q \cdot b)$ -source, we convert every pair of these sources into a single  $(l, 0.9l)$ -distributed source (where  $l = 2l_0(l_0/b_0) \log_2(l_0/b_0)$ ). This conversion is rate efficient, and its complexity does not depend on  $\varepsilon(n)$ .

**3.3. Slightly dependent sources.** In § 2, we showed how to extract unbiased bits from the output of two independent probability-bounded sources. A natural question is whether the independence requirement can be relaxed, and if so, to what extent. We suggest the following definition and investigate its ramifications.

**DEFINITION 10.** Let  $\delta \geq 0$ . We say that two variables  $X$  and  $Y$  are  $\delta$ -dependent if, for every  $\alpha, \beta \in \{0, 1\}^l$  with  $\Pr(X = \alpha) \cdot \Pr(Y = \beta) \neq 0$  the following holds:

$$(1 + \delta)^{-1} \leq \frac{\Pr(X = \alpha \text{ and } Y = \beta)}{\Pr(X = \alpha) \cdot \Pr(Y = \beta)} \leq (1 + \delta).$$

Thus, 0-dependence identifies with independence. Also, notice that this is a more refined measure of dependency than correlation. A different definition of slightly dependent SV-sources was presented in [29], and does not seem to extend to PRB-sources. The following lemma can be easily verified.

**LEMMA 18.** *Suppose that  $f$  is  $\varepsilon$ -robust for any two independent variables satisfying properties  $P_1$  and  $P_2$ , respectively. Then  $f$  is  $(\delta + (1 + \delta)\varepsilon)$ -robust for any two  $\delta$ -dependent variables satisfying properties  $P_1$  and  $P_2$ , respectively.*

Applications to extracting unbiased bits from slightly dependent functions follow immediately by combining Lemma 18 with Theorems 7, 9 and 15. Lemma 18 may seem weak at first glance. It only guarantees that, for small  $\delta$ , the added bias introduced by the  $\delta$ -dependency does not exceed  $\delta$ . However, this result is almost optimal! We will show that  $\delta$ -dependency may add an  $\Omega(\delta)$  term to the bias.

**THEOREM 19.** *Let  $0 < \delta \leq 4$  and  $f: \{0, 1\}^{2l} \mapsto \{0, 1\}$  be an arbitrary Boolean function. Then at least one of the following two conclusions holds:*

(1) *There exist a  $\sigma \in \{0, 1\}$  and a pair of  $\delta$ -dependent  $(l, l - 2)$ -distributed variables  $X$  and  $Y$  such that  $\Pr(f(X, Y) = \sigma) \geq \frac{1}{2} \cdot (1 + \delta/64)$ .*

(2) *There exist a  $\sigma \in \{0, 1\}$  and a pair of independent  $(l, l - 7 - \log_2 \delta^{-1})$ -distributed variables  $X$  and  $Y$  such that  $\Pr(f(X, Y) = \sigma) \leq \frac{1}{3}$ .*

*Proof.* Without loss of generality, we assume that  $|\{(i, j) \in \{0, 1\}^{2l} : f(i, j) = 1\}| \geq \frac{1}{2} \cdot 2^{2l}$ . The function  $f$  is represented as a bipartite graph  $G(V, E)$ , where  $V = A \cup B$  ( $A = \{a_i : i \in \{0, 1\}^l\}$  and  $B = \{b_j : j \in \{0, 1\}^l\}$ ) is the bipartition and the edge set  $E \subset A \times B$  satisfies

$$(a_i, b_j) \in E \text{ iff } f(i, j) = 1.$$

Throughout the rest of the proof, we assume that conclusion (2) of the theorem does not hold: namely, we assume that the function  $f$  is  $\frac{1}{3}$ -robust with respect to any two independent  $(l, l - 7 - \log_2 \delta^{-1})$ -distributed variables. The idea of the proof is to show the existence of a large regular subgraph with relatively high degree. Once this subgraph is found, the variables are defined to be flat on the vertex sets of the subgraph, and the dependency allowance is used to make the edges of the subgraph “heavy.” Thus, the probability mass is concentrated more on entries on which the function has value

1, and the function is biased towards 1 as required in conclusion (1) of the theorem. The following lemma guarantees the existence of such a subgraph.

LEMMA 20. *Let  $0 < \delta \leq 4$  and  $G((A, B), E)$  be a bipartite graph with  $n$  vertices on each side and at least  $\frac{1}{2}n^2$  edges. Suppose that the number of edges in every subgraph of  $G$ , with  $q \stackrel{\text{def}}{=} \delta/128 \cdot n$  vertices on each side, is in the range  $[q^2/3, 2q^2/3]$ . Then there exist  $m \geq n/4$  and  $k = m/32$  such that  $G$  contains a  $k$ -regular subgraph with  $m$  vertices on each side.*

The proof of Lemma 20 is given in the Appendix. A similar statement was attributed by the referee to Pyber and Rödel (as yet unpublished). They showed that every graph with  $n$  vertices and  $\Omega(n^2)$  edges contains an  $\Omega(n)$ -regular subgraph.

Let  $G_0((A_0, B_0), R)$  be the regular subgraph guaranteed by Lemma 20. We now use this regular subgraph to present a pair of  $\delta$ -dependent probability-bounded sources,  $X$  and  $Y$ , which make the function bias.  $X$  will be flat on  $A_0$  and  $Y$  will be flat on  $B_0$ ; that is,

$$\Pr(X = i) \stackrel{\text{def}}{=} \begin{cases} |A_0|^{-1} & \text{if } a_i \in A_0, \\ 0 & \text{otherwise,} \end{cases}$$

$$\Pr(Y = j) \stackrel{\text{def}}{=} \begin{cases} |B_0|^{-1} & \text{if } b_j \in B_0, \\ 0 & \text{otherwise.} \end{cases}$$

Now consider two cases. If the number of edges in the subgraph induced by  $A_0$  and  $B_0$  is smaller than  $\frac{1}{2}(1 - \delta/64) \cdot q^2$  then we let  $X$  and  $Y$  be independent and get the desired bias (towards 0). If the number of edges in the induced subgraph is  $\geq \frac{1}{2}(1 - \delta/64) \cdot q^2$  then we use the dependency allowance as follows:

$$\Pr(X = i, Y = j) \stackrel{\text{def}}{=} \begin{cases} (1 + \delta) \cdot \Pr(X = i) \cdot \Pr(Y = j) & \text{if } (a_i, b_j) \in R, \\ (1 - \delta/31) \cdot \Pr(X = i) \cdot \Pr(Y = j) & \text{otherwise.} \end{cases}$$

The reader may verify the validity of the above definition, by noting that  $1/32 \cdot (1 + \delta) + \frac{31}{32} \cdot (1 - \delta/31) = 1$ . It follows that

$$\Pr(f(X, Y) = 1) \geq \frac{1}{32} \cdot (1 + \delta) + \left(\frac{15}{32} - \frac{\delta}{128}\right) \cdot \left(1 - \frac{\delta}{31}\right)$$

$$> \frac{1}{2} + \frac{\delta}{128}.$$

The theorem follows.  $\square$

**3.4. Variations: entropy, varying length.** We conclude our investigation into the problem of extracting unbiased bits from weak sources of randomness by two remarks. The first remark concerns the probability bound  $b$ , while the second remark concerns both  $b$  and  $l$ .

We have defined probability-bounded variables as having an upper bound ( $2^{-b}$ ) on the probability for each individual  $l$ -bit string. A more “natural” but less convenient definition considers variables with a lower bound on the (information-theoretic) entropy. Every  $(l, b)$ -variable has entropy  $\geq b$ , but the converse does not hold. Nevertheless, every source which has nonzero entropy is in fact a probability-bounded source (a quantitative statement is omitted).

So far, we have considered sources where for some fixed integer  $l$ , given the history, the next  $l$  bits have a particular distribution. A natural question, raised by Jeff Lagarias, is what happens when the number of next bits is a variable. More precisely, let  $l(\cdot)$  and  $b(\cdot)$  be functions, and consider a source  $S$  with the following output distribution: for every integer  $n > 0$ , for every  $\alpha \in \{0, 1\}^n$  and every  $\beta \in \{0, 1\}^{l(n)}$ , the conditional probability that the next  $l(n)$  bits output by  $S$  equal  $\beta$  given that the first  $n$  bit output by  $S$  equals  $\alpha$  does not exceed  $2^{-b(n)}$ . We call this a varying probability-



bounded source (VPRB-source). Extending Theorems 7 and 4, we get an almost sharp threshold for the value of  $b(\cdot)$  which allows the extraction of almost-unbiased bits from two VPRB-sources. Such an extraction is possible whenever  $b(n) > 4 + \log_2 l(n)$ , and is impossible whenever  $b(n) < \log_2(l(n) - \log_2 l(n)) - 1$ .

**4. Communication complexity.** In this section, we present results concerning probabilistic communication complexity. In § 4.1, we recall the common definitions of communication complexity, present new definitions and compare them. In § 4.2, we prove lower bounds on the communication complexity of the functions considered in § 2. In § 4.3, we demonstrate the tightness of our results by presenting upper bounds on the communication complexity of all functions. In § 4.4, we suggest a robust notion of communication complexity and extend our lower bounds to it.

Consider two interactive parties  $A$  and  $B$ , such that  $A$  knows an input  $x \in \{0, 1\}^n$  and  $B$  knows an input  $y \in \{0, 1\}^n$ . The inputs are randomly and independently chosen, each with uniform probability distribution. Let  $f: \{0, 1\}^n \times \{0, 1\}^n \mapsto \{0, 1\}$  be a function and assume that  $A$  and  $B$  wish to compute  $f(x, y)$ . To this end they use a possibly *randomized* protocol  $P$ . As commonly assumed, the messages sent at each round are prefix-free. The protocol is terminated by party  $A$  and the last bit  $B$  sent to  $A$  is their *joint guess of the value of  $f(x, y)$* . A natural question is how many bits should be exchanged among the party so that their joint guess is significantly better than the a priori guess. The answer depends on the exact definitions of the notions “number of bits” and “success probability.”

**4.1. Definitions.** Let  $x, y \in \{0, 1\}^n$ . We consider the probability space defined by the coin tosses of the parties  $A$  and  $B$ . Let  $l_P(x, y)$  be the random variable denoting the number of bits  $A$  and  $B$  exchange on the pair  $(x, y)$ , using the protocol  $P$ . Let  $L_P(x, y)$  denote the expected value of  $l_P(x, y)$  and  $l_P^*(x, y)$  denote the supremum of  $l_P(x, y)$ . Let  $s_{P,f}(x, y)$  be the random variable denoting the success of  $P$  with respect to  $f$  on the pair  $(x, y)$ ; and let  $S_{P,f}(x, y)$  denote the expected value of  $s_{P,f}(x, y)$ . That is,  $S_{P,f}(x, y)$  is the probability that the last bit exchanged by  $A$  and  $B$  on inputs  $x$  and  $y$  equals  $f(x, y)$ .

The *average operator* (denoted *Ave*), and the *minimum* and *maximum operators* (denoted *Min* and *Max*, respectively) are defined in the obvious manner. These operators are used in defining the various measures. For example,  $Ave(L_P) = 2^{-2n} \sum_{x,y \in \{0,1\}^n} L_P(x, y)$  is the average number of bits exchanged in the protocol  $P$ ;  $Max(L_P)$  is the expected number of bits on the worst pair of inputs; and  $Max(l_P^*) = \max_{x,y \in \{0,1\}^n} \{l_P^*(x, y)\}$  is the maximum number of bits taken over all possible executions and pairs. Two measures for success are  $Min(S_{P,f})$  (worst pair) and  $Ave(S_{P,f})$  (averaged over all pairs).

*Previous definitions.* Various definitions of randomized communication complexity have appeared. We present some of them<sup>2</sup> (other definitions can be found in [21], [15] and [29]):

- Yao’s definition of randomized communication complexity [35] (hereby denoted  $C_\epsilon(f)$ ) is thus the infimum of  $Max(l_P)$ , when taken over all *randomized* protocols  $P$  satisfying  $Min(S_{P,f}) \geq \frac{1}{2} + \epsilon$ .
- Yao’s definition of distributed communication complexity [36] (hereby denoted  $D_\epsilon(f)$ ) is the infimum of  $Ave(L_P)$ , when taken over all *deterministic* protocols  $P$  satisfying  $Ave(S_{P,f}) \geq \frac{1}{2} + \epsilon$ .

---

<sup>2</sup> The role of  $\epsilon$  in our notation differs from its role in [35], [36], [20]. Here  $\epsilon$  denotes the advantage of the protocol over  $\frac{1}{2}$ , while originally it was used to denote the error probability.

- Orlitsky and El-Gamal [20] measure the *average communication complexity* (hereby denoted  $\bar{C}_\varepsilon(f)$ ) as the infimum of  $Ave(L_P)$ , when taken over all randomized protocols  $P$  satisfying  $Min(S_{P,f}) \geq \frac{1}{2} + \varepsilon$ .

- Paturi and Simon [23] define the *unbounded communication complexity* (hereby denoted  $U(f)$ ) to be the infimum of  $Max(I_P^*)$ , when taken over all randomized protocols  $P$  satisfying  $Min(S_{P,f}) > \frac{1}{2}$ .

*Our definitions.* We say that the protocol  $P$  has *average  $\varepsilon$ -advantage in guessing  $f$*  if  $Ave(S_{P,f}) \geq \frac{1}{2} + \varepsilon$ . In the following definitions we consider protocols with *average  $\varepsilon$ -advantage*.

- We define the *average randomized communication complexity of function  $f$*  (denoted  $A_\varepsilon^R(f)$ ) as the infimum of  $Ave(L_P)$ , when taken over all *randomized* protocols  $P$  which have average  $\varepsilon$ -advantage in guessing  $f$ .

- The *worst-case randomized communication complexity of function  $f$*  (denoted  $W_\varepsilon^R(f)$ ) is defined as the infimum of  $Max(I_P^*)$ , when taken over all *randomized* protocols  $P$  which have average  $\varepsilon$ -advantage in guessing  $f$ .

- The *deterministic communication complexities* ( $A_\varepsilon^D(f)$  and  $W_\varepsilon^D(f)$ ) of the function  $f$  are defined similarly, for deterministic protocols.

*Comparison of definitions.* For all functions  $f \in F_n$ , the following inequalities are immediate from the definitions;

$$A_\varepsilon^R(f) \leq \bar{C}_\varepsilon(f) \leq C_\varepsilon(f),$$

$$A_\varepsilon^R(f) \leq A_\varepsilon^D(f) = D_\varepsilon(f) \leq W_\varepsilon^D(f).$$

Yao showed that  $C_{\frac{1}{2}-\eta}(f) \geq \frac{1}{2} D_{\frac{1}{2}-2\eta}(f)$  [34], [36].

There are, however, functions for which  $A_\varepsilon^R(f) \ll C_\varepsilon(f), D_\varepsilon(f)$ . One such function is the ordering function  $g$  defined by  $g(x, y) = 1$  if and only if  $x \leq y$ . Yao showed that for any fixed  $\varepsilon > 0$ ,  $C_\varepsilon(g) = \Omega(\log n)$  [35]. The protocol in which  $A$  sends the most significant bit of  $x$  has a  $\frac{1}{4}$ -advantage in guessing  $g$ , and thus  $A_{1/4}^R(g) = 1$  (in fact,  $W_{1/4}^D(g) = 1$ ). (Paturi and Simon [23] showed that  $U(g) = 2$ .) The three measures  $W_\varepsilon^R(f), U(f)$  and  $C_\varepsilon(f)$  are not always comparable.

**4.2. Lower bounds.** We begin this section by stating our lower bounds, and comparing them to recent results of other researchers.

**THEOREM 21.** *Let  $0 < \varepsilon \leq \frac{1}{2}$ .*

(1) *For at least a  $1 - 2^{-2^n}$  fraction of the Boolean functions  $f \in F_n$ ,*

$$W_\varepsilon^R(f) > n - 7 - 3 \log_2 \varepsilon^{-1},$$

$$A_\varepsilon^R(f) > 2\varepsilon \cdot (n - 7 - 3 \log_2 \varepsilon^{-1}n) - 1.$$

(2) *For every  $f \in F_n$  representable by a Hadamard matrix, the following holds:*

$$W_\varepsilon^R(f) > n - 3 - 3 \log_2 \varepsilon^{-1},$$

$$A_\varepsilon^R(f) > 2\varepsilon \cdot (n - 3 - 3 \log_2 \varepsilon^{-1}n) - 1.$$

*In particular, this holds for the inner-product function.*

*Comparison to other works.* Our result implies that for almost all  $f \in F_n$ ,  $\bar{C}_\varepsilon(f) \geq 2\varepsilon(n - 7 - 3 \log_2 \varepsilon^{-1}n) - 1$ . This is related to a recent independent result of Orlitsky and El-Gamal [20], who showed that almost all  $f \in F_n$  have  $\bar{C}_\varepsilon(f) \geq 2\varepsilon(n - 1 - \log_2 n)$ . (Actually, they showed that for all  $2^{-n} < r \leq \frac{1}{2}$ , almost all functions  $f$  with  $r \cdot 2^{2^n}$  ones in their table, have  $\bar{C}_\varepsilon(f) \geq 2\varepsilon(n - \log_2 r^{-1}n)$ .)

Our bound on  $W_\epsilon^R(f)$  (for almost all  $f \in F_n$ ) is related to a recent independent result of Alon, Frankl and Rödl [5] who showed that almost all  $f \in F_n$  have  $U(f) \geq n - 5$ . Their result implies that  $C_\epsilon(f) > 2\epsilon(n - 5)$ .

All three works resolve Yao's open problem [35]: *What is  $C_\epsilon(f)$  for a random  $f \in F_n$ ?*

**4.2.1. Proof of Theorem 21.**

PROPOSITION 22. *Let  $0 < \delta \leq 1$ . Then  $A_\epsilon^R(f) \geq (2 - \delta)\epsilon \cdot W_{\delta\epsilon/2}^R(f)$ . (In particular,  $A_\epsilon^R(f) \geq \epsilon \cdot W_{\epsilon/2}^R(f)$ .)*

*Proof.* Consider runs of protocol  $P$  which have an  $\epsilon$ -advantage in guessing  $f$  and average length  $a$ . Truncate runs of  $P$  that exceed  $\epsilon^{-1}a/(2 - \delta)$  bits. In the event of a long run, flip a coin to determine the final guess. Such runs occur with probability  $\leq (2 - \delta)\epsilon$ , and by guessing at random we lose at most  $(2 - \delta)\epsilon/2$  of the average success probability. This yields a protocol with  $(\delta/2) \cdot \epsilon$ -advantage, the runs of which are no more than  $\epsilon^{-1}a/(2 - \delta)$  bits long.  $\square$

PROPOSITION 23.  $W_\epsilon^R(f) = W_\epsilon^D(f)$ .

*Proof.* For a randomized protocol, the average advantage is a sum over both the inputs and the coin tosses. There must be at least one sequence of coin tosses which does at least as well as the average. Using this string, we get a deterministic protocol with the same length and at least the same average advantage.  $\square$

Notice that both the above argument holds since we are interested in average advantage; the advantage on individual pairs may decrease. Using Propositions 22 and 23, we may concentrate on studying  $W_\epsilon^D(f)$ .

THEOREM 24. *Let  $k, n$  be integers, and  $0 < \epsilon \leq 1$ . Suppose that for every  $b_1 + b_2 \geq 2n - k - 1 + \log_2 \epsilon$ , the Boolean function  $f: \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  is  $\epsilon$ -robust with respect to any pair of independent random variables  $X, Y$  satisfying:  $X$  is  $(n, b_1)$ -distributed and  $Y$  is  $(n, b_2)$ -distributed. Then*

$$W_\epsilon^D(f) > k.$$

*Proof.* Suppose, towards a contradiction, that  $P$  is a deterministic protocol with average  $\epsilon$ -advantage in guessing  $f$ , such that  $\text{Max}(I_P^*) \leq k$ . Consider all possible executions of  $P$  and assume, without loss of generality, that  $A$  and  $B$  exchange exactly  $k$  bits on each pair of inputs.

For every  $\gamma \in \{0, 1\}^k$ , denote by  $C(\gamma)$  the set of  $(x, y)$  pairs on which the communication of  $A$  and  $B$  consists of  $\gamma$ . Note that by prefix freeness, the parsing of  $\gamma$  is unique. Let  $A(\gamma) = \{x: \exists y \text{ s.t. } (x, y) \in C(\gamma)\}$ , and  $B(\gamma) = \{y: \exists x \text{ s.t. } (x, y) \in C(\gamma)\}$ . By a cut-and-paste argument of Yao [35],  $C(\gamma) = A(\gamma) \times B(\gamma)$ .

Denote by  $\text{last}(\gamma)$  the last bit exchanged in the communication  $\gamma$ , and let  $G(\gamma) = \{(x, y) \in C(\gamma): \text{last}(\gamma) = f(x, y)\}$ . since  $P$  has  $\epsilon$ -advantage on  $f$ , we have that  $\sum_{\gamma \in \{0, 1\}^k} |G(\gamma)| \geq (\frac{1}{2} + \epsilon) \cdot 2^{2n}$ . Let us say that  $C(\gamma)$  is small if  $|C(\gamma)| < \epsilon \cdot 2^{2n-k-1}$ . Since there are at most  $2^k$  rectangles  $C(\gamma)$ , the number of points in all small rectangles is at most  $\epsilon \cdot 2^{2n-1}$ . Thus

$$\sum_{\gamma \text{ s.t. } |C(\gamma)| \geq \epsilon \cdot 2^{2n-k-1}} |G(\gamma)| \geq \left(\frac{1}{2} + \frac{\epsilon}{2}\right) \cdot 2^{2n}.$$

This implies that there exists a  $\gamma \in \{0, 1\}^k$  such that both  $|C(\gamma)| \geq \epsilon \cdot 2^{2n-k-1}$  and  $|G(\gamma)| \geq (\frac{1}{2} + \epsilon/2) \cdot |C(\gamma)|$  (i.e.,  $C(\gamma)$  is sufficiently large, and the protocol has nonnegligible advantage on the pairs in it). Fix such a  $\gamma$ .

Set  $X$  and  $Y$  to be two independent random variables, flat on  $A(\gamma)$  and  $B(\gamma)$ , respectively. Then  $\text{Pr}(f(X, Y) = \text{last}(\gamma)) \geq \frac{1}{2} \cdot (1 + \epsilon)$ . Let  $b_1 = \log_2 |A(\gamma)|$  and  $b_2 =$

$\log_2 |B(\gamma)|$ . Then  $X$  is  $(n, b_1)$ -distributed,  $Y$  is  $(n, b_2)$ -distributed, and  $b_1 + b_2 \geq \log_2(\varepsilon \cdot 2^{2n-k-1}) = 2n - k - 1 + \log_2 \varepsilon$ . This contradicts the  $\varepsilon$ -robustness of  $f$ .  $\square$

Theorem 21 (above) is a consequence of combining Theorem 24 (and Propositions 22 and 23) with Theorems 7 and 9 (of §§ 2.4 and 2.5, respectively). The arithmetic details are as follows.

Part (1): Let  $0 \leq \varepsilon \leq \frac{1}{2}$ . By Theorem 7 (see special case 4), all but at most  $2^{-2^n}$  of the functions  $f: \{0, 1\}^{2^n} \mapsto \{0, 1\}$  are  $\varepsilon$ -robust for any pair of  $(n, b_1), (n, b_2)$  sources satisfying  $b_1 + b_2 \geq n + 6 + 2 \log_2 \varepsilon^{-1}$ . Setting  $k = n - 7 - 3 \log_2 \varepsilon^{-1}$ , these functions  $f$  satisfy the condition of Theorem 24 (being  $\varepsilon$ -robust for sources with  $b_1 + b_2 \geq 2n - k - 1 - \log_2 \varepsilon^{-1}$ ). Thus, these functions  $f$  have  $W_\varepsilon^D(f) > n - 7 - 3 \log_2 \varepsilon^{-1}$ . To get the bound on  $A_\varepsilon^R(f)$ , we use Propositions 23 and 22:

$$\begin{aligned} A_\varepsilon^R(f) &\geq \max_{0 < \delta \leq 1} (2 - \delta)\varepsilon \cdot W_{\delta\varepsilon/2}^R(f) \\ &\geq (2 - 2/n)\varepsilon \cdot W_{\varepsilon/n}^D(f) \\ &> 2\varepsilon \cdot (n - 7 - 3 \log_2 \varepsilon^{-1}) - 1. \end{aligned}$$

Part (2): Similarly, by using Theorem 9, and setting  $k = n - 3 - 3 \log_2 \varepsilon^{-1}$ . This completes the proof of Theorem 21.  $\square$

**4.3. Upper bounds.** The lower bound on  $A_\varepsilon^R(f)$  for almost all  $f$ 's is nearly optimal, since Orlitsky and El-Gamal showed that most  $f \in F_n$  have  $C_\varepsilon(f) \leq 2\varepsilon(n + 6 \log_2 \varepsilon^{-1}n)$  [20] (recall that  $A_\varepsilon^R(f) \leq C_\varepsilon(f)$ ). The lower bound on  $W_\varepsilon^R(f)$  is also nearly optimal, since we have the following upper bounds.

**THEOREM 25.** (1) For every  $f \in F_n$  and every  $2^{-n/2+1} < \varepsilon < \frac{1}{32}$ ,  $W_\varepsilon^D(f) \leq n + 11 - 2 \log_2 \varepsilon^{-1}$ .

(2) For all  $f \in F_n$ ,  $W_{2^{-n/2+5.5}}^D(f) \leq 2$ .

*Proof.* For part (1), we use the following protocol. Party  $B$  sends the  $n + 9 - 2 \log_2 \varepsilon^{-1}$  most significant bits of  $y$  to party  $A$ . This defines a  $2^n \times 2 \cdot (32\varepsilon)^{-2}$  strip in  $f$ 's table. By Lemma 2 each such strip contains a  $2^{n-4} \times (32\varepsilon)^{-2}$  submatrix  $S$  with  $\frac{1}{2} + 32\varepsilon$  fraction of identical entries  $\sigma$  in it. In addition, party  $B$  sends a bit specifying whether  $y$  corresponds to a column in  $S$ . Party  $A$  replies by  $\sigma$  if  $(x, y)$  is in  $S$ , and by the outcome of a coin flip otherwise. In this way, we get an average  $\varepsilon$ -advantage.

For part (2), let  $S$  be a  $2^{n-4} \times 2^{n-1}$  submatrix containing a  $\frac{1}{2} + 2^{-\frac{1}{2}(n-1)}$  fraction of identical entries  $\sigma$  in it (Lemma 2 guarantees the existence of  $S$ ). Party  $B$  sends a bit specifying whether  $y$  corresponds to a column in  $S$ . Party  $A$  replies by  $\sigma$  if  $(x, y)$  is in  $S$ , and by the outcome of a coin flip otherwise. In this way, we get an average  $\frac{1}{32} \cdot 2^{(n-1)/2}$ -advantage.  $\square$

**4.4. Extension to  $(n, m)$ -distributions.** In the definitions and results presented above, we have assumed that the inputs to the protocol are uniformly distributed in  $\{0, 1\}^n$ . A natural question is what happens if we allow the inputs to be  $(n, m)$ -distributed. In particular, we consider protocols which have advantage with respect to some  $(n, m)$ -distributions, and study the infimum average number of bits they exchanged under these ‘‘advantageous’’ distributions. We show that most functions in  $F_n$  have  $\Theta(m)$  complexity, even in this weak measure.

**DEFINITION 11.** Let  $D_1$  and  $D_2$  be two  $(n, m)$ -distributions, and let  $P_i(z)$  be the probability that  $D_i$  assigns to  $z$ . For any protocol  $P$ , a function  $f \in F_n$ , and a metric  $M_{P,f}$  over runs of  $P$ , we define the *average operator on  $D_1 \times D_2$*

$$Ave_{D_1, D_2}(M_{P,f}) = \sum_{x, y \in \{0, 1\}^n} P_1(x) \cdot P_2(y) \cdot M_{P,f}(x, y).$$

Similarly, we define the *maximum operator on  $D_1 \times D_2$*

$$\text{Max}_{D_1, D_2}(M_{P,f}) = \max_{x \in \text{supp}(D_1), y \in \text{supp}(D_2)} \{M_{P,f}(x, y)\},$$

where  $\text{supp}(D_i) = \{z \in \{0, 1\}^n \mid P_i(z) > 0\}$ .

We say that the protocol  $P$  has  $D_1 \times D_2$ -average  $\varepsilon$ -advantage in guessing  $f$  if  $\text{Ave}_{D_1, D_2}(S_{P,f}) \geq \frac{1}{2} + \varepsilon$ .

- We define the *average randomized communication  $(n, m)$ -complexity of function  $f$*  (denoted  $(n, m)$ - $A_\varepsilon^R(f)$ ) as the infimum of  $\text{Ave}_{D_1, D_2}(L_P)$ , when taken over all  $(n, m)$ -distributions  $D_1, D_2$  and all *randomized* protocols  $P$  which have  $D_1 \times D_2$ -average  $\varepsilon$ -advantage in guessing  $f$ .

- The *worst-case randomized communication  $(n, m)$ -complexity of function  $f$*  (denoted  $(n, m)$ - $W_\varepsilon^R(f)$ ) is defined as the infimum of  $\text{Max}_{D_1, D_2}(L_P^*)$ , when taken over all  $(n, m)$ -distributions  $D_1, D_2$  and all *randomized* protocols  $P$  which have  $D_1 \times D_2$ -average  $\varepsilon$ -advantage in guessing  $f$ .

- The *deterministic communication  $(n, m)$ -complexities* of the function  $f$  are defined similarly, for deterministic protocols.

The key to dealing with  $(n, m)$ -complexities, is the fact that they are minimized on flat distributions (the proof is analogous to Lemma 5). Using the proof techniques of Theorem 21, the problem reduces to the existence of large submatrices with significant advantage inside the submatrix specified by the pair of flat distributions. We get the following.

**THEOREM 26.** *Let  $0 < \varepsilon \leq \frac{1}{2}$ .*

(1) *For at least a  $1 - 2^{-2^m}$  fraction of the Boolean functions  $f \in F_n$ ,*

$$(n, m)\text{-}W_\varepsilon^R(f) > m - 7 - 3 \log_2 \varepsilon^{-1},$$

$$(n, m)\text{-}A_\varepsilon^R(f) > 2\varepsilon \cdot (m - 7 - 3 \log_2 \varepsilon^{-1}m) - 1.$$

(2) *For every  $f \in F_n$  representable by a Hadamard matrix, the following holds:*

$$(n, m)\text{-}W_\varepsilon^R(f) > 2m - n - 3 - 3 \log_2 \varepsilon^{-1},$$

$$(n, m)\text{-}A_\varepsilon^R(f) > 2\varepsilon \cdot (2m - n - 3 - 3 \log_2 \varepsilon^{-1}m) - 1.$$

*In particular, this holds for the inner-product function.*

**5. On the robustness of BPP.** The class  $R$  [1] and its symmetric version BPP [13] consist of problems which can be solved with high probability in polynomial time, with the use of an unbiased coin. Recently, Vazirani and Vazirani [32] showed that all BPP problems can be efficiently solved even if a single SV-source is producing the coin tosses. In this section, we generalize their result by showing that BPP problems can be efficiently solved if a (single) PRB-source is producing the coin tosses.

The main idea of the proof is that *any* function which is robust with respect to *two independent* PRB-sources, can be used to produce *polynomially many* bits such that almost all of them are unbiased. Repeating this process  $m$  times, we get poly( $m$ ) strings of length  $m$  each. Most of these strings are almost uniformly distributed, and thus the fraction of these strings which hit the witness set  $W \subset \{0, 1\}^m$  is close to the density of  $W$ . If  $W$ 's density is large enough (say,  $\geq 0.8$ ) then with probability bounded away from 0.5 (e.g.,  $\geq 0.55$ ), the majority of the generated strings hit  $W$ . This argument needs careful formalization, which is carried out below. The final observation is that there are explicit and efficiently computable functions which are appropriate for the above procedure (e.g., the inner-product or the Paley graph functions).

The key technical lemma used in the proof is the following.

LEMMA 27. Let  $0 < \varepsilon < 1$  be a real and  $f: \{0, 1\}^l \times \{0, 1\}^l \mapsto \{0, 1\}$  be a Boolean function. Define  $f_i: \{0, 1\}^l \mapsto \{0, 1\}$  by  $f_i(j) = f(i, j)$  for every  $i, j \in \{0, 1\}^l$ . Suppose that the  $f$  is  $\varepsilon$ -robust with respect to any two independent random variables which are  $(l, l-1)$ -distributed and  $(l, b)$ -distributed, respectively. Then for every  $(l, b)$ -distributed  $Y$ , all but  $\sqrt{\varepsilon}$  fraction of the  $f_i$ 's are  $4\sqrt{\varepsilon}$ -robust on  $Y$ .

The reader may find it convenient to picture the two-argument Boolean function  $f: \{0, 1\}^l \times \{0, 1\}^l \mapsto \{0, 1\}$  as a table where the  $(i, j)$ -entry corresponds to  $f(i, j)$ . The lemma can be stated (informally) as follows: *if a function can be used for extracting almost unbiased bits from the output of any two independent PRB-sources, then most of its "rows" can be used for extracting an almost unbiased bit from a single PRB-source.* The identity of these "good" rows depends on the specific PRB-source, but for each source most of the rows will work. The proof is by contradiction, showing that if the conclusion of the lemma is violated, then it is possible to find a pair of probability bounded sources which falsify the robustness of  $f$ . While the proof of this lemma is rather simple, it seems much harder to prove a similar statement based merely on robustness with respect to SV-sources.

*Proof.* Let  $Y$  be an arbitrary  $(l, b)$ -distributed random variable. This defines a probability space on the entries of the rows. We will show that the number of rows which are biased too much towards 1 is small (rows with high 0 bias are treated identically). Let  $B$  denote the set of these rows, that is,

$$B = \{i: \Pr(f_i(Y) = 1) > \frac{1}{2}(1 + 4\sqrt{\varepsilon})\},$$

and let  $k$  denote the size of  $B$ .

We first show that  $k \leq 2^{l-1}$ . Assume, on the contrary, that  $k > 2^{l-1}$ . We will reach a contradiction by defining an  $(l, l-1)$ -distributed source  $X_1$  to be flat on  $B$ . Then

$$\begin{aligned} \Pr(f(X_1, Y) = 1) &= k^{-1} \sum_{i \in B} \Pr(f_i(Y) = 1) \\ &> \frac{1}{2} \cdot (1 + 4\sqrt{\varepsilon}) \\ &> \frac{1}{2} \cdot (1 + \varepsilon). \end{aligned}$$

Now that we know  $k \leq 2^{l-1}$ , we define an  $(l, l-1)$ -distributed source  $X_2$  to be flat on  $\{0, 1\}^l - B$ . Applying the  $\varepsilon$ -robustness of  $f$  to the pair  $(X_0, Y)$ , where  $X_0$  is the uniform source, we have

$$\begin{aligned} \frac{1}{2^l} \sum_{i \in \{0, 1\}^l} \Pr(f_i(Y) = 1) &= \Pr(f(X_0, Y) = 1) \\ &< \frac{1}{2}(1 + \varepsilon). \end{aligned}$$

In order to bound  $k$  from above, we first derive an upper bound on  $\Pr(f(X_2, Y) = 1)$ :

$$\begin{aligned} \Pr(f(X_2, Y) = 1) &= \frac{1}{2^l - k} \cdot \sum_{i \in \{0, 1\}^l - B} \Pr(f_i(Y) = 1) \\ &= \frac{\sum_{i \in \{0, 1\}^l} \Pr(f_i(Y) = 1) - \sum_{i \in B} \Pr(f_i(Y) = 1)}{2^l - k} \\ &< \frac{2^l \cdot (\frac{1}{2}(1 + \varepsilon)) - k \cdot (\frac{1}{2}(1 + 4\sqrt{\varepsilon}))}{2^l - k} \\ &= \frac{1}{2} \left( 1 + \frac{2^l \varepsilon - k \cdot 4\sqrt{\varepsilon}}{2^l - k} \right). \end{aligned}$$

Applying the  $\varepsilon$ -robustness of  $f$  to the pair  $(X_2, Y)$  we get

$$\Pr(f(X_2, Y) = 1) > \frac{1}{2}(1 - \varepsilon).$$

Combining the upper and lower bounds on  $\Pr(f(X_2, Y) = 1)$ , we have

$$\frac{1}{2} \left( 1 + \frac{2^l \varepsilon - 4k\sqrt{\varepsilon}}{2^l - k} \right) \cong \frac{1}{2}(1 - \varepsilon).$$

By a simple manipulation,  $k \leq \frac{1}{2}\sqrt{\varepsilon} \cdot 2^l$  follows.  $\square$

With Lemma 27 at our disposal, we can use a single source to generate many strings, most of which are almost unbiased. These strings are generated one bit at a time (i.e., in step  $t$ , the  $t$ th bit of all strings is generated). In the BPP application, the generated strings are tested for membership in a witness set  $W$ . A careful probabilistic analysis shows that if  $W$  is dense enough, then there is a fairly large probability that the majority of these strings will hit  $W$ .

**PROPOSITION 28.** (1) Let  $m$  be an integer, and  $W \subset \{0, 1\}^m$  be an arbitrary set. Denote by  $p$  the density of  $W$  (i.e.,  $p \stackrel{\text{def}}{=} |W|/2^m$ ), and let  $q \stackrel{\text{def}}{=} 1 - p$ .

(2) Let  $l$  be an integer, and  $0 < b < l$ . Let  $0 < \varepsilon < \frac{1}{2}$  and  $f: \{0, 1\}^l \mapsto \{0, 1\}$  be a Boolean function. Suppose that  $f$  is  $\varepsilon$ -robust with respect to any two independent random variables which are  $(l, l-1)$ -distributed and  $(l, b)$ -distributed, respectively.

(3) Let  $Y_1, Y_2, \dots, Y_m$  be a sequence of arbitrary random variables assuming values in  $\{0, 1\}^l$  such that for every  $1 \leq t \leq m$  the variable  $Y_t$  is  $(l, b)$ -distributed given  $Y_1, Y_2, \dots, Y_{t-1}$ . Let  $Y$  denote the concatenation of the  $Y_t$ 's.

(4) For every  $i \in \{0, 1\}^l$ , let  $f_i: \{0, 1\} \mapsto \{0, 1\}$  be defined as in Lemma 27 (i.e.,  $f_i(j) = f(i, j)$  for every  $i, j \in \{0, 1\}^l$ ). Let  $h_i(Y)$  be a random variable assuming values in  $\{0, 1\}^m$ , such that  $h_i(Y)$  is the concatenation of the random variables  $f_i(Y_1), f_i(Y_2), \dots, f_i(Y_m)$ .

Then the probability that a majority of the  $h_i(Y)$ 's miss  $W$ , does not exceed  $2q + 8m\sqrt{\varepsilon}$ . That is,

$$\Pr(|\{h_i(Y) : i \in \{0, 1\}^l\} \cap W| \leq 2^{l-1}) \leq 2q + 8m\sqrt{\varepsilon}.$$

**5.1. Proof of Proposition 28.** *Convention.* Throughout the proof, the probability space is the Cartesian product of  $Y$  (of item (3) above) with an  $(m, m)$ -distributed random variable  $Z$ . The random variable  $Y = Y_1 Y_2 \dots Y_m$  assumes values in  $\{0, 1\}^{m \cdot l}$ , and  $Z = Z_1 Z_2 \dots Z_m$  is uniformly distributed in  $\{0, 1\}^m$  independently of  $Y$ . A value which  $Y$  may assume will be denoted by  $\alpha = \alpha_1 \alpha_2 \dots \alpha_m$ , where  $\alpha_t \in \{0, 1\}^l$ . A value which  $Z$  may assume will be denoted by  $\beta = \beta_1 \beta_2 \dots \beta_m$ , where  $\beta_t \in \{0, 1\}$ . We will also use the notation  $Y'_t = Y_1 Y_2 \dots Y_t$  and  $\alpha'_t = \alpha_1 \alpha_2 \dots \alpha_t$  to denote the prefix consisting of first  $t$  elements of  $Y$  and  $\alpha$ , respectively.

**DEFINITIONS.** (1) For every  $i \in \{0, 1\}^l$ ,  $0 \leq t \leq m$ , we define a Boolean function  $\xi_{i,t}: \{0, 1\}^{lm} \times \{0, 1\}^m \mapsto \{0, 1\}$  as follows:

$$\xi_{i,t}(\alpha, \beta) = \begin{cases} 0 & \text{if } f_i(\alpha_1) f_i(\alpha_2) \dots f_i(\alpha_t) \beta_{t+1} \beta_{t+2} \dots \beta_m \in W, \\ 1 & \text{otherwise.} \end{cases}$$

(2) For every  $i \in \{0, 1\}^l$ ,  $0 \leq t \leq m$  we define a Boolean function  $\eta_{i,t}: \{0, 1\}^{lm} \mapsto \{0, 1\}$  as follows:

$$\eta_{i,t}(\alpha) = \begin{cases} 0 & \text{if } \frac{1}{2} - 2\sqrt{\varepsilon} \leq \Pr(f_i(Y_{t+1}) = 1 \mid Y'_t = \alpha'_t) \leq \frac{1}{2} + 2\sqrt{\varepsilon}, \\ 1 & \text{otherwise.} \end{cases}$$

*Explanation.* Letting  $\alpha$  assume values in  $\{0, 1\}^{lm}$  according to the random variable  $Y$ , and  $\beta$  assume values in  $\{0, 1\}^m$  according to the uniform distribution  $Z$ , the two functions above induce two random variables  $\xi_{i,t}(Y, Z)$  and  $\eta_{i,t}(Y)$ . These random

variables correspond to hybrids of the  $(l, b)$ -source and truly unbiased, independent coin tosses. The random variable  $\xi_{i,t}(Y, Z)$  equals 0 (“a success”) when a hybrid element, generated by applying the function  $f_i$  to the first  $t$  blocks output by the source  $Y$  and letting the rest be truly random, hits the set  $W$ . The random variable  $\eta_{i,t}(Y)$  equals 0 (“a good bit”) if, given the first  $t$  blocks of the source, the bit generated by applying  $f_i$  to the  $(t + 1)$ st block is almost unbiased.

**Elementary observations.**

FACT 1. For every  $i \in \{0, 1\}^l$ , we have

$$\Pr(\xi_{i,0}(Y, Z) = 0) = p.$$

*Proof.* Immediate by the definition of  $\xi_{i,0}(Y, Z)$  and  $|W| = p2^m$ .  $\square$

FACT 2.

$$\sum_{i \in \{0,1\}^l} \sum_{t=0}^{m-1} \exp(\eta_{i,t}(Y)) \leq 2^l m \sqrt{\epsilon}.$$

*Proof.* By Lemma 27, for every  $0 \leq t < m$  and every  $\alpha \in \{0, 1\}^{m-l}$ , we have

$$\sum_{i \in \{0,1\}^l} \eta_{i,t}(\alpha) \leq 2^l \sqrt{\epsilon}.$$

Thus, for every  $0 \leq t \leq m - 1$ ,

$$\sum_{i \in \{0,1\}^l} \eta_{i,t}(Y) \leq 2^l \sqrt{\epsilon}.$$

The sum of  $m$  such expressions (for the  $m$  values of  $t$ ) is thus bounded above by  $2^l m \sqrt{\epsilon}$ , and so is the expected value. Changing the order of summation, we get the claimed bound.  $\square$

The next fact formulates the intuition that, when the  $(t + 1)$ st bit produced in the  $i$ th row is almost unbiased, then the  $(t + 1)$ th hybrid of this row has almost the same success probability as the  $t$ th hybrid to hit  $W$ .

FACT 3. For every  $i \in \{0, 1\}^l$  and  $0 \leq t < m$ , we have

$$\Pr(\xi_{i,t+1}(Y, Z) = 0 \mid \eta_{i,t}(Y) = 0) \geq \Pr(\xi_{i,t}(Y, Z) = 0 \mid \eta_{i,t}(Y) = 0) - 2\sqrt{\epsilon}.$$

*Proof.* Consider an arbitrary  $\alpha \in \{0, 1\}^{m-l}$  such that  $\eta_{i,t}(\alpha) = 0$ . Let  $r = \Pr(f_i(Y_{t+1}) = 1 \mid Y'_t = \alpha'_t) - \frac{1}{2}$ . Then  $|r| \leq 2\sqrt{\epsilon}$ . Let  $s = \Pr(\xi_{i,t}(Y, Z) = 0 \mid Y'_t = \alpha'_t)$ ,  $s_0 = \Pr(\xi_{i,t}(Y, Z) = 0 \mid Y'_t = \alpha'_t, Z_{t+1} = 0)$  and  $s_1 = \Pr(\xi_{i,t}(Y, Z) = 0 \mid Y'_t = \alpha'_t, Z_{t+1} = 1)$ . By definition  $s = \frac{1}{2}s_0 + \frac{1}{2}s_1$ . Then

$$\begin{aligned} \Pr(\xi_{i,t+1}(Y, Z) = 0 \mid Y'_t = \alpha'_t) &= (\frac{1}{2} - r) \cdot s_0 + (\frac{1}{2} + r) \cdot s_1 \\ &= s - r(s_0 - s_1) \\ &\geq s - 2\sqrt{\epsilon}. \end{aligned}$$

Averaging over all such  $\alpha'_t$ 's, Fact 3 follows.  $\square$

**Probability calculation.** The next fact is crucial to our proof. It expresses the (unconditional) success probability of the  $(t + 1)$ st hybrid of the  $i$ th row, in terms of the  $t$ th hybrids of this row. The difference between the  $(t + 1)$ st and  $t$ th hybrids is bounded by the sum of a small error probability ( $\leq 2\sqrt{\epsilon}$ ), introduced by runs in which the  $(t + 1)$ st bit is almost unbiased, and the probability that the  $(t + 1)$ st bit is biased.

FACT 4. For every  $i \in \{0, 1\}^l$  and  $0 \leq t < m$ , we have

$$\Pr(\xi_{i,t+1}(Y, Z) = 0) \geq \Pr(\xi_{i,t}(Y, Z) = 0) - 2\sqrt{\epsilon} - 2 \cdot \Pr(\eta_{i,t}(Y) = 1).$$



*Proof.* By following manipulation, using Fact 3 (when passing from the first line to the second line), and the inequality  $\Pr(A|B) \geq \Pr(A) - \Pr(\bar{B})$  (passing from the third line to the fourth line), we have

$$\begin{aligned} \Pr(\xi_{i,t+1}(Y, Z) = 0) &\geq \Pr(\xi_{i,t+1}(Y, Z) = 0 | \eta_{i,t}(Y) = 0) \cdot \Pr(\eta_{i,t}(Y) = 0) \\ &\geq (\Pr(\xi_{i,t}(Y, Z) = 0 | \eta_{i,t}(Y) = 0) - 2\sqrt{\varepsilon}) \cdot (1 - \Pr(\eta_{i,t}(Y) = 1)) \\ &\geq \Pr(\xi_{i,t}(Y, Z) = 0 | \eta_{i,t}(Y) = 0) - 2\sqrt{\varepsilon} - \Pr(\eta_{i,t}(Y) = 1) \\ &\geq \Pr(\xi_{i,t}(Y, Z) = 0) - 2\sqrt{\varepsilon} - 2 \cdot \Pr(\eta_{i,t}(Y) = 1). \quad \square \end{aligned}$$

This yields an upper bound on the probability that the  $i$ th row does not hit  $W$ , when being generated from the blocks of the  $(l, b)$ -source using the function  $f_i$ .

FACT 5. For every  $i \in \{0, 1\}^l$

$$\text{Exp}(\xi_{i,m}(Y, Z)) \leq q + 2\sqrt{\varepsilon}m + 2 \sum_{t=0}^{m-1} \text{Exp}(\eta_{i,t}(Y)).$$

*Proof.* Follows by combining Fact 1 with repeated use of Fact 4, and using the fact that for any 0-1 random variable  $V$ ,  $\text{Exp}(V) = \Pr(V = 1)$ .  $\square$

**Conclusion.** We now bound the probability that the majority of rows produce elements which *do not hit* the set  $W$ .

FACT 6.

$$\Pr\left(\sum_{i \in \{0,1\}^l} \xi_{i,m}(Y, Z) \geq 2^{l-1}\right) \leq 2q + 8\sqrt{\varepsilon}m.$$

*Proof.* Applying the Markov inequality (the sum of the  $\xi_{i,t}(Y, Z)$ ), using Fact 5 (when passing from the first line to the second line) and Fact 2 (when passing from the second line to the third line), we have

$$\begin{aligned} \Pr\left(\sum_{i \in \{0,1\}^l} \xi_{i,m}(Y, Z) \geq 2^{l-1}\right) &\leq \frac{\text{Exp}(\sum_{i \in \{0,1\}^l} \xi_{i,m}(Y, Z))}{2^{l-1}} \\ &\leq 2^{-l+1} \cdot \left(2^l \cdot (q + 2\sqrt{\varepsilon}m) + 2 \cdot \sum_{i \in \{0,1\}^l} \sum_{t=0}^{m-1} \text{Exp}(\eta_{i,t}(Y))\right) \\ &\leq 2q + 4\sqrt{\varepsilon}m + 4 \cdot \sqrt{\varepsilon}m \\ &= 2q + 8\sqrt{\varepsilon}m. \quad \square \end{aligned}$$

Since  $\xi_{i,m}(Y, Z) = 1$  is just a fancy way of writing  $h_i(Y) \notin W$ , we get

$$\Pr(|\{h_i(Y) : i \in \{0, 1\}^l\} \cap W| \leq 2^{l-1}) \leq 2q + 8\sqrt{\varepsilon}m,$$

and Proposition 28 follows.  $\square$

**5.2. The transformation of BPP algorithms.** It will be convenient to consider randomized algorithms as deterministic algorithms with an auxiliary random input. The performance of such an algorithm (on input  $x$ ) will be evaluated with respect to the distribution of the auxiliary random input (denoted  $y$ ). The issue of the robustness of the class BPP is then stated as follows: *can a BPP algorithm be converted to a polynomial-time algorithm which has an advantage bounded away from  $\frac{1}{2}$  even when its random input is generated by a single probability-bounded source?*

DEFINITION. The class  $(l, b)$ -BPP consists of all decision problems  $D: \{0, 1\}^* \mapsto \{0, 1\}$  for which there exists polynomials  $P, Q$  and an algorithm  $A: \{0, 1\}^* \times \{0, 1\}^* \mapsto \{0, 1\}$  such that for every  $(l, b)$ -source  $Y$  the following holds:

(1) On each input of length  $n$ , algorithm  $A$  runs at most  $P(n)$  steps, and then stops, outputting a single bit.

(2) Let  $x \in \{0, 1\}^m$  be an input and let  $y \in \{0, 1\}^{P(n)}$  be the auxiliary random input generated by the  $(l, b)$ -source  $Y$ . Then

$$\Pr(A(x, y) = D(x)) \geq \frac{1}{2} + Q^{-1}(n).$$

Clearly,  $(l, l)$ -BPP is just a fancy way of writing BPP. Theorem 29 states that so is  $(l, b)$ -BPP.

**THEOREM 29.** *For every integer  $l$  and real  $0 < b \leq l$ ,*

$$(l, b)\text{-BPP} = \text{BPP}.$$

*Proof.* Let  $D$  be a decision problem in BPP, and  $A_0$  be a randomized polynomial-time algorithm for  $D$ . Let  $P(n)$  be the number of random bits used by the algorithm  $A_0$  on inputs of length  $n$ . Without loss of generality, we may assume that for every input  $x \in \{0, 1\}^m$ , the witness set  $W(x)$  for  $x$  (i.e., the  $y$ 's satisfying  $A_0(x, y) = D(x)$ ) contains  $p \geq 0.8$  of the strings in  $\{0, 1\}^{P(n)}$ .

Given  $l$  and  $b$ , let  $\varepsilon(n) \stackrel{\text{def}}{=} (160 \cdot P(n))^{-2}$ ,  $B(n) \stackrel{\text{def}}{=} 3 + 2 \log_2 \varepsilon^{-1}(n)$ , and  $L(n) \stackrel{\text{def}}{=} \lceil lB(n)/b \rceil$ . By Theorem 9, every function corresponding to a Hadamard matrix is  $\varepsilon(n)$ -robust with respect to any pair of independent random variables  $X, Y$  which are  $(L(n), L(n) - 1)$ -distributed and  $(L(n), B(n))$ -distributed, respectively. Furthermore, some of these families of functions, such as the inner-product modulo 2 or the quadratic residuosity modulo  $a$  prime (the Paley Graph function), can be computed by poly( $n$ )-time algorithms. Let  $f$  be one of these functions. Let  $F$  be an algorithm that on inputs  $n$  and  $i, j \in \{0, 1\}^{L(n)}$ , outputs  $f(i, j)$ .

By Proposition 28, we can use a single  $(L(n), B(n))$ -source to efficiently generate  $2^{L(n)}$  strings such that for every  $x \in \{0, 1\}^n$ , the majority of these strings hit the witness set  $W(x)$ , with probability greater than  $1 - 2q - 8P(n)\sqrt{\varepsilon(n)} \geq 0.55$  ( $q \leq 0.2$ ). We remind the reader that the  $(l, b)$ -source can be used as an  $(L(n), B(n))$ -source. Consider the following Algorithm A for deciding membership in  $D$  (with a two-sided error bounded above by 0.45).

1. **Algorithm A**
2. INPUT  $\leftarrow x$
- ; Let  $n = |x|$ , and  $m = P(n)$ .
3. AUXILIARY INPUT:  $y \in \{0, 1\}^{L(n)m}$  generated by an arbitrary  $(l, b)$ -source.
- ; Let  $y = y_1 y_2 \cdots y_m$ , where each  $y_i \in \{0, 1\}^{L(n)}$ .
4. For every  $i \in \{0, 1\}^{L(n)}$  and  $1 \leq j \leq m$ , compute  $f(i, y_j)$ , by invoking  $F(n, i, y_j)$ .
- ; For every  $i \in \{0, 1\}^{L(n)}$ , let  $w_i$  denote the concatenation  $f(i, y_1) \cdot f(i, y_2) \cdots f(i, y_m)$ .
5. For every  $i \in \{0, 1\}^{L(n)}$ , compute  $v_i \stackrel{\text{def}}{=} A_0(x, w_i)$ .
6. If  $\sum_{i \in \{0, 1\}^{L(n)}} v_i \leq 2^{L(n)-1}$  then  $d \leftarrow 0$  else  $d \leftarrow 1$ .
7. OUTPUT:  $d$ .

By the above discussion,  $\Pr(A(x, y) = D(x)) \geq 0.55$ . The running time of  $A(x, y)$  is polynomial in  $2^{L(n)} = n^{O(1)}$ , and in the running time of  $A_0(x, \cdot)$  and  $F(|x|, \cdot, \cdot)$ . The theorem follows.  $\square$

*Remarks.* (1) Algorithm A above is identical to the algorithm used in [32], [30], except that the auxiliary input is generated by a PRB-source instead of an SV-source. The difference is in the analysis of the success probability of this algorithm. Vazirani and Vazirani rely on properties of a particular function (inner-product modulo 2) to show that the different runs of  $A_0$  use auxiliary inputs related in a “good way.” We

reduce the existence of “good runs” of  $A_0$  to the robustness of the function with respect to two independent sources.

(2) Clearly, the same algorithm also works for the class  $R$ . It produces one-sided error, since for  $x \notin D$  the original algorithm  $A_0$  never errs.

(3) Proposition 28 can be viewed as proving a method for using a single PRB source to distinguish “high-density” sets from “low-density” sets. That is, given  $K \subset \{0, 1\}^k$  so that either  $|K| \geq p2^k$  or  $|K| \leq q2^k$ , determine which of the two cases occurs, with success probability about  $1 - 2q$ .

This viewpoint is helpful in solving the following additive approximation problem: for any  $\epsilon, \delta > 0$ , and any set  $S \subset \{0, 1\}^m$ , find an *additive*  $(\delta, \epsilon)$  approximation of the density of  $S$ , denoted  $\rho$ , using a single probability-bounded source (when we have an oracle for deciding membership in  $S$ ). By an *additive*  $(\delta, \epsilon)$ -approximation we mean that with probability  $\geq 1 - \epsilon$ ,  $|\rho - a| \leq \delta$  where  $a$  is the approximated values and  $\rho \stackrel{\text{def}}{=} |S|/2^m$  is the true density. To get this approximation, we first transform the problem of additive approximation into  $2/\delta$  problems of the form “ $P_j$ : is  $\rho \in [j \cdot \delta/2, j \cdot \delta/2 + \delta]$ ?” where  $0 \leq j \leq (2 - 2\delta)/\delta$  (notice that every pair of consecutive intervals overlaps by  $\delta/2$ ). By sampling  $k = O(\delta^{-2} \log(\epsilon\delta)^{-1})$  points in  $\{0, 1\}^m$ , and counting the number of times  $S$  is hit, every  $P_j$  has a corresponding witness set  $S_j \subset \{0, 1\}^{mk}$ . It is easy to see that for at most two consecutive  $j$ 's in the above range,  $S_j$  has more than  $(1 - \epsilon\delta)2^{kn}$  points, while all other  $j$ 's (except possibly another consecutive  $j$ ) has fewer than  $\epsilon\delta 2^{kn}$  points. Now we use the ideas above to try and hit all  $S_j$ 's by strings generated from a single probability-bounded source. With probability  $> 1 - \epsilon$ , we get positive answers only for  $j$ 's in a  $\delta$  neighbourhood of  $\rho$ . In case we get positive answers for several  $P_j$ 's, we choose the median  $j$ , and estimate  $\rho$  as being in the middle of the  $j$ th interval.

#### Appendix: Proof of Lemma 20.

LEMMA 20. Let  $0 < \delta \leq 4$  and  $G((A, B), E)$  be a bipartite graph with  $n$  vertices on each side and at least  $\frac{1}{2}n^2$  edges. Suppose that the number of edges in every subgraph of  $G$ , with  $q \stackrel{\text{def}}{=} \delta/128 \cdot n$  vertices on each side, is in the range  $[q^2/3, 2q^2/3]$ . Then there exist  $m \geq n/4$  and  $k = m/32$  such that  $G$  contains a  $k$ -regular subgraph with  $m$  vertices on each side.

*Proof.* Our proof is constructive. The construction proceeds in two phases. First, we use brute force to find a subgraph  $G'$  of  $G$ , which has  $m \geq n/4$  vertices on each side, average degree  $\geq (\frac{1}{2} - \delta/128) \cdot m$ , and minimum degree  $\geq m/8$ . Next we apply Hall's Theorem to find a spanning  $k$ -regular subgraph of the latter.

The subgraph  $G'$  is found by applying the following procedure.

1. **procedure 1: FIND LARGE SUBGRAPH  $G'$**   
; A vertex  $a \in A$  in a bipartite graph  $G((A, B), E)$  is called *bad* if its degree is  $< \frac{1}{4} \cdot |B|$ .
2. INPUT  $\leftarrow G(V, E)$   
; **Step 1—Omitting bad vertices from both sides**
3.  $G' \leftarrow G$
4. *While* both sides of  $G'(V, E')$  contain bad vertices *do begin*  
; Let  $L$  (resp.  $R$ ) be the set of bad vertices in the left (resp. right) side of  $G'$ .
5.  $\beta \leftarrow \min\{|L|, |R|\}$ ;
6. Omit  $\beta$  vertices of  $L$  and  $\beta$  vertices of  $R$  from  $G'$ .  
; The resulting graph is referred to as  $G'$ .
7. *end*;

- ; **Step 2—Omitting bad vertices from the remaining side**
- ; Let  $L$  (resp.  $R$ ) be the set of bad vertices in the left (resp. right) side of  $G'$ .  
 [By the above, either  $L$  or  $R$  is empty.  $W \log$  let  $R = \emptyset$ .]
- 8. Omit all remaining bad vertices (i.e.,  $L$ ) from the graph.
- 9. Omit, from the right side of the remaining graph,  $|L|$  vertices of minimum degree.
- 10. **return**  $G'$ .

Throughout the execution of Step (1) of the above procedure, we only omit vertices with degree not exceeding half the current average degree. Thus, at the end of Step (2) average degree in the remaining graph is no less than half the number of vertices in one side of the graph.

Let  $2t$  denote the number of vertices omitted in Step (1). Then the number of edges deleted in Step (1) is at most

$$t \cdot \frac{n}{4} + t \cdot (n - t).$$

(We charge deleted edges with a bad leftpoint to vertices omitted from the left side of the graph, while charging all other deleted edges to the vertices omitted from the right.) Using the above upper bound on  $|E - E'|$ , we get

$$\frac{n^2}{2} \leq |E| = |E - E'| + |E'| \leq t \cdot \frac{n}{4} + t \cdot (n - t) + (n - t)^2$$

which yields  $t \leq \frac{2}{3} \cdot n$ .

Now we claim that in Step (2),  $L$  could not be too big. If  $|L| \geq 3\delta/128 \cdot r$ , where  $r$  is the number of vertices in the right side of  $G'$  after Step (1), we reach contradiction by considering the subgraph induced on  $G'$  by all the vertices of the right side and the vertices in  $L$ . (This subgraph has at most  $\frac{1}{4}$  of all possible edges, contradicting the Lemma's hypothesis.) It is easy to see that after Step (2) the number of vertices on each side of the graph, denoted  $m$ , is  $\geq (1 - 3\delta/128)n/3 > n/4$ . Also, the minimum degree in the remaining graph is  $\geq r/4 - 3\delta \cdot r/128 \geq m/8$ , and the average degree is  $\geq (\frac{1}{2} - \delta/(32 \cdot 4)) \cdot m$ .

The second phase of our construction consists of finding a spanning  $k$ -regular graph of  $G_0 = G'$ . This is done by applying the following procedure.

1. **procedure 2:** FIND SPANNING  $k$ -REGULAR SUBGRAPH
2. INPUT  $\leftarrow G_0((A_0, B_0), E_0)$
3.  $R \leftarrow \emptyset$
4. For  $i = 1$  to  $k$  do begin
5. Find a perfect matching  $M_i$  in  $G_{i-1}$ .
6.  $R \leftarrow R \cup M_i$ .
7. Omit  $M_i$  from  $G_{i-1}$ , resulting in  $G_i$ .
8. end;
9. **return**  $R$ .

We now show that Procedure 2 does not fail. Assume on the contrary that in some iteration  $i + 1 \leq k$  a perfect matching is not found. Namely,  $G_i$  does not contain a perfect matching. By Hall's Theorem [7, § 5.2, p. 72], the left side of  $G_i$  (i.e.,  $A_0$ ) contains a set of vertices  $A'$  such that the neighbourhood of  $A'$  (denoted  $B'$ ) has cardinality smaller than  $|A'|$ . Since the residual degree of  $G_i$  is  $\geq m/8 - i$ , we get

$$|A'| > |B'| \geq \frac{m}{8} - k.$$

Consider the neighbourhood of a node in  $B_0 - B'$  (such a node does exist since  $|B'| < |A'| \leq |B_0|$ ). This neighbourhood has cardinality  $\geq m/8 - k$  and does not intersect with  $A'$ . We conclude that

$$|B_0 - B'| > |A_0 - A'| \geq \frac{m}{8} - k.$$

It should be noted that there are no edges in  $G_i$  between  $A'$  and  $B_0 - B'$ . Thus,  $G_0$  (or  $G$  for this matter) contain at most  $i \cdot \min\{|A'|, |B_0 - B'|\}$  edges between  $A'$  and  $B_0 - B'$ . This is at most one-third of  $|A'| \cdot |B_0 - B'|$  (since  $i < k = m/32$  and  $\min\{|A'|, |B_0 - B'|\} \geq m/8 - k = 3m/32$ ), which contradicts the lemma's hypothesis.  $\square$

**Acknowledgments.** We wish to thank Baruch Awerbuch for collaborating with us at the early stages of this research. We are very grateful to Noga Alon, László Babai and Carl Pomerance for sharing with us some of their mathematical knowledge. Noga Alon has referred us to the properties of Hadamard matrices, which turned out to be very suitable for our purposes. László Babai has referred us to the  $k$ th residues construction used in § 3 and Carl Pomerance has provided us with the estimates on the density of primes suitable for this construction. Their help enabled us to get the extraction scheme which is efficient in both rate and computation measures.

We are grateful to Jeff Lagarias for suggesting that we consider “extraction at the limit,” Nati Linial for pointing out Theorem 4, Mike Luby for pointing out that the entropy bound does suffice, Andrew Odlyzko for referring us to Paley Graph Conjecture, and Avi Wigderson for his interpretation of the result in [32].

We also wish to thank Reuven Bar-Yehuda, Shimon Even, Michael Fischer, Mihali Gerek, Shafi Goldwasser, Abraham Lempel, Leonid Levin, Silvio Micali, Ron Rivest, David Shmoys and Umesh Vazirani for very helpful discussions. Finally, we would like to thank the referees for their helpful comments.

#### REFERENCES

- [1] L. ADLEMAN, *Two theorems on random polynomial time*, Proc. 19th Annual IEEE Symposium on the Foundations of Computer Science, October 1978, pp. 75–83.
- [2] M. AJTÁI, L. BABAI, P. HAJNAL, J. KOMLÓS, P. PU DLÁK, V. RÖDL, E. SZEMERÉDI AND G. TURÁN, *Two lower bounds for branching programs*, Proc. 18th Annual ACM Symposium on the Theory of Computing, May 1986, pp. 30–38.
- [3] N. ALON, private communication, 1985.
- [4] ———, *Eigenvalues, geometric expanders, sorting in rounds and Ramsey theory*, *Combinatorica*, 6 (1986), pp. 231–243.
- [5] N. ALON, P. FRANKL AND V. RÖDL, *Geometric realization of set systems and probabilistic communication complexity*, Proc. 26th Annual IEEE Symposium on the Foundations of Computer Science, October 1985, pp. 277–280.
- [6] M. BLUM, *Independent unbiased coin flips from a correlated biased source: a finite state Markov chain*, *Combinatorica*, 6 (1986), pp. 97–108.
- [7] J. A. BONDY AND U. S. R. MURTY, *Graph Theory with Applications*, American Elsevier, New York, 1971.
- [8] E. R. CANFIELD, P. ERDÖS AND C. POMERANCE, *On a problem of Oppenheim concerning factorisation numerorum*, *J. Number Theory*, 17 (1983), pp. 1–28.
- [9] B. CHOR AND O. GOLDREICH, *Unbiased bits from sources of weak randomness and probabilistic communication complexity*, Proc. 26th Annual IEEE Symposium on the Foundations of Computer Science, October 1985, pp. 429–442.
- [10] J. D. DIXON, *Asymptotically fast factorization of integers*, *Math. Comp.*, 36 (1981), pp. 255–260.
- [11] P. ELIAS, *The efficient construction of an unbiased random sequence*, *Ann. Math. Statist.* 43 (1972), pp. 865–870.
- [12] P. ERDÖS AND J. SPENCER, *Probabilistic Methods in Combinatorics*, Academic Press, New York, 1974.
- [13] J. GILL, *Complexity of probabilistic Turing machines*, this Journal, 6 (1977), pp. 675–695.

- [14] M. HALL, JR., *Combinatorial Theory*, Blaisdell, New York, 1967.
- [15] J. JA'JA', V. K. PRASANNA KUMAR AND J. SIMON, *Information transfer under different sets of protocols*, this Journal, 13 (1984), pp. 840–849.
- [16] N. J. JOHNSON AND S. KOTZ, *Distributions in Statistics, Vol. 1, Discrete Distributions*, John Wiley, New York, 1969.
- [17] F. J. MCWILLIAMS AND N. J. A. SLOANE, *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, 1977.
- [18] J. VON NEUMANN, *Various techniques used in connection with random digits*, notes by G. E. Forsythe, Applied Math. Series Vol. 12, National Bureau of Standards, 1951, pp. 36–38, reprinted in *von Neumann's Collected Works*, Vol. 5, Pergamon Press, Elmsford, NY, 1963, pp. 768–770.
- [19] A. M. ODLYZKO, *Discrete logarithms in finite fields and their cryptographic significance*, Advances in Cryptology: Proceedings of EuroCrypt84, T. Beth et al., eds., Springer-Verlag, 1985, pp. 224–314.
- [20] A. ORLITSKY AND A. EL-GAMAL, *Randomized communication complexity*, preprint, (1985).
- [21] C. H. PAPADIMITRIOU AND M. SIPSER, *Communication complexity*, J. Comput. System Sci., 28 (1984), pp. 260–269.
- [22] C. H. PAPADIMITRIOU AND K. STEIGLITZ, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1984.
- [23] R. PATURI AND J. SIMON, *Probabilistic communication complexity*, Proc. 25th Annual IEEE Symposium on the Foundations of Computer Science, October 1984, pp. 118–126.
- [24] R. C. POHLIG AND M. HELLMAN, *An improved algorithm for computing logarithms over  $GF(p)$  and its cryptographic significance*, IEEE Trans. Inform. Theory, IT-24 1978, pp. 106–110.
- [25] V. PRATT, *Every prime has a succinct certificate*, this Journal, 8 (1975), pp. 214–220.
- [26] A. RÉNYI, *Probability Theory*, North-Holland, Amsterdam, 1970.
- [27] M. SANTHA AND U. V. VAZIRANI, *Generating quasi-random sequences from slightly-random sources*, Proc. 25th Annual IEEE Symposium on the Foundations of Computer Science, October 1984, pp. 434–440.
- [28] W. M. SCHMIDT, *Equations over Finite Fields: An Elementary Approach*, Lecture Notes in Mathematics 536, Springer-Verlag, Berlin, 1976.
- [29] U. V. VAZIRANI, *Towards a strong communication complexity theory or generating quasi-random sequences from two communicating slightly-random sources*, Proc. 17th Annual ACM Symposium on the Theory of Computing, May 1985, pp. 366–378.
- [30] ———, *Randomness, adversaries and computation*, Ph.D. dissertation, Computer Sci. Dept., Univ. of Calif., Berkeley, CA, 1986.
- [31] ———, *Efficiency considerations in using semi-random sources*, Proc. 19th Annual ACM Symposium on the Theory of Computing, May 1987, pp. 160–168.
- [32] U. V. VAZIRANI AND V. V. VAZIRANI, *Random polynomial time is equal to slightly-random polynomial time*, Proc. 26th Annual IEEE Symposium on the Foundations of Computer Science, October 1985, pp. 417–428.
- [33] ———, *Sampling a population with a single semi-random source*, Proc. 6th FST&TCS Conf., 1986.
- [34] A. C. YAO, *Probabilistic computations: towards a unified measure of complexity*, Proc. 18th Annual IEEE Symposium on the Foundations of Computer Science, October 1977, pp. 222–227.
- [35] ———, *Some complexity questions related to distributive computing*, Proc. 11th Annual ACM Symposium on the Theory of Computing, April 1979, pp. 209–213.
- [36] ———, *Lower bounds by probabilistic arguments*, Proc. 24th Annual IEEE Symposium on the Foundations of Computer Science, November 1983, pp. 420–428.

## RECONSTRUCTING TRUNCATED INTEGER VARIABLES SATISFYING LINEAR CONGRUENCES\*

ALAN M. FRIEZE†, JOHAN HASTAD‡, RAVI KANNAN§, JEFFREY C. LAGARIAS¶  
AND ADI SHAMIR||

**Abstract.** We propose a general polynomial time algorithm to find small integer solutions to systems of linear congruences. We use this algorithm to obtain two polynomial time algorithms for reconstructing the values of variables  $x_1, \dots, x_k$  when we are given some linear congruences relating them together with some bits obtained by truncating the binary expansions of the variables. The first algorithm reconstructs the variables when either the high order bits or the low order bits of the  $x_i$  are known. It is essentially optimal in its use of information in the sense that it will solve most problems almost as soon as the variables become uniquely determined by their constraints. The second algorithm reconstructs the variables when an arbitrary window of consecutive bits of the variables is known. This algorithm will solve most problems when twice as much information as that necessary to uniquely determine the variables is available. Two cryptanalytic applications of the algorithms are given: predicting linear congruential generators whose outputs are truncated and breaking the simplest version of Blum's protocol for exchanging secrets.

**Key words.** pseudorandom numbers, linear congruential generators, lattice basis reduction algorithm, cryptography

**AMS(MOS) subject classifications.** 11T71, 11K45

**1. Introduction.** The basic techniques of cryptanalysis are methods for solving various sorts of reconstruction problems. Given diverse kinds of information about a cryptosystem together with some enciphered messages, the cryptanalyst wishes to combine this information to recover the original plaintext messages, which is the *message reconstruction problem*. The cryptanalyst often accomplishes this by solving the possibly harder problem of finding the key used by the encipherer, which is the *key reconstruction problem*. From this perspective a general method of cryptanalysis is one that solves a wide class of reconstruction problems. General reconstruction methods serve as building blocks in the cryptanalysis of complex cryptosystems and also serve to set limitations of the possible design of secure cryptosystems.

This paper studies a reconstruction problem arising from the combination of two basic operations used in the design of pseudorandom number generators and cryptosystems. These two operations consist of modular arithmetic operations used as a computationally efficient way to “mix” the values of certain variables and the (non-linear) operation of truncating the binary representation of the results. A simple scheme of this type (which was used extensively on early computers) generates a pseudorandom sequence of integers by alternately squaring the previous  $n$ -bit value and discarding the top and bottom  $n/2$  bits of the  $2n$ -bit result. A related scheme is that of using the high-order bits of a linear congruential sequence, which is generally called a *truncated linear congruential pseudorandom number generator*. This was proposed by Knuth [10].

---

\* Received by the editors October 15, 1985; accepted for publication (in revised form) May 18, 1987.

† Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213.

‡ Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The work of this author was supported by an IBM fellowship.

§ Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213. The work of this author was supported in part by National Science Foundation grant ACS-8418392.

¶ AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

|| Weizmann Institute of Science, Rehovot, Israel.

The problem we consider is that of reconstructing a set  $(x_1, x_2, \dots, x_k)$  of integer variables given two sorts of information about them. First we are given a set of  $l$  modular equations

$$(1.1) \quad \sum_{j=1}^k a_{ij}x_j \equiv c_i \pmod{M} \quad \text{for } 1 \leq i \leq l,$$

that the variables satisfy. We assume that the  $a_{ij}$ ,  $c_i$  and  $M$  are known, and that the unknowns  $x_i$  satisfy the bounds

$$(1.2) \quad 0 \leq x_j < M \quad \text{for } 1 \leq j \leq k.$$

Second, we are given side information about some of the bits of the  $x_j$ , which consists of knowledge of blocks of consecutive binary digits of the variables  $x_j$ . More precisely, this partial information consists of knowledge of

$$(1.3) \quad y_j \equiv \left[ \frac{x_j}{2^{l_j}} \right] \pmod{2^{l_j}} \quad \text{for } 1 \leq j \leq k.$$

Here  $[z]$  denotes the greatest integer of a real number  $z$ . In this case we know a fraction  $\delta$  of the bits of each  $x_i$ , where  $\delta$  is given by

$$\delta = \frac{l_1}{[\log_2 M]}$$

whenever  $l_1 + l_2 \leq [\log_2 M]$ . The interesting case for cryptanalysis occurs when the number of equations  $l$  is less than the number of unknowns  $k$ . In this case the congruences  $\pmod{M}$  taken by themselves constrain the variables  $x_j \pmod{M}$  but do not determine them uniquely.

The first question to deal with is: how much side information is needed to make unique reconstruction possible? We may obtain an information-theoretic lower bound for the amount of side information required as follows. Suppose  $2^{n-1} \leq M < 2^n$  so that the  $x_j$  are  $n$  bit integers, and that we know a block of  $\delta n$  bits of each  $x_j$ . Now  $l$  modular equations  $\pmod{M}$  with side conditions  $0 \leq x_i < M$  can normally be used to eliminate  $l$  of the variables. The remaining  $k-l$  variables contain  $(k-l)n$  unknown bits of information which must be uniquely determined by the  $k\delta n$  bits given by the variables  $y_j$ . Consequently we infer that a necessary condition for unique reconstruction is that  $k\delta n \geq (k-l)n$ , which is

$$(1.4) \quad \delta \geq 1 - \frac{l}{k}.$$

In fact it turns out that the fraction  $\delta = 1 - (l/k) + \epsilon$  of the highest-order bits of each  $x_j$  suffice to guarantee unique reconstruction for the overwhelming majority of systems (1.1)–(1.3), in a sense made precise in § 2. However, there does exist a small minority of such systems which require a larger  $\delta$  than given by (1.4) to guarantee unique reconstructibility.

The main result of this paper is a general technique for solving this type of problem. It uses lattice basis reduction ideas and is guaranteed to run in polynomial time, but is not always guaranteed to produce a reconstruction. In the problems we are considering there are two major cases. The first case applies when the truncated variables  $y_j$  either consist of the highest-order bits of the  $x_j$ , or consist of the lowest-order bits of



the  $x_j$  and  $M$  is odd. We show in Theorem 2.1 that our general algorithm succeeds for “most” instances when

$$\delta > 1 - \frac{l}{k} + \varepsilon,$$

where

$$\varepsilon = \frac{c_k}{\log M}.$$

We will quantify what “most” means in § 2. The constants  $c_k$  are of size  $O(k)$ . Our second case applies to systems with arbitrarily truncated variables  $y_j$ . To be effective it requires that twice as much side information be given as that needed to guarantee uniqueness, that is,

$$\delta > 2 \left( 1 - \frac{l}{k} \right) + \varepsilon,$$

and it then works for “most” systems (Theorem 2.3). There is a small fraction of problems on which the algorithms fail, and there is a smaller fraction of exceptional problems for which unique reconstruction is not possible.

We demonstrate the usefulness of these reconstruction procedures with two applications:

(1) We show that truncated linear congruential pseudorandom number generators are cryptographically insecure in most cases.

(2) We show that the simplest version of Blum’s protocol [1] for exchanging secrets is insecure. We remark that Blum suggests other implementations of this protocol which do not seem vulnerable to the attack described here.

The two applications are described in §§ 3 and 4, respectively, where we give an analysis of our algorithm applied to these cases. The main technical difficulty is to analyze the behavior of the lattices arising in the special problems.

Some of the results of this paper appeared in preliminary form in Frieze, Kannan and Lagarias [5] and Hastad and Shamir [6].

**2. Reconstructing truncated variables satisfying linear congruences.** Let  $M$  be a given modulus and  $x_1, \dots, x_k$  unknown values in the range  $0 \leq x_j < M$  satisfying  $l$  independent linear congruences (mod  $M$ ):

$$(2.1) \quad \sum_{j=1}^k a_{ij} x_j \equiv c_i \pmod{M} \quad \text{for } 1 \leq i \leq l,$$

where  $l \leq k$ . The coefficients  $a_{ij}$  and  $c_i$  and the modulus  $M$  are assumed to be known. We are given (or somehow obtain) certain bits  $y_j$  of each  $x_j$  where

$$(2.2) \quad y_j \equiv \left\lfloor \frac{x_j}{2^{l_2}} \right\rfloor \pmod{2^{l_1}},$$

and our goal is to combine this partial knowledge with the given linear relationship to compute the remaining bits of all the  $x_j$ ’s. Our main tools to do this will come from the geometry of numbers, see [3], [4]. Let us recall some facts. A (full rank) *lattice*  $L$  is defined to be the set of points

$$L = \left\{ \mathbf{y}: \mathbf{y} = \sum_{i=1}^k n_i \mathbf{b}_i, n_i \in \mathbb{Z} \right\}$$

where the  $\mathbf{b}_i$  are linearly independent vectors in  $\mathbb{R}^k$ . The set  $\{\mathbf{b}_i; 1 \leq i \leq k\}$  is called a *basis* of  $L$  and  $k$  is the *dimension* of the lattice. The *determinant*  $d(L)$  of a lattice  $L$  is defined to be the absolute value of the determinant of a matrix whose rows are the  $\mathbf{b}_i$ . Geometrically the determinant can be interpreted as the volume of the parallelepiped spanned by the basis vectors. Using this interpretation it is possible to prove that the determinant is equal to the inverse of the density of the lattice (where the density is the average number of lattice points per unit volume). This characterization shows that the determinant is independent of the choice of basis.

We define the  $i$ th *successive minimum*  $\lambda_i = \lambda_i(L)$  of a lattice  $L$  to be the smallest radius  $r$  such that the sphere or radius  $r$  around 0 in  $\mathbb{R}^k$  contains  $i$  linearly independent points of  $L$  in it or on its boundary. We will be interested in lattices whose successive minima are roughly the same size.

We will be interested in bounding  $\lambda_k$  from above. To do so, we use the *dual lattice*  $L^*$  which is defined to be

$$L^* = \{\mathbf{y} \mid \langle \mathbf{y}, \mathbf{x} \rangle \in \mathbb{Z} \text{ for all } \mathbf{x} \in L\}.$$

It is well known that  $d(L^*) = d(L)^{-1}$ . A classical result asserts that  $\lambda_1 \lambda_k^* \leq k!$  [3, p. 371], and a recent result of Lagarias, Lenstra and Schnorr [12, Thm. 4.4] shows that  $\lambda_1^* \lambda_k \leq k^2/6$  for  $k \geq 7$ , where  $\lambda_1^*$  denotes the length of the shortest vector of  $L^*$ , and  $\lambda_1^* \lambda_k \leq k^2$  for all  $k$ . Thus a lower bound for  $\lambda_1^*$  gives the desired upper bound for  $\lambda_k$ .

The idea to use the dual lattice was suggested to us by C. P. Schnorr [17]. Our original method gave constants having a worse dependence on  $k$ .

Let us return to our problem. Let  $L(\mathbf{a}, M)$  be the lattice in  $\mathbb{R}^k$  spanned by the  $l$  vectors  $\mathbf{a}_i = (a_{i,1}, \dots, a_{i,k})$  (the coefficients of the known modular relations) in (2.1) and by the  $k$  vectors  $M\mathbf{e}_i$  where  $\mathbf{e}_i$  are the unit vectors along the coordinate axes. We will use this lattice in our algorithm, and the performance of the algorithm depends on properties of this lattice. Observe that the dual lattice  $L^* = L^*(\mathbf{a}, M)$  is

$$\left\{ \frac{1}{M} \mathbf{y} \mid \langle \mathbf{y}, \mathbf{a}_i \rangle \equiv 0 \pmod{M} \text{ for } 1 \leq i \leq l \right\},$$

where  $\langle \mathbf{y}, \mathbf{a}_i \rangle$  is the Euclidean inner product.

Let us start by giving the theorem which will be our main tool.

**THEOREM 2.1.** *The system of modular equations*

$$\sum_{j=1}^k a_{ij} x_j \equiv c_i \pmod{M}, \quad i = 1, 2, \dots, l$$

has at most one solution  $\mathbf{x} \in \mathbb{Z}^k$  satisfying the bound

$$(2.3) \quad \|\mathbf{x}\| \leq M \lambda_k^{-1} 2^{-(k/2)-1},$$

where  $\lambda_k$  is the largest successive minimum of the lattice  $L(\mathbf{a}, M)$ . If the  $a_{ij}$ ,  $c_i$  and  $M$  are known then there is a polynomial time algorithm that either finds  $\mathbf{x}$  or proves that no such  $\mathbf{x}$  exists.

*Proof.* We use a three-stage algorithm. First, we apply a lattice basis reduction algorithm to the lattice  $L = L(\mathbf{a}, M)$  of known modular relations to get modular relations with small coefficients. Second, we use size constraints on the  $x_j$  to transform these equations to equations over the integers. Third, we use these equations over the integers to recover the exact values of the  $x_j$ .

We apply the lattice basis reduction algorithm of Lenstra, Lenstra and Lovász [13] to the lattice  $L$  of modular relations to obtain a good basis. They prove the following result.

THEOREM 2.2. *There exists an algorithm, the  $L^3$ -algorithm, that when given as input a basis  $\{\mathbf{b}_i: 1 \leq i \leq k\}$  of an integer lattice  $L \subseteq \mathbb{R}^k$  finds a basis  $\{\mathbf{b}_i^*: 1 \leq i \leq k\}$  such that*

$$(2.4) \quad \|\mathbf{b}_i^*\| \leq 2^{k/2} \lambda_i(L) \quad \text{for } 1 \leq i \leq k.$$

*This algorithm always halts in  $O(n^6(\log B)^3)$  bit operations, where  $B^2 = \sum_{i=1}^k \|\mathbf{b}_i\|^2 = \sum_{i=1}^k \sum_{j=1}^k b_{ij}^2$ .*

We do not have a basis for the lattice  $L(\mathbf{a}, M)$  but we can obtain one as follows. Using a Hermite normal form reduction algorithm (see [8]) we obtain in polynomial time an integer matrix  $V$  in  $GL(k+l, \mathbb{Z})$  such that

$$\begin{bmatrix} \mathbf{w}'_1 \\ \mathbf{w}'_k \\ 0 \\ \vdots \\ 0 \end{bmatrix} = V \begin{bmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_l \\ M\mathbf{e}_1 \\ \vdots \\ M\mathbf{e}_k \end{bmatrix},$$

where the matrix on the left is in Hermite normal form and  $\{\mathbf{w}'_i: 1 \leq i \leq k\}$  is a basis of  $L(\mathbf{a}, M)$ . Now the  $L^3$ -algorithm applied to this basis produces an  $L^3$ -reduced basis  $\{\mathbf{w}_i: 1 \leq i \leq k\}$  and a unimodular matrix  $U$  in  $GL(k, \mathbb{Z})$  such that

$$\begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_k \end{bmatrix} = U \begin{bmatrix} \mathbf{w}'_1 \\ \vdots \\ \mathbf{w}'_k \end{bmatrix}$$

and

$$\|\mathbf{w}_i\| \leq 2^{k/2} \lambda_k, \quad 1 \leq i \leq k.$$

Combining these steps we obtain an integer matrix  $Y$  such that

$$(2.5) \quad \begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_k \end{bmatrix} = Y \begin{bmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_l \\ M\mathbf{e}_1 \\ \vdots \\ M\mathbf{e}_k \end{bmatrix}$$

(Alternatively the  $L^3$ -algorithm can be adapted to work on a set of generators of a lattice and produce (2.5) directly.) Now by multiplying (2.5) on the right by  $\mathbf{x}$  and reducing (mod  $M$ ) using (2.1) we obtain modular relations with small coefficients:

$$(2.6) \quad \sum_{j=1}^k w_{ij} x_j \equiv c'_i \pmod{M}, \quad 1 \leq i \leq k.$$

Note that, although we started with  $l$  modular equations in (2.1), we have now obtained a full set of  $k$  modular relations which are independent over the integers.

To perform the second stage of the algorithm we observe that

$$\left| \sum_{j=1}^k w_{ij} x_j \right| \leq \|\mathbf{w}_i\| \|\mathbf{x}\| < 2^{k/2} \lambda_k M \lambda_k^{-1} 2^{-(k/2)-1} < \frac{M}{2}.$$

Thus if we choose  $c'_i$  to satisfy  $|c'_i| < M/2$  we know that

$$\sum_{j=1}^k w_{ij} x_j = c'_i, \quad 1 \leq i \leq k,$$

holds over the integers.

Finally we solve this system of  $k$  linearly independent equations in  $k$  unknowns.  $\square$

Let us see how to use Theorem 2.1 if the variables are not small but some of the bits are known. Define for convenience

$$(2.7) \quad s_0 = \log \lambda_k + \frac{k}{2} + \frac{1}{2} \log k + 1,$$

where  $\lambda_k = \lambda_k(L(\mathbf{a}, M))$ .

COROLLARY 2.3. *The system of modular equations*

$$\sum_{j=1}^k a_{ij}x_j \equiv c_i \pmod{M}, \quad i = 1, 2, \dots, l$$

has at most one solution  $\mathbf{x}$  in which either of the following conditions holds:

- (i) The  $s_0$  most significant bits of each  $x_j$  are specified.
- (ii) The  $s_0$  least significant bits of each  $x_j$  are specified, and  $M$  is odd.

If the  $a_{ij}$ ,  $c_i$  and  $M$  are known there is a polynomial time algorithm that either finds  $\mathbf{x}$  or proves that no such  $\mathbf{x}$  exists.

*Proof.* To prove case (i) we just observe that  $x_i = x_i^{(1)} + x_i^{(2)}$  where  $x_i^{(1)}$  are the known most significant bits and  $|x_i^{(2)}| \leq M\lambda_k^{-1}2^{-(k/2)-1}\sqrt{k}^{-1}$ . Substituting the known  $x_i^{(1)}$  we come into position to use Theorem 2.1.

For the case (ii) write  $x_i = 2^{s_0}x_i^{(1)} + x_i^{(2)}$  where  $x_i^{(2)}$  are known and  $x_i^{(1)}$  satisfies the same size bounds. Since  $M$  is odd  $(2^{s_0})^{-1} \pmod{M}$  is defined, and after multiplying the equations by  $(2^{s_0})^{-1} \pmod{M}$  we can use Theorem 2.1.  $\square$

Note that the algorithm of Theorem 2.1 can be applied without knowing the value of  $\lambda_k$  or whether or not the bound (2.3) holds. To explain this, we associate with any basis  $\{\mathbf{b}_i: 1 \leq i \leq k\}$  of a lattice  $L$  in  $\mathbb{R}^k$  the quantity

$$(2.8) \quad \Delta_k(\mathbf{b}_1, \dots, \mathbf{b}_k) = \max_{1 \leq i \leq k} (\|\mathbf{b}_i\|).$$

Then by the proof of Theorem 2.1 the algorithm succeeds whenever

$$(2.9) \quad s_0 \geq \log \Delta_k(\mathbf{b}_1^*, \dots, \mathbf{b}_k^*) + \frac{1}{2} \log k + 1,$$

where  $\{\mathbf{b}_i^*: 1 \leq i \leq k\}$  is the  $L^3$ -reduced basis of the lattice  $L(\mathbf{a}, M)$  obtained in the algorithm. The bound (2.9) can be checked during the algorithm.

When can the algorithms of Corollary 2.2 be expected to succeed? This depends on the value of  $\lambda_k$ , and to get an idea how large it usually is we estimate it in the case where the modulus  $M$  is a fixed prime and the coefficients  $a_{ij}$  of the modular relations (2.1) are drawn independently from the uniform distribution on  $[0, M - 1]$ .

THEOREM 2.4. *Let  $p$  be prime. For the  $p^{kl}$  possible systems  $\mathbf{A}$  of modular equations*

$$\sum_{j=1}^k a_{ij}x_j \equiv 0 \pmod{p} \quad \text{for } 1 \leq i \leq l$$

arising by choosing

$$0 \leq a_{ij} < p \quad \text{for } 1 \leq i \leq l \text{ and } 1 \leq j \leq k$$

at least  $(1 - \varepsilon - O(p^{-1/k}))p^{kl}$  of these give rise to lattices  $L(\mathbf{A}, p)$  which have

$$(2.10) \quad \lambda_k < 5k^{3/2} \varepsilon^{-1/k} p^{1-1/k}.$$

*Proof.* We will use the previously mentioned result by Lagarias, Lenstra and Schnorr [12] that  $\lambda_1^* \lambda_k \leq k^2$  for all  $k \geq 1$ .

We estimate the probability that  $L^*$  contains a short vector. We know that  $pL^*$  is the integer lattice  $\{y \mid \langle y, \mathbf{a}_i \rangle \equiv 0 \pmod{p}; 1 \leq i \leq l\}$ . Take a sphere  $S$  centered around 0 of radius  $R$  where  $R < p$ . For any nonzero point  $\mathbf{z}$  in  $S$  the probability that  $\mathbf{z} \in pL^*$  is  $p^{-l}$ . Thus the probability that any point inside  $S$  is in  $pL^*$  is bounded by  $S_k(R) \cdot p^{-l}$ , where  $S_k(t)$  counts the number of lattice points in a  $k$ -dimensional sphere of radius  $t$  centered at the origin. Since  $S_k(t) = V_k t^k + O(kV_k t^{k-1})$  with  $V_k = \pi^{k/2} / \Gamma(k/2 + 1)$  as  $t \rightarrow \infty$  we conclude that if we choose  $R = (\pi^{k/2} / \Gamma(k/2 + 1))^{-1/k} \varepsilon^{1/k} p^{l/k}$  then  $p^{-l} S_k(R) = \varepsilon + O(p^{-l/k})$  as  $p \rightarrow \infty$ . Hence we conclude that with probability  $1 - \varepsilon - O(p^{-l/k})$  the inequality

$$\lambda_1^* \leq \frac{1}{p} \left( \frac{\pi^{k/2}}{\Gamma(k/2 + 1)} \right)^{-1/k} \varepsilon^{1/k} p^{l/k} \leq \frac{1}{5} \sqrt{k} \varepsilon^{1/k} p^{l/k-1}$$

holds, and thus

$$\lambda_k \leq 5k^{3/2} \varepsilon^{-1/k} p^{1-1/k}.$$

This completes the proof.  $\square$

A slightly weaker result than Theorem 2.4 can be proved to hold for all moduli  $M$ . We omit the details.

We may now infer that the algorithm of Corollary 2.3 succeeds in most cases for a random system (2.1) whenever the number  $s$  of known bits exceeds the information bound  $(1 - (l/k)) \log M$  by a small amount. Indeed for  $M$  a prime  $p$ , for most lattices  $L(\mathbf{a}, p)$  the bound (2.7) implies that this happens if  $s$  satisfies

$$(2.11) \quad s \geq \left(1 - \frac{l}{k}\right) \log p + \frac{k}{2} + 2 \log k + \frac{1}{k} |\log \varepsilon| + 3,$$

which exceeds the information bound by a constant depending only on the dimension  $k$  and desired failure rate  $\varepsilon$ .

In cryptanalytic applications, the set of problems (2.1) that arises may be distributed in an entirely different way than the uniform distribution studied in Theorem 2.4. For this reason, in §§ 3 and 4 we separately analyze the distributions of  $\lambda_k$  arising in our two applications. However in the absence of other information, the bound (2.11) is a useful heuristic to use.

We now describe and analyze our second algorithm, which applies to the set of modular equations

$$(2.12) \quad \sum_{j=1}^k a_{ij} x_j \equiv c_i \pmod{M}, \quad 1 \leq i \leq l$$

where we are given an arbitrarily located window of bits for each  $x_i$ . We suppose that the window of  $s$  truncated bits is from bit  $w$  to bit  $w + s - 1$ , i.e.,

$$x_i = x_i^{(1)} + 2^w x_i^{(2)} + 2^{w+s} x_i^{(3)}$$

where  $x_i^{(1)} < 2^w$ ,  $x_i^{(2)} < 2^s$  and  $x_i^{(3)} < M2^{-w-s}$ , and  $x_i^{(2)}$  is assumed to be known. Thus the unknown is  $x_i^{(1)} + 2^{w+s} x_i^{(3)}$ .

To use Theorem 2.1 we want to transform (2.12) to an equation with small unknowns. To do this we find  $a$  which satisfies

$$(2.13) \quad |a| \leq M2^{-w-s/2} \quad \text{and} \quad |a2^{w+s} \pmod{M}| \leq 2^{w+s/2}.$$

Such an  $a$  always exists and we can find it in polynomial time using the result of Lenstra [14] that there is a polynomial time algorithm for solving integer programs in

a fixed number of variables. This is because (2.13) can be written as the integer program in three variables  $(a, y_1, y_2)$  given by

$$\begin{aligned} -M2^{-w-(s/2)} &\leq a - My_1 \leq M2^{-w-(s/2)}, \\ -2^{w+(s/2)} &\leq a2^{w+s} - My_2 \leq 2^{w+(s/2)}, \\ 0 &< a < M. \end{aligned}$$

Multiplying the equation (2.12) by  $a$  and using the unknowns

$$z_i = ax_i^{(1)} + a2^{w+s}x_i^{(3)}$$

we obtain the modular equation

$$\sum_{j=1}^k a_{ij}z_j \equiv c'_i \pmod{M}, \quad 1 \leq i \leq l,$$

where the quantities

$$c'_i = a \left( c_i - 2^w \sum_{j=1}^k a_{ij}x_i^{(2)} \right)$$

are known. Now we know by (2.13) that

$$|z_i| \leq |a||x_i^{(1)}| + |a2^{w+s} \pmod{M}||x_i^{(3)}| \leq M2^{-s/2+1}.$$

Thus provided that

$$s > 2 \log \lambda_k + k + \log k + 4$$

holds that we can apply Theorem 2.1 and find  $z_i$ . If  $(a, M) = 1$  then all that remains is to compute  $a^{-1}z_i \pmod{M}$ . If  $M$  is prime then  $(a, M) = 1$  is guaranteed to hold and we have proved the following result.

**COROLLARY 2.5.** *Suppose  $M$  is prime and we are given a known system of modular equations*

$$\sum_{j=1}^k a_{ij}x_j \equiv c_i \pmod{M}, \quad 1 \leq i \leq l,$$

and a window of  $s$  truncated bits of each  $x_i$  consisting of bits  $w$  to  $w + s - 1$  where

$$s > 2 \log \lambda_k + k + \log k + 4.$$

Then there is a polynomial time algorithm that either finds a solution  $\mathbf{x}$  to the modular equations matching the truncated window data, or else proves that no such  $\mathbf{x}$  exists.

However in the case of a general modulus  $M$  we cannot assume that  $(a, M) = 1$ . There might not even exist an  $a$  such that  $(a, M) = 1$  which has the desired properties. To get around this problem we will use a different approach. We will prove a result for general  $M$  which depends on Diophantine approximation properties of the number  $M/2^{w+s}$ . Define for a real number  $\theta$  the quantity

$$(2.14) \quad \alpha(\theta, x) = \min_{\substack{m, n \in \mathbb{Z} \\ 1 \leq n \leq x}} |n\theta - m|.$$

We have the following result.

**THEOREM 2.6.** *Suppose that we are given a known system of modular equations*

$$\sum_{j=1}^k a_{ij}x_j \equiv c_i \pmod{M}, \quad i = 1, 2, \dots, l$$

and that we are given a window of  $s$  bits of each of the variables  $x_i$  whose largest bit is the  $(w + s)$ th bit. If

$$(2.15) \quad s \geq \log \lambda_k + \frac{k}{2} + \left| \log \alpha \left( \frac{M}{2^{w+s}}, \sqrt{k} 2^{(k/2)+1} \lambda_k \right) \right| + \frac{1}{2} \log k + 1$$

then the  $x_i$  are uniquely determined and can be found in polynomial time.

*Proof.* As in the proof of Theorem 2.1 we get a system of equations

$$\sum_{j=1}^k w_{ij} x_j \equiv c'_i \pmod{M}, \quad 1 \leq i \leq k,$$

where  $\|w_i\| \leq 2^{k/2} \lambda_k$ . Over the integers we can write this as

$$(2.16) \quad \sum_{j=1}^k w_{ij} x_j = c'_i + d_i M, \quad 1 \leq i \leq k,$$

where we know by the bound for the  $w_i$  that  $|d_i| \leq \sqrt{k} 2^{k/2} \lambda_k$ . Using  $x_j = x_j^{(1)} + x_j^{(2)} 2^w + x_j^{(3)} 2^{w+s}$  where  $x_j^{(2)}$  are known we get

$$(2.17) \quad \sum_{j=1}^k w_{ij} x_j^{(1)} \equiv c''_i + d_i M \pmod{2^{w+s}},$$

where the integers  $c''_i$  are known. Since  $|x_j^{(1)}| \leq 2^w$  the bounds on  $w_i$  imply that

$$(2.18) \quad \left| \sum_{j=1}^k w_{ij} x_j^{(1)} \right| \leq 2^{(k/2)+w} \lambda_k \sqrt{k}.$$

Using (2.17) we obtain

$$c''_i + d_i M \equiv z_i \pmod{2^{w+s}},$$

where  $|z_i| \leq \sqrt{k} 2^{(k/2)+w} \lambda_k$ . Now we know that each  $d_i$  satisfies the integer program in two variables  $(\tilde{d}_i, t_i)$  given by

$$(2.19a) \quad -\sqrt{k} 2^{(k/2)+w} \lambda_k \leq c'_i + \tilde{d}_i M + 2^{w+s} t_i \leq \sqrt{k} 2^{(k/2)+w} \lambda_k,$$

$$(2.19b) \quad -\sqrt{k} 2^{k/2} \lambda_k \leq \tilde{d}_i \leq \sqrt{k} 2^{k/2} \lambda_k.$$

We can find a solution  $(\tilde{d}_i, t_i)$  to this integer program in polynomial time by Lenstra [14] (see also [7]). We claim that all solutions of (2.19) have  $\tilde{d}_i = d_i$ . Suppose not, and let  $d_i^{(1)}, d_i^{(2)}$  be distinct solutions. Then their difference  $\bar{d}_i = d_i^{(1)} - d_i^{(2)}$  satisfies

$$(2.20a) \quad |\bar{d}_i M + 2^{w+s} (t_i^{(1)} - t_i^{(2)})| \leq \sqrt{k} 2^{(k/2)+w+1} \lambda_k,$$

$$(2.20b) \quad |\bar{d}_i| \leq \sqrt{k} 2^{(k/2)+1} \lambda_k.$$

Then

$$\left| \frac{\bar{d}_i M}{2^{w+s}} - r \right| \leq \sqrt{k} 2^{(k/2)-s+1} \lambda_k$$

for some integer  $r = t_i^{(1)} - t_i^{(2)}$ . Using the bound on  $\bar{d}$  and the definition of  $\alpha(M/2^{w+s}, \sqrt{k} 2^{(k/2)+1} \lambda_k)$  if  $\bar{d} \neq 0$  this yields

$$\alpha \left( \frac{M}{2^{s+w}}, \sqrt{k} 2^{(k/2)+1} \lambda_k \right) \leq \sqrt{k} 2^{(k/2)-s+1} \lambda_k.$$

This inequality contradicts the bound for  $s$  in (2.15). Consequently  $\tilde{d}_i = d_i$ . Hence we have found  $d_i$  by solving (2.19). Now we determine the  $x_j$  by solving the invertible linear system (2.16).  $\square$

To estimate the useful range of Theorem 2.6 we need information about the quantities  $\alpha(M/2^{s+w}, \sqrt{k} 2^{(k/2)+1} \lambda_k)$ . Dirichlet's theorem for Diophantine approximation (see [3, p. 165], [9]) asserts that for all real  $\theta$  one has

$$\alpha(\theta, x) \leq \frac{1}{x}$$

for integer  $x$ , and it is known that for most pairs  $(\theta, x)$  one has  $\alpha(\theta, x)$  of size about  $1/x$ . Hence one expects that for most triples  $(M, w, s)$  one has  $\alpha(M/2^{s+w}, \sqrt{k} 2^{(k/2)+1} \lambda_k) \sim k^{-1/2} 2^{-k/2} \lambda_k^{-1}$  and hence that the bound (2.15) is about twice that necessary to uniquely determine the variables  $x_i$ . The loss of efficiency of this algorithm in the information-theoretic sense arises in stage 2 of the algorithm. We are given a window of  $s$  truncated bits. The effect of the low-order bits  $y_j''$  is inflated by the coefficients  $w_{ij}$  of the reduced basis and in the integer program (2.19) they destroy the information in the bottom  $\log \lambda_k$  bits of the "window." Since we need about  $\log \lambda_k$  information bits to recover the input, the window must contain this many undestroyed bits of information, so it must have at least  $2 \log \lambda_k$  bits, i.e., its efficiency is halved.

**3. Cryptanalysis of truncated linear congruential pseudorandom number generators.** A linear congruential pseudorandom number generator is based on the recurrence

$$(3.1) \quad x_{i+1} \equiv ax_i + c \pmod{M}.$$

Several kinds of reconstruction problems relating to linear congruential generators have been studied previously. In the case where the parameters  $(a, c, M)$  are *unknown* and  $\{x_i: 1 \leq i \leq k\}$  are known, J. Boyar [2] shows that one can start predicting subsequent values of the sequence with high accuracy given a short initial segment. Her method is to find parameters  $(\hat{a}, \hat{c}, \hat{M})$  consistent with the available data (in polynomial time) and to extrapolate the sequence using these parameters. If a later disagreement occurs, the values  $(\hat{a}, \hat{c}, \hat{M})$  are changed to remain consistent with the new data. She shows that at most  $O(\log M)$  disagreements can ever occur, using this procedure. Knuth [11] considered problems arising when only truncated high-order bits  $y_i$  of the generator are known. He supposed that  $M = 2^n$  is known and that the parameters  $(a, c)$  are unknown, and he gave an attack which, when given  $\{y_i: 1 \leq i \leq k\}$  where  $y_i \equiv [x_i/2^l] \pmod{M}$ , will usually reconstruct the parameters  $(a, c)$  and seed  $x_0$  in  $O(n^2 2^{2l}/k^2)$  steps. This running time bound is exponential time, as may be seen for example in the case when half of the bits are truncated and when the number  $k$  of values  $y_i$  observed is small. Reeds [15], [16] was the first to study linear congruential generators from a cryptographic viewpoint. In [16] he studied a cryptosystem which in its simplest version enciphers the plaintext  $P_i$  as

$$E_i \equiv y_i + P_i \pmod{256},$$

where  $y_i = [257x_i/M]$  and  $x_i \equiv ax_{i-1} \pmod{M}$ . He showed how to break it in a reasonable time when both the modulus  $M = 2^{31} - 1$  and multiplier  $a = 7^5$  are *known*, using a partially known plaintext attack. His attack appears to take exponential time for general parameters  $(M, a, c)$ .

We consider here the situation in which the modulus  $M$  and multiplier  $a$  are *known*, the constant term  $c$  is *unknown* and a segment  $y_i$  of truncated high-order bits

$$(3.2) \quad y_i = \left[ \frac{x_i}{2^l} \right] \quad \text{for } 1 \leq i \leq k$$



of the linear congruential generator are given as data. We give a polynomial time reconstruction procedure and prove that it succeeds on nearly all problems in which sufficient data is available to permit unique reconstruction (Theorem 3.1), provided the modulus  $M$  is squarefree. We also prove a similar result which applies to *all* moduli  $M$ , provided that any fraction greater than one third of the bits of the original  $x_i$  is given as input (Theorem 3.5). In our analysis for simplicity we treat only the case that high-order truncated bits  $y_i$  are given as data, but our technique applies to the cases where the low-order truncated bits are given, or where an interior window of truncated bits is given. In the interior window case we would achieve only 50 percent efficiency in the use of the available information.

We will show that unique reconstruction of the sequence  $x_k$  is usually possible in the case that the parameter  $c = 0$ . The case  $c \neq 0$  is different. In the case  $c \neq 0$  we set  $x'_i = x_{i+1} - x_i$  and  $y'_i = y_{i+1} - y_i$  and observe that  $x'_i$  satisfies the recurrence

$$x'_{i+1} \equiv ax'_i \pmod{M}$$

with  $c = 0$  and  $y'_i$  is essentially a truncated version of  $x'_i$ . Now the methods of this section will show that  $x'_i$  can usually be uniquely reconstructed. However,  $x'_i$  does not determine the sequence  $x_i$  since  $x_i$  and  $x_i + d$  for any  $d$  will give the same  $x'_i$  and both are generated by linear congruential generators. In fact for small  $d$  the two sequences  $\{x_i\}$  and  $\{x_i + d\}$  will usually have the same  $s$  most significant bits, and so it is impossible to uniquely reconstruct the original  $\{x_i\}$  in this case. What we *can* do in the general case that  $c \neq 0$  is to *predict* future values of the truncated sequence  $\{y_i\}$  with great accuracy, using the  $s$  most significant bits of  $x_1$  together with the uniquely reconstructed sequence  $\{x'_i\}$ .

Now we consider the case where the parameter  $c = 0$ . The  $k$  unknowns  $\{x_i: 1 \leq i \leq k\}$  satisfy

$$x_{i+1} \equiv ax_i \pmod{M}$$

and consequently are related by the following system of  $k - 1$  independent homogeneous congruences:

$$(3.3) \quad a^{i-1}x_1 - x_i \equiv 0 \pmod{M} \quad \text{for } 2 \leq i \leq k.$$

Since we are given the high-order bits  $y_i$  of the  $x_i$  as input, we have exactly a problem of the kind analyzed in Corollary 2.2. In this case  $L(\mathbf{a}, M) = L_a$  is the lattice consisting of all vectors  $(\nu_1, \dots, \nu_k) \in \mathbb{Z}^k$  satisfying (3.3), which has as a basis the vectors

$$\begin{aligned} \mathbf{b}_1 &= (M, 0, 0, \dots, 0), \\ \mathbf{b}_2 &= (a, -1, 0, \dots, 0), \\ \mathbf{b}_3 &= (a^2, 0, -1, \dots, 0), \\ &\vdots \\ \mathbf{b}_k &= (a^{k-1}, 0, 0, \dots, -1). \end{aligned}$$

The determinant  $D = D(L_a)$  is given by

$$(3.4) \quad D(L_a) = M.$$

The analysis of the size of  $\lambda_k(L_a)$  is the only problem in applying Corollary 2.2. In the case where  $M$  is squarefree we are able to prove that  $\lambda_k$  is small for most  $L_a$ .

**THEOREM 3.1.** *For squarefree  $M > c(\varepsilon, k)$  there is an exceptional set  $E(M, \varepsilon, k)$  of multipliers of cardinality  $|E(M, \varepsilon, k)| \leq M^{1-\varepsilon}$  such that for any multiplier not in  $E(M, \varepsilon, k)$  the following is true. The  $x_i$  are uniquely determined by knowledge of the  $(1/k + \varepsilon) \log M + c(k)$  leading bits of all  $\{x_i: 1 \leq i \leq k\}$ , where*

$$c_k = \frac{k}{2} + (k-1) \log 3 + \frac{7}{2} \log k + 2.$$

Furthermore, there is an algorithm which runs in time polynomial in  $\log M + k$  which finds the  $x_i$ .

*Remarks.* (1) The number of bits needed for unique reconstruction is essentially optimal on information theory grounds, except for the presence of the  $\varepsilon$ .

(2) Some sort of exceptional set  $E(m, \varepsilon, k)$  is necessary because  $a = 1$  is always a “bad” multiplier. In the case  $a = 1$ , all the observed truncated bits  $y_i$  will be equal to the high-order bits of the seed  $x_0$ , and one never gets any information about the low-order bits of the seed. (Of course we can *extrapolate* future values of the generator very well in this case.) There are usually other multipliers  $a$  in the exceptional set  $E(M, \varepsilon, k)$  defined below, though they are in general not easy to characterize.

(3) The proof actually shows that we could take  $\varepsilon$  to be a constant times  $1/\log \log M$  as  $M \rightarrow \infty$ .

*Proof.* Our object is to apply Theorem 2.1, and our only problem is to bound the number of lattices  $L_a$  which have a large  $\lambda_k$ . We define the exceptional set

$$E(M, \varepsilon, k) = \{L_a: \lambda_k(L_a) > 2k^3 3^{k-1} M^{(1/k)+\varepsilon}\}$$

and our object will be to show that for squarefree  $M \geq c(\varepsilon, k)$  there are at most  $M^{1-\varepsilon}$  lattices  $L_a$  in the exceptional set  $E(M, \varepsilon, k)$ .

We will study the dual lattice  $L_a^*$ , which is generated by  $1/M(1, a, a^2, \dots, a^{k-1})$  and the unit vectors  $e_i, i = 1, \dots, k$ . For notational simplicity let us study  $ML_a^*$ . A short vector in  $ML_a^*$  corresponds to an integer  $t$  such that  $\{ta^i: 0 \leq i \leq k-1\}$  are all small (mod  $M$ ). For a fixed  $R$  we are interested in estimating the size of the set

$$S_R = \{a | \exists t, 1 \leq t < M, |ta^i \pmod M| < R \text{ for } 0 \leq i \leq k-1\}.$$

Define for  $d$  dividing  $M$  the sets

$$S_{R,d} = \{a: \exists t, 1 \leq t < M, (M, t) = d, |ta^i \pmod M| < R \text{ for } 0 \leq i \leq k-1\}.$$

It is clearly true that  $S_R = \bigcup_{d|M} S_{R,d}$ . Let us first estimate the size of  $S_{R,1}$ .

**LEMMA 3.2.** *If  $a \in S_{R,1}$  then  $a$  satisfies an equation*

$$(3.5) \quad \sum_{i=1}^k \nu_i a^{i-1} \equiv 0 \pmod M$$

with

$$(3.6) \quad |\nu_i| \leq (2kR)^{1/(k-1)}.$$

*Proof.* By assumption we have  $t$  such that  $(t, M) = 1$  and  $|ta^i| < R$  for  $0 \leq i \leq k-1$ . Consider all linear combinations  $\sum_{i=0}^{k-1} s_i ta^i$  with  $0 \leq s_i < (2kR)^{1/(k-1)}$  for  $0 \leq i \leq k-1$ . There are  $(2kR)^{k/(k-1)}$  such combinations and the value of any such combination is bounded in absolute value by  $kR(2kR)^{1/(k-1)}$ . Thus by the pigeonhole principle there are two different sets of  $s_i$ 's which give the same value. Subtracting the two expressions and dividing by  $t$  we get the desired solution to (3.5).  $\square$

To estimate the size of  $S_{R,1}$  we must estimate how many numbers  $a$  satisfy an equation (3.5) with small coefficients. Let  $M = \prod_{i=1}^f p_i$ . We count the number of  $a$ 's satisfying this equation having

$$(3.7) \quad d = \text{g.c.d.} (M, \nu_1 \cdots \nu_k).$$

If  $d$  is the product of  $g$  of the prime factors of  $M$  then an upper bound for the number of solutions is  $d(k-1)^{f-g}$ . The reason for this is that we have at most  $k-1$  solutions to the congruence

$$\sum_{i=1}^k \nu_i a^{i-1} \equiv 0 \pmod{p_i}$$

for each of the  $f-g$  primes  $p_i$  of  $M$  not dividing  $d$ . We next bound the number of vectors  $\mathbf{v} = (\nu_1, \dots, \nu_k)$  satisfying the condition (3.7), using the following lemma.

LEMMA 3.3. *If  $d$  divides  $M$  then the number of nonzero integer vectors satisfying*

$$\text{g.c.d.} (M, \nu_1, \dots, \nu_k) = d$$

and  $|\nu_i| \leq T$  for  $1 \leq i \leq k$  is less than  $(3T/d)^k$ .

*Proof.* Dividing all  $\nu_i$  and  $M$  by  $d$  shows it is enough to prove the lemma for  $d = 1$ . In this case the estimate follows from a trivial bound on the number of lattice points in the region considered.  $\square$

Combining the above results we get

$$\begin{aligned} |S_{R,1}| &\leq \sum_{d|M} k^{f-g} d \left( \frac{(2kR)^{1/(k-1)} \cdot 3}{d} \right)^k \\ &< k^f (2kR)^{k/(k-1)} \cdot 3^k \sum_{d|M} d^{1-k}. \end{aligned}$$

To simplify this further we use a well-known number-theoretic estimate valid for squarefree numbers  $M$  that shows that there is a constant  $c_0$  such that for  $M \geq 20$  one has

$$f \leq c_0 \frac{\log M}{\log \log M}.$$

Hence

$$k^f \leq M^{(c_0 \log k)/(\log \log M)}$$

and

$$\sum_{d|M} d^{1-k} \leq \sum_{d|M} 1 \leq 2^f \leq M^{(c_0 \log 2)/(\log \log M)},$$

giving

$$(3.8) \quad |S_{R,1}| < 3^k (2kR)^{k/(k-1)} M^{c_0 \log 2k / \log \log m}.$$

Next let us consider  $S_{R,d}$  for  $d > 1$ . Whether  $a \in S_{R,d}$  only depends on  $a \pmod{M/d}$ . To be more precise  $a \in S_{R,d}$  if and only if there exists an integer  $t$  with  $1 \leq t < M/d$ , and  $(t, M/d) = 1$  with

$$|ta^i \pmod{M/d}| \leq R/d, \quad 0 \leq i \leq k-1.$$

Reasoning as in Lemma 3.2 we find that  $a$  solves an equation with small coefficients (mod  $M/d$ ). Using that each solution (mod  $M/d$ ) lifts to at most  $d$  solutions (mod  $M$ ) we get

$$\begin{aligned} |S_{R,d}| &< d 3^k \left(\frac{2kR}{d}\right)^{k/(k-1)} M^{(c_0 \log 2k)/(\log \log M)} \\ &= 3^k d^{-1/(k-1)} (2kR)^{k/(k-1)} M^{(c_0 \log 2k)/(\log \log M)}. \end{aligned}$$

Thus we obtain

$$\begin{aligned} |S_R| &= \left| \bigcup_d S_{R,d} \right| \leq \sum_d |S_{R,d}| < 3^k (2kR)^{k/(k-1)} M^{(c_0 \log 2k)/(\log \log M)} \sum_{d|M} d^{-1/(k-1)} \\ (3.9) \quad &\leq 3^k (2kR)^{k/(k-1)} M^{(c_0 \log 2k)/(\log \log M)} 2^f \\ &\leq 3^k (2kR)^{k/(k-1)} M^{(c_0 \log 4k)/(\log \log M)}. \end{aligned}$$

Now choose  $R = M^{(k-1)/k} 3^{-(k-1)} (2k)^{-1}$ . Then (3.9) yields for  $M \geq c(\varepsilon, k)$  that

$$|S_R| < M^{1-\varepsilon},$$

and that for all  $a$  not in  $S_R$  we have

$$\lambda_1(L_a^*) \geq \frac{1}{2k} 3^{-(k-1)} M^{-1/k-\varepsilon}.$$

Then the inequality  $\lambda_1^* \lambda_k \leq k^2$  proved in [12] yields

$$(3.10) \quad \lambda_k(L_a) \leq 2k^3 3^{k-1} M^{1/k+\varepsilon}.$$

Hence  $E(M, \varepsilon, k) \subseteq S_R$  so  $|E(M, \varepsilon, k)| \leq M^{1-\varepsilon}$ .

To complete the proof of Theorem 3.1, we choose

$$s \geq \left(\frac{1}{k} + \varepsilon\right) \log M + c(k)$$

with  $c(k) = k/2 + (k-1) \log 3 + \frac{7}{2} \log k + 2$ , and apply Corollary 2.2, after observing that if  $a \notin E(M, \varepsilon, k)$  then (3.10) implies that

$$s \geq s_0 = \log \lambda_k + \frac{k}{2} + \frac{1}{2} \log k + 1. \quad \square$$

This proof of Theorem 3.1 does not carry over very well to non-squarefree moduli  $M$ , the worst case being  $M = p^e$  a prime power. The problem in that case is that polynomials (mod  $p^e$ ) may have many roots, e.g.,

$$\sum_{i=1}^k v_i a^{i-1} \equiv 0 \pmod{p^e}$$

may have up to  $kp^{e-1}$  roots, and we get a much weaker estimate for  $|S_R|$  in this case. We can still use this weaker bound to extend the proof of Theorem 3.1 to apply to moduli  $M$  which are almost squarefree. Define a number  $M$  to be  $\delta$ -squarefree if

$$M = \prod_{i=1}^f p_i^{e_i} \quad \text{and} \quad \prod_{i=1}^f p_i^{e_i-1} \leq M^\delta.$$

Then we have the following result.

**THEOREM 3.4.** *Suppose that the modulus  $M$  is  $\delta$ -squarefree and let the number of iterates  $k$  and a constant  $\varepsilon > 0$  be given. Then there exists a constant  $c_2(\varepsilon, \delta)$  such that*

for all such  $M > c_2(\epsilon, \delta)$  and all residues not in an exceptional set  $E(\epsilon, M)$  of cardinality at most  $M^{1-\epsilon}$  given knowledge of  $s$  leading bits of  $\{x_i: 1 \leq i \leq k\}$  where

$$s > \left(\frac{1}{k} + \epsilon + \delta\right) \log M + c(k)$$

suffices to determine the  $\{x_i: 1 \leq i \leq k\}$  uniquely. Furthermore, there is an algorithm that runs in time polynomial in  $\log M + k$  which always reconstructs all the  $x_i$  in this case.

The proof is essentially the same as that of Theorem 3.1, using the weaker bound for  $|S_R|$ , and we omit the details.

For the special case  $k=3$  we are able by a more careful argument to prove an essentially optimal bound valid for all moduli  $M$ .

**THEOREM 3.5.** *Given  $\epsilon > 0$ , for any  $M$  the knowledge of  $(\frac{1}{3} + \epsilon) \log M + c(k)$  leading bits of  $x_1, x_2$  and  $x_3$  allows the recovery of  $x_1, x_2$  and  $x_3$  in polynomial time for all multipliers a set of cardinality  $c(\epsilon)M^{1-\epsilon/2}$ .*

*Proof.* As we have seen the hard part of the proof will be to count the number of solutions to second-degree congruences when the modulus is highly composite. To fix notation let  $V(x) = \nu_0 + \nu_1 x + \nu_2 x^2$  and  $\mathbf{v} = (\nu_0, \nu_1, \nu_2)$  with  $\|\mathbf{v}\| = (\nu_0^2 + \nu_1^2 + \nu_2^2)^{1/2}$ . We want to estimate the size of the following set:

$$F(\mu) = \{a: 1 \leq a < M \text{ and } \exists \|\mathbf{v}\| \leq M^\mu \text{ with } V(a) \equiv 0 \pmod{M}\}.$$

Assume first that  $\text{g.c.d.}(\nu_0, \nu_1, \nu_2, M) = 1$ . Suppose  $M = \prod_{i=1}^f p_i^{e_i}$  where the  $p_i$  are distinct primes. We study the number of solutions to a quadratic equation modulo prime powers  $p^e$ . Let  $D(V) = \nu_1^2 - 4\nu_0\nu_2$  denote the discriminant of the quadratic polynomial  $V(x)$ . Note that the polynomial has a double root  $\pmod{p}$  if and only if  $D(V) \equiv 0 \pmod{p}$ . We have the following lemma.

**LEMMA 3.6.** *If  $p$  does not divide  $\text{g.c.d.}(\nu_0, \nu_1, \nu_2)$  then the number of solutions of*

$$\nu_2 x^2 + \nu_1 x + \nu_0 \equiv 0 \pmod{p^e}$$

*is at most  $2 \min(p^{\lfloor e/2 \rfloor}, p^{\lfloor r/2 \rfloor})$  where  $r$  is the largest integer such that*

$$D(V) \equiv 0 \pmod{p^r}.$$

*Proof.* We can assume that the highest-degree coefficient of  $V$  is not divisible by  $p$  since otherwise the congruence has at most one solution.

Suppose now that there is at least one solution, so that  $V$  factors as  $t(x+a)(x+b) \pmod{p^e}$  with  $(t, p) = 1$ . The discriminant  $D(V) \pmod{p^e}$  is  $t^2(a-b)^2$  and if  $s$  is the largest integer such that  $a \equiv b \pmod{p^s}$  then  $r \geq \min(2s, e)$ . We have two cases, depending on the size of  $2s$ .

If  $2s < e$  then the solutions of the congruence are precisely those  $x \equiv a \pmod{p^{e-s}}$  and those  $x \equiv b \pmod{p^{e-s}}$ . There are  $2p^s$  solutions in this case.

If  $2s \geq e$  then the solutions of the congruence are exactly those  $x \equiv a \equiv b \pmod{p^{\lfloor e/2 \rfloor}}$  and there are  $p^{\lfloor e/2 \rfloor}$  such  $x \pmod{p^e}$ .

In both these cases the bound of the lemma holds.  $\square$

We next estimate the frequency with which the condition in Lemma 3.6 is satisfied.

**LEMMA 3.7.** *Given  $\epsilon > 0$  and  $d < M^{2\mu}$  the number of  $\|\mathbf{v}\| \leq M^\mu$  that satisfy*

$$D(V) \equiv 0 \pmod{d}$$

*is  $O((1/d)M^{3\mu+\epsilon})$ .*

*Proof.* The congruence  $\nu_1^2 - 4\nu_0\nu_2 \equiv 0 \pmod{d}$  splits into the  $O((1/d)M^{2\mu})$  equations

$$\nu_1^2 - 4\nu_0\nu_2 = kd \quad \text{for } |k| \leq \frac{1}{d} M^{2\mu},$$

over the integers. For each fixed  $\nu_1$  the equation is of the form  $4\nu_0\nu_2 = c$ . If  $c \neq 0$  this equation has as many solutions as divisors of  $c$ , and it is not hard to see that this number is  $O(M^\varepsilon)$  since  $c \leq M^{2\mu}$ . The remaining case  $c = 0$  gives  $4M^\mu$  possibilities for  $\nu_0$  and  $\nu_2$  but in this case  $\nu_1$  is determined by  $k$  and hence the total number of solutions is  $O((1/d)M^{3\mu+\varepsilon})$ .  $\square$

Now we are able to estimate  $F(\mu)$  in the general case.

LEMMA 3.8. *For any  $\varepsilon > 0$  it is true that*

$$|F(\mu)| \leq c(\varepsilon) \max(M^{3\mu+\varepsilon}, M^{1/2+(3\mu/2)+\varepsilon}).$$

*Proof.* First observe that by Lemma 3.6 if  $(D(V), M) = d$  then the number of solutions to  $V(x) = 0$  is bounded by  $d2^f$ , where  $f$  is the number of prime factors of  $M$ . The contribution  $S_1(\mu)$  to  $|F(\mu)|$  from  $V(x)$  with  $\text{g.c.d.}(\nu_0, \nu_1, \nu_2, M) = 1$  and  $D(V) \neq 0$  can be estimated using Lemma 3.7 by

$$\begin{aligned} S_1(\mu) &\leq \sum_{\substack{d|M \\ d \leq M^{2\mu}}} c_1 d 2^f \left( \frac{1}{d} M^{3\mu+\varepsilon/2} \right) \\ (3.11) \qquad &\leq c_1 d(M) 2^f M^{3\mu+\varepsilon/2} \leq c_3 M^{3\mu+\varepsilon}, \end{aligned}$$

where  $d(M) = O(M^{f_0/(\log \log M)})$  denotes the number of divisors of  $M$ .

Now we let  $S_d(\mu)$  count the set of  $a$  in  $|F(\mu)|$  arising from polynomials  $V(x)$  with  $\text{g.c.d.}(M, \nu_0, \nu_1, \nu_2) = d$  and  $D(V) \neq 0$ . Dividing such an equation for  $V(x)$  by  $d$  we get a polynomial  $(1/d)V(x)$  with integer coefficients of size  $(1/d)M^\mu$  and modulus  $M/d$ , and  $\text{g.c.d.}(M/d, \nu_0/d, \nu_1/d, \nu_2/d) = 1$ . Looking at the corresponding  $F$ -set  $(\text{mod } (M/d))$  we see by the argument giving (3.11) that the  $F$ -set has cardinality  $O((1/d^3)M^{3\mu+\varepsilon})$  and furthermore that each solution to it  $(\text{mod } M/d)$  will lift to exactly  $d$  distinct solutions  $(\text{mod } M)$ . This leaves us with the bound

$$S_d(\mu) = O\left(\frac{1}{d^2} M^{3\mu+\varepsilon}\right).$$

We need also to estimate the number of  $a$  that satisfy the congruence  $V(x) \equiv 0 \pmod{M}$  with  $D(V) = 0$  and  $\|\mathbf{v}\| < M^\mu$ . If  $\text{g.c.d.}(\nu_0, \nu_1, \nu_2, M) = d$  then the congruence is

$$(3.12) \qquad \frac{\nu_2}{d} x^2 + \frac{\nu_1}{d} x + \frac{\nu_0}{d} \equiv 0 \pmod{\frac{M}{d}}.$$

By the analysis of the second case in Lemma 3.6 this congruence has at most  $2^{\omega(M/d)}(M/d)^{1/2}$  solutions  $(\text{mod } M/d)$ , where  $\omega(M/d)$  counts the number of distinct prime divisors of  $M/d$ . These lift to a total of at most  $2^{\omega(M)}\sqrt{Md}$  solutions  $x \pmod{M}$ . Since  $d \leq M^\mu$ , we have at most  $2^{\omega(M)}d(M)M^{1/2+\mu/2}$  solutions to (3.12). This is  $O(M^{1/2+\mu/2+\varepsilon/2})$  for any fixed  $\varepsilon$ , as  $M \rightarrow \infty$ . Finally the number of polynomials  $V$  that satisfy  $D(V) = 0$  and  $\|\mathbf{v}\| < M^\mu$  is  $O(M^{\mu+\varepsilon/2})$  for any fixed  $\varepsilon$ , by similar reasoning to that of Lemma 3.7.

The total count of solutions  $F(\mu)$  therefore satisfies

$$\begin{aligned} |F(\mu)| &\leq \sum_{d|M} S_d(\mu) + M^{1/2+\mu/2+\varepsilon/2} \#\{V: D(V) = 0\} \\ &\leq c_3 M^{3\mu+\varepsilon} \left( \sum_{d|M} \frac{1}{d^2} \right) + c_2 M^{1/2+3\mu/2+\varepsilon} \\ &\leq c_4 (M^{3\mu+\varepsilon} + M^{1/2+3\mu/2+\varepsilon}). \end{aligned}$$

This proves Lemma 3.8.  $\square$

Now Lemma 3.8 implies Theorem 3.5 by a proof similar to that of Theorem 3.1.  $\square$

**4. Cryptanalysis of Blum’s protocol for exchanging secrets.** Blum’s paper [1] was one of the first to deal with the issue of simultaneity in sequential processes. He proposed several versions of a protocol to enable two parties to exchange the factorizations of their two published moduli  $m_A$  and  $m_B$ , which are the products of two large primes in a fair and verifiable way. The simplest of these is as follows. Let  $n = \log m_A = \log m_B$  be the size parameter. The protocol is symmetric, and the two parties alternately perform the following steps:

- (1) Choose  $k$  random numbers  $y_1, \dots, y_k$  and send their squares modulo the opponent’s modulus to the other party.
- (2) Extract the four square roots modulo your own number of each number  $y_i^2$  received from the other party. This is possible since you know the factorization. Now write the  $4k$  square roots in a  $4k \times n$  binary matrix where the least significant bits are in the last column.
- (3) Send the  $i$ th column of the matrix to the other party (for  $i = 1, \dots, n$ ).

The idea behind this procedure is that by having one of the square roots of  $y_i^2$  at hand it is possible to check that what you receive is correct information. If  $B$  wants to cheat he can guess which square root  $A$  has and send that square root and its negation correctly while the rest are unrelated bits. The probability that such cheating would not be detected by  $A$  is  $2^{-k}$ . The security of the protocol depends on the inability of the parties to factor efficiently before all (or almost all) the columns have been exchanged. Blum stated this as an assumption in the proof of correctness in this protocol. We show that this assumption is incorrect.

**THEOREM 4.1.** *There is an algorithm which when given as input  $k$  random numbers  $y_i$  and the  $n/k + c_{k,\epsilon}$  most significant bits of all square roots of the  $y_i^2 \pmod{M}$  factors  $M$  with probability  $1 - \epsilon$ . The probability is taken over the probability distribution on the  $y_i$  and the running time of the algorithm is polynomial in  $n$  but not in  $k$ .*

*Proof.* For each  $i$  the four square roots of  $y_i^2$  can be denoted by  $y_i, -y_i, x_i \equiv ry_i$  and  $-x_i \equiv -ry_i \pmod{M}$  where  $r$  is a square root of 1  $\pmod{M}$  different from  $\pm 1$ . Since  $y_i$  is known, we can easily pair  $y_i$  and  $-y_i$  with their  $N/k + c_{k,\epsilon}$  most significant bits in the data received. However, we do not know how to pair the remaining two sets of most significant bits with  $x_i$  and  $-x_i$ , so we must guess. However, for fixed  $k$  the total number of guesses is the constant  $2^k$ . When the correct guess is made we have paired each  $x_i$  with its  $N/k + c_{k,\epsilon}$  most significant bits. Observe that if we can recover any of the  $x_i$  we can factor  $M$  since g.c.d.  $(x_i - y_i, M)$  will be nontrivial.

Since the unknown values of the  $x_i$  are fixed multiples of the known values of the  $y_i$ , they are related by  $k - 1$  modular linear equations:

$$y_i x_1 - y_1 x_i \equiv 0 \pmod{M} \quad \text{for } i = 2, \dots, k.$$

The lattice  $L_y$  spanned by the  $k - 1$  coefficient vectors

$$(y_i, 0, \dots, 0, -y_1, 0, \dots, 0)$$

together with the vectors  $M\mathbf{e}_1, \dots, M\mathbf{e}_k$  is the set of vectors

$$\left( \sum_{i=2}^k y_i v_i, -y_1 v_2, \dots, -y_1 v_k \right) + (a_1 M, \dots, a_k M)$$

for all possible choices of  $v_2, \dots, v_k$  and  $a_1, \dots, a_k$  in  $\mathbb{Z}$ . When  $y_1$  is invertible  $\pmod{M}$ , as is usually the case, we obtain the following characterization of this lattice:

$$L_y = \left\{ \mathbf{v} \in \mathbb{Z}^k \mid \sum_{i=1}^k y_i v_i \equiv 0 \pmod{M} \right\},$$

and  $L_y$  has determinant  $M$ . To apply our general technique, we only have to bound  $\lambda_k(L_y)$  from above for almost all choices of  $y$ .

As usual we do this by bounding from below the length of the shortest vector in the dual lattice  $L_y^*$ . In this case  $L_y^*$  is the lattice spanned by  $(1/M)(y_1, \dots, y_k)$  and the unit vectors  $e_i$ . We use the following lemma.

LEMMA 4.2. *Let  $\varepsilon > 0$  and  $k$  be given. Then there is a positive constant  $d_{k,\varepsilon}$  such that for a random drawing of integers  $(y_1, \dots, y_k)$  in  $(\mathbb{Z}/M\mathbb{Z})^k$  where  $M = p_1 p_2$  and  $\min(p_1, p_2) \geq \frac{1}{2} d_{k,\varepsilon} M^{1/k}$ , then with probability at least  $1 - \varepsilon$  the inequalities*

$$(4.1) \quad |ty_i \pmod{M}| \leq d_{k,\varepsilon} M^{(k-1)/k}$$

cannot be solved with  $0 < t < M$ , and so

$$\lambda_1(L_y^*) \geq d_{k,\varepsilon} M^{-1/k}.$$

*Proof.* For each fixed  $t$  and a fixed index  $i$  the probability that

$$|ty_i \pmod{M}| \leq d_{k,\varepsilon} M^{(k-1)/k}$$

is at most

$$\frac{(t, M)}{M} + 2d_{k,\varepsilon} M^{-1/k} \leq 3d_{k,\varepsilon} M^{-1/k},$$

since

$$\frac{(t, M)}{M} \leq \frac{M}{p_1} + \frac{M}{p_2} \leq d_{k,\varepsilon} M^{(k-1)/k}.$$

Since the draws for different  $i$  are independent, the probability that (4.1) holds for fixed  $t$  is  $\leq (3d_{k,\varepsilon})^k M^{-1}$ . Summing over  $0 < t < M$  the total probability that (4.1) holds is at most  $\leq (3d_{k,\varepsilon})^k$ , and we may choose  $d_{k,\varepsilon} = \frac{1}{3} \varepsilon^{1/k}$ .  $\square$

Now we complete the proof of Theorem 4.1. By Corollary 2.2 we can recover  $(x_1, \dots, x_k)$  if we know  $s_0 = \log \lambda_k + (k/2) + \frac{1}{2} \log k + 1$  significant bits of each  $x_i$ . Using Lemma 4.2 and the bound  $\lambda_1^* \lambda_k \leq k$  we obtain

$$\lambda_k(L_y) \leq k^2 d_{k,\varepsilon}^{-1} M^{1/k}.$$

Since  $n = \log M$  this yields

$$s_0 \leq \frac{n}{k} + \frac{k}{2} + \frac{3}{2} \log k - \log d_{k,\varepsilon} + 1.$$

We are given  $s_1 = n/k + c_{k,\varepsilon}$  significant bits, so on choosing

$$c_{k,\varepsilon} = \frac{k}{2} + \frac{3}{2} \log k - \log d_{k,\varepsilon} + 1$$

we can find the  $x_i$ . As pointed out earlier, this enables us to factor  $M$  by calculating g.c.d.  $(x_i - y_i, M)$  so the theorem follows.  $\square$

Theorem 4.1 shows that Blum's original protocol can be broken by somebody who only deviates from the protocol by stopping early and using this algorithm—there is no need to control the choice of random bits or to lie to the other party.

The alternative protocol in which the columns of the matrices are exchanged in reverse order (from least significant bits to most significant bits) is just as insecure, again using the algorithm of Corollary 2.2, noting that  $M$  is odd.



*Discussion.* Blum proved that his protocol is secure assuming the truth of certain (unproved) assumptions. Due to the care with which Blum listed his assumptions, it is easy to trace the source of the cryptographic weakness we exploit to the following one [1, p. 187]:

“Alice cannot use the  $100 \times k$  most significant bits,  $y_1^k, \dots, y_{100}^k$ , to split  $M_B$  any better than she can use just the  $k$  most significant bits  $y_i^k$ .”

Our attack shows that this assumption was too strong. Blum considered the possibility that his original protocol might be insecure, and in his paper he described a modified protocol in which the participants use several moduli each. A second possible modification is to ask the parties to exchange fewer columns from their matrices and to use our algorithm to factor the moduli at an earlier stage. Neither of these variants seems to be vulnerable to the cryptanalytic attack proposed in this paper.

The existence of this cryptanalytic attack demonstrates once more the extremely delicate nature of proofs of security in cryptology. It also shows the importance when proposing cryptographic protocols to clearly distinguish sources of insecurity. Blum’s paper certainly does this.

**Acknowledgments.** We thank M. Blum, J. Boyer, O. Goldreich and S. Micali for bringing this problem to our attention. We thank Shafi Goldwasser and Rick Statman for useful discussion and comments.

#### REFERENCES

- [1] M. BLUM, *How to exchange (secret) keys*, ACM Trans. Comput. Systems, 1 (1983), pp. 175–193.
- [2] J. BOYAR, *Inferring sequences produced by pseudo-random number generators*, Proc. 23rd IEEE Conference on Foundations of Computer Science, 1982, pp. 153–159.
- [3] J. W. S. CASSELS, *Geometry of Numbers*, Springer-Verlag, New York, 1971.
- [4] ———, *Rational Quadratic Forms*, Academic Press, London, 1978.
- [5] A. M. FRIEZE, R. KANNAN AND J. C. LAGARIAS, *Linear congruential generators do not produce random sequences*, Proc. 25th IEEE Symposium on Theory of Computing, 1984, pp. 480–484.
- [6] J. HASTAD AND A. SHAMIR, *The cryptographic security of truncated linearly related variables*, Proc. 17th Annual ACM Symposium on the Theory of Computing, 1985, pp. 356–362.
- [7] R. KANNAN, *Improved algorithms for integer programming and related lattice problems*, Proc. 15th Annual ACM Symposium on the Theory of Computing, 1983, pp. 193–206.
- [8] R. KANNAN AND A. BACHEM, *Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix*, this Journal, 8 (1979), pp. 499–507.
- [9] A. YA. KHINCHIN, *Continued Fractions*, University of Chicago Press, Chicago, 1964.
- [10] D. E. KNUTH, *The Art of Computer Programming*, Vol. 2, Addison-Wesley, Reading, MA, 1980.
- [11] ———, *Deciphering a linear congruential encryption*, IEEE Trans. Inform. Theory, IT-31 (1985), pp. 49–52.
- [12] J. C. LAGARIAS, H. W. LENSTRA, JR., AND C. P. SCHNORR, *Korkine–Zolotarev bases and successive minima of a lattice and its reciprocal lattice*, preprint.
- [13] A. K. LENSTRA, H. W. LENSTRA, JR., AND L. LOVÁSZ, *Factoring polynomials with integer coefficients*, Math. Ann., 261 (1982), pp. 513–534.
- [14] H. W. LENSTRA, JR., *Integer programming in a fixed number of variables*, Math. Oper. Res., 8 (1983), pp. 538–548.
- [15] J. REEDS, “Cracking” a random number generator, Cryptologia, 1 (1977), pp. 20–26.
- [16] ———, *Cracking a multiplicative congruential encryption algorithm*, in Information Linkage Between Applied Mathematics and Industry, P. C. Wong, ed., Academic Press, New York, 1979, pp. 467–472.
- [17] C. P. SCHNORR, private communication.

## A DIGITAL SIGNATURE SCHEME SECURE AGAINST ADAPTIVE CHOSEN-MESSAGE ATTACKS\*

SHAFI GOLDWASSER<sup>†</sup>, SILVIO MICALI<sup>†</sup> AND RONALD L. RIVEST<sup>†</sup>

**Abstract.** We present a digital signature scheme based on the computational difficulty of integer factorization.

The scheme possesses the novel property of being robust against an adaptive chosen-message attack: an adversary who receives signatures for messages of his choice (where each message may be chosen in a way that depends on the signatures of previously chosen messages) cannot later forge the signature of even a single additional message. This may be somewhat surprising, since in the folklore the properties of having forgery being equivalent to factoring and being invulnerable to an adaptive chosen-message attack were considered to be contradictory.

More generally, we show how to construct a signature scheme with such properties based on the existence of a “claw-free” pair of permutations—a potentially weaker assumption than the intractibility of integer factorization.

The new scheme is potentially practical: signing and verifying signatures are reasonably fast, and signatures are compact.

**Key words.** cryptography, digital signatures, factoring, chosen-message attacks, authentication, trap-door permutations, randomization

**AMS(MOS) subject classification.** 94A05

**1. Introduction.** The idea of a “digital signature” first appeared in Diffie and Hellman’s seminal paper, *New Directions in Cryptography* [DH76]. They propose that each user  $A$  publish a “public key” (used for validating signatures), while keeping secret a “secret key” (used for producing signatures). In their scheme user  $A$ ’s signature for a message  $M$  is a value which depends on  $M$  and on  $A$ ’s secret key, such that anyone can verify the validity of  $A$ ’s signature using  $A$ ’s public key. However, while knowing  $A$ ’s public key is sufficient to allow one to validate  $A$ ’s signatures, it does not allow one to easily forge  $A$ ’s signatures. They also proposed a way of implementing signatures based on “trap-door functions” (see § 2.1.1).

The notion of a digital signature is useful and is a legal replacement for handwritten signatures [LM78], [Ma79]. However, a number of technical problems arise if digital signatures are implemented using trap-door functions as suggested by Diffie and Hellman [DH76]; these problems have been addressed and solved in part elsewhere. For example, [GMY83] showed how to handle arbitrary or sparse message sets and how to ensure that if an enemy sees previous signatures (for messages that he has not chosen) it does not help him to forge new signatures (this is a “nonadaptive chosen-message attack”; see § 2.2).

The signature scheme presented here, using fundamentally different ideas than those presented by Diffie and Hellman, advances the state of the art of signature schemes with provable security properties even further; it has the following important characteristics:

- What we prove to be difficult is *forgery*, and not merely obtaining the secret key used by the signing algorithm (or obtaining an efficient equivalent algorithm).

---

\* Received by the editors August 26, 1985; accepted for publication (in revised form) June 12, 1987. This research was supported by National Science Foundation grants MCS-80-06938, DCR-8607494 and DCR-8413577, an IBM/MIT Faculty Development Award and Defense Advanced Research Projects Agency contract N00014-85-K-0125.

<sup>†</sup> Massachusetts Institute of Technology Laboratory for Computer Science, Cambridge, Massachusetts 02139.

- Forgery is proven to be difficult for a “most general” enemy who can mount an *adaptive chosen-message attack*. (An enemy who can use the real signer as “an oracle” cannot in time polynomial in the size of the public key forge a signature for any message whose signature was not obtained from the real signer.) In contrast to all previous published work on this problem, we prove the scheme invulnerable against such an adaptive attack where each message whose signature is requested may depend on all signatures previously obtained from the real signer. We believe that an adaptive chosen-message attack is the most powerful attack possible for an enemy who is restricted to using the signature scheme in a natural manner.

- The properties we prove about the new signature scheme do not depend in any way on the set of messages to be signed or on any assumptions about a probability distribution on the message set.

- Our scheme can be generalized so that it can be based on “hard” problems other than factoring whenever one can create claw-free trap-door pair generators.

Our scheme can be based on any family of pairs of claw-free permutations, yielding a signature scheme that is *invulnerable* to a chosen-message attack even if the claw-free permutations are *vulnerable* to a chosen-message attack when used to make a trap-door signature scheme (see § 2.1.1).

Fundamental ideas in the construction are the use of randomization, signing by using two authentication steps (the first step authenticates a random value which is used in the second step to authenticate the message), and the use of a treelike branching authentication structure to produce short signatures.

We note that our signature scheme is not of the simple Diffie–Hellman “trap-door” type. For example, a given message can have many signatures.

Our signature scheme is seemingly “paradoxical,” in that we prove that forgery is equivalent to factoring even if the enemy uses an *adaptive* chosen-message attack. We can restate the paradox as follows:

- Any general technique for forging signatures can be used as a “black box” in a construction that enables the enemy to factor one of the signer’s public moduli (he has two in our scheme), but

- The technique of “forging” signatures by getting the real signer to play the role of the “black box” (i.e., getting the real signer to produce some desired genuine signatures) does not help the enemy to factor either of the signer’s moduli.

Resolving this paradox was previously believed to be impossible and contradictory ([Wi80] misled by Rivest).

The rest of this paper is organized as follows. In § 2 we present definitions of what it means to “break” a signature scheme and what it means to “attack” a signature scheme. In § 3 we review previously proposed signature schemes. In § 4 we review more closely the nature of the “paradox,” and discuss how it can be resolved. Section 5 defines some useful conventions and notation, and § 6 describes the complexity-theoretic foundations of our scheme. In § 7 we give some of the fundamental notions for our signature scheme, and in § 8 we give the details. In § 9 we prove that it has the desired properties. In the last section we discuss some ways to improve the running time and memory requirements of this scheme.

**2. Fundamental notions.** To properly characterize the results of this paper, it is helpful to answer the following questions:

- What is a digital signature scheme?
- What kinds of attacks can the enemy mount against a digital signature scheme?
- What is meant by “breaking” the signature scheme?

Little attention has been devoted so far to precisely answering these questions. For instance, signature schemes have been generically called “secure” without specifying against what kind of attack. This way, it would not be surprising that “secure” signature schemes were later broken by an unforeseen attack. We hope that the classification we propose in this section may prove useful in resolving unpleasant ambiguities.

**2.1. What is a digital signature scheme?** A *digital signature scheme* contains the following components:

- A *security parameter*  $k$ , which is chosen by the user when he creates his public and secret keys. The parameter  $k$  determines a number of quantities (length of signatures, length of signable messages, running time of the signing algorithm, overall security, etc).
- A *message space*  $\mathcal{M}$ , which is the set of messages to which the signature algorithm may be applied. Without loss of generality, we assume in this paper that all messages are represented as binary strings, that is,  $\mathcal{M} \subseteq \{0, 1\}^+$ . To ensure that the entire signing process is polynomial in the security parameter, we assume that the length of the messages to be signed is bounded by  $k^c$ , for some constant  $c > 0$ .
- A *signature bound*  $B$ , which is an integer bounding the total number of signatures that can be produced with an instance of the signature scheme. This value is typically bounded above by a low-degree polynomial in  $k$ , but may be infinite.
- A *key generation algorithm*  $G$ , which any user  $A$  can use on input  $1^k$  (i.e.,  $k$  in unary) to generate in polynomial time a pair  $(P_A^k, S_A^k)$  of matching *public* and *secret* keys. The secret key is sometimes called the *trap-door information*.
- A *signature algorithm*  $\sigma$ , which produces a signature  $\sigma(M, S_A)$  for a message  $M$  using the secret key  $S_A$ . Here  $\sigma$  may receive other inputs as well. For example, in the scheme we propose first,  $\sigma$  has an additional input which is the number of previously signed messages.
- A *verification algorithm*  $V$ , which tests whether  $S$  is a valid signature for message  $M$  using the public key  $P_A$ . (That is,  $V(S, M, P_A)$  will be **true** if and only if it is valid.) Any of the above algorithms may be “randomized” algorithms that make use of auxiliary random bit stream inputs. We note that  $G$  *must* be a randomized algorithm, since part of its output is the secret key, which must be unpredictable to an adversary. The signing algorithm  $\sigma$  may be randomized—we note in particular that our signing algorithm is randomized and is capable of producing many different signatures for the same message. In general, the verification algorithm need not be randomized, and ours is not.

We note that there are other kinds of “signature” problems that are not dealt with here; the most notable being the “contract-signing problem” where two parties wish to exchange their signatures to an agreed-upon contract *simultaneously* (for example, see [BI83], [EGL82], [BGMR85]).

**2.1.1. A classical example: trap-door signatures.** To create a signature scheme, Diffie and Hellman proposed that  $A$  use a “trap-door function”  $f$ : informally, a function for which it is easy to evaluate  $f(x)$  for any argument  $x$  but for which, given only  $f(x)$ , it is computationally infeasible to find *any*  $y$  with  $f(y) = f(x)$  without the secret “trap-door” information. According to their suggestion,  $A$  publishes  $f$  and any one can validate a signature by checking that  $f(\text{signature}) = \text{message}$ . Only  $A$  possesses the “trap-door” information allowing him to invert  $f$ :  $f^{-1}(\text{message}) = \text{signature}$ . (Trap-door functions will be formally defined in § 6.) We call any signature scheme that fits into this model (i.e., uses trap-door functions and signs by applying  $f^{-1}$  to the message) a *trap-door signature scheme*.

We note that not all signature schemes are trap-door schemes, although most of the ones proposed in the literature are of this type.

**2.2. Kinds of attacks.** We distinguish two basic kinds of attacks:

- *Key-only attacks* in which the enemy knows only the real signer's public key, and
- *Message attacks* where the enemy is able to examine some signatures corresponding to either known or chosen-messages before his attempt to break the scheme.

We identify the following four kinds of message attacks, which are characterized by how the messages whose signatures the enemy sees are chosen. Here  $A$  denotes the user whose signature method is being attacked.

- *Known-message attack.* The enemy is given access to signatures for a set of messages  $m_1, \dots, m_t$ . The messages are known to the enemy but are not chosen by him.
- *Generic chosen-message attack.* Here the enemy is allowed to obtain from  $A$  valid signatures for a chosen list of messages  $m_1, \dots, m_t$  before he attempts to break  $A$ 's signature scheme. These messages are *chosen* by the enemy, but they are *fixed* and *independent* of  $A$ 's public key (for example the  $m_i$ 's may be chosen at random). This attack is *nonadaptive*: the entire message list is constructed before any signatures are seen. This attack is "generic" since it does not depend on the  $A$ 's public key; the same attack is used against everyone.
- *Directed chosen-message attack.* This is similar to the generic chosen-message attack, except that the list of messages to be signed may be created after seeing  $A$ 's public key but before any signatures are seen. (The attack is still nonadaptive.) This attack is "directed" against a particular user  $A$ .
- *Adaptive chosen-message attack.* This is more general yet: here the enemy is also allowed to use  $A$  as an "oracle"; not only may he request from  $A$  signatures of messages which depend on  $A$ 's public key but he may also request signatures of messages which depend additionally on previously obtained signatures.

The above attacks are listed in order of increasing severity, with the adaptive chosen-message attack being the most severe natural attack an enemy can mount. That the adaptive chosen-message attack is a natural one can be seen by considering the case of a notary public who must sign more-or-less arbitrary documents on demand. In general, the user of a signature scheme would like to feel that he may sign arbitrary documents prepared by others without fear of compromising his security.

**2.3. What does it mean to "break" a signature scheme?** One might say that the enemy has "broken" user  $A$ 's signature scheme if his attack allows him to do any of the following with a nonnegligible probability:

- *A total break.* Compute  $A$ 's secret trap-door information.
- *Universal forgery.* Find an efficient signing algorithm functionally equivalent to  $A$ 's signing algorithm (based on possibly different but equivalent trap-door information).
- *Selective forgery.* Forge a signature for a particular message chosen a priori by the enemy.
- *Existential forgery.* Forge a signature for at least one message. The enemy has no control over the message whose signature he obtains, so it may be random or nonsensical. Consequently this forgery may only be a minor nuisance to  $A$ .

Note that to forge a signature means to produce a *new* signature; it is not forgery to obtain from  $A$  a valid signature for a message and then claim that he has now "forged" that signature, any more than passing around an authentic handwritten signature is an instance of forgery. For example, in a chosen-message attack it does

not constitute selective forgery to obtain from the real signer a signature for the target message  $M$ .

Clearly, the kinds of “breaks” are listed above in order of decreasing severity; the least the enemy might hope for is to succeed with an existential forgery.

We say that a scheme is respectively *totally breakable*, *universally forgeable*, *selectively forgeable* or *existentially forgeable* if it is breakable in one of the above senses. Note that it is more desirable to prove that a scheme is not even existentially forgeable than to prove that it is not totally breakable. The above list is not exhaustive; there may be other ways of “breaking” a signature scheme which fit in between those listed, or are somehow different in character.

We utilize here the most realistic notion of forgery, in which we say that a forgery algorithm succeeds if it succeeds probabilistically with a nonnegligible probability. To make this notion precise, we say that the forgery algorithm succeeds if its chance of success is at least as large as one over a polynomial in the security parameter  $k$ .

To say that the scheme is “broken,” we not only insist that the forgery algorithm succeed with a nonnegligible probability, but also that it must run in probabilistic polynomial time.

We note here that the characteristics of the signature scheme may depend on its message space in subtle ways. For example, a scheme may be existentially forgeable for a message space  $\mathcal{M}$  but not existentially forgeable if restricted to a message space which is a sufficiently small subset of  $\mathcal{M}$ .

The next section exemplifies these notions by reviewing previously proposed signature schemes.

**3. Previous signature schemes and their security.** In this section we list a number of previously proposed signature schemes and briefly review some facts about their security.

(1) *Trap-door signature schemes* [DH76]. Any trap-door signature scheme is existentially forgeable with a key-only attack since a valid (message, signature) pair can be created by beginning with a random “signature” and applying the public verification algorithm to obtain the corresponding “message.” A common heuristic for handling this problem in practice is to require that the message space be sparse (i.e., requiring that very few strings actually represent messages—for example this can be enforced by having each message contain a reasonably long checksum). In this case this specific attack is not likely to result in a successful existential forgery.

(2) *Rivest–Shamir–Adleman* [RSA78]. The RSA scheme is selectively forgeable using a directed chosen-message attack, since RSA is *multiplicative*: the signature of a product is the product of the signatures. (This can be handled in practice as above using a sparse message space.)

(3) *Merkle–Hellman* [MH78]. Shamir showed the basic Merkle–Hellman “knapsack” scheme to be universally forgeable using just a key-only attack [Sh82]. (This scheme was perhaps more an encryption scheme than a signature scheme, but had been proposed for use as a signature scheme as well.)

(4) *Rabin* [Ra79]. Rabin’s signature scheme is totally breakable if the enemy uses a directed chosen-message attack (see § 4). However, for nonsparse message spaces selective forgery is as hard as factoring if the enemy is restricted to a known-message attack.

(5) *Williams* [Wi80]. This scheme is similar to Rabin’s. The proof that selective forgery is as hard as factoring is slightly stronger, since here only a single instance of selective forgery guarantees factoring (Rabin needed a probabilistic argument).

Williams uses effectively (as we do) the properties of numbers which are the product of a prime  $p \equiv 3 \pmod{8}$  and a prime  $q \equiv 7 \pmod{8}$ . Again, this scheme is totally breakable with a directed chosen-message attack.

(6) *Lieberherr* [Li81]. This scheme is similar to Rabin's and Williams', and is totally breakable with a directed chosen-message attack.

(7) *Shamir* [Sh78]. This knapsack-type signature scheme has recently been shown by Tulpan [Tu84] to be universally forgeable with a key-only attack for any practical values of the security parameter.

(8) *Goldwasser-Micali-Yao* [GMY83]. This paper presents for the first time signature schemes which are not of the trap-door type, and which have the interesting property that their security characteristics hold for *any* message space. The first signature scheme presented in [GMY83] was proven not to be even existentially forgeable against a *generic* chosen-message attack unless factoring is easy. However, it is not known to what extent *directed* chosen-message attacks or adaptive chosen-message attacks might aid an enemy in "breaking" the scheme.

The second scheme presented there (based on the RSA function) was also proven not to be even existentially forgeable against a generic chosen-message attack. This scheme may also resist existential forgery against an adaptive chosen-message attack, although this has not been proved. (A proof would require showing certain properties about the density of prime numbers and making a stronger intractability assumption about inverting RSA.) We might note that, by comparison, the scheme presented here is much faster, produces much more compact signatures, and is based on much simpler assumptions (only the difficulty of factoring or more generally the existence of claw-free permutation pair generators).

Several of the ideas and techniques presented in [GMY83], such as bit-by-bit authentication, are used in the present paper.

(9) *Ong-Schnorr-Shamir* [OSS84a]. Totally breaking this scheme using an adaptive chosen-message attack has been shown to be as hard as factoring. However, Pollard [Po84] has recently been able to show that the "OSS" signature scheme is universally forgeable in practice using just a key-only attack; he developed an algorithm to forge a signature for any given message without obtaining the secret trap-door information. A more recent "cubic" version has recently been shown to be universally forgeable in practice using just a key-only attack (also by Pollard). An even more recent version [OSS84b] based on polynomial equations was similarly broken by Estes, Adleman, Kompella, McCurley and Miller [EAKMM85] for quadratic number fields.

(10) *El-Gamal* [EG84]. This scheme, based on the difficulty of computing discrete logarithms, is existentially forgeable with a generic message attack and selectively forgeable using a directed chosen-message attack.

(11) *Okamoto-Shiraishi* [OS85]. This scheme, based on the difficulty of solving quadratic inequalities modulo a composite modulus, was shown to be universally forgeable by Brickell and DeLaurentis [BD85].

**4. The paradox of proving signature schemes secure.** The paradoxical nature of signature schemes which are provably secure against chosen-message attacks made its first appearance in Rabin's paper, *Digitalized Signatures as Intractable as Factorization* [Ra79]. The signature scheme proposed there works as follows. User  $A$  publishes a number  $n$  which is the product of two large primes. To sign a message  $M$ ,  $A$  computes as  $M$ 's signature one of  $M$ 's square roots modulo  $n$ . (When  $M$  is not a square modulo  $n$ ,  $A$  modifies a few bits of  $M$  to find a "nearby" square.) Here signing is essentially just extracting square roots modulo  $n$ . Using the fact that extracting square roots

modulo  $n$  enables one to factor  $n$ , it follows that selective forgery in Rabin's scheme is equivalent to factoring if the enemy is restricted to at most a known-message attack.

However, it is true (and was noticed by Rabin) that an enemy might totally break the scheme using a directed chosen-message attack. By asking  $A$  to sign a value  $x^2 \bmod n$  where  $x$  was picked at random, the enemy would obtain with probability  $\frac{1}{2}$  another square root  $y$  of  $x^2$  such that  $\gcd(x+y, n)$  was a prime factor of  $n$ .

Rabin suggested that one could overcome this problem by, for example, having the signer concatenate a fairly long randomly chosen pad  $U$  to the message before signing it. In this way the enemy cannot force  $A$  to extract a square root of any particular number.

However, the reader may now observe that the proof of the equivalence of selective forgery to factoring no longer works for the modified scheme. That is, being able to selectively forge no longer enables the enemy to directly extract square roots and thus to factor. Of course, breaking this equivalence was really the whole point of making the modification.

**4.1. The paradox.** We now “prove” that it is impossible to have a signature scheme for which it is both true that forgery is provably equivalent to factoring, and yet the scheme is invulnerable to adaptive chosen-message attacks. The argument is essentially the same as the one given in [Wi80]. By *forgery* we mean in this section any of universal, selective, or existential forgery; we assume that we are given a proof that forgery of the specified type is equivalent to factoring.

Let us begin by considering this given proof. The main part of the proof presumably goes as follows: given a subroutine for forging signatures, a constructive method is specified for factoring. (The other part of the equivalence, which shows that factoring enables forgery, is usually easy, since factoring usually enables the enemy to totally break the scheme.)

But it is trivial then to show that an adaptive chosen-message attack enables an enemy to totally break the scheme. The enemy merely executes the constructive method for factoring given in the proof, using the real signer instead of the forgery subroutine! That is, whenever he needs to execute the forgery subroutine to obtain the signature of a message, he merely performs an “adaptive chosen-message attack” step—getting the real user to sign the desired message. In the end the unwary user has enabled the enemy to factor his modulus! (If the proof reduces factoring to universal or selective forgery, the enemy has to get the real user to sign a particular message. If the proof reduces factoring to existential forgery, the enemy need only get him to sign anything at all.)

**4.2. Breaking the paradox.** How can one hope to get around the apparent contradictory natures of equivalence to factoring and invulnerability to an adaptive chosen-message attack?

The key idea in resolving the paradox is to have the constructive proof that forgery is as hard as factoring be a *uniform* proof which makes *essential* use of the fact that the forger can forge for *arbitrary* public keys with a nonnegligible probability of success. However, in “real life” a signer will only produce signatures for a *particular* public key. Thus the constructive proof cannot be applied in “real life” (by asking the real signer to unwittingly play the role of the forger) to factor.

In our scheme this concept is implemented using the notion of “random rooting.” Each user publishes not only his two composite moduli  $n_1$  and  $n_2$ , but also a “random root”  $r$ . This value  $r$  is used when validating the user's signatures. The paradox is resolved in our case as follows:



- It is provably equivalent to factoring for an enemy to have a *uniform* algorithm for forging; uniform in the sense that if for all pairs of composite numbers  $n_1$  and  $n_2$  if the enemy can randomly forge signatures for a significant fraction of the possible random roots  $r$ , then he can factor either  $n_1$  or  $n_2$ .

- The above proof *requires* that the enemy be able to pick  $r$  himself—the forgery subroutine is fed triples  $(n_1, n_2, r)$ , where the  $r$  part is chosen by the enemy according to the procedure specified in the constructive proof. However, in “real life” the user has picked a *fixed*  $r$  at random to put in his public key, so an adaptive chosen-message attack will not enable the enemy to “forge” signatures corresponding to any other values of  $r$ . Thus the constructive method given in the proof cannot be applied! More details can be found in § 9.

## 5. General notation and conventions.

**5.1. Notation and conventions for strings.** Let  $\alpha = \alpha_0\alpha_1 \cdots \alpha_x$  be a binary string, then  $\bar{\alpha}$  will denote the integer  $\sum_{k=0}^x \alpha_k 2^{x-k}$ . (Note that a given integer may have several denotations, but only one of a given length.) The strings in  $\{0, 1\}^*$  are ordered as follows: if  $\alpha$  and  $\beta$  are binary strings, we write  $\alpha < \beta$  if there exists a string  $\gamma$  such that  $\alpha$  is a prefix of  $\gamma$ ,  $\gamma$  has exactly the same length as  $\beta$ , and  $\bar{\gamma} < \bar{\beta}$ .

If  $i$  is a  $k$ -bit string, we let  $\text{DFS}(i) = \{\beta \mid \beta \preceq i\}$ . (Imagine a full binary tree of depth  $k$  whose root is labelled  $\varepsilon$ , and the left (right) son of a node labelled  $\alpha$  is  $\alpha 0$  ( $\alpha 1$ ) and let  $\text{DFS}$  be the Depth First Search algorithm that starts at the root and explores the left son of any node before the right son of that node. Then  $\text{DFS}(i)$  represents the set of nodes visited by  $\text{DFS}$  up to and including the time when it reaches node  $i$ .) Note that  $\text{DFS}(i)$  contains the empty string.

**5.2. Notation and conventions for probabilistic algorithms.** We introduce some generally useful notation and conventions for discussing probabilistic algorithms. (We make the natural assumption that all parties, including the enemy, may make use of probabilistic methods.)

We emphasize the number of inputs received by an algorithm as follows. If algorithm  $A$  receives only one input we write “ $A(\cdot)$ ,” if it receives two inputs we write “ $A(\cdot, \cdot)$ ,” and so on.

We write “PS” for “probability space;” in this paper we only consider countable probability spaces. In fact, we only deal with probability spaces arising from probabilistic algorithms.

If  $A(\cdot)$  is a probabilistic algorithm then, for any input  $i$ , the notation  $A(i)$  refers to the PS which assigns to the string  $\sigma$  the probability that  $A$ , on input  $i$ , outputs  $\sigma$ . We point out the special case that  $A$  takes no inputs; in this case the notation  $A$  refers to the algorithm itself, whereas the notation  $A(\cdot)$  refers to the PS defined by running  $A$  with no input. If  $S$  is a PS, we denote by  $\mathbf{P}_S(e)$  the probability that  $S$  associates with element  $e$ . Also, we denote by  $[S]$  the set of elements which  $S$  gives positive probability. In the case that  $[S]$  is a singleton set  $\{e\}$  we will use  $S$  to denote the value  $e$ ; this is in agreement with traditional notation. (For instance, if  $A(\cdot)$  is an algorithm that, on input  $i$ , outputs  $i^3$ , then we may write  $A(2) = 8$  instead of  $[A(2)] = \{8\}$ .)

If  $f(\cdot)$  and  $g(\cdot, \cdots)$  are probabilistic algorithms then  $f(g(\cdot, \cdots))$  is the probabilistic algorithm obtained by composing  $f$  and  $g$  (i.e., running  $f$  on  $g$ 's output). For any inputs  $x, y, \cdots$  the associated probability space is denoted  $f(g(x, y, \cdots))$ .

If  $S$  is a PS, then  $x \leftarrow S$  denotes the algorithm which assigns to  $x$  an element randomly selected according to  $S$ ; that is,  $x$  is assigned the value  $e$  with probability  $\mathbf{P}_S(e)$ .

The notation  $\mathbf{P}(p(x, y, \dots) | x \leftarrow S; y \leftarrow T; \dots)$  will then denote the probability that the predicate  $p(x, y, \dots)$  will be true, after the (ordered) execution of the algorithms  $x \leftarrow S, y \leftarrow T$ , etc.

We let  $\mathcal{RA}$  denote the set of probabilistic polynomial-time algorithms. We assume that a natural representation of these algorithms as binary strings is used. By  $1^k$  we denote the binary representation of integer  $k$ , i.e.,

$$\underbrace{11 \cdots 1}_k.$$

**6. The complexity theoretic basis of the new scheme.** A particular instance of our scheme can be constructed if integer factorization is computationally difficult. However, we will present our scheme in a general manner without assuming any particular problem to be intractable. This clarifies the exposition, and helps to establish the true generality of the proposed scheme. We do this by introducing the notion of a “claw-free permutation pair,” and by showing the existence of such objects under the assumption that integer factorization is difficult.

This section builds up the relevant concepts and definitions in stages. In § 6.1 we give a careful definition of the notions of a trap-door permutation and a trap-door permutation generator. These notions are not directly used in this paper, but serve as a simple example of the use of our notation. (Furthermore, no previous definition in the literature was quite so comprehensive.) The reader may, if he wishes, skip § 6.1 without great loss.

In § 6.2 we define claw-free permutation pairs and claw-free permutation pair generators.

In § 6.3 we show how to construct claw-free permutation pair generators under the assumption that factoring is difficult.

Finally, in § 6.4 we show how to construct an infinite family of pairwise claw-free permutations, given a generating pair  $f_0, f_1$ , of claw-free permutations.

Altogether, then, this section provides the underlying definitions and assumptions required for constructing our signature scheme. The actual construction of our signature scheme will be given in §§ 7 and 8.

**6.1. Trap-door permutations.** Informally, a family of trap-door permutations is a family of permutations  $f$  possessing the following properties:

- It is easy, given an integer  $k$ , to randomly select permutations  $f$  in the family which have  $k$  as their security parameter, together with some extra “trap-door” information allowing easy inversion of the permutations chosen.
- It is hard to invert  $f$  without knowing  $f$ ’s trapdoor.

We can interpret the two properties above by saying that any user  $A$  can easily randomly select a pair of permutations,  $(f, f^{-1})$ , inverses of each other. This will enable  $A$  to easily evaluate and invert  $f$ ; if now  $A$  publicizes  $f$  and keeps secret  $f^{-1}$ , then inverting  $f$  will be hard for all other users.

In the informal discussion above, we used the terms “easy” and “hard.” The term “easy” can be interpreted as “in polynomial time”; “hard,” however, is of more difficult interpretation. By saying that  $f$  is hard to invert we cannot possibly mean that  $f^{-1}$  cannot be easily evaluated at any of its arguments.<sup>1</sup> We mean, instead, that  $f^{-1}$  is hard to evaluate at a *random* argument. Thus, if one wants (as we do) to use trap-door functions to generate problems computationally hard for an “adversary,” he must be

<sup>1</sup> For example, any  $f$  can be easily inverted at the image of a fixed argument, say 0. In fact, we may consider inverting algorithms that, on inputs  $x$  and  $f$ , first check whether  $x = f(0)$ .

able to randomly select a point in the domain of  $f$  and  $f^{-1}$ . This operation is easy for all currently known candidates of a trap-door permutation, and we explicitly assume it to be easy in our formal treatment.

DEFINITION. Let  $G$  be an algorithm in  $\mathcal{RA}$  that on input  $1^k$ , outputs an ordered triple  $(d, f, f^{-1})$  of algorithms. (Here  $D = [d(\cdot)]$  will denote the domain of the trap-door permutation  $f$  and its inverse  $f^{-1}$ .) We say that  $G$  is a *trap-door permutation generator* if there is a polynomial  $p$  such that

(1) Algorithm  $d$  always halts within  $p(k)$  steps and defines a uniform probability distribution over the finite set  $D = [d(\cdot)]$ . (That is, running  $d$  with no inputs uniformly selects an element from  $D$ .)

(2) Algorithms  $f$  and  $f^{-1}$  halt within  $p(k)$  steps on any input  $x \in D$ . (For inputs  $x$  not in  $D$ , the algorithms  $f$  and  $f^{-1}$  either loop forever or halt and print an error message that the input is not in the appropriate domain.) Furthermore, the functions  $x \mapsto f(x)$  and  $x \mapsto f^{-1}(x)$  are permutations of  $D$  which are inverses of each other.

(3) For all (inverting) algorithms  $I(\cdot, \cdot, \cdot, \cdot) \in \mathcal{RA}$ , for all  $c$  and sufficiently large  $k$ :

$$P(y = f^{-1}(z) | (d, f, f^{-1}) \leftarrow G(1^k); z \leftarrow d(\cdot); y \leftarrow I(1^k, d, f, z)) < k^{-c}.$$

We make the following informal remarks corresponding to parts of the above definition.

(1) This condition makes it explicit that it is possible to sample the domain of  $f$  in a uniform manner.

(3) This part of the definition states that if we run the experiment of generating  $(d, f, f^{-1})$  using the generator  $G$  and security parameter  $k$ , and then randomly generating an element  $z$  in the range of  $f$ , and then running the “inverting” algorithm  $I$  (for polynomially in  $k$  many steps) on inputs  $d, f$ , and  $z$ , the chance that  $I$  will successfully invert  $f$  at the point  $z$  is vanishingly small as a function of  $k$ .

DEFINITION. If  $G$  is a trap-door permutation generator, we say that  $[G(1^k)]$  is a *family of trap-door permutations*. We say that  $f$  and  $f^{-1}$  are *trap-door permutations* if  $(d, f, f^{-1}) \in [G(1^k)]$  for some  $k$  and trap-door permutation generator  $G$ .

**6.2. “Claw-free” permutation pairs.** The signature scheme we propose is based on the existence of “claw-free” permutation pairs; informally, these are permutations  $f_0$  and  $f_1$  over a common domain for which it is computationally infeasible to find a triple  $x, y$ , and  $z$  such that  $f_0(x) = f_1(y) = z$  (a “claw” or “ $f$ -claw”—see Fig. 1).

DEFINITION. Let  $G$  be an algorithm in  $\mathcal{RA}$  that, on input  $1^k$ , outputs an ordered quintuple  $(d, f_0, f_0^{-1}, f_1, f_1^{-1})$  of algorithms. We say that  $G$  is a *claw-free permutation pair generator* if there is a polynomial  $p$  such that:

(1) Algorithm  $d$  always halts within  $p(k)$  steps and defines a uniform probability distribution over the finite set  $D = [d(\cdot)]$ .

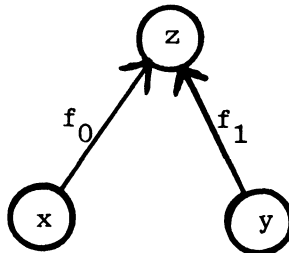


FIG. 1. A claw.

(2) Algorithms  $f_0, f_0^{-1}, f_1$  and  $f_1^{-1}$  halt within  $p(k)$  steps on any input  $x \in D$ . (For inputs  $x$  not in  $D$ , these algorithms either loop forever or halt with an error message that the input is not in the necessary domain.) Furthermore, the functions  $x \mapsto f_0(x)$  and  $x \mapsto f_0^{-1}(x)$  are permutations of  $D$  which are inverses of each other, as are  $x \mapsto f_1(x)$  and  $x \mapsto f_1^{-1}(x)$ .

(3) For all (claw-making) algorithms  $I(\cdot, \cdot, \cdot, \cdot) \in \mathcal{RA}$ , for all  $c$  and sufficiently large  $k$ :

$$\mathbf{P}(f_0(x) = f_1(y) = z \mid (d, f_0, f_0^{-1}, f_1, f_1^{-1}) \leftarrow G(1^k); (x, y, z) \leftarrow I(1^k, d, f_0, f_1)) < k^{-c}.$$

*Note.* It would be possible to use a variant of the above definition, in which the function  $f$  may actually return answers for inputs outside of  $D$  as long as it is understood that the difficulty of creating a “claw” applies to all  $x, y$  for which the function  $f$  returns an answer. Thus, it should be hard to find *any* triplet  $(x, y, z)$  such that  $f_0(x) = f_1(y) = z$  even when  $x, y$  are not in  $D$ . We do not pursue this variation further in this paper.

**DEFINITION.** We say that  $f = (d, f_0, f_1)$  is a *claw-free permutation pair* (or *claw-free pair* for short) if  $(d, f_0, f_0^{-1}, f_1, f_1^{-1}) \in [G(1^k)]$  for some  $k$  and claw-free permutation pair generator  $G$ . In this case,  $f^{-1}$  will denote the pair of permutations  $(f_0^{-1}, f_1^{-1})$ .

**6.2.1. Claw-free permutation pairs versus trap-door permutations.** In this subsection we clarify the relation between the notions of claw-free permutation pairs and trap-door permutations, by showing that the existence of the former ones implies the existence of the latter ones. (Since trap-door permutations are not used in our signature scheme, this subsection can be skipped by the reader without loss of clarity.)

**CLAIM.** Let  $G \in \mathcal{RA}$  be a claw-free permutation generator. Then there exists a  $\bar{G} \in \mathcal{RA}$  which is a trap-door permutation generator.

*Proof.* The algorithm  $\bar{G}$  is defined as follows on input  $1^k$ : Run  $G$  on input  $1^k$ . Say,  $G$  outputs the ordered tuple  $(d, f_0, f_0^{-1}, f_1, f_1^{-1})$ . Then,  $\bar{G}$  outputs  $(d, f_0, f_0^{-1})$ .

We now show that  $\bar{G}$  is a trap-door permutation generator. Assume for contradiction that it is not the case. Namely, there exists a constant  $c > 0$  and an inverting algorithm  $\bar{I}(\cdot, \cdot, \cdot, \cdot) \in \mathcal{RA}$  such that for infinitely many  $k$ :

$$\mathbf{P}(f_0(y) = z \mid (d, f_0, f_0^{-1}) \leftarrow \bar{G}(1^k); z \leftarrow d(\cdot); y \leftarrow \bar{I}(1^k, d, f_0, z)) \geq k^{-c}.$$

Note now, that since  $f_1$  is a permutation, algorithms  $f_1(d(\cdot))$  and  $d(\cdot)$  both define the uniform probability distribution over  $[d(\cdot)]$ . Thus, for infinitely many  $k$ ,

$$\begin{aligned} \mathbf{P}(f_1(x) = f_0(y) \\ = z \mid (d, f_0, f_0^{-1}, f_1, f_1^{-1}) \leftarrow G(1^k); x \leftarrow d(\cdot); z \leftarrow f_1(x); y \leftarrow \bar{I}(1^k, d, f_0, z)) \geq k^{-c}. \end{aligned}$$

Let  $I(\cdot, \cdot, \cdot, \cdot)$  be the following inverting algorithm: On input  $1^k, d, f_0$  and  $f_1$ , compute  $x \leftarrow d(\cdot), z \leftarrow f_1(x), y \leftarrow \bar{I}(1^k, d, f_0, z)$  and output  $(x, y, z)$ .

Then,  $I$  is in  $\mathcal{RA}$  and for infinitely many  $k$ ,

$$\mathbf{P}(f_0(x) = f_1(y) = z \mid (d, f_0, f_0^{-1}, f_1, f_1^{-1}) \leftarrow G(1^k); (x, y, z) \leftarrow I(1^k, d, f_0, f_1)) \geq k^{-c}.$$

This contradicts  $G$  being a claw-free permutation generator and thus  $\bar{G}$  must be a trap-door permutation generator.  $\square$

We note, however, that the converse to the above claim may be false. For example, the pair of (“RSA”) permutations over  $Z_n^* = \{1 \leq x \leq n: \gcd(x, n) = 1\}$ , defined by

$$f_0(x) \equiv x^3 \pmod{n} \quad \text{and} \quad f_1(x) \equiv x^5 \pmod{n}$$

(where  $\gcd(\phi(n), 15) = 1$ ) is not claw-free: since the two functions commute it is easy

to create a claw by choosing  $w$  at random and then defining  $x \equiv f_1(w)$ ,  $y \equiv f_0(w)$ , and  $z \equiv f_0(x) \equiv f_1(y) \equiv w^{15} \pmod{n}$ .

However, it is likely that  $f_0$  and  $f_1$  are trap-door permutations.

In practice, one may want to relax the definition of a claw-free permutation pair generator slightly, to allow the generator to have a very small chance of outputting functions  $f_0$  and  $f_1$  which are not permutations. We do not pursue this line of development in this paper.

**6.3. Claw-free permutations exist if factoring is hard.** The assumption of the existence of claw-free pairs is made in this paper in a general manner, independent of any particular number-theoretic assumptions. Thus, instances of our scheme may be secure even if factoring integers turns out to be easy. However, for concretely implementing our scheme the following is suggested.

We first make an assumption about the intractability of factoring, and then exhibit a claw-free permutation pair generator based on the difficulty of factoring.

*Notation.* Let

$$H_k = \{n = p \cdot q \mid |p| = |q| = k, p \equiv 3 \pmod{8}, q \equiv 7 \pmod{8}\}$$

(the set of composite numbers which are the product of two  $k$ -bit primes which are both congruent to 3 modulo 4 but not congruent to each other modulo 8), and let  $H = \bigcup_k H_k$ .

*Remark.* One way to choose “hard” instances for all known factoring algorithms seems to be to choose  $k$  to be large enough and then to choose  $n$  randomly from  $H_k$ .

These numbers were used in [Wi80] and their wide applicability to cryptography was demonstrated by Blum in [Bl82]; hence, they are commonly referred to as “Blum integers.”

Let  $Q_n$  denote the set of quadratic residues  $\pmod{n}$ . We note that for  $n \in H$ :

- 1 has Jacobi symbol +1 but is not in  $Q_n$ ;
- 2 has Jacobi symbol -1 (and is not in  $Q_n$ ).

We also note that every  $x \in Q_n$  has exactly one square root  $y \in Q_n$ , but has four square roots  $y, -y, w, -w$  altogether (see [Bl82] for proof). Roots  $w$  and  $-w$  have Jacobi symbol -1, while  $y$  and  $-y$  have Jacobi symbol +1.

The following assumption about the intractability of factoring is made throughout this section.

**Intractability Assumption for Factoring (IAF).** Let  $A$  be a probabilistic polynomial-time (factoring) algorithm. Then for all constants  $c > 0$  and sufficiently large  $k$

$$P(x \text{ is a nontrivial divisor of } n \mid n \leftarrow H_k(\cdot); x \leftarrow A(n)) < \frac{1}{k^c}.$$

(Here we have used the notation  $n \leftarrow H_k(\cdot)$  to denote the operation of selecting an element of  $H_k$  uniformly at random.)

Define  $f_{0,n}$  and  $f_{1,n}$  as follows:

$$f_{0,n}(x) = \begin{cases} x^2 \pmod{n} & \text{if } x^2 \pmod{n} < n/2, \\ -x^2 \pmod{n} & \text{if } x^2 \pmod{n} \geq n/2, \end{cases}$$

$$f_{1,n}(x) = \begin{cases} 4x^2 \pmod{n} & \text{if } 4x^2 \pmod{n} < n/2, \\ -4x^2 \pmod{n} & \text{if } 4x^2 \pmod{n} \geq n/2. \end{cases}$$

The common domain of these functions is

$$D_n = \left\{ x \in Z_n \mid \left( \frac{x}{n} \right) = 1 \text{ and } 0 < x < n/2 \right\};$$

it is easy to see that the range of these functions is included in  $D_n$  for  $n \in H$ . Note also that it is easy to test whether or not a given element  $x$  is a member of  $D_n$ , since Jacobi symbols can be evaluated in polynomial time.

We now show that  $f_{0,n}$  and  $f_{1,n}$  are actually permutations of  $D_n$  for  $n \in H$ . Suppose  $f_{0,n}$  is not a permutation of  $D_n$ ; then there exist distinct elements  $x, y$  in  $D_n$  such that  $f_{0,n}(x) = f_{0,n}(y)$ . This can only happen if  $x^2 \equiv y^2 \pmod{n}$ , which would imply that  $x \equiv \pm y \pmod{n}$ . But this is impossible if  $x$  and  $y$  are both in  $D_n$ , thus proving that  $f_{0,n}$  is a permutation. The proof for  $f_{1,n}$  is similar.

Not only are  $f_{0,n}$  and  $f_{1,n}$  permutations of  $D_n$  when  $n \in H$ , but their inverses are easily computed, given knowledge of  $p$  and  $q$ . Given  $p$  and  $q$ , it is easy to distinguish quadratic residues (mod  $n$ ) from nonresidues with Jacobi symbol equal to 1; this ability enables one to negate the input to the inverse function if necessary in order to obtain a quadratic residue (mod  $n$ ). Of course, dividing by 4 is easy—this step is needed only for inverting  $f_{1,n}$ . Next, taking square roots (mod  $n$ ) is easy, since we can take square roots modulo  $p$  and  $q$  separately (making sure to pick the square root which is itself a quadratic residue) and combine the results using the Chinese Remainder Theorem. Finally, the result can be negated (mod  $n$ ) as necessary in order to obtain a result in  $D_n$ . Since all of these steps are computable in polynomial time, each of the inverse functions  $f_{0,n}^{-1}$  and  $f_{1,n}^{-1}$  is computable in polynomial time, given  $p$  and  $q$  as additional inputs.

**THEOREM 1.** *Under the IAF, the following algorithm  $G$  is a claw-free permutation pair generator. On input  $1^k$ ,  $G$ .*

- (1) *Generates two random primes  $p$  and  $q$  of length  $k$ , where  $p \equiv 3 \pmod{8}$  and  $q \equiv 7 \pmod{8}$ ,*
- (2) *Outputs the quintuple*

$$(d, f_{0,n}, f_{0,n}^{-1}, f_{1,n}, f_{1,n}^{-1})$$

where

- (a) *Algorithm  $d$  generates elements uniformly at random in  $Q_n$ ,*
- (b) *Algorithms  $f_{0,n}$  and  $f_{1,n}$  are as described in the above equations,*
- (c) *Algorithms  $f_{0,n}^{-1}$  and  $f_{1,n}^{-1}$  are algorithms for the inverse functions (these algorithms make use of  $p$  and  $q$ ).*

*Proof.* We first note that uniformly selecting  $k$ -bit guaranteed primes can be accomplished in expected polynomial (in  $k$ ) time (by the recent work of Goldwasser and Kilian [GK86]) and that asymptotically one-quarter of these will be congruent to 3 (mod 8) (similarly for those congruent to 7 (mod 8)). (In practice, one would use a faster probabilistic primality test such as the one proposed by Solovay and Strassen [SS77] or Rabin [Ra80].)

Let  $n \in H$  and  $(d, f_{0,n}, f_{0,n}^{-1}, f_{1,n}, f_{1,n}^{-1}) \in [G(1^k)]$ . First,  $f_{0,n}$  and  $f_{1,n}$  are permutations of  $D_n = [d(\cdot)]$ . Then, we need only show that if there exists a fast algorithm that finds  $x$  and  $y$  in  $D_n$  such that  $f_{0,n}(x) \equiv f_{1,n}(y) \pmod{n}$  (i.e., a claw-creating algorithm) then factoring is easy. Suppose such an  $x$  and  $y$  have been found. Then  $x^2 \equiv 4y^2 \pmod{n}$ . (Note that  $x^2 \equiv -4y^2 \pmod{n}$  is impossible: since  $4y^2$  is a quadratic residue (mod  $n$ ),  $-4y^2$  cannot be a quadratic residue (mod  $n$ ), for  $n \in H$ .) This implies that  $(x + 2y) \times (x - 2y) \equiv 0 \pmod{n}$ . Moreover, we also know that  $x \not\equiv \pm 2y \pmod{n}$ , since  $(x/n) = 1$  and  $(2y/n) = -1$ . Thus  $\gcd(x \pm 2y, n)$  will produce a nontrivial factor of  $n$ .  $\square$

**6.4. An infinite set of pairwise claw-free permutations.** For our scheme we need not only claw-free pairs of permutations, but an infinite family of permutations which are pairwise claw-free and generated by a single claw-free pair  $f = (d, f_0, f_1)$ .

We define the function  $f_i(\cdot)$  for any string  $i \in \{0, 1\}^+$  by the equation:

$$f_i(x) = f_{i_0}(f_{i_1}(\dots(f_{i_{d-1}}(f_{i_d}(x)) \dots)))$$

if  $i = i_0i_1 \dots i_{d-1}i_d$ . (Also, read  $f_i^{-1}$  as  $(f_i)^{-1}$  so that  $f_i^{-1}(f_i(x)) = x$ .)

Each  $f_i$  is a trap-door permutation: it is easy to compute  $f_i(x)$  given  $f_0, f_1, i$ , and  $x$ , and to compute  $f_i^{-1}(x)$  if  $f_0^{-1}$  and  $f_1^{-1}$  are available. However, given only  $f_0$  and  $f_1$  it should be hard to invert  $f_i$  on a random input  $z$ , or else  $f_0$  and  $f_1$  are not trap-door permutations. (By inverting  $f_i$  on a random input we also effectively invert  $f_{i_0}$  on a random input, where  $i_0$  is the first bit of  $i$ .)

This way of generating an infinite family of trap-door permutations was also used in [GMY83].

Looking ahead, we shall see that a user  $A$  of our scheme can use the  $f_i$ 's to perform basic authentication steps as follows. Let us presume that  $A$  has published  $f_0$  and  $f_1$  as part of his public key, and has kept their inverses  $f_0^{-1}$  and  $f_1^{-1}$  secret. If user  $A$  is known to have authenticated a string  $y$ , then by publishing strings  $i$  and  $x$  such that

$$f_i(x) = y,$$

he authenticates the new strings  $i$  and  $x$ .

For this to work, when the signer  $A$  reveals  $f_i^{-1}(y)$ , he should not enable anyone else to compute  $f_j^{-1}(y)$  for any other  $j$ .

The signer achieves this in our scheme by coding  $i$  using a prefix-free mapping  $\langle \cdot \rangle$ . This prevents an enemy from computing  $f_{\langle j \rangle}^{-1}(x)$  from  $f_{\langle i \rangle}^{-1}(x)$  in an obvious way since  $\langle j \rangle$  is never a prefix of  $\langle i \rangle$ . The following Lemma 1 shows that this approach is not only necessary but sufficient.

*Note.* Actually, the mapping  $\langle \cdot \rangle$  that we use is a one-to-one mapping from *tuples* of strings of bits to strings of bits. The mapping  $\langle \cdot \rangle$  is prefix-free in the sense that  $\langle a_1, \dots, a_n \rangle$  is never a prefix of  $\langle b_1, \dots, b_m \rangle$  unless  $n = m$  and  $a_1 = b_1, \dots, a_n = b_n$ . Any prefix-free mapping is usable if it and its (partial) inverses are polynomial-time computable and the lengths of  $a_1, \dots, a_n$  and  $\langle a_1, \dots, a_n \rangle$  are polynomially related. For concreteness, we suggest the following encoding scheme for the tuple of strings  $a_1, \dots, a_n$ . Each string  $a_i$  is encoded by changing each 0 to 00 and each 1 to 11, and the encoding is followed by 01. The encodings of  $a_1, \dots, a_n$  are concatenated and followed by 10.

Lemma 1 essentially says that if  $(d, f_0, f_1)$  is a claw-free pair, then it will be hard to find two different tuples of strings  $i$  and  $j$ , and elements  $x$  and  $y$  such that  $f_{\langle i \rangle}(x) = f_{\langle j \rangle}(y)$ .

**LEMMA 1.** *Let  $f = (d, f_0, f_1)$  be a claw-free pair,  $x$  and  $y$  be elements of  $d$  and  $i, j$  two different tuples of binary strings such that there exists a string  $z$  such that  $z = f_{\langle i \rangle}(x) = f_{\langle j \rangle}(y)$ . Then there exists an  $f$ -claw  $(x_1, x_2, x_3)$  where  $x_3 = f_c^{-1}(z)$  for some prefix  $c$  of  $\langle i \rangle$ .*

*Proof.* Let  $c \in \{0, 1\}^*$  be the longest common prefix of  $\langle i \rangle$  and  $\langle j \rangle$ . Such a  $c$  must exist since  $\langle \cdot \rangle$  is a prefix-free encoding scheme. Thus, setting  $x_3 \leftarrow f_c^{-1}(z)$ ,  $x_1 \leftarrow f_{c0}^{-1}(z)$ , and  $x_2 \leftarrow f_{c1}^{-1}(z)$ , we obtain an  $f$ -claw  $(x_1, x_2, x_3)$ . (If  $c$  is the empty string then  $f_c^{-1}$  denotes the identity function, so  $x_3 = z$ .) Note that the  $f$ -claw is easily computed from  $f, x$ , and  $y$ .  $\square$

**7. Building blocks for signing.** In this section we define the basic building blocks needed for describing our signature scheme. In § 8, we will define what a signature is and how to sign, using the objects and data structures introduced here.

*Assumption.* We assume from here on that all claw-free functions used are defined over domains which do *not* include the empty string  $\epsilon$ .

This assumption is necessary since we use  $\epsilon$  as a “marker” in our construction; note that it is easy, via simple recodings, to enforce this construction if necessary.

We begin by defining the essential notion of an  $f$ -item.

DEFINITION. Let  $f = (d_f, f_0, f_1)$  be a claw-free pair. A tuple of strings  $(t, r; c_1, \dots, c_m)$  is an  $f$ -item if

$$f_{(c_1, \dots, c_m)}(t) = r.$$

DEFINITION. In an  $f$ -item  $(t, r; c_1, \dots, c_m)$ ,

- $t$  is called the *tag* of the item,
- $r$  is called the *root* of the item, and
- the  $c_i$ 's are the *children* of the item. We note that the children are ordered, so that we can speak of the first child or the second child of the item.

Note that given a claw free pair  $f$  and a tuple it is easy to check if the tuple is an  $f$ -item by applying the appropriate  $f_{(i)}$  to the tag, and checking if the correct root is obtained.

Figure 2 gives our graphic representation of an  $f$ -item  $(t, r; c_1, c_2)$  with two children.

DEFINITION. We say that a sequence of  $f$ -items  $L_1, L_2, \dots, L_b$  is an  $f$ -chain starting at  $y$  if, for  $i = 1, \dots, b - 1$ , the root of  $L_{i+1}$  is one of the children of  $L_i$  and  $y$  is the root of  $L_1$ . We say the chain ends at  $x$  if  $x$  is one of the children of the item  $L_b$ .

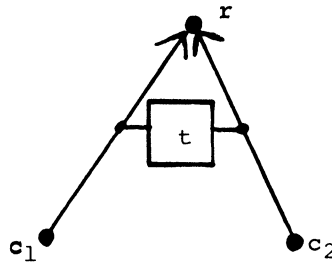


FIG. 2. An  $f$ -item with two children.

For efficiency considerations, our signature scheme will organize a collection of a special-type of  $f$ -chains in the treelike structure defined below.

DEFINITION. Let  $i$  be a binary string of length  $b$  and  $f$  a claw-free pair. An  $f$ - $i$ -tree is a bijection  $T$  between DFS( $i$ ) and a set of  $f$ -items such that:

- (1) if string  $j$  has length  $b$ , then  $T(j)$  is an  $f$ -item with exactly two children, exactly one of which is  $\epsilon$ , the empty string. These  $f$ -items are called *bridge items*.
- (2) if string  $j$  has length less than  $b$ , then  $T(j)$  is an  $f$ -item with exactly two children,  $c_0$  and  $c_1$ , both of which are nonempty strings. Moreover,  $c_0$ , the  $0$ th child, is the root of  $T(j0)$  and  $c_1$ , the  $1$ st child, the root of  $T(j1)$ .

The  $f$ -item  $T(j)$  is said to be of *depth*  $b$  if string  $j$  has length  $d$ . (The bridge items are thus the items of depth  $b$ .) The *root* of  $T$  is the root of the  $f$ -item  $T(\epsilon)$ . The *internal nodes* of  $T$  are the root and the children of the  $f$ -items of depth less than  $b$ . The *leaves* of  $T$  are the nonempty children of the bridge items. Thus the internal nodes and the leaves of an  $f$ - $i$ -tree are actual values and not  $f$ -items. Leaves possess binary names of length  $b$ ; leaf  $j$  is the nonempty child of bridge item  $T(j)$ . The *path to leaf*  $j = j_0 \dots j_b$  is the  $f$ -chain  $T(\epsilon), T(j_0), \dots, T(j_0 \dots j_b)$ .



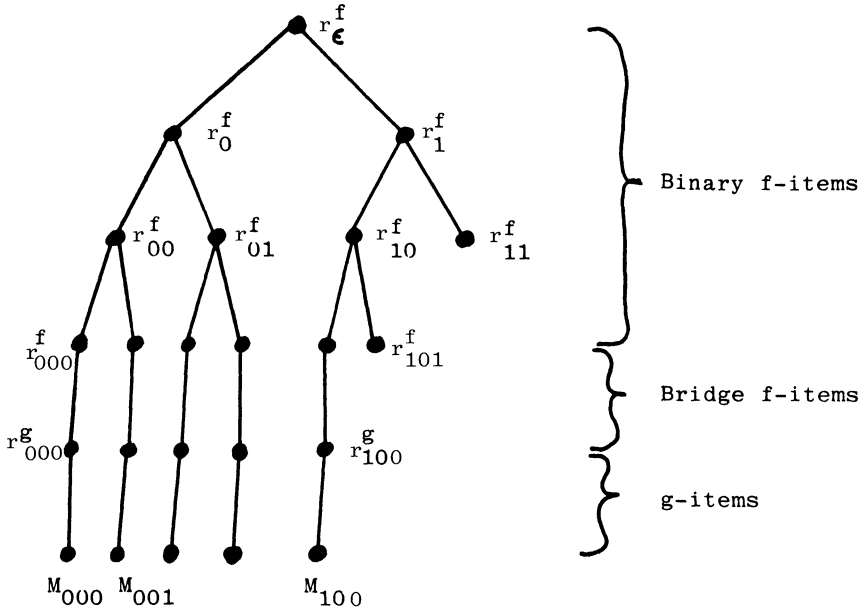


FIG. 3. An  $f$ -100-tree.

Figure 3 gives our graphic representation of an  $f$ -100-tree, as it would be used in our signature scheme. In this figure we denote by  $r_i^f$  the root of  $f$ -item  $T(i)$ , and by  $r_i^g$  the leaf (nonempty) child of bridge item  $T(i)$ . (Also present in Fig. 3 are a number of “ $g$ -items,” which are not part of the  $f$ -100 tree but are attached to it in a manner to be described.)

There are two reasons for letting the bridge items of an  $f$ -tree have the empty string as one of their children. First, it makes them de facto  $f$ -items with only one child, a subtle point in our proof of security that is pointed out in Remark 1. Second, it makes them distinguishable from items with two children, a simple point used, for instance, in Lemma 2.

**8. Description of our signature scheme.**

**8.1. Message spaces.** The security properties of the new signatures scheme hold for any nonempty message space  $M \subset \{0, 1\}^+$ .

**8.2. How to generate keys.** We assume the existence of a claw-free permutation pair generator  $G$  and, without loss of generality, that the bound  $B$  on the number of signatures that can be produced is a power of 2:  $B = 2^b$ .

The key-generation algorithm  $K$  runs as follows on inputs  $1^k$  and  $2^b$ :

- (1)  $K$  runs  $G$  twice on input  $1^k$  to secretly and randomly select two quintuples

$$(d_f, f_0, f_0^{-1}, f_1, f_1^{-1}) \text{ and } (d_g, g_0, g_0^{-1}, g_1, g_1^{-1}) \in [G(1^k)].$$

- (2)  $K$  then randomly selects  $r_\epsilon^f$  in  $D_f = [d_f(\cdot)]$ .

(3)  $K$  outputs the public key  $PK = (f, r_\epsilon^f, g, 2^b)$  where  $f$  is the claw-free pair  $(d_f, f_0, f_1)$  and  $g$  is the claw-free pair  $(d_g, g_0, g_1)$ .

- (4)  $K$  outputs the secret key  $SK = (f^{-1}, g^{-1})$ .

The  $PK$  and  $SK$  so produced are said to be (matching) keys of size  $k$ .

**8.3. What is a signature?** A *signature* of a message  $m$  with respect to a public key  $(f, r_\varepsilon^f, g, 2^b)$  consists of the following:

- (1) An  $f$ -chain of length  $b+1$  starting at a string  $r_\varepsilon^f$  and ending at  $r^g$ , and
- (2) A  $g$ -item with  $r^g$  as its root and  $m$  as its only child.

**8.4. How to sign.** In the remainder of this section we shall presuppose that user  $A$ 's public key is  $PK = (f, r_\varepsilon^f, g, 2^b)$  where  $f = (d_f, f_0, f_1)$  and  $g = (d_g, g_0, g_1)$ . User  $A$ 's secret key is  $SK = (f^{-1}, g^{-1})$ . We denote by  $D_f$  the domain  $[d_f(\cdot)]$ , and denote by  $D_g$  the domain  $[d_g(\cdot)]$  similarly.

Conceptually, user  $A$  creates an  $f-1^b$ -tree  $T$ , which has  $2^b$  leaves. The root of  $T$  will be  $r_\varepsilon^f$ . The other internal nodes of  $T$  are randomly selected elements of  $D_f$ . The leaves of  $T$  are randomly selected elements of  $D_g$ .

To sign  $m_i$ , the  $i$ th message in the chronological order, user  $A$  computes a  $g$ -item  $G_i$  whose root  $r_i^g \in D_g$  is the  $i$ th leaf of  $T$ , and whose only child is the message  $m_i$ . He then outputs, as the signature of  $m_i$ ,  $G_i$  and the  $f$ -chain in  $T$  starting at root  $r_\varepsilon^f$  and ending at leaf  $r_i^g$ .

In practice, it will be undesirable for user  $A$  to precompute and store all of  $T$ . He will instead "grow"  $T$  as needed and try to optimize his use of storage and time. This is taken into account by our signing procedure. In what follows, we describe a variation of our signing method that requires the signer to remember just his secret key and his most recently produced signature, in order to produce his next signature. The reader may find it helpful to refer to Fig. 3 while reading this description.

**The signing procedure (also called  $\mathcal{SP}$ ).** We presume that the procedure is initialized with the values of the public key  $PK$  and the corresponding secret key  $SK$  in its local private storage, that has already signed messages  $m_0, m_1, \dots, m_{i-1}$  and kept track of the number of previous messages signed (i.e., the variable  $i = i_0 \dots i_{b-1}$ , which is a  $b$ -long bit string, which may contain leading 0's), and the most recent signature produced.

To compute a signature for message  $m_i$ , the  $i$ th message, user  $A$  performs the following steps.

- (1) (*Output  $f$ -chain.*)
  - (1.1) (*Output  $f$ -items in common with previous signature.*) If  $i = 0^b$  this substep is skipped, and control passes to step (1.2). Otherwise, for each string  $j$  which is a common prefix of  $i$  and  $i-1$ , he outputs the  $f$ -item  $(t_j^f, r_j^f; r_{j_0}^f, r_{j_1}^f)$  which was part of the signature for message  $m_{i-1}$ , in order of increasing length of  $j$ .
  - (1.2) (*Output new  $f$ -items in  $f$ -tree.*) For each string  $j$  (if any) which is a proper prefix of  $i$ , but not a prefix of  $i-1$ , user  $A$  creates and outputs an  $f$ -item  $T(j)$ , in order of increasing length of  $j$ . The  $f$ -item  $T(j) = (t_j^f, r_j^f; r_{j_0}^f, r_{j_1}^f)$  is created as follows: If  $j = \varepsilon$  its root  $r_j^f$  is the  $r_\varepsilon^f$  from the public key; otherwise it is the  $k$ th child of the most recently output  $f$ -item, where  $k$  is the last bit of the string  $j$ . The children  $r_{j_0}^f$  and  $r_{j_1}^f$  of the  $f$ -item with root  $r_j^f$  are chosen at random from  $D_f$ . The tag  $t_j^f = f_{(r_{j_0}^f, r_{j_1}^f)}^{-1}(r_j^f)$  is computed using  $f_0^{-1}$  and  $f_1^{-1}$  from the secret key. Note that the last item output (by either step (1.1) or (1.2)) has  $r_i^f$  as one of its children.
- (1.3) (*Output bridge  $f$ -item.*) User  $A$  next outputs a single  $f$ -item with root  $r_i^f$  and whose children are  $\varepsilon$  and  $r_i^g$ , a randomly chosen element from  $D_g$ . The tag  $t_i^f$  for this item is again computed using the secret trap-door information for inverting  $f_0$  and  $f_1$ .
- (2) (*Output  $g$ -item.*) Finally, user  $A$  outputs the  $g$ -item  $G_i = (t_i^g, r_i^g; m_i)$ . The tag  $t_i^g$  for this item is computed using the  $g^{-1}$  from the secret key.

The items output by the above procedure constitute a signature for  $m_i$ . Notice that there are many possible signatures (among which  $A$  chooses one at random) for each occurrence of each message, but only one signature is actually output.

The reader may verify that the above procedure for producing a signature will have a total running time which is bounded by a polynomial in  $k$  and  $b$ .

Notice that if  $A$  has signed  $i$  messages, the function  $T$  mapping each string  $j \in \text{DFS}(i)$  to  $f$ -item  $T(j)$  is an  $f$ - $i$ -tree as defined in § 7.

**8.5. How to verify a signature.** Given  $A$ 's public key  $(f, r_e^f, g, 2^b)$ , anyone can easily verify that the first  $b+1$  elements in the signature of  $m_i$  are  $f$ -items forming an  $f$ -chain starting at  $r_e^f$  and ending at  $r_i^g$ , and that the  $g$ -item in the signature has  $r_i^g$  as its root and  $m_i$  as its only child. If these checks are all satisfied, the given sequence of items is accepted as an authentic signature by  $A$  of the message  $m_i$ .

It is easy to confirm that these operations take time proportional to  $b$  times some polynomial in  $k$ , the size of the public key.

**8.6. Efficiency of the proposed signature scheme.** Assume that if  $f = (d_f, f_0, f_1)$  is a claw-free pair of size  $k$ , then an element of  $D_f$  is specified by a  $k$ -bit string. Then the time to compute a signature for a message  $m$  of length  $l$  is  $O(bk)$   $f$ -inversions (i.e., inversions of  $f_0$  or  $f_1$ ) and  $O(l)$   $g$ -inversions.

Another relevant measure of efficiency is "amortized" time. That is, the time used for producing all possible  $2^b$  signatures divided by  $2^b$ . In our scheme, the amortized " $f$ -inversion" cost is  $O(k)$ . The amortized " $g$ -inversion" cost is  $O(l)$  if the average length of a message is  $l$ .

The length of the signature for  $m$  is  $O(bk+l)$ , where  $l$  is the length of  $m$ , as  $m$  is included in  $m$ 's signature as the child of the  $g$ -item. Clearly, if  $m$  is known to the signature recipient, the  $g$ -item need not include  $m$ : it suffices to give its root and its tag. This way the length of the signature can be only  $O(bk)$  long, which is independent of the length of  $m$  and possibly much shorter.

The memory required by the signing algorithm is  $O(bk)$ , since it consists of storing (the  $f$ -items in) the most recently produced signature.

**9. Proof of security.** Let us start by establishing a convenient terminology.

**DEFINITION.** We call *signature corpus* the first  $i$  (for some  $i > 0$ ) signatures output by our signing procedure  $\mathcal{SP}$ . We shall generally use the symbol  $\mathcal{S}$  to denote a signature corpus.

We define the following quantities relative to a signature corpus  $\mathcal{S}$ , consisting of  $i$  signatures relative to a public key  $PK = (f, r_e^f, g, 2^b)$ .

- (1) The set of *items* of  $\mathcal{S}$ , denoted by  $\mathcal{I}(\mathcal{S})$ , is the set of the items in the signatures of  $\mathcal{S}$ .
- (2) The set of  *$f$ -items* of  $\mathcal{S}$ , denoted by  $f(\mathcal{S})$ , is the set of  $f$ -items in  $\mathcal{I}(\mathcal{S})$ .
- (3) The set of  *$g$ -items* of  $\mathcal{S}$ , denoted by  $g(\mathcal{S})$ , is the set of  $g$ -items in  $\mathcal{I}(\mathcal{S})$ .
- (4) The set of *messages* of  $\mathcal{S}$ , denoted  $M(\mathcal{S})$ , is the set of messages signed by  $\mathcal{S}$ , i.e., the set of children of the  $g$ -items of  $\mathcal{S}$ .
- (5) The  *$f$ -tree* of  $\mathcal{S}$ , denoted by  $\mathcal{T}^f(\mathcal{S})$ , is the  $f$ - $i$ -tree having root  $r_e^f$  and, as path to leaf  $j$  ( $j = 0, \dots, i$ ), the  $f$ -chain of the  $j$ th signature of  $\mathcal{S}$ .
- (6) The set of *internal nodes* of  $\mathcal{S}$ , denoted by  $\mathcal{IN}(\mathcal{S})$ , is the set of the internal nodes of  $\mathcal{T}^f(\mathcal{S})$ .
- (7) The set of *nonroots* of  $\mathcal{S}$ , denoted by  $\mathcal{NR}(\mathcal{S})$ , is the set of those internal nodes of  $\mathcal{T}^f(\mathcal{S})$  that are not the root of any  $f$ -item of  $\mathcal{S}$ . We may think of these nodes as "hooks" from which additional  $f$ -items will be grown as new signatures are created.
- (8) The set of *leaves* of  $\mathcal{S}$ , denoted  $\mathcal{L}(\mathcal{S})$ , is the set of leaves of  $\mathcal{T}^f(\mathcal{S})$ .

Notice that all the above sets are unambiguously defined. For instance, an item in  $f(\mathcal{S})$  has exactly two children while an item in  $g(\mathcal{S})$  only one, the bridge elements of  $\mathcal{I}(\mathcal{S})$  have exactly one empty child and thus are distinguishable from other items in  $f(\mathcal{S})$ , and so on.

Some of these definitions can be observed in Fig. 3. For example, the leaves of the  $f$ -100-tree in Fig. 3 are  $r_{000}^f, r_{001}^f, r_{010}^f, r_{011}^f, r_{100}^f$  and its nonroots are  $r_{101}^f$  and  $r_{111}^f$ .

Let us now see how the signature of a message never signed before relates to a given signature corpus.

LEMMA 2. *Let  $\mathcal{S}$  be a signature corpus relative to a public key  $PK = (f, r_\epsilon^f, g, 2^b)$  and let  $\sigma$  be a signature (relative to the same public key) of a message  $m$  not belonging to  $M(\mathcal{S})$ . Denote by  $\mathcal{I}(\sigma)$  the set of items in  $\sigma$ . Then  $\mathcal{I}(\sigma) - \mathcal{I}(\mathcal{S})$  (the set of new items) contains either*

- (1) a  $g$ -item with root  $r \in \mathcal{L}(\mathcal{S})$ , or
- (2) an  $f$ -item with root  $r \in \mathcal{IN}(\mathcal{S})$ .

*Proof.* First notice that  $\mathcal{I}(\sigma) - \mathcal{I}(\mathcal{S})$  is not empty as it contains  $G$ , the  $g$ -item of  $\sigma$ . In fact,  $G$  cannot belong to  $f(\mathcal{S})$ , as it is a  $g$ -item, and cannot belong to  $g(\mathcal{S})$ , as  $m$  is its only child and all items in  $g(\mathcal{S})$  have elements of  $M(\mathcal{S})$  as their children. Assume  $\mathcal{I}(\sigma) - \mathcal{I}(\mathcal{S})$  also contains an  $f$ -item. Then this  $f$ -item belongs to  $F$ , the  $f$ -chain of  $\sigma$  whose first item has  $r_\epsilon^f$  as root, one of the internal nodes of  $\mathcal{S}$ . Thus, for some item in  $F$ , (2) holds. Assume now that  $\mathcal{I}(\sigma) - \mathcal{I}(\mathcal{S}) = G$ . Then the root of  $G$  is the nonempty child of  $B$ , the bridge  $f$ -item of  $\sigma$ . By hypothesis,  $B$  is in  $\mathcal{I}(\mathcal{S})$ ; thus, the root of  $G$  belongs to  $\mathcal{L}(\mathcal{S})$  and (1) holds.  $\square$

Recall Lemma 1 from § 6.4:

LEMMA 1. *Let  $f = (d, f_0, f_1)$  be a claw-free pair,  $x$  and  $y$  be elements of  $d$  and  $i, j$  be two different tuples of binary strings such that there exists a string  $z$  such that  $z = f_{\langle i \rangle}(x) = f_{\langle j \rangle}(y)$ . Then there exists an  $f$ -claw  $(x_1, x_2, x_3)$  where  $x_3 = f_c^{-1}(z)$  for some prefix  $c$  of  $\langle i \rangle$ .*

We can now prove Lemma 3:

LEMMA 3. *There exists a polynomial-time algorithm  $A$  that, on input, a corpus  $\mathcal{S}$  relative to a public key  $PK = (f, r_\epsilon^f, g, 2^b)$  and the signature  $\sigma$  of message not belonging to  $M(\mathcal{S})$ , finds either*

- (1) a  $g$ -claw, or
- (2a) an  $f$ -claw, or
- (2b) an  $f$ -item whose root belongs to  $\mathcal{NR}(\mathcal{S})$ .

*Proof.* (The cases are numbered according to the corresponding cases in Lemma 2.) If case (1) of Lemma 2 holds for  $\mathcal{S}$  and  $\sigma$ , then we have two  $g$ -items with the same root  $r$  in  $\mathcal{L}(\mathcal{S})$ . Namely, an  $i, j, x$  and  $y$  such that  $g_{\langle i \rangle}(x) = g_{\langle j \rangle}(y) = r$  and we get a  $g$ -claw by Lemma 1. Otherwise, if case (2) of Lemma 2 holds, let  $F$  be the  $f$ -item that satisfies condition (2) of Lemma 2. If  $F$  has the same root as some  $F' \in f(\mathcal{S})$ , then again by Lemma 1, we get an  $f$ -claw; otherwise, we get an  $f$ -item whose root belongs to  $\mathcal{NR}(\mathcal{S})$ .  $\square$

*Remark 1.* Notice that if  $\sigma$  is generated by the legal signer (i.e., the  $\mathcal{SP}$  procedure) then, with very high probability, case (2b) will hold in Lemma 3.

In the proof of the main theorem we will assume that there exists a successful adaptive chosen-message attack, and derive a contradiction by showing that this attack would enable an enemy to easily create either an  $f$ -claw or a  $g$ -claw with sufficiently high probability. Recall that in an adaptive chosen-message attack the enemy can repeatedly use the real signer as an “oracle” before attempting to forge a new signature. The next lemma, Lemma 4, essentially states that the signing process can be simulated perfectly by an efficient algorithm that knows the public key and only half of the secret

key: the inverses of the first claw-free pair. (That is, in some sense, this algorithm is a forger.)

To state Lemma 4, additional notation regarding “interactive” probabilistic algorithms, needs to be introduced. The notion of an adaptive, chosen-message attack involves the interaction of two algorithms:  $\mathcal{S}$  (the signer) and  $\mathcal{R}$  (the signature requestor). These algorithms “take turns”:  $\mathcal{R}$  requests a signature of a given message,  $\mathcal{S}$  signs it,  $\mathcal{R}$  requests a second signature,  $\mathcal{S}$  signs it, and so on. We might view the two routines as “co-routines” that pass control back and forth while preserving their own state. We formalize this interaction by means of the *combining algorithm*  $\mathcal{C}$  that defines a composite algorithm from two auxiliary ones. The combining algorithm  $\mathcal{C}$  will invoke repeatedly  $\mathcal{S}$  and  $\mathcal{R}$  in alternation, corresponding to their taking turns. The algorithms  $\mathcal{S}$  and  $\mathcal{R}$  have private-state variables (denoted  $V_{\mathcal{S}}$  and  $V_{\mathcal{R}}$ ) that are preserved from invocation to invocation. Algorithm  $\mathcal{S}$  (which produces signatures) takes as input a public key  $PK$ , an auxiliary input  $X$  (which for the moment is unspecified but will later denote either the corresponding secret key  $SK$  or part of it), a new message to sign, and its private-state variable. It produces as output a signature for the new message and an updated version of its private-state variable. Similarly,  $\mathcal{R}$  is a probabilistic algorithm which takes as input a public key, a sequence of previous signatures relative to that public key, and its private-state variable, and produces as output a message to be signed and an updated version of its private-state variable.

The following algorithm makes specific the process of combining  $\mathcal{S}$  and  $\mathcal{R}$ :

ALGORITHM  $\mathcal{C}(\mathcal{S}, \mathcal{R}; PK, X, i)$

Set  $\mathcal{S}_0 \leftarrow \emptyset$ .

Set  $V_{\mathcal{R}}$  and  $V_{\mathcal{S}}$  to  $\emptyset$ .

for  $j = 0$  to  $i$  do:

$(m_j, V_{\mathcal{R}}) \leftarrow \mathcal{R}(PK, \{\mathcal{S}_1, \dots, \mathcal{S}_{j-1}\}, V_{\mathcal{R}})$  (*Request signature for message  $m_j$ .*)

$(\mathcal{S}_j, V_{\mathcal{S}}) \leftarrow \mathcal{S}(PK, m_j, V_{\mathcal{S}}, X)$ . (*Produce signature for message  $m_j$ .*)

Output  $\mathcal{S}_j$ .

Here  $\mathcal{S}_j$  denotes the signature of the  $j$ th message.

We extend our notation of probabilistic algorithm in a natural way by letting  $\mathcal{C}(\mathcal{S}, \mathcal{R}; PK, X, i)$  represent the probability space that assigns the sequence  $\sigma$  the probability that  $\mathcal{C}$  outputs  $\sigma$  after invoking alternatively (for  $i$  times)  $\mathcal{S}$  (with initial input  $PK$  and  $X$ ) and  $\mathcal{R}$  (with initial inputs  $PK$ ).

We can now state Lemma 4, stating that the signing process can be simulated effectively if the  $f_i$ 's inverses are known but the  $g_i$  inverses are not.

LEMMA 4. *There exists an algorithm  $\mathcal{A}$  in  $\mathcal{RA}$  such that for all requestors  $\mathcal{R} \in \mathcal{RA}$ , for all public keys  $PK = (f, r_e^f, g, 2^b)$  and for all nonnegative integers  $i < 2^b$ ,*

$$\mathcal{C}(\mathcal{A}, \mathcal{R}; PK, \{f^{-1}\}, i) = \mathcal{C}(\mathcal{SP}, \mathcal{R}; PK, SK, i)$$

(where  $\mathcal{SP}$  is the legal signing process of § 8, and  $SK$  is the corresponding secret key to  $PK$ ).

*Proof.* Consider the following algorithm  $\mathcal{A}$ . We inductively assume that

$$\mathcal{C}(\mathcal{A}, \mathcal{R}; PK, \{f^{-1}\}, i - 1) = \mathcal{C}(\mathcal{SP}, \mathcal{R}; PK, SK, i - 1).$$

Thus the  $f$ -chains in the first  $i - 1$  signatures output by  $\mathcal{C}$  uniquely define an  $f - (i - 1)$ -tree  $T$ . Algorithm  $\mathcal{A}$  stores  $i - 1$  and the  $f$ -chain of the last produced signature and executes the following instructions to sign  $m_i$ , the  $i$ th message, where  $i = i_0 \cdots i_b$ .

- (1) (*Authenticate  $m_i$  with a  $g$ -item.*) Pick an element  $t_i^g$  at random in  $D_g$  and compute  $r_i^g = g_{(m_i)}(t_i^g)$  so as to generate the  $g$ -item  $(t_i^g, r_i^g; m_i)$ .

- (2) (*Build the  $f$ -chain from  $r_\varepsilon^f$  to  $r_i^f$  to the extent that it is not already done.*) Compute  $i_0 i_1 \cdots i_j$ , the longest proper prefix of  $i$  that is also a prefix of  $i-1$ . For  $x = 1$  to  $b-j$ , generate  $T(i_0 \cdots i_{j+x})$ , an  $f$ -item whose root is the  $i_{j+x}$ th child of  $T(i_0 \cdots i_{j+x-1})$ , and whose two children are independently and randomly selected elements of  $D_f$ . (Algorithm  $\mathcal{A}$  easily computes the tag of this new  $f$ -item by using  $f^{-1}$ .)
- (3) (*Create the bridge item authenticating  $r_i^g$ .*) Using  $f_0^{-1}$  and  $f_1^{-1}$ , create an  $f$ -item with children  $\varepsilon$  and  $r_i^g$  and having as root the  $i_b$ th child of  $T(i_0 \cdots i_{b-1})$ .
- (4) (*Output signature of  $m_i$* ) Output  $T(\varepsilon)$ ,  $T(i_0)$ ,  $\cdots$ ,  $T(i_0 \cdots i_{b-1})$ , the new bridge item  $T(i_0, \cdots, i_b)$  and the new  $g$ -item.  $\square$

In Lemma 6 we show a similar result: the signing process can be simulated if  $g^{-1}$  is known, but  $f^{-1}$  is not. The proof of Lemma 6 makes essential use of the fact that there is a known upper bound on the number of signatures to be produced. (The bound provides a limit on the amount of a preprocessing step that is the subject of Lemma 5.)

There is, however, a very important difference between the signing simulation procedure described in Lemma 4 (which uses  $f^{-1}$  but not  $g^{-1}$ ) and that of Lemma 6 (which uses  $g^{-1}$  but not  $f^{-1}$ ). The proof of Lemma 4 works with any fixed root  $r_\varepsilon^f$ , which can be fixed arbitrarily before the simulation procedure is invoked.

By contrast, the signing simulation procedure of Lemmas 5 and 6 actually *produces* the necessary root  $r_\varepsilon^f$  to be part of the public key in its preprocessing step. The root produced is uniformly distributed over  $D_f$ . Thus, from the point of view of an observer that monitors the behaviour of the signer when he publishes his public key, the preprocessing step is indistinguishable from a genuine key generation step. Moreover, by monitoring the signing process, the observer cannot tell whether the signer really knows  $f^{-1}$  or if he has first applied the preprocessing procedure of Lemma 5 to produce his public file and only then applied the simulation procedure of Lemma 6.

DEFINITION. For all strings  $m_1, \cdots, m_i$ , let *sequence*( $m_1, \cdots, m_i$ ) denote the trivial interactive algorithm that, no matter what inputs it gets, when invoked for the  $j$ th time ( $j = 1, \cdots, i$ ) outputs the string  $m_j$ .

Let us define two probability spaces over the  $f$ - $i$ -trees which are crucial to our analysis.

DEFINITION. Let  $PK = (f, r_\varepsilon^f, g, 2^b)$  and  $SK = (f^{-1}, g^{-1})$  be a pair of matching public and secret key, where  $f = (d_f, f_0, f_1)$ . Recall that  $\mathcal{C}$  is the combining algorithm. Define two probability spaces,  $\mathcal{T}_{i,PK}$  and  $\mathcal{T}_{i,f,g,2^b}$ , as follows:

$\mathcal{T}_{i,PK}$  is generated by randomly selecting  $\mathcal{S}$  in  $\mathcal{C}(\mathcal{SP}, \text{sequence}(m_1, \cdots, m_i); PK, SK, i)$  and then computing  $\mathcal{T}^f(\mathcal{S})$ . (Note that  $\mathcal{T}_{i,PK}$  does not depend on the values of the messages  $m_1, \cdots, m_i$  but it does depend on  $i$ , the number of messages.)

$\mathcal{T}_{i,f,g,2^b}$  is generated by randomly selecting  $\mathcal{S}$  in  $\mathcal{C}(\mathcal{SP}, \text{sequence}(m_1, \cdots, m_i); (f, d_f(\cdot), g, 2^b), SK, i)$  and then computing  $\mathcal{T}^f(\mathcal{S})$ .

Informally,  $\mathcal{T}_{i,PK}$  is the probability space obtained from  $\mathcal{T}_{i,f,g,2^b}$  by randomly picking  $r_\varepsilon^f \in D_f$  and fixing it in  $PK$ .

Notice that both probability spaces are easily generated if the secret key  $SK = (f^{-1}, g^{-1})$  is among the available inputs. However, both probability spaces remain easy to generate on a more restricted set of inputs. It has been implicitly proved in Lemma 2 that  $\mathcal{T}_{i,PK}$  can be generated in probabilistic polynomial-time on inputs  $i, PK$  and  $f^{-1}$  alone. The following lemma shows that  $\mathcal{T}_{i,f,g,2^b}$  is easily generated on inputs  $i, f, g, 2^b$  alone.

LEMMA 5. *There exists  $\mathcal{T} \in \mathcal{RA}$  such that for all claw-free pairs  $f = (d_f, f_0, f_1)$  and  $g = (d_g, g_0, g_1)$  and for all integers  $i < 2^b$ ,*

$$\mathcal{T}(i, f, g, 2^b) = \mathcal{T}_{i,f,g,2^b}$$

*Proof.* Consider the following algorithm  $\mathcal{T}$  that constructs an  $f$ - $i$ -tree  $T$  in “reverse order”; that is, it constructs  $f$ -item  $T(x)$  before  $f$ -item  $T(y)$  if  $y < x$ . (This is necessary since  $\mathcal{T}$  does not have access to  $f^{-1}$ .) The construction goes as follows.

If string  $j \in \text{DFS}(i)$  has length  $b$ ,  $\mathcal{T}$  selects the nonempty child of  $T(j)$  at random in  $D_g$ . Otherwise (if  $j$  has length shorter than  $b$ ),  $\mathcal{T}$  selects, as 0th child of  $T(j)$ , the root of  $T(j0)$  and, as 1st child, the root of  $T(j1)$ . In case  $j1$  does not belong to  $\text{DFS}(i)$ ,  $\mathcal{T}$  selects the second child of  $T(j)$  at random in  $D_f$ .

Having selected the two children  $c_0$  and  $c_1$  of  $T(j)$ ,  $\mathcal{T}$  selects its tag  $t$  at random in  $D_f$ . Then it computes the prefix-free encoding  $\langle (c_0, c_1) \rangle$  and selects as the root of  $T(j)$  the element  $f_{(c)}(t)$ , which  $\mathcal{T}$  easily computes using  $f_0$  and  $f_1$ .

Notice that each  $T(j)$  so computed is a proper  $f$ -item and that the resulting  $T$  is a proper  $f$ - $i$ -tree belonging to  $[\mathcal{T}_{i,f,g,2^b}]$ . Let us now analyze the probability distribution according to which  $T$  has been selected.

First notice that the leaves of  $T$  (that is the nonempty children of the items of depth  $b$ ) have the same distribution of the leaves of an  $f$ - $i$ -tree randomly selected in  $\mathcal{T}_{i,f,g,2^b}$ . In fact, in both cases, all leaves are uniformly and independently selected elements of  $D_g$ . Then notice that the roots of the items of  $T$  of depth  $k$  (that is, the children of the items of  $T$  of depth  $k - 1$ ) are selected uniformly and independently in  $D_f$ . In fact, the root of each item is obtained by applying  $f_{(x)}$ , a permutation of  $D_f$  randomly selected from some probability space, to an element  $t$  (the tag) independently and uniformly selected in  $D_f$ . From this it easily follows that  $\mathcal{T}$  selects  $T$  at random in  $\mathcal{T}_{i,f,g,2^b}$ . It is easily seen that  $\mathcal{T} \in \mathcal{RA}$  and thus satisfies all the required properties of our lemma.  $\square$

LEMMA 6. *There exists an algorithm  $\mathcal{A} \in \mathcal{RA}$  such that for all signature requestors  $\mathcal{SR} \in \mathcal{RA}$ , for all claw-free pairs  $f = (d_f, f_0, f_1)$  and  $g = (d_g, g_0, g_1)$ , and for all nonnegative integers  $i < 2^b$ ,*

$$\mathcal{C}(\mathcal{A}, \mathcal{SR}; (f, d_f(\cdot), g, 2^b), \{g^{-1}\}, i) = \mathcal{C}(\mathcal{SP}, \mathcal{SR}; (f, d_f(\cdot), g, 2^b), \{f^{-1}, g^{-1}\}, i).$$

*Proof.* Consider the following algorithm  $\mathcal{A}$ . In a preprocessing step,  $\mathcal{A}$  runs algorithm  $\mathcal{T}$  of Lemma 5 to randomly select an  $f$ - $i$ -tree  $T$  from  $\mathcal{T}_{i,f,g,2^b}$ . Let  $r_e^f$  be the root of  $T$ . This root is used to construct the public file  $PK = (f, r_e^f, g, 2^b)$ , with respect to which all subsequent signatures will be produced as follows.  $\mathcal{A}$  starts the signature requestor  $\mathcal{SR}$  on input  $PK$ . Then it simulates the signing procedure with initial inputs  $PK$  and the corresponding secret key  $SK = (f^{-1}, g^{-1})$  without using  $f^{-1}$  in the following way. When  $\mathcal{SR}$  outputs  $m_j$ , the  $j$ th message to be signed,  $\mathcal{A}$  retrieves the  $f$ -chain  $T_j$ , the path from the root of  $T$  to leaf  $j$ . Then  $\mathcal{A}$  computes the necessary  $g$ -item by using  $g^{-1}$ .  $\square$

Before stating and proving our Main Theorem, let us single out a simple lemma stating that one cannot invert a claw-free pair on a randomly selected input of its domain.

LEMMA 7. *Let  $G$  be a claw-free permutation pair generator. Then, for any inverting algorithm  $I \in \mathcal{RA}$ , any  $c > 0$  and sufficiently large  $k$ ,*

$$\mathbf{P}(h_0(z) = x \text{ or } h_1(z) = x | (d, h_0, h_0^{-1}, h_1, h_1^{-1}) \leftarrow G(1^k);$$

$$x \leftarrow d_h(\cdot); z \leftarrow I(1^k, d, h_0, h_1)) < k^{-c}.$$

*Proof.* Otherwise the following algorithm would find a claw with too high a probability: randomly select  $y$  in  $d_h$ , randomly select  $i$  between 1 and 2, compute  $x = h_i(y)$  and run  $I$  to get  $z$  such that  $h_i(z) = x$  for  $j \neq i$ .  $\square$

We are now ready to formally state and prove our Main Theorem. We start by strengthening the definition of existentially forgeable to include probabilistic success on the part of the forger.

**DEFINITION.** We say that a signature scheme is  $\varepsilon$ -*existentially forgeable* if it is existentially forgeable with probability  $\varepsilon$  where the probability space includes the random choices of the adaptive chosen-message attack, the random choices made by the legal signer in the creation of the public key, and the random choice made by the legal signer in producing signatures.

It is very important to note that the random choices made in creating the public key are included in the probability space; our proof depends critically on this definition. The Main Theorem of this paper is the following.

**MAIN THEOREM.** *Assuming that claw-free permutation pair generators exist, the signature scheme described in § 8 is not even  $1/Q(k)$ -existentially forgeable under an adaptive chosen-message attack, for all polynomials  $Q$  and for all sufficiently large  $k$ .*

*Proof of the Main Theorem.* The proof proceeds by contradiction. We assume, for the sake of contradiction, that for some polynomial  $Q$  and for infinitely many  $k$  our signature scheme is  $1/Q(k)$ -existentially forgeable under an adaptive chosen-message attack by an algorithm  $\mathcal{F}$  in  $\mathcal{RA}$ .

By definition, the forging algorithm  $\mathcal{F}$  consists of two algorithms in  $\mathcal{RA}$ : a signature requestor  $\mathcal{FR}$ , which is active in a first phase when it adaptively asks and receives signatures of messages of its choice, and a signature finder  $\mathcal{FF}$ , which is active in a second phase when it attempts to forge a signature of a message not asked about by  $\mathcal{FR}$ .

Let  $PK = (f, r_{\varepsilon}^f, g, 2^b)$  and  $SK$  be a public/secret-key pair of size  $k$ , randomly selected by our key generator using a claw-free permutation pair generator  $G$ . In the first phase a signature corpus  $\mathcal{S} \leftarrow \mathcal{C}(\mathcal{SP}, \mathcal{FR}; PK, SK, i)$  is generated, where  $i < 2^b$ . Then  $\mathcal{FF}$  is run on input  $\mathcal{S}$  and  $PK$ . Let  $\varepsilon_k$  denote the probability that  $\mathcal{FF}$  outputs  $\sigma$ , a legal signature, with respect to  $PK$ , for a message  $m \notin M(\mathcal{S})$ . (This probability is taken over all the coin tosses of  $G$ ,  $\mathcal{FR}$ ,  $\mathcal{FF}$  and  $\mathcal{SP}$ .)

What we have assumed is that, for infinitely many  $k$ ,

$$\varepsilon_k \cong \frac{1}{Q(k)}.$$

By Lemma 3, given  $\mathcal{S}$  and  $\sigma$ , it is now easy to compute either

- (1) a  $g$ -claw (i.e., a claw for the second claw-free pair in  $PK$ ), or
- (2) an  $f$ -claw (i.e., a claw for the first claw-free pair in  $PK$ ), or
- (3) an  $f$ -item whose root belongs to  $\mathcal{NR}(\mathcal{S})$ .

Denote the probability that case (1), (2) or (3) hold, respectively, by  $\delta_1$ ,  $\delta_2$  and  $\delta_3$ . Then, for infinitely many  $k$ , we have

$$\delta_1(k) + \delta_2(k) + \delta_3(k) \cong \varepsilon_k > \frac{1}{Q(k)}.$$

Thus either

- (1') there is an infinite set  $K_1$  so that for  $k \in K_1$   $\delta_1(k) > 1/3Q(k)$ , or
- (2') there is an infinite set  $K_2$  so that for  $k \in K_2$   $\delta_2(k) > 1/3Q(k)$ , or
- (3') there is an infinite set  $K_3$  so that for  $k \in K_3$   $\delta_3(k) > 1/3Q(k)$ .

We will show that either case leads to a contradiction.



Assume case (1') holds. Then consider the following algorithm in  $\mathcal{RA}$  that, on input  $1^k$  and a claw-free pair  $h = (d_h, h_0, h_1)$  of size  $k$ , randomly selected by  $G$ , finds an  $h$ -claw with sufficiently high probability.

ALGORITHM 1. Run  $G$  on input  $1^k$  to randomly select a quintuple  $(d_f, f_0, f_0^{-1}, f_1, f_1^{-1})$ . Select  $r_\epsilon^f \in D_f$  at random and construct the public key  $PK = (d_f, f_0, f_1, r_\epsilon^f, d_h, h_0, h_1, 2^b)$ . (Notice that  $PK$  is a random public key of size  $k$  of our signature scheme.) Randomly select the signature corpus  $\mathcal{S} \leftarrow \mathcal{C}(\mathcal{SP}, \mathcal{FR}; PK, SK, i)$ . Though  $PK$ 's matching secret key  $SK$  is not totally known, this random selection can be efficiently done as, by Lemma 4, there exists an  $\mathcal{A} \in \mathcal{RA}$  such that  $\mathcal{C}(\mathcal{SP}, \mathcal{FR}; PK, SK, i) = \mathcal{C}(\mathcal{A}, \mathcal{FR}; PK, f^{-1}, i)$ . Now run  $\mathcal{FF}$  on input  $\mathcal{S}$  and  $PK$  to sign a new message. From this last signature and  $\mathcal{S}$ , try to compute an  $h$ -claw.

Notice that, for  $k \in K_1$ , Algorithm 1 will successfully compute an  $h$ -claw with probability  $\delta_1(k) > 1/3Q(k)$ . This contradicts the claw-freeness of  $G$ .

Assume now that either (2') or (3') holds. Consider the following algorithm in  $\mathcal{RA}$ , whose input is  $1^k$  and a claw-free pair  $h = (d_h, h_0, h_1)$  of size  $k$  randomly selected by  $G$ .

ALGORITHM 2. Run  $G$  on input  $1^k$  to randomly select a quintuple  $(d_g, g_0, g_0^{-1}, g_1, g_1^{-1})$ . Randomly select the signature corpus

$$\mathcal{S} \leftarrow \mathcal{C}(\mathcal{SP}, \mathcal{SR}; (h, d_h(\cdot), g, 2^b), \{h^{-1}, g^{-1}\}, i),$$

which can be done as by Lemma 6 there exists an algorithm  $\mathcal{A} \in \mathcal{RA}$  such that

$$\mathcal{C}(\mathcal{SP}, \mathcal{SR}; (h, dh(\cdot), g, 2^b), \{h^{-1}, g^{-1}\}, i) = \mathcal{C}(\mathcal{A}, \mathcal{SR}; (h, d_h(\cdot), g, 2^b), g^{-1}, i).$$

Then run  $\mathcal{FF}$  on input  $\mathcal{S}$  and  $PK$ .

Assume that case (2') holds. Then, for  $k \in K_2$ , from the output of Algorithm 2 an  $h$ -claw can be computed with sufficiently high probability to violate the claw-freeness of  $G$ .

Finally, assume that case (3') holds and  $k \in K_3$ . Then, given a random  $x \leftarrow d_h(\cdot)$ , the following algorithm  $\mathcal{S}$  will invert  $h$  on  $x$  with nonnegligible probability (contradicting Lemma 7).  $\mathcal{S}$  runs Algorithm 2 except that, when constructing  $\mathcal{T}^h(\mathcal{S})$  as in Lemma 5, it makes  $x$  the value of a randomly selected nonroot of  $\mathcal{S}$ . Notice that this operation does not change the probability distribution of  $\mathcal{S}$ . (Recall that the preprocessing procedure of Lemma 5 just picks at random all the internal nodes of  $\mathcal{S}$ .) Thus  $\mathcal{S}$  is a random signature corpus with respect to a randomly selected public key of size  $k$ . Thus, from the output of Algorithm 2,  $\mathcal{S}$  computes an  $h$ -item with root  $r \in \mathcal{NR}(\mathcal{S})$  with probability  $\delta_3(k) > 1/3Q(k)$ . When this happens, with probability  $1/|\mathcal{NR}(\mathcal{S})|$ , we have  $r = x$ . Now, given the  $h$ -item computed,  $\mathcal{S}$  can easily compute either  $h_0^{-1}(x)$  or  $h_1^{-1}(x)$ , and Lemma 7 is contradicted. This completes the proof of the Main Theorem.  $\square$

**10. Variations and improvements.** In this section we describe ways to improve the efficiency of the proposed signature scheme without affecting its security.

**10.1. Using  $g_i$ 's to sign rather than  $g_i^{-1}$ 's.** This variation is of interest if it is substantially easier to compute  $g_0$  or  $g_1$  than to compute their inverses. In this case steps (3) and (4) in the signing procedure can be replaced by:

- (3) (*Output g-item.*) User  $A$  selects a random  $t_i^g \in D_g$ , and (using  $g_0$  and  $g_1$ ) computes the root  $r_i^g$  of the  $g$ -item  $(t_i^g, r_i^g; m_i)$ , and outputs this item.
- (4) (*Output bridge f-item.*) Using his knowledge of  $f_0^{-1}$  and  $f_1^{-1}$ , user  $A$  outputs an  $f$ -item with root  $r_i^f$  and an only child  $r_i^g$ .

Now each usage of  $g_0^{-1}$  or  $g_1^{-1}$  has been replaced by a usage of  $g_0$  or  $g_1$ .

Although one might be tempted to use this variation using one-way permutations instead of trap-door permutations for the  $g_i$ 's, this temptation should be resisted, since our proof of security does not hold if this change is made.

**10.2. Fast iterated square roots.** As we saw in § 6.3, if factoring is computationally hard, a particular family of trap-door permutations is claw-free. By using these permutations in a straightforward manner, we obtain a particular instance of our signature scheme. Let us discuss the efficiency of this instance. The computation of  $f_0^{-1}(x)$  consists of computing the square root which has Jacobi symbol 1 and is less than  $n/2$ , modulo a Blum-integer  $n$ . We can compute  $f_1^{-1}(x)$  as  $f_0^{-1}(x/4)$ . Computing  $g_0^{-1}(x)$  and  $g_1^{-1}(x)$  is the same, except for using the appropriate  $n$ . If  $n$  is  $k$ -bits long, this can be done in  $O(k^3)$  steps. Thus the signature of a  $k$ -bit message can be computed in time  $O(b \cdot k^4)$ , or in  $O(k^4)$  amortized time.

This particular instance of our scheme can be improved in a manner suggested in discussions with Oded Goldreich (see [Go86])—we appreciate his permission to quote these results here). The improvement relates to the computation of  $f_{\langle y \rangle}^{-1}(x)$  (or  $g_{\langle y \rangle}^{-1}(x)$ ).

We note first of all that taking square roots modulo  $n$  is equivalent to taking  $u$ th powers modulo  $n$ , where  $u \cdot 2 \equiv 1 \pmod{\phi(n)}$ , and where  $\phi(n)$  is Euler's phi function. More generally, to find a  $2^m$ th root  $w$  of  $x$  modulo  $n$  one can raise  $x$  to the  $v$ th power modulo  $n$ , where  $v \equiv u^m \pmod{\phi(n)}$ . Computing  $w$  by first computing  $v$  and then raising  $x$  to the  $v$ th power is substantially faster than repeatedly taking square roots.

To apply this observation, we note that the functions  $f$  defined in § 6.3 satisfy

$$f_{\langle y \rangle}^{-1}(x) = \pm \left( \frac{x}{r^{\text{rev}(\langle y \rangle)}} \right)^{2^{-m}},$$

where “rev” is the operation which reverses strings and interprets the result as an integer, where  $m$  is the length of  $\langle y \rangle$ , where all operations are performed modulo  $n$ , and where the final sign is chosen to make the result less than  $n/2$ . The only computationally difficult portion here is computing a  $2^m$ th root. Using the observation of the previous paragraph, the computation of such an  $f$ -inverse can be performed in time proportional to the cube of the length of  $n$ , in the case that messages have the same length  $k$  as  $n$ . Using these ideas, the signature of a  $k$ -bit message can be computed in time  $O(b \cdot k^3)$ , or in  $O(k^3)$  amortized time.

**10.3. “Memoryless” version of the proposed signature scheme.** The concept of a *random function* was introduced by Goldreich, Goldwasser and Micali in [GGM84].

Let  $I_k$  denote the set of  $k$ -bit integers. Let  $W_k$  denote the set of all functions from  $I_k$  to  $I_k$ , and let  $F_k \subseteq W_k$  be a set of functions from  $I_k$  to  $I_k$ . We say that  $F = \cup_k F_k$  is a *polyrandom* collection if:

(1) Each function in  $F_k$  has a unique  $k$ -bit index associated with it. Furthermore, picking such an index at random (thereby picking an  $f \in F_k$  at random) is easy.

(2) There exists a deterministic polynomial time algorithm that given as input an index of a function  $f \in F_k$  and an argument  $x$  computes  $f(x)$ .

(3) No probabilistic polynomial in  $k$ -time algorithm can “distinguish” between  $W_k$  and  $F_k$ . Formally, let  $T$  be a probabilistic polynomial-time algorithm, that on input  $k$  and access to an oracle  $O_f$  for a function  $f: I_k \rightarrow I_k$  outputs 0 or 1. Then, for all  $T$ ,

for all polynomials  $Q$ , for all sufficiently large  $k$ , the difference between the probability that  $T$  outputs 1 on access to an oracle  $O_f$  when  $f$  was randomly picked in  $F_k$  and the probability that  $T$  outputs 1 on access to an oracle  $O_f$  when  $f$  was randomly picked in  $W_k$  is less than  $1/Q(k)$ .

In [GGM84] it was shown how to construct a polyrandom collection assuming the existence of one-way functions. The existence of claw-free permutation pairs is a stronger assumption, and thus implies the existence of a polyrandom collection. See § 5.4 for an implementation of a claw-free family of functions based on factoring and [GGM84] for details on how to construct a polyrandom collection.

Leonid Levin suggested the following use of a polyrandom collection in order to reduce the amount of storage that a signer must keep from  $O(bk)$  to  $O(b)$  bits. His suggestion also eliminates the need to generate new random numbers (e.g.,  $r_i^g$ ) during the signing process.

Let  $k$  denote the security parameter. In the secret key generation phase, in addition to computing the secret trap-door pairs  $(f_0^{-1}, f_1^{-1}), (g_0^{-1}, g_1^{-1})$  user  $A$  also picks a random function  $h$  in a polyrandom collection  $F_k$ , and keeps  $h$  secret. (We assume that  $k > b$ .) During the signing process,  $A$  keeps a counter  $i$  to denote the number of times the signing algorithm has been invoked. To sign message  $m_i$ ,  $A$  signs as before, except that (using  $m$  to denote the length of  $j$ ):

- Instead of picking values  $r_j^f$  at random from  $D_f$ , he *computes* them as  $r_j^f = h(0^{k-m_j})$ .
- Instead of picking values  $r_j^g$  at random from  $D_g$ , he *computes* them as  $r_j^g = h(1^{k-m_j})$ .

We claim that the “memoryless” version of the signature scheme described above enjoys the same security properties as our original scheme. The proof (which we shall not give in detail) is based on the observation that if the memoryless scheme was vulnerable to an adaptive chosen-message attack, then it would be possible to efficiently distinguish pseudorandom functions from truly random functions.

A further improvement (due to Oded Goldreich [Go86]) removes even the necessity of remembering the number of previous signatures by picking the index  $i$  for a message  $M$  as a random  $b$ -bit string. To make this work, the maximum number of signatures that can be produced by an instance of this scheme is limited to  $2^{\sqrt{b}}$ , so that it is extremely unlikely that two messages would have the same index chosen for them. The security proof can be modified to accommodate these changes. (Note that in the preprocessing step that builds an  $f$ -tree, we would now only build a portion of it consisting of  $2^{\sqrt{b}}$  randomly chosen paths of length  $b$ .)

### 11. Open problems.

- It is an open question whether the RSA scheme is universally forgeable under an adaptive chosen-message attack.
- Can an encryption scheme be developed for which decryption is provably equivalent to factoring yet for which an adaptive chosen ciphertext attack is of no help to the enemy?

**Acknowledgments.** We are most grateful to Leonid Levin for his suggestion of how to use random functions in our scheme, in order to (almost) completely eliminate the need for storage in our signature scheme. We are also very grateful to Oded Goldreich for many valuable suggestions concerning the presentation of these results, and for suggesting the speedup described in § 10.3. We are also thankful to Avi Wigderson for helpful suggestions on the presentation of this research. Special thanks go to an anonymous referee for his very careful reading of our paper and suggested improvements.

## REFERENCES

- [BGM85] M. BEN-OR, O. GOLDBREICH, S. MICALI AND R. L. RIVEST, *A fair protocol for signing contracts*, Proc. 12th ICALP Conference, Nafplion, Greece, July 1985, pp. 43-52.
- [B182] M. BLUM, *Coin flipping by telephone*, Proc. IEEE Spring COMPCOM, 1982, San Francisco, pp. 133-137.
- [B183] ———, *How to exchange (secret) keys*, ACM Trans. Comput. Systems, 1 (1983), pp. 175-193.
- [BD85] E. BRICKELL AND J. DELAURENTIS, *An attack on a signature scheme proposed by Okamoto and Shiraishi*, Proc. Crypto 85, Springer-Verlag, New York, Heidelberg, Berlin, 1986.
- [Ch82] D. CHAUM, *Blind signatures and untraceable payments*, in Advances in Cryptography—Proc. Crypto 82, D. Chaum, R. Rivest and A. Sherman, eds., Plenum Press, New York, 1983.
- [De82] D. DENNING, *Cryptography and Data Security*, Addison-Wesley, Reading, MA, 1982.
- [DH76] W. DIFFIE AND M. E. HELLMAN, *New directions in cryptography*, IEEE Trans. Inform. Theory, IT-22, 6 (1976), pp. 644-654.
- [EAKMM85] D. ESTES, L. ADLEMAN, K. KOMPPELLA, K. MCCURLEY AND G. MILLER, *Breaking the Ong-Schnorr-Shamir signature scheme for quadratic number fields*, Proc. Crypto 85, Springer-Verlag, New York, Heidelberg, Berlin, 1986, pp. 3-13.
- [EG84] T. EL-GAMAL, *A public key cryptosystem and a signature scheme based on discrete logarithms*, Proc. Crypto 84, Springer-Verlag, New York, Heidelberg, Berlin, 1985, pp. 10-18.
- [EGL82] S. EVEN, O. GOLDBREICH AND A. LEMPEL, *A randomized protocol for signing contracts*, in Advances in Cryptology—Proc. Crypto 82, D. Chaum, R. Rivest and A. Sherman, eds., Plenum Press, New York, 1983, pp. 205-210.
- [GGM84] O. GOLDBREICH, S. GOLDWASSER AND S. MICALI, *How to construct random functions*, Proc. 25th Annual IEEE Symposium on the Foundations of Computer Science, FL, November 1984, pp. 464-479.
- [Go86] ODED GOLDBREICH, *Two remarks concerning the GMR signature scheme*, MIT Lab. for Computer Science Technical Report 715, September, 1986.
- [GK86] S. GOLDWASSER AND J. KILIAN, *Almost all primes can be quickly certified*, Proc. 18th Annual ACM Symposium on the Theory of Computing, Berkeley, CA, 1986.
- [GM82] S. GOLDWASSER AND S. MICALI, *Probabilistic encryption*, J. Comput. Systems Sci., 28 (1984), pp. 270-299.
- [GMR84] S. GOLDWASSER, S. MICALI AND R. L. RIVEST, *A “paradoxical” solution to the signature problem*, Proc. 25th Annual IEEE Symposium on the Foundations of Computer Science, Singer Island, FL, 1984, pp. 441-448.
- [GMY83] S. GOLDWASSER, S. MICALI AND A. YAO, *Strong signature schemes*, Proc. 15th Annual ACM Symposium on the Theory of Computing, Boston, MA, April 1983, pp. 431-439.
- [La79] L. LAMPORT, *Constructing digital signatures from a one-way function*, SRI Intl. CSL-98, October 1979.
- [Li81] K. LIEBERHERR, *Uniform complexity and digital signatures*, Theoret. Computer. Sci., 16 (1981) pp. 99-110.
- [LM78] S. LIPTON AND S. MATYAS, *Making the digital signature legal—and safeguarded*, Data Communications, (1978), pp. 41-52.
- [Ma79] S. MATYAS, *Digital signatures—an overview*, Computer Networks, 3 (1979), pp. 87-94.
- [MH78] R. MERKLE AND M. HELLMAN, *Hiding information and signatures in trap-door knapsacks*, IEEE Trans. Inform. Theory, IT-24 (1978), pp. 525-530.
- [Me79] R. MERKLE, *Secrecy, authentication, and public-key systems*, Ph.D. dissertation, Electrical Engineering Department, ISL SEL 79-017, 1982, Stanford University, Stanford, CA.
- [MM82] C. MEYER AND S. MATYAS, *Cryptography: A New Dimension in Data Security*, John Wiley, New York, 1982.
- [MGR85] S. MICALI, S. GOLDWASSER AND C. RACKOFF, *The knowledge complexity of interactive proof systems*, Proc. 17th Annual ACM Symposium on the Theory of Computing, Providence, RI, May 1985, pp. 291-304.
- [OS85] T. OKAMOTO AND A. SHIRAISHI, *A fast signature scheme based on quadratic inequalities*, Proc. 1985 Symposium on Security and Privacy, Oakland, CA, April 1985.
- [OSS84a] H. ONG, C. SCHNORR AND A. SHAMIR, *An efficient signature scheme based on quadratic equations*, Proc. 16th Annual ACM Symposium on the Theory of Computing, Washington, D.C., April 1984, pp. 208-217.
- [OSS84b] ———, *An efficient signature scheme based on polynomial equations*, Proc. Crypto 84, Springer-Verlag, New York, Heidelberg, Berlin, 1985, pp. 37-46.
- [Po84] J. POLLARD, *How to break the “OSS” signature scheme*, private communication, 1984.

- [Ra78] M. RABIN, *Digitalized signatures*, in Foundations of Secure Computation, R. A. DeMillo, D. Dobkin, A. Jones and R. Lipton, eds., Academic Press, New York, 1978, pp. 133–153.
- [Ra79] ———, *Digitalized signatures as intractable as factorization*, MIT Laboratory for Computer Science Technical Report MIT/LCS/TR-212, Massachusetts Institute of Technology Cambridge, MA, January 1979.
- [Ra80] ———, *Probabilistic algorithms for testing primality*, J. Number Theory, 12 (1980), pp. 128–138.
- [RV83] J. REIF AND L. VALIANT, *A logarithmic time sort for linear size networks*, Proc. 15th Annual ACM Symposium on the Theory of Computing, Boston, MA, April 1983, pp. 10–16.
- [RSA78] R. RIVEST, A. SHAMIR AND L. ADLEMAN, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. ACM, 21 (1978), pp. 120–126.
- [Sh78] A. SHAMIR, *A fast signature scheme*, MIT, Laboratory for Computer Science Technical Memo MIT/LCS/TM-107, Massachusetts Institute of Technology, Cambridge, MA, July, 1978.
- [Sh82] ———, *A polynomial time algorithm for breaking the basic Merkle–Hellman cryptosystem*, Proc. 23rd Annual IEEE Symposium on the Foundations of Computer Science, Chicago, IL, November 1982, pp. 145–152.
- [SS77] R. SOLOVAY AND V. STRASSEN, *A fast Monte-Carlo test for primality*, this Journal, 6 (1977), pp. 84–85.
- [Tu84] Y. TULPAN, *Fast cryptoanalysis of a fast signature system*, Master’s Thesis in Applied Mathematics, Weizmann Institute, Israel, 1984.
- [Wi80] H. C. WILLIAMS, *A modification of the RSA public-key cryptosystem*, IEEE Trans. Inform. Theory, IT-26 (1980), pp. 726–729.
- [Yu79] G. YUVAL, *How to swindle Rabin*, Cryptologia, 3 (1979), pp. 187–189.

## COMPLEXITY MEASURES FOR PUBLIC-KEY CRYPTOSYSTEMS\*

JOACHIM GROLLMANN† AND ALAN L. SELMAN‡

**Abstract.** A general theory of public-key cryptography is developed that is based on the mathematical framework of complexity theory. Two related approaches are taken to the development of this theory, and these approaches correspond to different but equivalent formulations of the problem of cracking a public-key cryptosystem (PKCS). The first approach is to model the cracking problem as a partial decision problem called a “promise problem.” Every NP-hard promise problem is shown to be uniformly NP-hard, and a number of results and a conjecture about promise problems are shown to be equivalent to separability assertions for sets in NP that are the natural analogues of well-known results in classical recursion theory. The conjecture, if it is true, implies nonexistence of PKCS having NP-hard cracking problems. The second approach represents the cracking problem of a PKCS as a partial computational problem directly. Using this approach, it is shown that one-way functions exist if and only if  $P \neq UP$  and that one-way functions with greater cryptographic significance exist if and only if NP contains disjoint P-inseparable sets. The paper concludes with a discussion of almost-everywhere security measures for PKCS.

**Key words.** public-key cryptography, promise problems, one-way function, UP, security, inseparable, NP-hard, uniformly NP-hard

**1. Introduction.** Since any public-key cryptosystem (PKCS) can be cracked non-deterministically in polynomial time, and since, minimally, a PKCS should not be crackable deterministically in polynomial time, a proof that a given system is secure would imply  $P \neq NP$ . Several concrete systems have been created in recent years, but none have been proved secure in this weak sense, even if  $P \neq NP$  is assumed as a hypothesis. The reasons for this vary, but some general remarks can be made. The Merkle–Hellman schemes [MH78] are based on an NP-complete problem (Knapsack), but the problem of cracking these schemes is not necessarily polynomially equivalent to the original problem [Sha82]. Similarly, the cracking problem for the Rivest–Shamir–Adleman scheme [RSA78] is not known to be equivalent to integer factorization. Even if it were, as is true of the Rabin and Williams variants [Rab79], [Wil80], it is very unlikely that intractability of integer factorization is equivalent to the  $P \neq NP$  problem itself. (It is known [Bra79] that integer factorization cannot be NP-hard unless  $NP = co-NP$ .) Security of a concrete PKCS depends on intractability of the concrete problem on which it is based, and, in particular, no concrete PKCS has been designed whose security is equivalent to any NP-hard problem.

Very informally, the “cracking problem” for a PKCS is the problem of computing the message  $M$  that is encoded by a cryptogram  $C$  without knowledge of the current decryption key. In order to make such a definition precise, it is necessary to determine appropriate complexity measures according to which the cracking problem is to be hard. Typically, computational problems (e.g., given a graph, output a hamiltonian if one exists) are transformed into polynomial time equivalent decision problems (e.g.,

---

\* Received by the editors November 24, 1985; accepted for publication May 14, 1987. Some of the results of this paper were presented in preliminary form at the 25th IEEE Symposium on Foundations of Computer Science. This research was supported by the National Science Foundation under grants DCR84-02033 and DCR86-96082.

† Siemens AG, Centralbereich Technik, D-8000 Munich 80, West Germany. The work of this author was supported by the Fulbright Kommission of West Germany.

‡ College of Computer Science, Northeastern University, Boston, Massachusetts 02115.

given a graph, does it have a hamiltonian), and the complexity of the original computational problem is measured by classifying the associated decision problem. When this transformation works, as it does for computational problems that are NP-complete, it is because computing a partial function and accepting membership in its domain are polynomially equivalent. However, this program does not work in general and does not work for PKCS. In general, computational problems are not polynomially equivalent to their corresponding domain recognition problems [Val76], [Mil76], [Bra79] (and cf. § 4 below). Also, we will argue in the course of this paper that it is more important to know the complexity of the class of extensions of a partial function than it is to know the complexity of the given function. General studies of public-key cryptography based on these observations were initiated by Brassard [Bra79], [Bra81], [Bra83a], [Bra83b] and by Even and Yacobi [EY80a], [EY80b]. In [EY80a], Even and Yacobi propose a definition for public-key cryptography, formulate the cracking problem of a PKCS as a partial decision problem called a “promise problem,” and raise the question of whether there exist NP-hard PKCS. In this approach certain hardness criteria for cryptosystems (Is a PKCS crackable in polynomial time? Is there an NP-hard PKCS?) can be translated into properties about the associated promise problems. In [ESY84], Even, Selman, and Yacobi present a plausible conjecture about promise problems that implies that no PKCS has an NP-hard cracking problem. A major component of the work in this paper consists of further analysis of this conjecture. The main technical result is that every NP-hard promise problem is uniformly NP-hard. This result may be a step toward proving the conjecture raised in [ESY84] and consequently toward proving that PKCS that are NP-hard to crack do not exist.

The broader motivation of the research project described herein is to develop complexity theory that has cryptographic significance. Our approach is to base the investigation of questions such as “What is a secure PKCS?” and “Do secure PKCS exist?” directly on structural properties of complexity classes such as NP and on separation assertions about complexity classes, rather than on presumed intractability of individual concrete problems.

Two complexity theoretic hypotheses emerge as most relevant for capturing the complexity issues that surround public-key cryptography. These hypotheses are that  $P \neq UP$  and that NP contains P-inseparable sets. (Two disjoint sets A and B are *P-inseparable* if every set that includes one of them and is disjoint from the other does not belong to P. UP is the set of problems in NP that are accepted by unambiguous polynomial time-bounded Turing machines (UTMs) [Val76].) To see the relevance of these hypotheses, consider, for example, the “one-way function.” Informally, a one-way function is one that is easy to compute but hard to invert. To make this notion precise, let us define a (partial) function  $f$  to be a *one-way* function if  $f$  is one-one,  $f$  is honest (to be defined in § 2.2),  $f$  can be computed in polynomial time, and  $f^{-1}$  is *not* computable in polynomial time. We will prove that one-way functions exist if and only if  $P \neq UP$  and that one-way functions with range in P exist if and only if  $P \neq UP \cap \text{co-UP}$ . According to this definition, since a one-way function is polynomial time computable, its domain belongs to P (this point will be clarified in § 2.2), and so a polynomial time Turing machine that computes a one-way function must behave in a specified manner on all input instances not in its domain (for example, output a special marker). When we analyze public-key cryptosystems we will see that one should not care what happens on inputs that are not in the domain of a proposed one-way function. This defect will be remedied with a definition of a *weak* one-way function. With this notion in hand, we will prove that there exist public-key cryptosystems that cannot be cracked in polynomial time only if there exist P-inseparable sets in NP, and we will prove that

the latter property is equivalent to the existence of weak one-way functions. Furthermore, since every one-way function is a weak one-way function,  $P \neq UP$  implies all of these properties.

We cannot say that the complexity class  $UP$  forms the exact boundary of interest with regard to public-key cryptography, but we will show that there exist PKCS that cannot be cracked in polynomial time and that also satisfy reasonable restrictions only if  $P \neq UP$ , and, as described above, every problem in  $UP - P$  yields some PKCS that does not have a polynomial time solvable cracking problem. To say the least, we would know a lot more about what kinds of cryptographic systems are possible if we knew more about the class  $UP$ .

A PKCS may be uncrackable in polynomial time and yet there may be exist a polynomial time algorithm that cracks most messages. (Such a system will be exhibited in § 4.) For this reason it is understood that infinitely often (i.o.) complexity measures are insufficient for cryptography. Nevertheless, much effort is spent in this paper studying i.o. measures, and for this we give the following justification. Cryptosystems that are secure cannot exist unless there exist systems that cannot be cracked in polynomial time. In this paper we are not interested in the development of secure systems; our concern is with the development of complexity theoretic underpinnings. What we see in this paper is that questioning the existence of such intractable systems already entails difficult structural properties of  $NP$ .

We will consider measures that are more appropriate. We will see that the usual immunity and bi-immunity notions for specifying complexity almost everywhere (a.e.) do not work, and in particular, we will see that there is no elegant way to formulate a.e. security for PKCS in terms of promise problems. Primarily for this reason, we include here a development of complexity classes of functions. Once we relate function classes with the familiar classes of decision problems, then we will give definitions of security measures for PKCS directly in terms of function classes. We will demonstrate that the strongest possible sense in which a PKCS can be secure with respect to message cracking is that every deterministic, randomized, or probabilistic algorithm computable in polynomial time cracks at most a sparse subset of the message space. It will be seen finally that natural PKCS can exist that are secure in this strongest possible sense only if they are based on problems in  $NP$  that possess specific complexity theoretic properties (no nonsparse subset can be recognized by any probabilistic polynomial time Turing machine).

The paper is organized as follows. The next section is of a preliminary nature. At the risk of tediousness, it is rather long; the only way to carefully lay foundations is to be careful about the foundations. Here we collect various specialized complexity theoretic notions, present a framework for specifying computational complexity of functions, and review definitions and results from [ESY84], [EY80b] about promise problems and their relation to public-key cryptography. This synopsis is needed for completeness. Also, a number of interesting corollaries will follow from the work in those papers. Complexity issues concerning promise problems are taken up in § 3. There we focus attention on the i.o. measures “not crackable in polynomial time” and “ $NP$ -hard,” and there the proof that every  $NP$ -hard promise problem is uniformly  $NP$ -hard can be found. Sections 4 and 5 study the cracking problem directly as a computational problem, making use of the framework that was developed in § 2. Section 4 contains definitions of one-way functions and weak one-way functions, their relationships to hypotheses about  $UP$  and about  $P$ -inseparable sets, and discusses connections between one-way functions and public-key cryptography. Section 5 discusses a.e. complexity measures.



The reader will quickly observe that not every question that will be raised has been answered. In § 6, the last section of this paper, a number of relativization results are stated that suggest that answers are not readily forthcoming, for they cannot be obtained by proof techniques that relativize to oracles.

**2. Preliminaries.** Unless otherwise specified, sets are subsets of  $\Sigma^*$ ,  $\Sigma = \{0, 1\}$ , and predicates vary over  $\Sigma^*$ . The empty word is denoted by  $\lambda$ ,  $\Sigma^* - A$  is denoted by  $\bar{A}$ , and  $\leq$  denotes lexicographic ordering defined on  $\Sigma^*$ . Let  $(x, y)$  be the ordered pair with components  $x$  and  $y$  and let  $\langle \cdot, \cdot \rangle$  be a polynomial time computable pairing function with polynomial time computable inverses. The reader is assumed to be familiar with standard concepts and notations of polynomial time-bounded complexity theory [HU79], [GJ79], but a number of specialized notions are described below. The subsections on promise problems and public-key cryptosystems comprise a synopsis of the work by Even, Selman, and Yacobi [ESY84], [EY80a].

**2.1. Complexity theoretic notions.** The notations  $\leq_T^P$  and  $\leq_m^P$  denote polynomial time-bounded Turing and many-one reducibility, respectively. We follow the conventions that a set  $L$  is NP-complete if  $L \in \text{NP}$  and  $\text{SAT} \leq_m^P L$ , and that a set is NP-hard if  $\text{SAT} \leq_T^P L$ , where SAT denotes the well-known satisfiability problem.

A set  $A$  is *a.e. complex* if for every Turing machine  $M$  that accepts  $A$  and every polynomial  $p$ ,  $M$  runs for more than  $p(|x|)$  steps for all but finitely many strings  $x$  [Ber76], [Rab60]. For an arbitrary class of languages  $\mathcal{C}$ , an infinite set  $A$  is  $\mathcal{C}$ -immune (or immune to  $\mathcal{C}$ ) if no infinite subset of  $A$  is in  $\mathcal{C}$ . Immunity, and the special case of P-immunity, are defined and discussed in [Ber76], [FS74], [KM81].  $A$  is *bi-immune* to  $\mathcal{C}$  if  $A$  and  $\bar{A}$  are both immune to  $\mathcal{C}$ ; see [BS85], [KM81]. It is observed in [BS85] that a set is a.e. complex if and only if it is bi-immune to P.

Two disjoint sets  $A$  and  $B$  are called P-inseparable if no set  $L$  with the property  $A \subseteq L \subseteq \bar{B}$  is in P. Otherwise,  $A$  and  $B$  are called P-separable.

A nondeterministic Turing machine (NTM) that has at most one accepting computation for any input is called by Valiant [Val76] an *unambiguous* Turing machine (UTM). Let UP denote the set of languages accepted by UTMs in polynomial time. Obviously,  $P \subseteq \text{UP} \subseteq \text{NP}$ , and it is not known whether either inclusion is proper.

Following Papadimitriou and Yannakakis [PY84] define  $D^P = \{L_1 \cap L_2 \mid L_1 \in \text{NP} \text{ and } L_2 \in \text{co-NP}\}$ .  $D^P$  contains interesting combinatorial problems not known to belong to NP. Trivially,  $\text{NP} \subseteq D^P$ ,  $\text{co-NP} \subseteq D^P$ , and  $D^P \subseteq \Delta_2^P$  (where  $\Delta_2^P$  is defined to be  $P^{\text{NP}}$ ), and it is expected that each of these inclusions is proper. Indeed,  $\text{NP} = D^P$  if and only if  $\text{NP} = \text{co-NP}$ .

Let BPP be the class of sets that can be decided by a probabilistic Turing machine with bounded error probability in polynomial time. The definition is due to Gill [Gil77] and the reader is referred to [Gil77] for a detailed exposition. For a problem in BPP any small probability of error can be achieved by running the given machine many times on the same input and taking the majority result. For this reason, problems in BPP are tractable in practice. Gacs (see reference in [Sip83]) and Lautemann [Lau83] showed that  $\text{BPP} \subseteq \Sigma_2^P (= \text{NP}^{\text{NP}})$ . The relationship between BPP and NP is not known, but Ko [Ko82] has shown that the supposition  $\text{BPP} = \text{NP}$  has consequences that are unlikely to be true. Our work will not directly involve this class, but we must keep in mind that cryptosystems should not be susceptible to cryptanalyst attack by any tractable process.

In § 4 the reader will need to be familiar with the class R of problems that have efficient randomized algorithms. The definition of this class is due to Adleman and

Manders [AM77]. The definition implies that  $P \subseteq R \subseteq NP$  and it is well known that  $R \subseteq BPP$  [Gil77].

**2.2. Complexity classes of functions.** We take a look now at some complexity classes of functions. The development is repeated in part from [SXB83], [BLS84], [BLS85] and is also in the spirit of [Val76].

Consider nondeterministic transducers with accepting states. A transducer  $T$  computes a value  $y$  on an input string  $x$  if there is an accepting computation of  $T$  on  $x$  for which  $y$  is the final contents of  $T$ 's output tape. In general, such transducers compute partial multivalued functions, but we will be concerned only with single-valued functions.

**DEFINITION 1.** (i) NPSV is the set of all partial single-valued functions computed by nondeterministic polynomial time-bounded transducers.

(ii) UPSV is the set of all functions  $f$  in NPSV such that for some nondeterministic polynomial time-bounded transducer that computes  $f$ , for every  $x \in \text{dom}(f)$  there is exactly one accepting computation.

(iii) PSV is the set of all partial single-valued functions computed by deterministic polynomial time-bounded transducers.

If  $T$  is a transducer that computes a partial function  $f$ , then the domain of  $f$  is the set of words accepted by  $T$ . If, in addition,  $T$  is polynomial time-bounded, then there is a polynomial  $p$  such that every word  $x$  accepted by  $T$  is accepted within  $p(|x|)$  steps. Given a polynomial  $p = n^c$  there is a Turing machine  $M_p$  that on every input of length  $n$  will execute for exactly  $n^c$  steps. It is possible to design a Turing machine  $M'$  that simulates  $T$  and uses the polynomial "clock"  $M_p$  to guarantee that  $T$  is not simulated on an input  $x$  for more than  $|x|^c$  steps. Clearly  $L(M') = L(T)$ . In the case that  $T$  is deterministic, so is  $M'$ , and since  $M'$  halts within  $n^c$  steps on every input word of length  $n$ ,  $M'$  can be designed to enter a reject state for every word that is not accepted. Hence, the domain of a partial function in PSV is a set in P. Similarly, the domain of a partial function in NPSV (UPSV) is a set in NP (UP, respectively). In part, these considerations are in analogy with recursive function theory. If a Turing machine  $T$  computes a partial recursive function  $f$ , then the domain of  $f$  is the set of words accepted by  $T$ . The analogy breaks down, however, for the domain of a partial recursive function is not necessarily a recursive set. After all, since the running time of  $T$  is not necessarily recursively bounded, no Turing machine clock can be used to detect nonacceptance.

A partial function  $f$  is *honest* if there is a polynomial  $q$  such that for every  $y \in \text{range}(f)$  there exists  $x \in \text{dom}(f)$  such that  $f(x) = y$  and  $|x| \leq q(|y|)$ . If  $f$  is an honest function in PSV and  $B \in NP$ , then  $f(B) \in NP$ .

For any partial function  $f$ ,  $\text{graph}(f) = \{(x, y) \mid x \in \text{dom}(f) \text{ and } y = f(x)\}$ . Given an arbitrary class  $\mathcal{F}$  of partial functions, let  $\mathcal{F}_g = \{f \in \mathcal{F} \mid \text{graph}(f) \in \mathcal{F}\}$ . It is easy to see that  $\text{NPSV}_g = \text{UPSV}_g$ .

The *projection* of a function  $f$  is the set  $R_f = \{(x, y) \mid x \in \text{dom}(f) \text{ and } y \leq f(x)\}$ . This construction is defined in [Mil76]. Miller shows for total functions  $f$  that are polynomial length-bounded (i.e., there is a polynomial  $q$  such that for all  $x$ ,  $|f(x)| \leq q(|x|)$ ), that  $R_f \in P$  if and only if  $f \in \text{PSV}$ . The equivalence holds for partial functions as well without any change of proof. (The proof from right to left is trivial and from left to right employs a binary search algorithm.)

**PROPOSITION 1.** (i) [SXB83]  $P = NP$  if and only if  $\text{PSV} = \text{NPSV}$ .

(ii) *The following are equivalent:*

(a)  $P = UP$ .

(b)  $PSV = UPSV$ .

(c)  $PSV = UPSV_g$ .

*Proof.* We prove (ii). Obviously (b) implies (c). To show that (c) implies (a), let  $S \in UP$  and let  $M$  be a UTM that accepts  $S$  in polynomial time. For each word  $x$  in  $S$ , let  $comp_M(x)$  denote the unique accepting computation of  $M$  on input  $x$ . Then, the function  $comp_M$  belongs to  $UPS\mathcal{V}_g$  and  $dom(comp_M) = S$ . Since  $comp_M \in PSV$  follows, we conclude that  $S \in P$ .

To show that (a) implies (b), let  $f \in UPSV$ . The projection  $R_f$  is in  $UP$  as witnessed by an unambiguous machine that behaves as follows:

On input  $(x, y)$ , unambiguously compute  $f(x)$ . If successful, then determine from the output tape whether  $y \cong f(x)$ .

By assumption,  $R_f \in P$ , and so  $f \in PSV$  follows immediately from the above remarks.  $\square$

The equivalence of (a) with (c) was first proved in [Val76]. Stronger relations than given by Proposition 1 will be developed in § 4.

If primality testing belongs to  $UP$  as some apparently believe (assuming the extended Riemann hypothesis it belongs to  $P$  [Mil76]), then integer factorization belongs to  $UPS\mathcal{V}$ . A very reasonable candidate for a function in  $UPS\mathcal{V} - PSV$ , which will be discussed in detail in a later section, is the discrete logarithm problem.

A partial function  $g$  is an *extension* of a partial function  $f$  if  $dom(f) \subseteq dom(g)$  and  $f$  and  $g$  coincide on  $dom(f)$ . If  $f$  has an extension in  $PSV$ , then  $f$  also has a total extension in  $PSV$  because the domain of a function in  $PSV$  belongs to  $P$ . A function  $f \in UPS\mathcal{V}$  with  $dom(f) \notin P$  and hence  $f \notin PSV$  may have an extension in  $PSV$ ; for example, if  $S \in UP - P$ , then the partial function  $f$  defined by

$$f(x) = \begin{cases} 1 & \text{if } x \in S, \\ \uparrow & \text{else} \end{cases}$$

(we use “ $\uparrow$ ” as an abbreviation for “undefined”) belongs to  $UPS\mathcal{V} - PSV$  and obviously has a total extension in  $PSV$ . The behavior of functions in  $UPS\mathcal{V}_g$  is quite different than this, because if  $f \in UPS\mathcal{V}_g$  and  $dom(f) \notin P$ , then no extension of  $f$  is in  $PSV$ .

**2.3. Promise problems.** A promise problem is a formulation of a partial decision problem that has the structure

<b>input</b>	$x$ ;
<b>promise</b>	$Q(x)$ ;
<b>property</b>	$R(x)$ ;

where  $Q$  and  $R$  are predicates. Formally, a *promise problem* is a pair of predicates  $(Q, R)$ . A deterministic Turing machine  $M$  *solves*  $(Q, R)$  if

$$\forall x [Q(x) \rightarrow [M(x) \text{ converges} \wedge [M(x) = \text{“yes”} \leftrightarrow R(x)]]].$$

A promise problem  $(Q, R)$  is *solvable* if there is a Turing machine  $M$  that solves it. If  $(Q, R)$  is solved by  $M$ , then the language  $L(M)$  accepted by  $M$  is called a *solution* of  $(Q, R)$ . Note that the behavior of  $M$  is of interest only for those input values  $x$  for which the promise  $Q(x)$  is true.  $M$  need not give the correct answer or even halt, for instances that do not satisfy the promise.

Our interest is with promise problems  $(Q, R)$  such that  $Q$  and  $R$  are recursive predicates and the only solutions of interest are recursive sets. We assume these conditions to hold for all promise problems considered. It is easy to see that every

recursive solution of  $(Q, R)$  is of the form  $(Q \cap R) \cup X$  where  $X$  is some recursive subset of  $\bar{Q}$ . We frequently take the liberty of writing predicates as sets and vice versa.

NPP is the class of all promise problems  $(Q, R)$  that have a solution in NP. Co-NPP is the class of all promise problems  $(Q, R)$  such that  $(Q, \bar{R})$  is in NPP; equivalently, co-NPP is the class of all promise problems  $(Q, R)$  that have a solution in co-NP. This follows from the fact that a recursive set  $L$  is a solution in NP of  $(Q, \bar{R})$  if and only if  $\bar{L}$  is a solution in co-NP of  $(Q, R)$ . Every set  $S$  in NP (co-NP) may be thought of as the promise problem  $(\Sigma^*, S)$  with the universal promise  $\Sigma^*$ , and  $(\Sigma^*, S) \in$  NPP (co-NPP, respectively). From these remarks it follows that  $\text{NPP} = \text{co-NPP}$  if and only if  $\text{NP} = \text{co-NP}$ .

DEFINITION 2. (i) A promise problem  $(Q, R)$  is *Turing reducible in polynomial time* to a promise problem  $(S, T)$ , in symbols,  $(Q, R) \leq_T^{\text{PP}} (S, T)$ , if, for every solution  $A$  of  $(S, T)$ , there is a deterministic polynomial time-bounded oracle Turing machine  $M$  such that  $M$  with oracle  $A$  solves  $(Q, R)$ .

(ii) A promise problem  $(Q, R)$  is *uniformly Turing reducible in polynomial time* to a promise problem  $(S, T)$ , in symbols,  $(Q, R) \leq_{\text{UT}}^{\text{PP}} (S, T)$ , if there is a deterministic polynomial time-bounded oracle Turing machine  $M$  such that, for every solution  $A$  of  $(S, T)$ , The language recognized by  $M$  with oracle  $A$  is a solution of  $(Q, R)$ .

(iii) A promise problem  $(Q, R)$  is *NP-hard* if every solution is NP-hard, i.e., for every solution  $A$  of  $(Q, R)$  there is a deterministic polynomial time-bounded oracle Turing machine  $M$  such that  $\text{SAT} = L(M, A)$ .

(iv) A promise problem  $(Q, R)$  is *uniformly NP-hard* if there exists a deterministic polynomial time-bounded oracle Turing machine  $M$  such that for every solution  $A$  of  $(Q, R)$ ,  $\text{SAT} = L(M, A)$ .

If an NP-hard promise problem could be solved in polynomial time by some deterministic Turing machine  $M$ , then by definition  $L(M)$  is an NP-hard set, and therefore  $\text{P} = \text{NP}$  follows. So, assuming  $\text{P} \neq \text{NP}$ , no NP-hard promise problem can be solved in polynomial time. Obviously, “uniformly NP-hard” implies “NP-hard” and “uniformly Turing reducible” implies “Turing reducible,” and one would guess the converse to be false, as in fact is conjectured in [ESY84]. We will nevertheless show in § 3 that these notions are equivalent. A set  $S \subseteq \Sigma^*$  is NP-hard if and only if the promise problem  $(\Sigma^*, S)$  is NP-hard. A promise problem  $(S, T)$  is NP-hard if and only if every promise problem  $(Q, R) \in \text{NPP}$  is Turing-reducible to  $(S, T)$  in polynomial time.

The next proposition shows that finding an algorithm to compute a function in NPSV is equivalent in a natural way to finding an algorithm to solve a promise problem. Given  $f \in \text{NPSV}$  define  $Q_f = \text{dom}(f) \times \Sigma^*$  and let  $R_f$  be the projection of  $f$ . Let us call  $(Q_f, R_f)$  the promise problem *associated with*  $f$ .

PROPOSITION 2. (i) If  $f \in \text{NPSV}$  and  $A$  is a solution of  $(Q_f, R_f)$ , then there is a total function  $h$  that extends  $f$  such that  $h \leq_T^{\text{P}} A$ .

(ii) If  $f \in \text{NPSV}$  and  $h$  is a total extension of  $f$ , then there is a solution  $A$  of  $(Q_f, R_f)$  such that  $A \leq_T^{\text{P}} h$ .

*Proof.* (i) On an input string  $x$ , apply a binary search algorithm to find the largest  $y$  (whose length is within the right polynomial length of  $x$ ) such that  $(x, y) \in A$ . Let  $h(x) = y$ . If no such  $y$  exists, then define  $h(x)$  to be  $\lambda$ . Since  $A$  is a solution,  $y = h(x) = f(x)$  for  $x \in \text{dom}(f)$ .

(ii) Given  $h$ , define  $A = R_h$ , the only solution of  $(Q_h, R_h)$ . It is easy to see that  $A$  is a solution of  $(Q_f, R_f)$  and  $A \leq_T^{\text{P}} h$ .  $\square$

The next observations are most important for what will follow. Given  $f \in \text{NPSV}$ , one can conclude  $Q_f \in \text{NP}$  and  $(Q_f, R_f) \in \text{NPP} \cap \text{co-NPP}$ . Namely,  $\text{dom}(f) \in \text{NP}$ ,  $R_f$  is a solution in NP of  $(Q_f, R_f)$ , and  $\{(x, y) \mid x \in \text{dom}(f) \text{ and } y > f(x)\}$  is a solution in

NP of  $(Q_f, \bar{R}_f)$ . The cracking problem of a PKCS will be a function in NPSV, and so we will concentrate our efforts on the study of promise problems that satisfy these two properties.

It is well known that  $NP \cap \text{co-NP}$  contains an NP-hard set if and only if NP is closed under complements [Bra79], [Sel78]. Hence, the former property is unlikely to be true. To the extent that one expects complexity classes of promise problems to behave as do the corresponding classes of decision problems, it is reasonable to conjecture that no promise problem  $(Q, R)$  such that  $Q \in NP$  and  $(Q, R) \in NPP \cap \text{co-NPP}$  is also NP-hard. Indeed, this conjecture is made in [ESY84]. Now let us state it formally.

*Conjecture 1.* There exists no promise problem  $(Q, R)$  such that

- (i)  $Q \in NP$ ,
- (ii)  $(Q, R) \in NPP \cap \text{co-NPP}$ , and
- (iii)  $(Q, R)$  is NP-hard.

As a consequence of this conjecture, if  $f \in NPSV$ , then  $(Q_f, R_f)$  is not NP-hard; that is, there must be some total extension  $h$  of  $f$  that cannot be used as an oracle for deciding satisfiability in polynomial time. Also, the conjecture trivially implies  $NP \neq \text{co-NP}$  and it is proved in [ESY84] that the conjecture implies  $UP \neq NP$ .

We need to recall the following two main results of [ESY84], which we will state here as propositions. The first of these results states that a variant of the conjecture is true, assuming  $NP \neq \text{co-NP}$ .

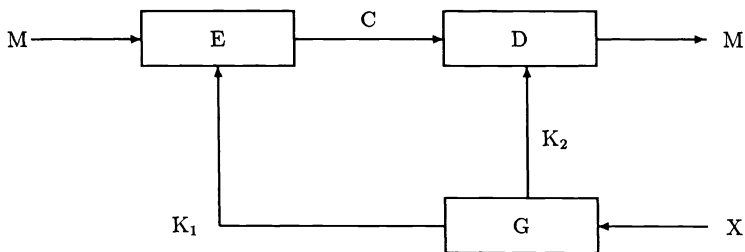
**PROPOSITION 3.** [ESY84] *NP  $\neq$  co-NP if and only if there exists no promise problem  $(Q, R)$  such that*

- (i)  $Q \in \text{co-NP}$ ,
- (ii)  $(Q, R) \in NPP \cap \text{co-NPP}$ , and
- (iii)  $(Q, R)$  is uniformly NP-hard.

In contrast, the next result states that there do exist promise problems  $(Q, R)$  such that  $Q \in D^P$  and conditions (ii) and (iii) of the conjecture hold. Let  $\oplus$  denote the logical operator “exclusive or.” We take the liberty of writing SAT as a predicate so that  $SAT(x)$  asserts that  $x$  is the string encoding of a satisfiable formula of propositional logic. The predicates EX and SAT-1 over  $\Sigma^* \times \Sigma^*$  are defined by  $EX(x, y) \leftrightarrow SAT(x) \oplus SAT(y)$  and  $SAT-1(x, y) \leftrightarrow SAT(x)$ .

**PROPOSITION 4.** *EX is a complete set for  $D^P$  and the promise problem  $(EX, SAT-1)$  belongs to  $NPP \cap \text{co-NPP}$  and is uniformly NP-hard.*

**2.4. Public-key cryptosystems.** A PKCS consists of three deterministic and publicly known algorithms,  $E$ ,  $D$ , and  $G$ , that operate in polynomial time. Figure 1 gives the basic layout.  $E$  is the *encryption algorithm*,  $D$  is the *decryption algorithm*, and  $G$  is the



Transmitter

Receiver

FIG. 1. Layout of a public-key cryptosystem.

key generator.  $M$ ,  $C$ ,  $K_1$ ,  $K_2$ , and  $X$  will be binary words, called the *message*, *cryptogram*, *encryption key*, *decryption key*, and *trapdoor*, respectively.

Prior to transmission of messages  $M$  of length  $n$ , the receiver randomly generates some  $X$ , where  $|X|$  is bounded by a polynomial in  $n$ , and then uses  $X$  to compute the pair  $G(X) = (K_1, K_2)$ . Pairs that can be generated in this way are called *legal* pairs. Components,  $K_1$ , and  $K_2$ , of legal pairs are called *legal* encryption keys, decryption keys, respectively.  $K_1$  and  $n$ , which is assumed to be computable in polynomial time from  $K_1$ , are made publicly known, but  $X$  and  $K_2$  remain private. Legal pairs may be used for encoding and decoding purpose for a relatively long time. We also call triples  $(n, K_1, M)$  *legal*, if  $|M| = n$ , and  $K_1$  can be used for encryption of messages of length  $n$ .

When a transmitter wants to send a message  $M$  of length  $n$  to a receiver who published key  $K_1$ , he computes  $C = E(K_1, M)$  and sends  $C$  on an open channel. The receiver, knowing  $K_2$ , reconstructs  $M$  by  $M = D(K_2, C)$ . It is assumed that, if  $(K_1, K_2)$  is a legal pair for messages of length  $n$  and if  $|M| = n$ , then  $M = D(K_2, E(K_1, M))$ . This inverse condition guarantees that the function  $\lambda M E(K_1, M)$  is one-one for each legal key  $K_1$ .

**2.5. Cracking problems.** Informally, the cracking problem is the problem of computing  $M$  such that  $E(K_1, M) = C$  if such  $M$  exists, and a solution to the cracking problem is an algorithm that on input  $(n, K_1, C)$ , where  $K_1$  is a legal encryption key for messages of length  $n$ , correctly computes  $M$  such that  $E(K_1, M) = C$ . That is, a *solution* of the cracking problem is an algorithm that computes an extension (not necessarily total) of the following partial function, called “Crack.”

$\text{Dom}(\text{Crack}) = \{(n, K_1, C) \mid K_1 \text{ is legal for messages of length } n \text{ and there is a message } M, |M| = n, \text{ such that } E(K_1, M) = C\}$ . For  $(n, K_1, C) \in \text{dom}(\text{Crack})$ ,  $\text{Crack}(n, K_1, C) = M \leftrightarrow |M| = n \text{ and } E(K_1, M) = C$ .

We assume Crack is polynomial length-bounded (i.e.,  $|M|$  is bounded by some polynomial in  $|K_1| + |C|$ ), so as to avoid the uninteresting case that Crack has no solution that is computable in polynomial time simply because  $|M|$  for which  $E(K_1, M) = C$  is too long.

Clearly  $\text{Crack} \in \text{NPSV}$ . Let us define the promise problem version of the cracking problem to be the promise problem  $\text{CP} = (Q_{\text{Crack}}, R_{\text{Crack}})$  that is associated with Crack. Explicitly stated, CP is the following promise problem.

**input**  $n, K_1, C$ , and  $M'$

**promise**  $K_1$  is a legal encryption key for  
messages of length  $n$  and there exists a message  $M$ ,  
 $|M| = n$ , such that  $E(K_1, M) = C$ .

**property**  $M' \equiv M$ , where  $M$  is the message which satisfies  $E(K_1, M) = C$ .

We conclude immediately that  $Q_{\text{Crack}} \in \text{NP}$  and  $\text{CP} \in \text{NPP} \cap \text{co-NPP}$ . Recall also that solving CP is polynomial-time equivalent to computing an extension of Crack.

Attempts to formulate the cracking problem as an ordinary decision problem would lead to faulty considerations. To see this, tentatively define the conjunction  $Q_{\text{Crack}} \cap R_{\text{Crack}}$  to be the cracking decision problem. One might have a PKCS, say  $S$ , such that the cracking decision problem  $Q_{\text{Crack}} \cap R_{\text{Crack}}$  is difficult to solve. But if there is an efficient algorithm that solves Crack, i.e., that efficiently finds  $M$  when the promise predicate  $Q_{\text{Crack}}$  is true and that gives “garbage” output when  $Q_{\text{Crack}}$  is false, then  $S$  is not a usable system. On the other hand, if Crack has no efficient algorithm, then the promise problem  $(Q_{\text{Crack}}, R_{\text{Crack}})$  has no efficient solution, and so the cracking

decision problem  $Q_{\text{Crack}} \cap R_{\text{Crack}}$  is difficult to solve also, because any algorithm that solves  $Q_{\text{Crack}} \cap R_{\text{Crack}}$  also solves  $(Q_{\text{Crack}}, R_{\text{Crack}})$ .

**3. Complexity of promise problems.** In this section we present new results about promise problems—all with an eye toward setting the conjecture raised in § 2.3. Recall that the cracking promise problem CP of a PKCS has promise  $Q_{\text{Crack}} \in \text{NP}$  and  $\text{CP} \in \text{NPP} \cap \text{co-NPP}$ . The conjecture that no such promise problem can be NP-hard implies that no PKCS can be NP-hard to crack. Analysis of the conjecture will lead toward development of a theory of disjoint pairs of sets in NP, and this development in turn will give further justification for its reasonableness. Also, this development will yield complexity theoretic conditions for the existence of PKCS that have intractable cracking problems (i.e., such that CP has no solution in P).

**3.1. On NP-hard promise problems.** A corollary of our first significant result states that every NP-hard promise problem is uniformly NP-hard.

LEMMA 1.  $(Q, R)$  and  $(Q, R \cap Q)$  have the same solutions. Thus,  $(Q, R)$  is (uniformly) NP-hard if and only if  $(Q, R \cap Q)$  is (uniformly) NP-hard. And,  $(Q, R)$  is (uniformly) Turing reducible to  $(S, T')$  if and only if  $(Q, R')$  is (uniformly) Turing reducible to  $(S, T)$ , where  $R' \in \{R, R \cap Q\}$  and  $T' \in \{T, T \cap S\}$ .

Assume without loss of generality, therefore, that  $R \subseteq Q$ . Then  $S$  is a solution of  $(Q, R)$  if and only if  $S = R \cup X$ , where  $X$  is a recursive subset of  $\bar{Q}$ . Also, for “reducibility” read “polynomial time-bounded reducibility” throughout this section.

LEMMA 2. “Uniform Turing reducibility is closed under finite variations.”  $(Q, R)$  is uniformly Turing reducible to  $(S, T)$  if and only if for any finite sets  $Q', R', S', T'$  with  $R' \subseteq Q' \subseteq \bar{Q}$  and  $T' \subseteq S' \subseteq \bar{S}$ ,  $(Q \cup Q', R \cup R')$  is uniformly Turing reducible to  $(S \cup S', T \cup T')$ .

*Proof.* First observe that it suffices to show that for any promise problem  $(X, Y)$ ,  $(X, Y)$  is uniformly Turing equivalent to  $(X \cup X', Y \cup Y')$ , for any finite sets  $X'$  and  $Y'$  with  $Y' \subseteq X' \subseteq \bar{X}$ . To see that  $(X, Y)$  is uniformly Turing reducible to  $(X \cup X', Y \cup Y')$  is in fact trivial, since every solution of  $(X \cup X', Y \cup Y')$  is also a solution of  $(X, Y)$ . Hence, we only have to show that  $(X \cup X', Y \cup Y')$  is uniformly Turing reducible to  $(X, Y)$ . To this end we must construct a polynomial time-bounded deterministic oracle Turing machine  $M$  such that for every solution  $S$  of  $(X, Y)$ , the set  $L(M, S)$  (i.e., the language recognized by  $M$  with oracle  $S$ ) is a solution of  $(X \cup X', Y \cup Y')$ . The following is an algorithm describing such a machine.

```

begin
input  $x$ ;
if  $x \in S \cup Y'$  and  $x \notin X' - Y'$ 
then accept else reject
end.
```

Since  $X'$  and  $Y'$  are finite sets this algorithm can be executed in polynomial time with respect to the oracle  $S$ . For any solution  $S$  of  $(X, Y)$ ,  $L(M, S) = (S \cup Y') - (X' - Y')$  is a solution of  $(X \cup X', Y \cup Y')$ .  $\square$

THEOREM 1. If  $(Q, R)$  is Turing reducible to  $(S, T)$ , then  $(Q, R)$  is uniformly Turing reducible to  $(S, T)$ .

*Proof.* Assume that  $(Q, R)$  is not uniformly Turing reducible to  $(S, T)$ . That is, assume that for every polynomial time-bounded deterministic oracle Turing machine  $M$  there is some solution  $A$  of  $(S, T)$  such that  $L(M, A)$  does not solve  $(Q, R)$ . Let us expand this assumption one more time: For every polynomial time-bounded deter-

ministic oracle Turing machine  $M$  there exists a solution  $A$  of  $(S, T)$  and there exists a string  $y \in \Sigma^*$  such that

$$y \in R \cap \overline{L(M, A)} \quad \text{or} \quad y \in (Q - R) \cap L(M, A).$$

We want to show with this assumption that  $(Q, R)$  is not Turing reducible to  $(S, T)$ . To do so we will construct one recursive solution  $A$  of  $(S, T)$  such that for every polynomial time-bounded deterministic oracle Turing machine  $M$ ,  $L(M, A)$  is not a solution of  $(Q, R)$ . Let  $\{M_i\}_{i \geq 1}$  be an effective enumeration of polynomial time-bounded deterministic oracle Turing machines with associated polynomial time bounds  $\{p_i\}_{i \geq 1}$ .

We want to show with this assumption that  $(Q, R)$  is not Turing reducible to  $(S, T)$ . To do so we will construct one recursive solution  $A$  of  $(S, T)$  such that for every polynomial time-bounded deterministic oracle Turing machine  $M$ ,  $L(M, A)$  is not a solution of  $(Q, R)$ . Let  $\{M_i\}_{i \geq 1}$  be an effective enumeration of polynomial time-bounded deterministic oracle Turing machines with associated polynomial time bounds  $\{p_i\}_{i \geq 1}$ .

The solution  $A$  of  $(S, T)$  will be constructed inductively to be of the form  $T \cup \bigcup\{Y_i \mid i \geq 1\}$ , where  $\bigcup\{Y_i \mid i \geq 1\}$  is a recursive subset of  $\bar{S}$ . At stage  $i$  of the construction,  $i \geq 1$ , finite subset  $Y_i$  of  $\bar{S}$  will be chosen so that  $L(M_i, T \cup Y_i)$  is not a solution of  $(Q, R)$ .

*Stage 0.* Define  $Y_0 = \emptyset$  and  $n_0 = 0$ .

*Stage  $i$  ( $i \geq 1$ ).* By induction hypothesis,  $Y_{i-1}$  is defined,  $Y_{i-1}$  is a finite set,  $n_{i-1} \geq 0$  is defined, and  $Y_{i-1} \subseteq \bar{S} \cap \Sigma^{\leq n_{i-1}}$ .

Now comes a sequence of claims.

*Claim 1.* There is a recursive set  $X_i \subseteq \overline{(S \cup \Sigma^{\leq n_{i-1}})}$  and a string  $y_i \in \Sigma^*$  such that

$$(1) \quad y_i \in R \cup \overline{L(M_i, T \cup Y_{i-1} \cup X_i)} \quad \text{or} \quad y_i \in (Q - R) \cap L(M_i, T \cup Y_{i-1} \cup X_i).$$

If the claim is correct, then  $y_i$  is a witness to the fact that  $L(M_i, T \cup Y_{i-1} \cup X_i)$  is not a solution of  $(Q, R)$ . Suppose the claim is false. Then, for every recursive subset  $X_i$  of  $\overline{(S \cup \Sigma^{\leq n_{i-1}})}$ ,  $R \cap \overline{L(M_i, T \cup Y_{i-1} \cup X_i)} = \emptyset$  and  $(Q - R) \cap L(M_i, T \cup Y_{i-1} \cup X_i) = \emptyset$ . Thus,  $R \subseteq L(M_i, T \cup Y_{i-1} \cup X_i) \subseteq \overline{(Q - R)}$  and so  $L(M_i, T \cup Y_{i-1} \cup X_i)$  is a recursive solution of  $(Q, R)$ . The set of all languages  $T \cup Y_{i-1} \cup X_i$ , where  $X_i$  is a recursive subset of  $\overline{(S \cup \Sigma^{\leq n_{i-1}})}$  is exactly the set of all solutions of the promise problem  $(S \cup \Sigma^{\leq n_{i-1}}, T \cup Y_{i-1})$ . Thus,  $(Q, R)$  is uniformly Turing reducible to this promise problem. But this promise problem is a finite variation of  $(S, T)$ , and so, by Lemma 2,  $(Q, R)$  must be uniformly Turing reducible to  $(S, T)$ . This is a contradiction to our assumption. Hence, the claim is true.

*Claim 2.* There is a finite set  $X_i \subseteq \overline{(S \cup \Sigma^{\leq n_{i-1}})}$  and a string  $y_i \in \Sigma^*$  that satisfy condition (1).

For the set  $X_i$  and a string  $y_i$  whose existence is guaranteed by Claim 1, machine  $M_i$  on input  $y_i$  may query strings of length at most  $p_i(|y_i|)$ . So, for  $X'_i = X_i \cap \Sigma^{\leq p_i(|y_i|)}$  and  $y_i$ , assertion (1) is still true.

*Claim 3.* A finite set  $X_i \subseteq \overline{(S \cup \Sigma^{\leq n_{i-1}})}$  and string  $y_i$  that satisfy assertion (1) can be found effectively.

This is trivial. Effectively enumerate all pairs of finite sets and strings until a pair with the desired properties is found.

At Stage  $i$ , apply Claim 3; define  $Y_i = Y_{i-1} \cup X_i$  and define  $n_i = \max\{2^{n_{i-1}}, p_i(|y_i|)\} + 1$ .

End of Stage  $i$ .



Define  $A = T \cup \bigcup \{Y_i \mid i \geq 1\}$ . Since  $\bigcup \{Y_i \mid i \geq 1\} \subseteq \bar{S}$ ,  $A$  is a solution of  $(S, T)$ .  $A$  is a recursive set because a string  $w$  belongs to  $A$  if and only if  $w \in R$  or  $w \in Y_i$ , where  $i$  is the least index of an  $n_i$  such that  $n_i \geq |w|$ . For every  $i$ ,  $L(M_i, A)$  does not solve  $(Q, R)$  because a string  $y_i$  exists such that  $y_i \in R \cap \overline{L(M_i, T \cup Y_i)}$  or  $y_i \in (Q - R) \cap L(M_i, T \cup Y_i)$ , and therefore  $y_i \in R \cap \overline{L(M_i, A)}$  or  $y_i \in (Q - R) \cap L(M_i, A)$ .

This holds because  $(T \cup Y_i) \cap \Sigma^{\leq p_i(|y_i|)} = A \cap \Sigma^{\leq p_i(|y_i|)}$ . Furthermore, strings added to the solution in future stages cannot disturb this relation because only strings are added that are too long to be queried on input  $y_i$ . Thus,  $A$  is a recursive solution of  $(S, T)$  such that  $L(M, A)$  is not a solution of  $(Q, R)$  for any deterministic polynomial time-bounded oracle Turing machine  $M$ .

Therefore,  $(Q, R)$  is not Turing reducible to  $(S, T)$ .  $\square$

**COROLLARY 1.** *A promise problem is NP-hard if and only if it is uniformly NP-hard.*<sup>1</sup>

*Proof.*  $(Q, R)$  is NP-hard if and only if  $(\Sigma^*, \text{SAT}) \leq_T^{\text{PP}} (Q, R)$ , and  $(Q, R)$  is uniformly NP-hard if and only if  $(\Sigma^*, \text{SAT}) \leq_{\text{UT}}^{\text{PP}} (Q, R)$ .  $\square$

Corollary 1 suggests a fascinating approach to proving the conjecture. Corollary 1 states that if  $(Q, R)$  is NP-hard, then there is *one* reduction  $M$  from SAT to every solution of  $(Q, R)$ . Thus, the result of  $M$ 's computation on an input string  $x$  does not depend on the answer of the oracle to queries outside of  $Q$ . If an oracle responds correctly to queries in  $Q$  and arbitrarily to queries outside of  $Q$ , then  $M$  accepts an input string  $x$  if and only if  $x \in \text{SAT}$ . Queries to strings in  $\bar{Q}$  are "meaningless."

Given a promise problem  $(Q, R)$  such that  $R \subseteq Q$  and a language  $L$ , say there is a *smart* reduction from  $L$  to  $(Q, R)$  if there is a polynomial time-bounded oracle Turing machine  $M$  that witnesses  $(\Sigma^*, L) \leq_{\text{UT}}^{\text{PP}} (Q, R)$  such that  $M$  never queries any word in  $\bar{Q}$ . (Smart reductions never ask meaningless questions.) The next theorem states that knowledge about the existence of smart reductions might help to solve the conjecture. First observe that if  $\text{NP} = \text{co-NP}$ , then the promise problem  $(Q, R)$ , where  $Q = \Sigma^*$  and  $R = \text{SAT}$  satisfies each of the following properties.

- (i)  $Q \in \text{NP}$ ,
- (ii)  $(Q, R) \in \text{NPP} \cap \text{co-NPP}$ , and
- (iii) there is a smart reduction from SAT to  $(Q, R)$ .

**THEOREM 2.** *If there is a promise problem  $(Q, R)$  in  $\text{NPP} \cap \text{co-NPP}$  and there is a smart reduction from SAT to  $(Q, R)$ , then  $\text{NP} = \text{co-NP}$ .*

If there exists  $(Q, R)$  such that  $Q \in \text{NP}$ ,  $(Q, R) \in \text{NPP} \cap \text{co-NPP}$ , and  $(Q, R)$  is NP-hard, then, by Corollary 1,  $(Q, R)$  is uniformly NP-hard. If these three conditions can be shown to imply existence of a smart reduction from SAT to  $(Q, R)$ , then, by Theorem 2,  $\text{NP} = \text{co-NP}$ , and so  $\text{NP} \neq \text{co-NP}$  would imply that the conjecture is true. However, the situation is probably more subtle than this, for in [ESY84] it is shown that  $\text{UP} = \text{NP}$  implies existence of a promise problem that satisfies the conditions listed in the conjecture, and intuition does not suggest that  $\text{NP} \neq \text{co-NP}$  implies  $\text{UP} \neq \text{NP}$ .

*Proof.* Let  $(Q, R)$  be a promise problem in  $\text{NPP} \cap \text{co-NPP}$  for which there is a smart reduction from SAT to  $(Q, R)$ . We will show that  $\overline{\text{SAT}} \in \text{NP}$  and conclude therefore that  $\text{NP} = \text{co-NP}$ . Let  $M$  be a deterministic oracle Turing machine that uniformly reduces SAT to solutions of  $(Q, R)$  and that never queries any word in  $\bar{Q}$ . Then,  $M$  with its accepting and rejecting states reversed (call this machine  $M'$ ) reduces  $\overline{\text{SAT}}$  to solutions of  $(Q, R)$  and never queries any word in  $\bar{Q}$ . By Lemma 1, assume without loss of generality that  $R \subseteq Q$ . Let  $M_i$ ,  $i = 1, 2$ , be NP-acceptors that solve

<sup>1</sup> A noneffective version of this result has been obtained independently by Regan [Reg86].

$(Q, R)$  and  $(Q, \bar{R})$ , respectively. We now describe an NP-acceptor  $N$  for  $\overline{\text{SAT}}$ . On input  $x$ ,  $N$  begins a simulation of  $M'$  on  $x$  but replaces each query  $w$  to the oracle by simulations of  $w$  on the NP-acceptors  $M_i$ ,  $i=1, 2$ , according to the following nondeterministic algorithm.

- if  $M_1$  accepts  $w$ 
  - continue simulation of  $M'$  in the YES state
- $M_2$  accepts  $w$ 
  - continue simulation of  $M'$  in the NO state

If the algorithm is executed on a word  $w$  and the guard that is chosen does not evaluate to *true*, then the simulation by  $N$  of  $M'$  on input  $x$  is to terminate without accepting. Since  $M_1$  has an accepting computation on a query string  $w$ , if  $w \in R$ ,  $M_2$  has an accepting computation on  $w$ , if  $w \in Q - R$ , and every word queried belongs to either  $R$  or  $Q - R$ , it follows, for every input word  $x$  to  $N$ , that the simulation by  $N$  of  $M'$  on  $x$  can be completed.

It is obvious that the language accepted by  $N$  belongs to NP. To see that this language is  $\overline{\text{SAT}}$ , simply observe for each input word  $x$  to  $N$  that  $N$  simulates  $M'$ , relative to some solution of  $(Q, R)$ . The latter fact follows readily because for every solution  $L$  of  $(Q, R)$ ,  $R \subseteq L$  and  $Q - R \subseteq \bar{L}$ . Thus,  $\overline{\text{SAT}} \in \text{NP}$  and so  $\text{NP} = \text{co-NP}$ . □

The next corollary follows from Proposition 4 in § 2.3.

**COROLLARY 2.** *If there is a smart reduction from SAT to (EX, SAT-1), then  $\text{NP} = \text{co-NP}$ .*

Thus, smart reductions cannot be expected to exist for all NP-hard promise problems in  $\text{NPP} \cap \text{co-NPP}$ .

**3.2. P-separability.** Recall that a disjoint pair of sets  $A$  and  $B$  are P-separable if there exists a set  $L$  in P such that  $A \subseteq L$  and  $B \subseteq \bar{L}$ ; they are P-inseparable otherwise. The following theorem gives a characterization of the conjecture that is strictly in terms of the fine structure of NP.

**THEOREM 3.** *There exists a promise problem  $(Q, R)$  satisfying the conditions of the conjecture if and only if there exist disjoint  $\cong_P^P$ -complete sets  $A$  and  $B$  in NP such that for every recursive set  $L$ , if  $A \subseteq L$  and  $B \subseteq \bar{L}$ , then  $L$  is NP-hard.*

*Proof.* Given a promise problem  $(Q, R)$  that satisfies the conditions of the conjecture, the sets  $A = Q \cap R$  and  $B = Q - R$  are  $\cong_P^P$ -complete sets in NP. The sets that separate  $A$  and  $B$  are precisely the solutions of  $(Q, R)$  so every set that separates  $A$  and  $B$  is NP-hard. Conversely, given such  $A$  and  $B$ ,  $(A \cup B, A)$  satisfies the conditions of the conjecture. □

We will be showing that P-inseparable sets in NP probably do exist. But, the property given in Theorem 3 is a much stronger constraint than P-inseparability. It is tempting to compare P-inseparability and “separability by NP-hard sets only” with the analogous notions from classical recursive function theory (recursive sets and recursively enumerable sets for P and NP, respectively). It is a fundamental result that there exist disjoint recursively enumerable sets that are recursively inseparable. On the other hand, Shoenfield [Sho60] has shown that every disjoint pair of recursively enumerable sets can be separated by some set whose degree is strictly less than  $\mathbf{0}'$ . Thus, the analogue of our conjecture for the recursively enumerable sets is a theorem.

The following corollaries are reformulations of one of the principal results (Proposition 4) proved in [ESY84]. Let  $\text{SAT-UNSAT} = \{(x, y) | x \text{ encodes a satisfiable formula of propositional logic while } y \text{ encodes an unsatisfiable formula}\}$  and let  $\text{UNSAT-SAT} = \{(x, y) | x \text{ is unsatisfiable and } y \text{ is satisfiable}\}$ . It is shown in [PY84] that  $\text{SAT-UNSAT}$

is complete for  $D^P$ , and trivially UNSAT-SAT is complete for  $D^P$  also. Now,  $SAT-UNSAT = EX \cap SAT-1$  and  $UNSAT-SAT = EX - SAT-1$ . Thus, from Theorem 3 applied to Proposition 4 we immediately obtain the following.

**COROLLARY 3.**  $D^P$  contains disjoint complete sets SAT-UNSAT and UNSAT-SAT such that every set that separates them is NP-hard.

Here is a property about  $D^P$  that very likely does not hold for NP and so Corollary 3 provides structural evidence that  $D^P$  is not equal to NP. Corollary 3 yields the following result also.

**COROLLARY 4.**  $P = NP$  if and only if SAT-UNSAT and UNSAT-SAT are P-separable.

The following corollary to Theorem 3 presents a provocative connection to the structure of the class of all NP-hard sets in NP.

**COROLLARY 5.** Statement (a) implies statement (b).

(a) For all sets  $A$  and  $B$ , if  $A$  and  $B$  are disjoint  $\cong_T^P$ -complete sets in NP, then  $A \cup B$  is  $\cong_T^P$ -complete in NP.

(b) If there exists a promise problem  $(Q, R)$  that satisfies the conditions of the conjecture, then  $Q$  is a  $\cong_T^P$ -complete set in NP.

The proof follows from the observation that  $Q \cap R$  and  $Q - R$  are both  $\cong_T^P$ -complete sets. It is an interesting open question whether statement (a) is true, but statement (b) would be a surprising consequence. The cracking problem for a PKCS is formulated as a promise problem specifically (i) so that its complexity is determined by the effort required to crack cryptograms correctly, and (ii) because we do not want the complexity to depend on the effort required to determine whether an arbitrary string is a legitimate cryptogram or a sham. Nevertheless, if statement (b) is true, then a given PKCS does have an NP-hard cracking problem only if its promise predicate is NP-hard.

Now we turn our attention to the existence of PKCS with intractable cracking problems, for surely a PKCS cannot be deemed secure if it can be cracked in polynomial time. For such a PKCS,  $CP = (Q_{Crack}, R_{Crack})$  and in addition to the conditions (i)  $Q_{Crack} \in NP$ , and (ii)  $CP \in NPP \cap co-NPP$ , the following property holds also: (iii)' No solution of CP is in P.

**THEOREM 4.** There is a promise problem  $(Q, R)$  satisfying conditions (i), (ii), and (iii)' if and only if  $NP - P$  contains disjoint P-inseparable sets.

Thus, PKCS with intractable cracking problems exist only if there exist P-inseparable sets. The proof is a straightforward variation of the proof of Theorem 3.

Do there exist P-inseparable sets in NP? There probably do. First of all because it is essentially trivial that if  $A \in (NP \cap co-NP) - P$ , then  $A$  and  $\bar{A}$  are P-inseparable and, secondly, because of the following theorem.

**THEOREM 5.** If  $P \neq UP$ , then NP contains P-inseparable sets.

*Proof.* Assume  $P \neq UP$  and by Proposition 1, let  $f \in NPSV_g - PSV$ . By Theorem 4, it suffices to show that  $(Q_f, R_f)$  has no solution in P. By Proposition 2, this is equivalent to showing that  $f$  has no extension in P. Since  $graph(f) \in P$  but  $f \notin PSV$ , it follows that the last assertion is true.  $\square$

**THEOREM 6.** If NP contains P-inseparable sets, then there exist NP-complete P-inseparable sets.

This is a very intriguing result. Theorems 5 and 6 together tell us that information about a subset of NP that probably does not contain the NP-complete sets, yet nevertheless influences the structure of the NP-complete sets.

*Proof.* Let  $\{M_i\}_{i \geq 1}$  be an enumeration of polynomial time-bounded nondeterministic Turing machines with associated polynomial time bounds  $\{p_i\}_{i \geq 1}$ . It is well known

that the set  $K = \{\langle i, x, 0^n \rangle \mid \text{some computation of } M_i \text{ accepts } x \text{ in at most } n \text{ steps}\}$  is NP-complete [BGS75]. For every set  $A$  in NP there exists  $i \geq 1$  such that  $A = L(M_i)$ , and there exists an honest many-one reduction  $f$  from  $A$  to  $K$  defined by  $f(x) = \langle i, x, 0^{p_i(|x|)} \rangle$ . Let  $A$  and  $B$  be P-inseparable sets in NP and let  $f$  be an honest reduction from  $A$  to  $K$ . Our first goal is to show that the sets  $f(B)$  and  $K$  are P-inseparable sets in NP. Since  $B \in \text{NP}$  and  $f$  is honest,  $f(B) \in \text{NP}$ . Since  $f$  is a reduction from  $A$  to  $K$  and  $A \cap B = \emptyset$ ,  $f(A) \subseteq K$  and  $f(B) \subseteq \bar{K}$ , and so  $f(B)$  and  $K$  are disjoint sets. In order to prove that  $f(B)$  and  $K$  are P-inseparable, assume that  $K \subseteq L \subseteq \bar{f(B)}$  and  $L \in \text{P}$ . Then,  $A \subseteq f^{-1}(L) \subseteq \bar{B}$ . Also,  $f^{-1}(L) \in \text{P}$  because  $f^{-1}(L) \leq_m^{\text{P}} L$ . Thus, P-inseparability of  $A$  and  $B$  are contradicted. Hence,  $f(B)$  and  $K$  are P-inseparable.

Now we have obtained two disjoint P-inseparable sets in NP and one of them is NP-complete. To prove the theorem, apply the construction once again, this time with an honest reduction  $g$  from  $f(B)$  to  $K$ . Namely,  $g(f(B)) \subseteq K$  and  $g(K) \subseteq \bar{K}$ . Then,  $K$  and  $g(K)$  are disjoint NP-complete sets and the argument already given shows that they are P-inseparable.  $\square$

**3.3. Almost-everywhere complexity.** If a PKCS cannot be cracked in polynomial time, then for every polynomial time-bounded algorithm there exist infinitely many messages whose codes the algorithm cannot crack. This leaves open the possibility that there exist some algorithm and infinitely many messages whose codes this algorithm can crack in polynomial time. Indeed, this leaves open the possibility that there exists some algorithm that can crack codes for *most* messages in polynomial time. NP-hardness is not a good measure for the security of a cryptographic system, for it is at the same time both too strong and too weak. It is too strong because there is no need to require that every set in NP is reducible to every solution of CP. It is too weak because it too is an i.o. measure and therefore allows the possibility of cracking nearly all codes in polynomial time. (An NP-hard classical cryptosystem that exhibits this feature has been constructed by Even and Yacobi and is described in the survey article by Lempel [Lem79].) Rather, a PKCS should be difficult to crack on as many instances of its message space as possible. Therefore, it seems worthwhile to formulate a.e. complexity measures for promise problems.

Define a promise problem to be *immune* to a complexity class  $\mathcal{C}$  if every solution is immune to  $\mathcal{C}$  and to be *bi-immune* to  $\mathcal{C}$  if every solution is bi-immune to  $\mathcal{C}$ . Here we will just consider immunity and bi-immunity to P, but our remarks hold for other classes as well.

An adversary who has available a P-immune solution  $S$  to CP may still be able to efficiently compute Crack for many input instances by applying a reduction technique (as indicated in the proof of Proposition 2) that uses easy instances of  $\bar{S}$ . Thus, immunity to P is not a sufficiently strong measure for public-key cryptography.

Analogous to Theorems 3 and 4, it is easy to show that there exists a bi-immune to P promise problem  $(Q, R)$  such that  $Q \in \text{NP}$  and  $(Q, R) \in \text{NPP} \cap \text{co-NPP}$  if and only if there exist disjoint sets  $A$  and  $B$  in NP such that every set which separates them is bi-immune to P. Furthermore, the latter condition is equivalent to the existence of disjoint sets in NP which are both bi-immune to P. Thus, we state the following proposition.

**PROPOSITION 5.** *There exists a bi-immune to P promise problem  $(Q, R)$  such that  $Q \in \text{NP}$  and  $(Q, R) \in \text{NPP} \cap \text{co-NPP}$  if and only if NP contains a disjoint pair of sets that are both bi-immune to P.*

Unfortunately, our interest in bi-immune promise problems is diminished by the following proposition.

PROPOSITION 6. For  $f \in \text{NPSV}$ ,  $(Q_f, R_f)$  is not bi-immune to  $P$ . Every solution of  $(Q_f, R_f)$  contains a nonsparse subset in  $P$ .

The proof is easy. Given  $x \in \text{dom}(f)$ , every solution of  $(Q_f, \bar{R}_f)$  includes  $\{(x, y) \mid y > f(x)\}$ . But this set is only useful for computing one value of  $f$ . The moral is this: The existence of solutions of CP with large easy subsets or with large easy subsets of the complement does not give information about the density of the domain of easy instances of Crack. Promise problems do not permit elegant ways to characterize almost-everywhere security measures which impart information about the security of the underlying functions. Primarily for this reason, the complexity of Crack is analyzed in the next two sections directly as a computational problem.

**4. One-way functions.** Recall that a function  $f$  is honest if there is a polynomial  $q$  such that for every  $y \in \text{range}(f)$  there exists  $x \in \text{dom}(f)$  such that  $f(x) = y$  and  $|x| \leq q(|y|)$ . We define a partial function  $f$  to be *one-way* if  $f$  is one-one,  $f$  is honest,  $f \in \text{PSV}$ , and  $f^{-1} \notin \text{PSV}$ . This is a common formulation that has been useful in several complexity theoretic investigations [BL86], [Ko85], [KLD86], [Wat85], [You83].

If  $f$  is a one-way function, then  $f^{-1} \in \text{UPSV}_g - \text{PSV}$ . In fact, no extension of  $f^{-1}$  is in  $\text{PSV}$ . Also,  $f^{-1} \in \text{UPSV}_g - \text{PSV}$  implies  $\text{UP} \neq P$ , by Proposition 1. The converse holds as well.

THEOREM 7. The following are equivalent.<sup>2</sup>

- (a)  $P \neq \text{UP}$ .
- (b) There exists a one-way function.
- (c) There exists a total one-way function.

*Proof.* Clearly (c) implies (b), and we just observed that (b) implies (a). Let  $L \in \text{UP} - P$  be witnessed by a UTM  $M$ . The total function  $f$  defined by

$$f(x) = \begin{cases} y0 & \text{if } x = \text{comp}_M(y), \\ x1 & \text{otherwise} \end{cases}$$

is a total one-way function.  $\square$

In some sense, for the function  $f$  just constructed,  $f^{-1}$  is hard to compute because one cannot even determine whether a string  $x$  belongs to  $\text{dom}(f^{-1})$ . Thus, it is interesting to know whether there exist one-way functions with range belonging to  $P$ .

THEOREM 8. The following are equivalent.

- (a)  $P \neq \text{UP} \cap \text{co-UP}$ .
- (b) There is a one-way function  $f$  such that  $\text{range}(f) \in P$ .
- (c) There is a one-way function  $f$  such that  $\text{range}(f) = \Sigma^*$ .
- (d) There is a total one-way function  $f$  such that  $\text{range}(f) \in P$ .

*Proof.* The equivalences can be proved from the following three implications.

(b) implies (a): Let  $f$  be a one-way function such that  $\text{range}(f) \in P$ , and let  $(Q, R)$  be the promise problem associated with  $f^{-1}$ . Then,  $Q \in P$ , because  $Q = \text{dom}(f^{-1}) \times \Sigma^* = \text{range}(f) \times \Sigma^*$ . Since no extension of  $f^{-1}$  belongs to  $\text{PSV}$ ,  $(Q, R)$  has no solution in  $P$ , by Proposition 2. Therefore, in particular,  $Q \cap R \notin P$ . On the other hand, it is easy to see that  $Q \cap R \in \text{UP}$ . Similarly,  $Q - R \in \text{UP} - P$ . Assertion (a) follows directly.

(a) implies (c): Let  $L \in (\text{UP} \cap \text{co-UP}) - P$ . Let  $M$  and  $N$  be UTMs that witness  $L \in \text{UP}$  and  $\bar{L} \in \text{UP}$ , respectively. The function  $f$  defined as follows satisfies condition (c):

<sup>2</sup> This result has been obtained independently by Ko [Ko85].

$$f(x) = \begin{cases} y & \text{if } x = \text{comp}_M(y) \text{ or } x = \text{comp}_N(y), \\ \uparrow & \text{otherwise.} \end{cases}$$

(a) implies (d): Let  $L$ ,  $N$ , and  $M$  be as in the previous implication. This time define  $f$  as follows:

$$f(x) = \begin{cases} y0 & \text{if } x = \text{comp}_M(y) \text{ or } x = \text{comp}_N(y), \\ x1 & \text{otherwise.} \end{cases} \quad \square$$

THEOREM 9. *The following are equivalent.*

(a)  $\text{UP} = \text{NP}$ .

(b) *There exists a one-one, honest function  $f \in \text{PSV}$  such that  $\text{dom}(f^{-1})$  is NP-complete.*

(c) *There exists a function  $f \in \text{UPSV}_g$  such that  $\text{dom}(f)$  is NP-complete.*

*Proof.* To see that (a) implies (b), let  $L$  be an NP-complete set. Since  $L \in \text{UP}$  is assumed, it follows that  $L = \text{dom}(f^{-1})$ , where  $f$  is defined by

$$f(x) = \begin{cases} y & \text{if } x = \text{comp}_M(y), \\ \uparrow & \text{otherwise.} \end{cases}$$

We observed previously that (b) implies (c).

Let  $f \in \text{UPSV}_g$  such that  $\text{dom}(f)$  is NP-complete. To see that  $\text{UP} = \text{NP}$ , observe first that  $\text{dom}(f) \in \text{UP}$  and secondly that

$$(A \leq_m^P B \text{ and } B \in \text{UP}) \rightarrow A \in \text{UP}.$$

Thus (c) implies (a).  $\square$

Next we will observe that one of the main results of [ESY84] follows quickly when Theorem 9 is applied to what we learned in § 2 about the promise problem  $(R_f, Q_f)$  that is associated with a function  $f$ .

COROLLARY 6 [ESY84]. *The conjecture implies  $\text{UP} \neq \text{NP}$ .*

*Proof.* Suppose  $\text{UP} = \text{NP}$  and let  $f \in \text{UPSV}_g$  such that  $\text{dom}(f)$  is NP-complete. We know already that  $Q_f \in \text{NP}$  and  $(Q_f, R_f) \in \text{NPP} \cap \text{co-NPP}$ .

To show that  $(Q_f, R_f)$  is NP-hard, by Proposition 2 it suffices to show that  $\text{dom}(f) \leq_T^P h$ , for every extension  $h$  of  $f$ . The reduction is given by the following algorithm:

**input**  $x$ ;  
**if**  $(x, h(x)) \in \text{graph}(f)$  **then** accept **else** reject.  $\square$

COROLLARY 7. *If  $\text{P} \neq \text{NP}$ , then  $\text{NP} = \text{UP}$  if and only if there is a one-way function  $f$  such that  $\text{dom}(f^{-1})$  is NP-complete.*

Now we concern ourselves with the following important question. Do one-way functions play a role in public-key cryptography? The answer is not at all obvious. The function  $\lambda ME(K_1, M)$  is not a one-way function because the inverse  $\lambda CD(K_2, C)$  is easy to compute, assuming  $(K_1, K_2)$  is a legal pair. The encoding function  $E$  is not a one-way function because it is not one-one. Consider the function  $E'$  defined as

follows:

$$E'(n, K_1, M) = \begin{cases} (n, K_1, E(K_1, M)) & \text{if } K_1 \text{ is a legal encryption key for messages} \\ & \text{of length } n, \text{ and } |M| = n, \\ \uparrow & \text{otherwise.} \end{cases}$$

$E'$  is one-one, is honest (by an assumption made in § 2.4) and has an extension in PSV.  $E'$  is a candidate therefore for being a one-way function, but  $E'$  is not necessarily in PSV, because  $\text{dom}(E')$  is not necessarily in P.

The inverse of  $E'$  is the function Crack' defined as follows:

$$\text{Crack}'(n, K_1, c) = \begin{cases} (n, K_1, \text{Crack}(n, K_1, C)) & \text{if } (n, K_1, C) \in \text{dom}(\text{Crack}), \\ \uparrow & \text{otherwise.} \end{cases}$$

$\text{Dom}(\text{Crack}') = \text{dom}(\text{Crack})$ , and it should be obvious that Crack' is computable in polynomial time if and only if Crack is computable in polynomial time. Therefore, if for a given PKCS,  $\text{Crack} \notin \text{PSV}$ , then  $E'$  will be a one-way function if and only if  $\text{dom}(E') \in \text{P}$ . In fact, the latter condition holds if and only if  $\text{Crack} \in \text{UPSV}_g$ . We conclude: If  $\text{Crack} \in \text{UPSV}_g - \text{PSV}$ , then  $E'$  is a one-way function and, furthermore, for this to occur it is necessary that  $\text{P} \neq \text{UP}$ . Indeed, even if  $\text{Crack} \in \text{UPSV} - \text{PSV}$ , it must be the case that  $\text{P} \neq \text{UP}$ .

The following propositions give some equivalent conditions for Crack to belong to one of the classes UPSV or  $\text{UPSV}_g$ .

**PROPOSITION 7.** *The following are equivalent.*

- (a)  $\text{Crack} \in \text{UPSV}$ .
- (b)  $\text{dom}(\text{Crack}) \in \text{UP}$ .
- (c)  $\text{dom}(E') \in \text{UP}$ .
- (d) *The set of legal encryption keys is in UP.*

**PROPOSITION 8.** *The following are equivalent.*

- (a)  $\text{Crack} \in \text{UPSV}_g$ .
- (b)  $\text{dom}(E') \in \text{P}$ .
- (c)  $E' \in \text{PSV}$ .
- (d) *The set of legal encryption keys is in P.*

Reasonable assumptions can be made about a PKCS which guarantee that Crack  $\in$  UPSV. One constraint to insure this is that the first component of the key generator be one-one.

**4.1. Weak one-way functions.** If the conditions listed in Proposition 8 do not hold, then we cannot conclude that  $E'$  is a one-way function. A PKCS may be intractable without the conditions listed in Proposition 7 being true. Therefore, existence of a PKCS that cannot be cracked in polynomial time does not necessarily imply  $\text{P} \neq \text{UP}$ , and so does not necessarily imply existence of one-way functions. Public-key cryptography without one-way functions is possible.

But let us focus attention on the function  $E'$  and its inverse Crack' once more. We have seen already that  $E'$  is one-one, is honest, and has an extension in PSV. Clearly,  $E' \in \text{NPSV}$ . Since a PKCS can be cracked in polynomial time if and only if some extension of Crack belongs to PSV, and since Crack has an extension in PSV if and only if Crack' has an extension in PSV, a necessary and sufficient condition that a PKCS is intractable is that Crack' cannot be extended to any function in PSV.

With these properties of  $E'$  and Crack' in mind we make the following definition. A function  $f \in \text{NPSV}$  is *weak one-way* if  $f$  is one-one,  $f$  is honest, an extension of  $f$  is in PSV, and no extension of  $f^{-1}$  is in PSV. Every one-way function is a weak one-way

function. It follows from the discussion that existence of intractable public-key cryptosystems implies the existence of weak one-way functions. In light of the fact that existence of intractable public-key cryptosystems implies the existence of P-inseparable sets in NP, the next theorem is particularly interesting.

**THEOREM 10.** *The following are equivalent.*

- (a) NP contains disjoint P-inseparable sets.
- (b) There exist functions in NPSV that have no extension in PSV.
- (c) There exist weak one-way functions.

*Proof.* To prove that (a) implies (b), let  $A$  and  $B$  be disjoint P-inseparable sets in NP. Define  $f$  as follows:

$$f(x) = \begin{cases} 1, & x \in A, \\ 0, & x \in B. \end{cases}$$

Clearly  $f \in \text{NPSV}$ . If  $f_1$  is any extension of  $f$ , then  $\{x \mid f_1(x) = 1\}$  separates  $A$  and  $B$ . Therefore, this set does not belong to P and so  $f_1$  does not belong to PSV.

Now we prove that (b) implies (c). Let  $f \in \text{NPSV}$  so that no extension of  $f$  belongs to PSV. Define  $g$  to be the natural mapping from  $\text{dom}(f)$  to  $\text{graph}(f)$ . That is,

$$g(x) = \begin{cases} \langle x, f(x) \rangle, & x \in \text{dom}(f), \\ \uparrow & \text{otherwise.} \end{cases}$$

Finally, define  $h = g^{-1}$ . Then,

$$h(\langle x, y \rangle) = \begin{cases} x, & x \in \text{dom}(f) \text{ and } y = f(x), \\ \uparrow & \text{otherwise.} \end{cases}$$

We claim that  $h$  is a weak one-way function. Clearly  $h$  is one-one, is honest, and has an extension in PSV. If  $h^{-1} = g$  has an extension in PSV, then it follows that  $f$  has an extension in PSV. Thus, no extension of  $h^{-1}$  belongs to PSV, and so  $h$  is a weak one-way function.

To prove that (c) implies (a), let  $f$  be a weak one-way function and observe that  $f^{-1} \in \text{NPSV}$  and  $f^{-1}$  has no extension in PSV. Therefore, the promise problem  $(Q_{f^{-1}}, R_{f^{-1}})$ , by Proposition 2, has no solution in P. So, by Theorem 4, NP contains disjoint P-inseparable sets.  $\square$

**4.2. Summary.** It is useful to summarize the results of this and the previous section in the following manner. Consider the following assertions of existence.

- (A) There exist PKCS satisfying reasonable conditions (e.g., the set of legal encryption keys is in UP) that cannot be cracked in polynomial time.
- (B) There exist one-way functions. Equivalently,  $\text{P} \neq \text{UP}$ .
- (C) There exist PKCS that cannot be cracked in polynomial time.
- (D) There exist weak one-way functions. Equivalently, NP contains disjoint P-inseparable sets.

We proved that (A) implies (B) and that (C) implies (D). Now we will show that (B) implies (C), thereby completing a sequence of implications. The next result also helps to explain why none of the converse implications seem to hold.

**THEOREM 11.** *If there exists a function  $f \in \text{NPSV}$  such that*

- (i)  $f$  is one-one,
- (ii)  $f$  is honest,
- (iii) there is a one-one extension of  $f$  in PSV,
- (iv) no extension of  $f^{-1}$  is in PSV,

*then there exists a PKCS that cannot be cracked in polynomial time.*



Observe that every one-way function satisfies the hypothesis of Theorem 11 and every function that satisfies these conditions is a weak one-way function. A function  $f$  may have an extension in PSV but not have any extension in PSV that is one-one, and so we can prove neither that every weak one-way function satisfies the hypothesis of Theorem 11 nor that existence of weak one-way functions imply existence of PKCS that cannot be cracked in polynomial time.

*Proof.* Let  $f$  satisfy the conditions listed and let  $f_1$  be an extension of  $f$  such that  $f_1$  is one-one and  $f_1 \in \text{PSV}$ . Since  $f \in \text{NPSV}$ ,  $\text{dom}(f) \in \text{NP}$  and so some total function  $g \in \text{PSV}$  exists such that  $\text{range}(g) = \text{dom}(f)$  [Sel78].

Define a key generator  $G$  as follows:  $G(x) = (f(g(x)), g(x))$ . Since  $f_1(g(x)) = f(g(x))$  for all  $x$ ,  $G$  is computable in polynomial time. If  $K_1 = f(g(x))$  is an enciphering key, then  $K_2 = g(x)$  is the deciphering key. (Observe that if  $(K_1, K_2)$  is a legal pair, then  $f(K_2) = K_1$  and hence “key-cracking” is equivalent to inverting  $f$ .)

As blocklength for an enciphering key  $K_1 = f(g(x))$ ,  $|K_2| = |g(x)|$  is published. Encoding and decoding functions are defined as follows, where  $|M| = |f_1^{-1}(K_1)|$ .

$$E(K_1, M) = \begin{cases} \#^{|K_1|} & \text{if } f_1(M) = K_1, \\ M & \text{otherwise,} \end{cases}$$

and

$$D(K_2, C) = \begin{cases} K_2 & \text{if } C = \#^{|K_1|}, \\ C & \text{otherwise.} \end{cases}$$

The functions  $E$  and  $D$  obviously are computable in polynomial time. Since  $f$  is an honest function, it is easy to see that the function Crack for this system is polynomial length-bounded.

To prove that the system inverse condition holds, let  $(K_1, K_2)$  be a legal pair of keys and let  $|M| = |f_1^{-1}(K_1)|$ . Consider the case that  $f_1(M) = K_1$ . Then,  $D(K_2, E(K_1, M)) = D(K_2, \#^{|K_1|}) = K_2$ . Since  $(K_1, K_2)$  is a legal pair,  $f(K_2) = K_1$ , and since  $f_1$  is one-one,  $K_2 = M$ , and so  $D(K_2, E(K_1, M)) = M$ . The other case is trivial. Thus, the system inverse condition holds and therefore  $G$ ,  $E$ , and  $D$  form a viable definition of a PKCS.

Now assume that there is an algorithm to solve Crack in polynomial time, i.e., some extension of Crack is in PSV. This algorithm can be used to compute an extension of  $f^{-1}$  in polynomial time as follows: if  $y \in \text{range}(f)$ , then  $(y, f^{-1}(y)) = (f(x), x)$ , for some  $x \in \text{range}(g)$ . Thus  $(y, f^{-1}(y))$  is a legal pair. Thus, Crack  $(|y|, y, \#^{|y|})$  is defined and has value  $f^{-1}(y)$ .  $\square$

Condition (iii) is somewhat stronger than needed, for it suffices to assume that an extension  $f'$  of  $f$  exists such that

$$\forall x \forall y [x \in \text{dom}(f) \wedge y \in \text{dom}(f') - \text{dom}(f) \rightarrow f'(x) \neq f'(y)]$$

and such that  $f' \in \text{PSV}$ .

Consider how this system might be used. Suppose  $K_1$  is the public-key and  $n = |f^{-1}(K_1)|$ . A sender who wants to send the message  $M$  with  $|M| = n$ , first checks whether  $f_1(M) = K_1$ . If so, then  $\#^{|K_1|}$  is the cyphertext transmitted and the receiver immediately knows that the cleartext is just  $f_1^{-1}(K_1) = K_2$ . If  $f_1(M) \neq K_1$ , and this is the case nearly always, then  $M$  itself is transmitted. Thus, this system is not a *usable* PKCS, rather, it illustrates by example what was said in § 3.3. Namely, intractability is a necessary component of security, but public-key cryptography requires a.e. security measures.

**4.3. Discrete logarithms.** We provide candidates for weak one-way functions, one-way functions, and problems in  $UP-P$ .

The following partial function  $f$  is examined in [Bra79].

$$\text{Dom}(f) = \{\langle p, g, c \rangle \mid p \text{ is a prime number,} \\ g \text{ is a primitive root mod } p, \text{ and } 1 \leq c \leq p-1\}$$

such that for  $\langle p, g, c \rangle \in \text{dom}(f)$ ,

$$f(\langle p, g, c \rangle) = \langle p, g, g^c(\text{mod } p) \rangle.$$

This partial function is one-one, is honest, and has an extension in PSV (perform the displayed computation for every  $p$ ,  $g$ , and  $c$ ).  $f$  is probably not in PSV because  $\text{dom}(f)$  is not in  $P$  unless both primality and primitive roots can be recognized in polynomial time.

The *discrete logarithm* problem is the problem of computing an extension of the following partial function  $h$ , where  $\text{dom}(h) = \text{dom}(f)$  and for  $\langle p, g, x \rangle \in \text{dom}(h)$ ,

$$h(\langle p, g, x \rangle) = \text{the unique integer } c, \quad 1 \leq c \leq p-1, \quad \text{such that } g^c = x(\text{mod } p).$$

Equivalently, the discrete logarithm problem is the following promise problem.

**input** integers  $p$ ,  $g$ , and  $b$

**promise**  $p$  is a prime number,  $g$  is a primitive root mod  $p$ , and  $1 \leq b \leq p-1$ .

**output** unique  $c$ ,  $1 \leq c \leq p-1$ , such that  $g^c = b(\text{mod } p)$ .

The partial function  $f^{-1}$  has no extension in PSV unless there is a polynomial time algorithm to solve the discrete logarithm problem. Therefore,  $f$  is a candidate for being a weak one-way function.

Feigenbaum [Fei85] has shown that  $f$  can be extended to a total function that is both one-one and computable in polynomial time only if there is a randomized polynomial time algorithm for testing whether an integer  $g$ ,  $1 \leq g \leq p-1$ , is a primitive root for a prime  $p$ . Therefore,  $f$  probably does not satisfy the conditions of Theorem 11.

Here is Feigenbaum's argument. Consider the following *primitive root* problem.

**input** integers  $p$  and  $g$

**promise**  $p$  is a prime number.

**property**  $g$  is a primitive root mod  $p$ .

Suppose that  $\bar{f}$  is an extension of  $f$  that is total, one-one, and computable in polynomial time. If  $g$  is *not* a primitive root, then the mapping  $c \mapsto g^c(\text{mod } p)$  is (at least two)-to-one, when we restrict both domain and range to the interval  $[1, p-1]$ . If, for every  $c$  in  $\{1, \dots, p-1\}$ ,  $\bar{f}(\langle p, g, c \rangle) = \langle p, g, y \rangle$  where  $y = g^c(\text{mod } p)$  and  $1 \leq y \leq p-1$ , then  $\bar{f}$  cannot be one-one. Thus, for a *fixed* prime, a *fixed* nonprimitive root  $g$ , and a *random*  $c \in [1, p-1]$ , the probability is at least  $1/2$  that if  $\bar{f}(\langle p, g, c \rangle) = \langle a, b, y \rangle$ , then one of these three conditions holds:

$$(1) \quad a \neq p \text{ or } b \neq g;$$

$$(2) \quad y > p-1;$$

$$(3) \quad y \neq g^c(\text{mod } p).$$

Thus, the discrete logarithm problem can be solved as follows: For  $k$  independent iterations, choose a random  $c \in [1, p-1]$  and compute  $\bar{f}(\langle p, g, c \rangle)$ . If  $g$  passes all  $k$  tests, then, with probability of error at most  $1/2^k$ ,  $g$  is a primitive root mod  $p$ . If one of the results satisfies one of conditions 1, 2, or 3, then  $g$  is not a primitive root mod  $p$ .

The primitive root problem is not believed to have a solution in  $R$ ; hence this is evidence against the existence of  $\bar{f}$ .

Now we will show that a modification of  $f$  is a candidate for being a one-way function. Observe that  $\text{dom}(f) \in NP$  by a result of Pratt [Pra75]. Hence, there exists

a polynomial  $q$  and a polynomial time recognizable relation  $R$  so that  $p$  is a prime number and  $g$  is a primitive root mod  $p \Leftrightarrow \exists y_{|y| \leq q(|p|)} R(p, g, y)$ .

Johnson [Joh85] suggests that the following partial function is a candidate for being a one-way function.

$$f_1(\langle p, g, y, c \rangle) = \begin{cases} \langle p, g, y, g^c \pmod{p} \rangle & \text{if } 1 \leq c \leq p-1, |y| \leq q(|p|), \text{ and } R(p, g, y), \\ \uparrow & \text{else.} \end{cases}$$

The partial function  $f_1$  is in PSV, and, of course,  $f_1$  is one-one and honest. It is believed that discrete logarithms remain difficult to compute even given a “short proof” that  $g$  is a primitive root mod  $p$  for a prime number  $p$ . Thus,  $f_1^{-1}$  is not believed to be computable in polynomial time.

Consider the promise problem  $(Q_{f_1^{-1}}, R_{f_1^{-1}})$  (except that the output is not padded). Then,  $R_{f_1^{-1}} = \{(p, g, y, b, x) \mid 1 \leq b \leq p-1, 1 \leq x \leq p-1, |y| \leq q(|p|), R(p, g, y), \text{ and, the unique } c \text{ such that } 1 \leq c \leq p-1 \text{ and } g^c = b \pmod{p} \text{ is less than } x\}$ . Hence,  $R_{f_1^{-1}} \in \text{UP}$ . Therefore, if  $f_1$  is a one-way function, then, since  $(Q_{f_1^{-1}}, R_{f_1^{-1}})$  has no solution in P, the set  $R_{f_1^{-1}} \in \text{UP-P}$ .

**5. Almost-everywhere complexity measures.** Given two partial functions  $f$  and  $g$ ,  $g$  is an *approximation* of  $f$  for all  $x \in \text{dom}(g) \cap \text{dom}(f)$ ,  $g(x) = f(x)$ . One might wish to design a PKCS such that every easily computable function  $g$  approximates Crack only for finitely many inputs, i.e.,  $\text{dom}(g) \cap \text{dom}(\text{Crack})$  is finite. This is an unobtainable goal. Namely, for any polynomial  $q$ , Crack can be computed in polynomial time for  $q(n)$  many input triples of the form  $(n, K_1, C)$  by the following trivial algorithm. Sequentially search all strings of length  $n$  until either a string  $M$  is found such that  $E(K_1, M) = C$  or  $q(n)$  many strings are searched. For every  $n$ , the algorithm solves the cracking problem in polynomial time for a polynomial number of input triples of the form  $(n, K_1, C)$ . Therefore, it is not possible to have  $\text{dom}(f) \cap \text{dom}(\text{Crack})$  finite for arbitrary approximations.

On the other hand, observe that the domain of the procedure just described is a sparse set. We want to consider security measures that reflect the limit of what can be achieved, and so we are led to the following definitions.

A partial function  $f$  is *partially immune* to a complexity class of functions  $\mathcal{F}$  if  $\text{dom}(f)$  is not sparse and for every approximation  $g$  of  $f$ ,  $g \in \mathcal{F}$  implies  $\text{dom}(g) \cap \text{dom}(f)$  is a sparse set. A set  $A$  is *partially immune* to a complexity class of sets  $\mathcal{C}$  if  $A$  is not sparse and every subset of  $A$  in  $\mathcal{C}$  is a sparse set.

If a partial function  $f$  is partially immune to PSV, then no extension of  $f$  is in PSV. It is not good enough to consider only approximations  $g$  which fulfill  $\text{dom}(g) \subseteq \text{dom}(\text{Crack})$ . If  $\text{dom}(\text{Crack})$  is immune to P, then for every approximation  $g \in \text{PSV}$  of Crack such that  $\text{dom}(g) \subseteq \text{dom}(\text{Crack})$ ,  $\text{dom}(g)$  is finite. Nevertheless, Crack might be extendable to a total function in PSV, and therefore not secure at all.

A partial function  $f$  may be partially immune to PSV without  $\text{dom}(f)$  being partially immune to P and vice versa. According to the following theorem the implication in one direction holds for PKCS such that  $\text{Crack} \in \text{UPSV}_g$  (cf. Proposition 8).

**THEOREM 12.** *If  $f \in \text{UPSV}_g$  and  $\text{dom}(f)$  is partially immune to P, then  $f$  is partially immune to PSV.*

*Proof.* Let  $g \in \text{PSV}$  be an approximation to  $f$ . The following algorithm shows that  $\text{dom}(f) \cap \text{dom}(g) \in \text{P}$ : on input  $x$ , **if**  $(x, g(x)) \in \text{graph}(f)$  **then** accept **else** reject. It follows that this set is sparse.  $\square$

Recall that a set  $A$  is bi-immune to P if and only if it is a.e. complex. The following proposition states that partial immunity to PSV enjoys the analogous property.

**PROPOSITION 9.** *The following are equivalent.*

- (a)  $f$  is partially immune to PSV.

(b) If  $M$  computes an extension of  $f$ , then  $M$  requires more than polynomial time on  $\text{dom}(f)$  except on a sparse set.

Now, suppose  $f$  is partially immune to PSV, and let  $g$  be any approximation of  $f$ . For most input words  $x$  in the domain of  $f$ ,  $g(x)$  is undefined, but when  $g(x)$  is defined, the output value is correct, i.e.,  $g(x) = f(x)$ . It may seem that  $f$  cannot have any good heuristics, but this is not in general true. Approximations as we have defined them satisfy a severe condition—they never give the wrong answer. In many applications, cryptography included, it may be more appropriate to consider polynomial time-bounded Turing machines that compute the correct answer for some inputs and that compute wrong answers for other inputs, and to concern oneself with the relative frequency of these two events. These considerations as applied to languages have led to a study of polynomial time random languages [Wil83], [Huy85]. Briefly, a language  $L$  is *polynomial time random* if every polynomial time algorithm is no more successful at recognizing  $L$  than is guessing. Huynh [Huy85] shows that bi-immunity and polynomial time randomness are orthogonal properties.

Now we consider public-key cryptosystems for which good heuristics to compute Crack do not exist. We formulate this for arbitrary  $f$  as follows:

(2) For every partial function  $g \in \text{PSV}$   $\{x \mid g(x) = f(x)\}$  is a sparse set.

Condition (2) is not in general equivalent to partial-immunity to PSV. But, surprisingly, as a consequence of the next theorem, these two conditions are equivalent for the cracking problem of public-key cryptosystems. Therefore, we define a PKCS to be *secure with respect to PSV* if Crack satisfies condition (2).

**THEOREM 13.** *If  $f$  is a partial function such that the promise problem  $(\text{dom}(f) \times \Sigma^*$ ,  $\text{graph}(f))$  has a solution in  $\text{P}$ , then each of the following is equivalent.*

- (a) *For every total function  $g$ ,  $g \in \text{PSV}$ ,  $\{x \mid g(x) = f(x)\}$  is a sparse set.*
- (b) *For every partial function  $g$ ,  $g \in \text{PSV}$ ,  $\{x \mid g(x) = f(x)\}$  is a sparse set.*
- (c)  *$f$  is partially immune to PSV.*

*Proof.* To show that (a) implies (b), assume (a) and let  $g \in \text{PSV}$ . Define  $g'(x) = g(x)$ , if  $x \in \text{dom}(g)$ , and  $g'(x) = x$ , if  $x \notin \text{dom}(g)$ .  $g'$  is defined for all  $x$  and, since  $\text{dom}(g) \in \text{P}$ ,  $g' \in \text{PSV}$ . Thus, by assumption,  $\{x \mid g'(x) = f(x)\}$  is sparse. Since  $\{x \mid g(x) = f(x)\} \subseteq \{x \mid g'(x) = f(x)\}$ ,  $\{x \mid g(x) = f(x)\}$  is sparse as well.

Now assume (b) and let us show that  $f$  is partially immune. Let  $g \in \text{PSV}$  such that  $\forall x(x \in \text{dom}(f) \cap \text{dom}(g) \rightarrow f(x) = g(x))$ . Then,  $\text{dom}(f) \cap \text{dom}(g) = \{x \mid g(x) = f(x)\}$  and so, by assumption,  $\text{dom}(f) \cap \text{dom}(g)$  is sparse.

Finally, we use the hypothesis that  $(\text{dom}(f) \times \Sigma^*$ ,  $\text{graph}(f))$  has a solution in  $\text{P}$  to show that with this assumption (c) implies (a). Let  $S \in \text{P}$  be a solution, assume (c), and let  $g \in \text{PSV}$  be a total function. Define  $g'(x) = g(x)$ , if  $(x, g(x)) \in S$ , and to be undefined otherwise. Clearly,  $g' \in \text{PSV}$  and we show now that for every word  $x$ ,  $x \in \text{dom}(f) \cap \text{dom}(g')$  implies  $f(x) = g'(x)$ . Given  $x \in \text{dom}(f) \cap \text{dom}(g')$ ,  $(x, g(x)) \in \text{dom}(f) \times \Sigma^*$  and so the promise is satisfied. Therefore,  $(x, g(x)) \in S \Leftrightarrow f(x) = g(x)$ . Since  $x \in \text{dom}(g')$ ,  $g'(x) = g(x)$ , and  $(x, g(x)) \in S$ , and therefore  $g'(x) = g(x) = f(x)$ .

It follows that  $\text{dom}(f) \cap \text{dom}(g')$  is sparse, for we are assuming that  $f$  is partially immune. Next we will observe that  $\{x \mid f(x) = g(x)\} \subseteq \text{dom}(f) \cap \text{dom}(g')$ , from which the claim in assertion (a) follows. If  $f(x) = g(x)$ , then  $x \in \text{dom}(f)$  and so  $(x, g(x)) \in S \Leftrightarrow f(x) = g(x)$ . Then, by definition  $x \in \text{dom}(g')$  and so  $x \in \text{dom}(f) \cap \text{dom}(g')$ , and the proof is complete.  $\square$

Observe from the proof of the theorem that condition (2) implies  $f$  is partially immune to PSV for all  $f$ . To complete the argument that condition (2) is equivalent to partial immunity for the cracking problem of public-key cryptosystems, recall that

the encoding algorithm  $E$  is computable in polynomial time and observe that the set  $S = \{(n, K_1, C, M) \mid |M| = n \text{ and } E(K_1, M) = C\}$  is a solution in  $P$  of  $(\text{Crack} \times \Sigma^*, \text{graph}(\text{Crack}))$ . Therefore, the following corollary is proved.

**COROLLARY 8.** *A PKCS is secure with respect to PSV if and only if Crack is partially immune to PSV.*

Next we will examine security with respect to functions that are computed probabilistically in polynomial time with bounded error. With that exception, we stress that the security measure we have been discussing represents a theoretical limit of what can be achieved and may even be stronger than what a usable system need attain. For example, Goldwasser and Micali [GM84] permit (when the message space has normal distribution) a super-polynomial number of messages of size  $n$  to be cracked.

**5.1. Cryptanalysis by probabilistic algorithms.** The informal view does not change. If a cryptanalyst has available a transducer  $T$  that on many inputs to Crack probabilistically computes Crack with bounded error probability, then the PKCS under attack is rendered useless, even if  $T$  behaves arbitrarily on other inputs, and even if it is not possible to decide when  $T$  is correctly computing Crack. Formally, we make the following definitions.

Let  $T$  be a nondeterministic transducer as described in § 2.2 and let  $p$  be a polynomial. A partial function  $f_{T,p}$  is defined as follows:

$$\text{dom}(f_{T,p}) = \{x \mid \text{every computation of } T \text{ on input } x \text{ halts within } p(|x|) \text{ steps} \\ \text{and there exists a string } y \text{ such that } \Pr(T(x) = y) \geq 3/4\}.$$

For each string  $x \in \text{dom}(f_{T,p})$ ,

$$f_{T,p}(x) = \text{unique } y \text{ such that } \Pr(T(x) = y) \geq 3/4.$$

(The choice of  $3/4$  is somewhat arbitrary and  $1/2 + \epsilon$ , for fixed  $\epsilon > 0$ , would suffice [BG81].)

**DEFINITION 3.**  $\text{BPPSV} = \{f_{T,p} \mid T \text{ is a transducer and } p \text{ is a polynomial}\}$ .

A definition for total functions was given in [RZ83] and this definition agrees with that one for total functions. Also, note that if a language  $L$  belongs to BPP, then the characteristic function of  $L$  belongs to BPPSV.

**THEOREM 14.** *If the promise problem  $(\text{dom}(f) \times \Sigma^*, \text{graph}(f))$  has a solution in  $P$ , then  $f$  is partially immune to BPPSV if and only if for every partial function  $g \in \text{BPPSV}$ ,  $\{x \mid g(x) = f(x)\}$  is a sparse set.*

The reader can construct a proof of this theorem easily from the proof of Theorem 13. In Theorem 13, the proof that (a) implies (b) depends on the fact that  $f \in \text{PSV}$  implies  $\text{dom}(f) \in P$ . This property does not seem to hold for  $f \in \text{BPPSV}$ , and so we do not include the corresponding assertion in this theorem.

Define a PKCS to be *secure with respect to BPPSV* if for every partial function  $g \in \text{BPPSV}$ ,  $\{(n, K_1, C) \mid g(n, K_1, C) = \text{Crack}(n, K_1, C)\}$  is a sparse set.

**COROLLARY 9.** *A PKCS is secure with respect to BPPSV if and only if Crack is partially immune to BPPSV.*

**5.2. Discussion.** Now that almost everywhere security measures have been studied, it might be appropriate to revisit the notion of one-way function. Suppose one-way functions are redefined based on the intuition that they should be difficult to invert on *most* inputs. Namely, let us say that a function is one-way if it is one-one, honest, computable in polynomial time, and its inverse is partially immune to BPPSV. Though we will not carry out the development here, it is possible to revisit § 4 and to obtain analogous conclusions.

The salient feature of public-key cryptosystems, as studied in this paper, is that

messages are encrypted deterministically. A number of researchers have been actively investigating systems that encrypt messages probabilistically [Gol84], [GM84], [BG85]. An advantage of this technique is that it makes it possible to send single bits securely over insecure channels. Improving a result of Shamir [Sha81], Blum and Micali [BM84] introduced the notion of *strong pseudorandom generator*. Here we do not need to say what strong pseudorandom generators are; we mention only that they have known applications to cryptography, that they have been used to design public-key cryptosystems [BM84], [BG85], but that their exact relationship to public-key cryptosystems remains unclear. By the latter assertion we mean that there is no theorem known of the form “strong pseudorandom generators imply secure public-key cryptosystems.” The papers [Yao82], [Lev85] each show that the existence of certain types of one-way functions imply the existence of pseudorandom generators. Thus, even though existence of one-way functions as they are defined in [Lev85] are in fact equivalent to existence of strong pseudorandom generators, the exact relationship between one-way functions and public-key cryptosystems remains unclear. It is interesting to observe that Yao’s notion of one-way function is similar to the one given in the previous paragraph in that both notions assert that one-way functions are hard to invert on most inputs. The fractions of inputs for which the inverses are difficult to compute are different.

**6. Oracles.** The first significant relativization results having to do with cryptography are due to Brassard [Bra83b], [Bra81]. In [Bra83b] it is shown that there is a recursive oracle relative to which there is a public-key cryptosystem that is secure with respect to PSV, and in [Bra81] this result is extended to security with respect to BPPSV.

We showed in this paper that there exist PKCS such that  $\text{Crack} \in \text{UPS}V - \text{PS}V$  only if  $P \neq \text{UP}$ , that there exist PKCS that cannot be cracked in polynomial time only if  $\text{NP} - P$  contains a disjoint pair of  $P$ -inseparable sets, and that there exist PKCS that are secure with respect to PSV only if there exist functions that are partially immune to PSV. Here we state relativization results that establish possibilities of existence and relationships between the pertinent complexity theoretic necessary conditions.

**6.1. The class UP.** Rackoff [Rac82] obtained oracles  $A$  and  $B$  such that  $P^A \neq \text{UP}^A = \text{NP}^A$  and  $P^B = \text{UP}^B \neq \text{NP}^B$ . Geske and Grollmann [GG86] obtained a number of strong relativized separation results for UP, among which we mention only the following: There exists a recursive oracle  $C$  and a nonsparse language  $L$  such that  $P^C \neq \text{UP}^C \neq \text{NP}^C$  and  $L \in \text{NP}^C$  is partially immune to  $\text{UP}^C$ . (In fact,  $L \in \text{R}^C$ , cf. [Gil77] and [AM77].)

The existence of functions in UPSV that are partially immune to BPPSV is a necessary condition for there to exist secure PKCS with  $\text{Crack} \in \text{UPS}V$ . Such functions exist if UP contains a set which is partially immune to R for it is well known that  $\text{R} \subseteq \text{BPP}$  [Gil77].

**6.2. Inseparability.** Since Theorem 5 holds for all oracles, the oracle  $C$  for which  $P^C \neq \text{UP}^C$  is also an oracle for which  $P$ -inseparable sets in NP exist. We do not believe that  $P$ -inseparable sets can be obtained from the hypothesis “ $P \neq \text{NP}$ .” Note that this conviction is paramount to saying that in order for there to exist PKCS that are secure in even the weakest sense,  $P \neq \text{NP}$  is not a sufficiently strong complexity theoretic hypothesis. However, we have not been successful at obtaining an oracle relative to which  $P \neq \text{NP}$  and  $P$ -inseparable sets do not exist. Such an oracle  $X$  also must satisfy  $P^X = \text{UP}^X = \text{NP}^X \cap \text{co-NP}^X$  and, trivially,  $\text{EXP}^X \not\subseteq \text{NP}^X$ , where  $\text{EXP} = \bigcup \{ \text{DTIME}(2^{cn}) \mid c > 0 \}$  (because  $A \in \text{EXP} - P$  implies  $\bar{A} \in \text{EXP} - P$ , and  $A$  and  $\bar{A}$  are  $P$ -inseparable).

The NP-complete set  $K$  (cf. Theorem 6) belongs to EXP. Hence  $\bar{K}$  is in EXP and

$K$  and  $\bar{K}$  can be separated only by  $K$  which is NP-hard. Thus, EXP contains disjoint sets which can be separated only by NP-hard sets. An oracle is constructed in [GH83] relative to which  $\text{EXP} \subseteq \text{NP}$ . Therefore, relative to this oracle NP contains disjoint sets that can be separated by NP-hard sets only. *We have not been able to construct an oracle  $Y$  relative to which the conjecture is true.* Such an oracle must satisfy  $\text{UP}^X \neq \text{NP}^X \neq \text{co-NP}^X$  and  $\text{EXP}^X \not\subseteq \text{NP}^X$ .

## REFERENCES

- [AM77] L. ADLEMAN AND K. MANDERS, *Reducibility, randomness, and intractability*, Proc. 9th ACM Symposium on Theory of Computing, 1977, pp. 151–163.
- [Ber76] L. BERMAN, *On the structure of complete sets: almost everywhere complexity and infinitely often speedup*, Proc. 17th IEEE Symposium on Foundations of Computing, 1976, pp. 76–80.
- [BG81] G. BENNETT AND J. GILL, *Relative to a random oracle  $A$ ,  $\text{P}^A \neq \text{NP}^A \neq \text{co-NP}^A$  with probability 1*, SIAM J. Comput., 10 (1981), pp. 96–113.
- [BG85] M. BLUM AND S. GOLDWASSER, *An efficient probabilistic public-key encryption scheme which hides all partial information*, in Proc. Crypto 84, Springer-Verlag, Berlin, 1985, pp. 289–295.
- [BGS75] T. BAKER, J. GILL AND R. SOLOVAY, *Relativizations of the  $\text{P} = ? \text{NP}$  question*, SIAM J. Comput., 4 (1975), pp. 431–441.
- [BL86] R. BOPPANA AND J. C. LAGARIAS, *One-way functions and circuit complexity*, in Structure in Complexity Theory. Lecture Notes in Computer Science, A. Selman, ed., Springer-Verlag, Berlin, 1986, pp. 51–65.
- [BLS84] R. BOOK, T. LONG AND A. SELMAN, *Quantitative relativizations of complexity classes*, SIAM J. Comput., 13 (1984), pp. 461–487.
- [BLS85] ———, *Qualitative relativizations of complexity classes*, J. Comput. System Sci., 30 (1985), pp. 395–413.
- [BM84] M. BLUM AND S. MICALI, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. Comput., 13 (1984), pp. 850–864.
- [Bra79] G. BRASSARD, *A note on the complexity of cryptography*, IEEE Trans. Inform. Theory, 25 (1979), pp. 232–233.
- [Bra81] ———, *A time-luck tradeoff in relativized cryptography*, J. Comput. System Sci., 22 (1981), pp. 280–311.
- [Bra83a] ———, *An optimally secure relativized cryptosystem*, in SIGACT news, pages 28–33, Crypto 81 Workshop on Cryptology, Univ. of California, Santa Barbara, Winter-Spring 1983.
- [Bra83b] ———, *Relativized cryptography*, IEEE Trans. Inform. Theory, 29 (1983), pp. 877–894.
- [BS85] J. BALCÁZAR AND U. SCHÖNING, *Bi-immune sets for complexity classes*, Math. Systems Theory, 18 (1985), pp. 1–18.
- [ESY84] S. EVEN, A. SELMAN AND Y. YACOBI, *The complexity of promise problems with applications to public-key cryptography*, Inform. and Control, 61 (1984), pp. 159–173.
- [EY80a] S. EVEN AND Y. YACOBI, *Cryptocomplexity and NP-completeness*, in Proc. 8th Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1980, pp. 195–207.
- [EY80b] ———, *An observation concerning the complexity of problems with few solutions and its applications to cryptography*, Technical Report 197, Computer Science Department, Technion, Haifa, Israel, 1980.
- [Fei85] J. FEIGENBAUM, personal communication, 1985.
- [FS74] P. FLAJOLET AND J. STEYAERT, *On sets having only hard subsets*, in Proc. 2nd Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1974, pp. 446–457.
- [GG86] J. GESKE AND J. GROLLMANN, *Relativizations of unambiguous and random polynomial time classes*, SIAM J. Comput., 15 (1986), pp. 511–519.
- [GH83] W. GASARCH AND S. HOMER, *Relativizations comparing NP and exponential time*, Inform. and Control, 58 (1983), pp. 88–100.
- [Gil77] J. GILL, *Computational complexity of probabilistic Turing machines*, SIAM J. Comput., 6 (1977), pp. 675–695.
- [GJ79] M. GAREY AND D. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
- [GM84] S. GOLDWASSER AND S. MICALI, *Probabilistic encryption*, J. Comput. System Sci., 28 (1984), pp. 270–299.
- [Gol84] S. GOLDWASSER, *Probabilistic encryption: Theory and applications*, Ph.D. thesis, University of California, Berkeley, CA, 1984.

- [HU79] J. HOPCROFT AND J. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.
- [Huy85] D. HUYNH, *Some observations about the randomness of hard problems*, TR 85-5, Iowa State University, Department of Computer Science, Ames, IA, February 1985.
- [Joh85] D. JOHNSON, *The NP-completeness column: an ongoing guide*, J. Algorithms, 6 (1985), pp. 291-305.
- [KLD86] K. KO, T. LONG AND D. DU, *A note on one-way functions and polynomial-time isomorphisms*, in Proc. Eighteenth ACM Symposium on Theory of Computing, 1986, pp. 295-302.
- [KM81] K. KO AND D. MOORE, *Completeness, approximation and density*, SIAM J. Comput., 10 (1981), pp. 787-796.
- [Ko82] K. KO, *Some observations on the probabilistic algorithms and NP-hard problems*, Inform. Process. Lett., 14 (1982), pp. 39-43.
- [Ko85] ———, *On some natural complete operators*, Theoret. Comput. Sci., 37 (1985), pp. 1-30.
- [Lau83] C. LAUTEMANN, *BPP and the polynomial hierarchy*, Inform. Process. Lett., 17 (1983), pp. 215-217.
- [Lem79] A. LEMPEL, *Cryptology in transition*, Comput. Surveys, 11 (1979), pp. 205-303.
- [Lev85] L. LEVIN, *One-way functions and pseudorandom generators*, in Proc. 17th Annual ACM Symposium on Theory of Computing, 1985, pp. 363-365.
- [MH78] R. MERKLE AND M. HELLMAN, *Hiding information and signatures in trapdoor knapsacks*, IEEE Trans. Inform. Theory, IT-24 (1978), pp. 525-530.
- [Mil76] G. MILLER, *Reimann's hypothesis and tests for primality*, J. Comput. System Sci., 13 (1976), pp. 300-317.
- [Pra75] V. PRATT, *Every prime has a succinct certificate*, SIAM J. Comput., 4 (1975), pp. 214-220.
- [PY84] C. PAPADIMITRIOU AND M. YANNAKAKIS, *The complexity of facets (and some facets of complexity)*, J. Comput. System Sci., 28 (1984), pp. 244-259.
- [Rab60] M. RABIN, *Degree of Difficulty of Computing a Function and a Partial Ordering of Recursive Sets*, 2, The Hebrew University, Jerusalem, 1960.
- [Rab79] ———, *Digitalized signatures and public-key functions as intractable as factorization.*, Technical Report MIT/LCS/TR-212, MIT Lab. for Comp. Sci., Cambridge, MA, 1979.
- [Rac82] C. RACKOFF, *Relativized questions involving probabilistic algorithms*, J. Assoc. Comput. Mach., 29 (1982), pp. 261-268.
- [Reg86] K. REGAN, *A uniform reduction theorem: extending a result of J. Grollmann and A. Selman*, in Proc. 13th Colloquium on Automata, Languages, and Programming. Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1986, pp. 324-333.
- [RSA78] R. RIVEST, A. SHAMIR AND L. ADLEMAN, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. ACM, 21 (1978), pp. 120-126.
- [RZ83] D. RUSSO AND S. ZACHOS, *Positive relativizations of probabilistic polynomial time*, Technical Report, University of California, Department of Mathematics, University of California, Santa Barbara, CA, 1983.
- [Sel78] A. SELMAN, *Polynomial time enumeration reducibility*, SIAM J. Comput., 7 (1978), pp. 440-457.
- [Sha81] A. SHAMIR, *On the generation of cryptographically strong pseudo-random sequences*, in Proc. 8th Colloquium on Automata, Languages, and Programming. Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1981, pp. 550-554.
- [Sha82] ———, *A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem*, in Proc. 23rd IEEE Symposium Found. of Computer Science, 1982, pp. 145-152.
- [Sho60] J. SHOENFIELD, *Degrees of models*, J. Symbolic Logic, 25 (1960), pp. 233-237.
- [Sip83] M. SIPSER, *A complexity theoretic approach to randomness*, in Proc. 15th ACM Symposium. Theory of Computing, 1983, pp. 330-335.
- [SXB83] A. SELMAN, XU M.-R. AND R. BOOK, *Positive relativizations of complexity classes*, SIAM J. Comput., 12 (1983), pp. 465-479.
- [Val76] L. VALIANT, *Relative complexity of checking and evaluating*, Inform. Process., 5 (1976), pp. 20-23.
- [Wat85] O. WATANABE, *On one-one polynomial time equivalence relations*, Theoret. Comput. Sci., 38 (1985), pp. 157-165.
- [Wil80] H. WILLIAMS, *A modification of the RSA public-key encryption procedure*, IEEE Trans. Inform. Theory, 26 (1980), pp. 726-729.
- [Wil83] R. WILBER, *Randomness and the density of hard problems*, in Proc. 24th IEEE Symposium on Foundations of Computer Science, 1983, pp. 335-342.
- [Yao82] A. YAO, *Theory and applications of trapdoor functions*, in Proc. 23rd IEEE Symposium on Foundations of Computer Science, 1982, pp. 80-91.
- [You83] P. YOUNG, *Some structural properties of polynomial reducibilities and sets in NP*, in Proc. Fifteenth ACM Symposium on Theory of Computing, 1983, pp. 392-401.



## SOLVING SIMULTANEOUS MODULAR EQUATIONS OF LOW DEGREE\*

JOHAN HASTAD†

**Abstract.** We consider the problem of solving systems of equations  $P_i(x) \equiv 0 \pmod{n_i}$   $i = 1 \cdots k$  where  $P_i$  are polynomials of degree  $d$  and the  $n_i$  are distinct relatively prime numbers and  $x < \min(n_i)$ . We prove that if  $k > d(d+1)/2$  we can recover  $x$  in polynomial time provided  $\min(n_i) > 2^{d^2}$ . As a consequence the RSA cryptosystem used with a small exponent is not a good choice to use as a public-key cryptosystem in a large network. We also show that a protocol by Broder and Dolev [Proceedings on the 25th Annual IEEE Symposium on the Foundations of Computer Science, 1984] is insecure if RSA with a small exponent is used.

**Key words.** cryptography, lattice algorithm, RSA, coin flipping, modular equations

**AMS(MOS) subject classifications.** 68Q20, 11H99, 11Y99

**1. Introduction.** Let us start with some cryptographic motivation. The RSA function [10] is defined as  $f(x) \equiv x^e \pmod{n}$ . Here  $n$  is usually taken of the form  $n = pq$  where  $p$  and  $q$  are two large primes and  $e$  is an integer relatively prime to  $(p-1)(q-1)$ . Using these parameters the function is 1-1 when restricted to  $1 \leq x \leq n$ ,  $(x, n) = 1$ . Furthermore the function is widely believed to be a *trapdoor function*, i.e., given  $n$  and  $e$  it is easy to compute  $f(x)$  and given  $f(x)$  it is also easy to recover  $x$  provided one has some secret information but otherwise it is difficult to compute  $x$ . In this case the secret information is the factorization of  $n$ .

The RSA function can be used to construct a deterministic Public Key Cryptosystem (PKC) in the following way: Each user  $B$  in a communication network chooses two large primes  $p$  and  $q$  and multiplies them together and publishes the result  $n_B$  together with a number  $e_B$  which is relatively prime to  $(p-1)(q-1)$ . He keeps the factorization of  $n_B$  as his private secret information. If any user  $A$  in the system wants to send a secret message  $m$  to another user  $B$  she retrieves  $B$ 's published information, computes  $y \equiv m^{e_B} \pmod{n_B}$  and sends  $y$  to  $B$ .  $B$  now obtains the original message using his secret information, while somebody who does not know the secret information presumably faces an intractable computational task.

Public Key Cryptosystems are different and more complex objects than are trapdoor functions. The reason is that a PKC involves a protocol consisting of several steps. For example the use of RSA in a PKC may present obstacles that did not occur when we considered it as a trapdoor function. Several people (including Blum, Lieberherr and Williams) have observed the following possible attack. Assume that 3 is chosen as the exponent and that  $A$  wants to send the same message  $m$  to users  $U_1$ ,  $U_2$  and  $U_3$ . She will compute and send  $y_i \equiv m^3 \pmod{n_i}$ ,  $i = 1, 2, 3$ . If someone gains access to  $y_1$ ,  $y_2$  and  $y_3$  then by using the fact that  $n_1$ ,  $n_2$  and  $n_3$  will be relatively prime he can combine the messages by Chinese remaindering to get  $m^3 \pmod{n_1 n_2 n_3}$  and since  $m^3 < n_1 n_2 n_3$  he can recover  $m$ . In general if the exponent is  $e$  the number of messages needed is  $e$ .

A natural question is therefore: Is there a better way to send the same message to many people using this PKC?

\* Received by the editors September 16, 1985; accepted for publication (in revised form) July 7, 1986.

† Massachusetts Institute of Technology, Laboratory for Computer Science, Cambridge, Massachusetts 02139. This work was supported by an IBM fellowship, and was partially supported by National Science Foundation grant DCR-8509905.

A common heuristic tells us to use a “time stamp.” Instead of sending the same message  $m$  to everybody one attaches the time and thus sends the encryption of  $2^{|t_i|}m + t_i$  where  $2^{|t_i|}m$  is the shifted message and  $t_i$  is the time when the message is sent to user  $U_i$ . This time will be different for the different receivers. The previous attack then fails.

If we assume that the times  $t_i$  are known to the cryptanalyst we are led to consider the following computational problem (for  $e = 3$ ).

Given  $(a_i m + b_i)^3 \pmod{n_i}$  where all the  $a_i$  and  $b_i$  are known is it possible to recover  $m$  in polynomial time?

We will prove in § 3 that the answer is YES if the number of similar messages is at least 7. In fact we will prove this as a special case of our main result, which is: Given a set of  $k$  polynomial equations

$$P_i(x) \equiv 0 \pmod{n_i}, \quad i = 1, \dots, k$$

each of degree  $\leq d$ , it is possible to recover all solutions in time polynomial in both  $k$  and  $\log n_i$  if  $k > d(d + 1)/2$  provided  $\min(n_i) > 2^{d^2}$ .

Observe that the described attack does not work if the values of the  $t_i$  are not known to the cryptanalyst. Thus if a random padding were used or if the time stamp were unknown then the present attack would not work. However, this weakness seems severe enough so that if one uses RSA as a PKC then, as a matter of prudence, one should use a large exponent or, even better, a probabilistic encryption scheme [3], [7] based on RSA. By [1], [3] this can be done with as much efficiency as in the deterministic case.

The outline of the paper is as follows. In § 2 we state some results from geometry of numbers which will be needed in later sections. In § 3 we state and prove our main result and in § 4 we derive some cryptographic applications.

**2. Background from geometry of numbers.** The main tool in our algorithm will be the use of lattices. In this section we will gather the relevant background information. A *lattice*  $L$  is defined to be the set of points

$$L = \left\{ y \mid y = \sum_{i=1}^n a_i \vec{b}_i, a_i \in \mathbf{Z} \right\}$$

where  $\vec{b}_i$  are linearly independent vectors in  $\mathbf{R}^n$ . The set  $\vec{b}_i$  is called a *basis* for the lattice and  $n$  is the dimension. The *determinant* of a lattice is defined as the absolute value of the determinant of the matrix with rows  $\vec{b}_i$ . It is not hard to see that the determinant is independent of the choice of basis. The length of the shortest nonzero vector in the lattice is denoted by  $\lambda_1$ . Let us recall the following well-known fact:

**THEOREM (Minkowski).**  $\lambda_1 \leq \gamma_n^{1/2} (\det(L))^{1/n}$  where  $\gamma_n$  is Hermite’s constant.

Hermite’s constant is not known exactly for  $n > 8$  but Minkowski’s convex body theorem [5, ix.7] implies that  $\gamma_n \leq n$ . Lenstra et al. showed in [8] that it was possible to find a vector in  $L$  of length at most  $2^{(n-1)/2} \lambda_1$  in polynomial time. From their proof we can, however, derive a slightly better bound in the present case.

**THEOREM (LLL).** *Given a lattice  $L$  as a basis for integer vectors of length at most  $B$  we can find a vector  $\vec{b}$  in time  $O(n^6 (\log B)^3)$  which satisfies  $\|\vec{b}\| \leq 2^{(n+1)/4} (\det(L))^{1/n}$ .*

This gives an effective variant of Minkowski’s theorem. Here  $\|\vec{b}\|$  is the euclidean length of the vector  $\vec{b}$ . The bound on the running time assumes that multiplication of  $r$  bit numbers is done by classical arithmetic taking  $O(r^2)$  steps. Using faster multiplication routines the bounds can be improved by a factor close to  $n \log B$ . Armed with this information we return to the original problem.

**3. Main theorem.** Let us start by fixing some notation. Let  $N = \prod_{i=1}^k n_i$  and  $n = \min n_i$ . Now we can state the problem formally:

*Problem.* Given a set of  $k$  equations  $\sum_{j=0}^d a_{ij}x^j \equiv 0 \pmod{n_i}, i = 1, \dots, k$ . Suppose that the system has a solution  $x < n$  and the numbers  $n_i$  are pairwise relatively prime. Can we find such a solution efficiently?

Before we state our main result let us give the basic ideas. Define  $u_j < N$  to be the Chinese remaindering coefficients, i.e.,  $u_j \equiv \delta_{ij} \pmod{n_i}$  ( $\delta_{ij} = 1$  if  $i = j$  and 0 otherwise). We can combine the equations to a single equation using the Chinese Remainder Theorem.

$$0 \equiv \sum_{j=0}^d x^j \sum_{i=1}^k u_i a_{ij} \equiv \sum_{j=0}^d x^j c_j \pmod{N}.$$

One of the important parts of the entire paper is the following simple lemma.

LEMMA 1. *If  $|c_j| < N/(d+1)n^j$  and we have at least one nonzero  $c_j$  then we can find all  $x$  satisfying  $x < n$  and  $\sum_{j=0}^d c_j x^j \equiv 0 \pmod{N}$  in time  $O((d \log N)^3)$ .*

*Proof.* If  $|c_j| < N/(d+1)n^j$  then

$$\left| \sum_{j=0}^d c_j x^j \right| \leq \sum_{j=0}^d |c_j| n^j < N.$$

Thus the condition  $\sum_{j=0}^d c_j x^j \equiv 0 \pmod{N}$  implies  $\sum_{j=0}^d c_j x^j = 0$ . In other words  $x$  solves the equation over the integers and to prove the lemma we just need the fact that we can solve polynomial equations over the integers quickly. Since this is a special case in which we are looking for an integer solution, we can proceed as follows. Find all linear factors modulo a small prime. Now apply Hensel lifting to obtain these factors modulo a large power of the prime and finally check if any of the roots is a root over the integers. The estimate for the running time in the lemma is correct but not the best possible.  $\square$

The condition of Lemma 1 is quite unlikely to be fulfilled when we start with a general set of equations. In spite of this, Lemma 1 will be one of our main tools for proving our main result, which is as follows.

THEOREM. *Given a set of equations  $\sum_{j=0}^d a_{ij}x^j \equiv 0 \pmod{n_i}, i = 1, 2, \dots, k$  where the moduli  $n_i$  are pairwise relatively prime and  $\gcd(\langle a_{ij} \rangle_{j=0}^d, n_i) = 1$  for all  $i$ . Then we can find all  $x < n$  satisfying the equations in time  $O(d^6(\log N)^3)$  if*

$$N > n^{d(d+1)/2} 2^{(d+2)(d+1)/4} (d+1)^{(d+1)}.$$

As before,  $N = \prod_{i=1}^k n_i, n = \min n_i, d$  is the degree of the equations and  $k$  is the number of equations. By  $\gcd(\langle a_{ij} \rangle_{j=0}^d, n_i)$  we mean the greatest common divisor of all  $d+2$  numbers.

*Proof.* The idea is to use Lemma 1, though it is unlikely that it will apply to our equations directly. However, we have an extra degree of freedom. We can multiply the equation by an arbitrary constant  $S$  and we still get a valid equation. Using this trick we will be able to make the coefficients small. Thus, we want to make  $Sc_i \pmod{N}$  less than  $N/n^i(d+1)$  in absolute value. Set up the following lattice  $L$  of dimension  $d+2$ :

$$\begin{aligned} \vec{b}_1 &= (c_0, nc_1, n^2c_2, \dots, n^dc_d, 1/(d+1)), \\ \vec{b}_2 &= (N, 0, 0, \dots, 0, 0), \\ \vec{b}_3 &= (0, nN, 0, \dots, 0, 0), \\ \vec{b}_4 &= (0, 0, n^2N, \dots, 0, 0), \\ &\vdots \\ \vec{b}_{d+2} &= (0, 0, 0, \dots, n^dN, 0). \end{aligned}$$

Let us see why this lattice is relevant to our purposes. Look at a generic vector  $S\vec{b}_1 + \sum_{i=2}^{d+2} s_i \vec{b}_i$ . Call the  $i$ th coordinate  $d_i$ . From the definition, it follows that  $d_i$  is

divisible by  $n^{i-1}$  and  $d_i/n^{i-1} \equiv S c_{i-1} \pmod{N}$ . Thus if we find a vector  $\vec{b} \in L$  satisfying  $\|\vec{b}\| < N/(d+1)$  we know that  $|d_i| < N/(d+1)$  and we get the desired bound for  $S c_i \pmod{N}$ . The last coordinate is there to prevent  $S = N$ , which would make  $d_i = 0$  for  $i = 1, \dots, d+1$ .

We have only one term in the expansion of the determinant and we get

$$\text{Det}(L) = \frac{n^{d(d+1)/2} N^{d+1}}{(d+1)}.$$

Using the theorem of LLL in § 2 we know that we can find a vector  $\vec{b}$  in  $L$  that satisfies

$$\|\vec{b}\| \leq 2^{(d+1)/4} \left( \frac{n^{d(d+1)/2} N^{d+1}}{d+1} \right)^{1/(d+2)}.$$

As observed above we need  $\|\vec{b}\| < N/(d+1)$  in order to get the desired bound for the coefficients and thus we need,

$$2^{(d+1)/4} \left( \frac{n^{d(d+1)/2} N^{d+1}}{d+1} \right)^{1/(d+2)} < \frac{N}{d+1}.$$

Raising both sides to the  $d+2$  power and rearranging we see that this is equivalent to the condition in the theorem.

To finish the proof we need to prove that we have at least one nonzero coefficient. Since  $\|\vec{b}\| < N/(d+1)$ , we see by looking at the last coordinate that the coefficient  $S$  multiplying  $\vec{b}_1$  satisfies  $|S| < N$ . Furthermore we know that  $S \neq 0$  since all nonzero vectors with  $S = 0$  are of length at least  $N$ . This means that there is an  $n_i$  such that  $S \not\equiv 0 \pmod{n_i}$ . Look at the equation modulo this  $n_i$ . Using that  $\gcd(\langle a_{ij} \rangle_{j=0}^d, n_i) = 1$  we see that the equation is nontrivial. The bottleneck in the computation is the lattice computation and this gives the running time of the algorithm.  $\square$

*Remark.* One interesting open question is whether we can solve the problem with fewer equations. It does not seem possible to use this line of attack with substantially fewer equations. To see this one might argue as follows:

The probability that  $|c_j| < N/(d+1)n^j$  for  $j = 0, 1, \dots, d$  for a fixed  $S$  is approximately  $n^{-d(d+1)/2}$  and this would indicate that we should have  $n^{d(d+1)/2}$  different  $S$  to choose between; we would, therefore, need at least  $d(d+1)/2$  equations.

**4. Cryptographic applications.** We get some immediate applications of the main theorem.

*APPLICATION 1.* *Sending linearly related messages using RSA with low exponent  $e$  is insecure. Sending more than  $e(e+1)/2$  messages enables an adversary to recover the messages provided that the moduli  $n_i$  satisfy  $n_i > 2^{(e+2)(e+1)/4}(e+1)^{(e+1)}$ .*

*Proof.* Suppose we are given the encryption of  $k$  linearly dependent messages. We expand the  $e$ th power and we get  $k$  equations of degree  $e$  with the different moduli used  $n_i$ . We now apply the main theorem. We need to verify that the conditions of the main theorem are satisfied. If one of the gcd conditions is not satisfied we can obtain the message by factoring one of the  $n_i$ . Finally,

$$N = \prod_{i=1}^k n_i \geq n_1 \prod_{i=2}^{d(d+1)/2+1} n_i > 2^{(e+1)(e+2)/4}(e+1)^{(e+1)} n^{d(d+1)/2}$$

and hence we can apply our main theorem.  $\square$

If one is prepared to do computation which is exponential in the number of equations, one can also attack the cryptosystem given exactly  $e(e+1)/2$  messages. The way to proceed is to use an almost identical lattice. The only difference is to replace

the last coordinate of the first vector with  $2^{(e+2)/4}$ . Now the algorithm of LLL finds a vector in the lattice of length at most  $N2^{(e+2)/4}$ . This implies in particular that  $c_i < N2^{(e+2)/4}n^{-i}$ . Now it is no longer possible to conclude that  $x$  solves the equation over the integers. We do know, however, that the right-hand side is a multiple of  $N$  not exceeding  $e2^{(e+2)/4}N$  and so we try all  $e2^{(e+2)/4}$  possibilities.

Another way of encrypting messages was proposed by Rabin [9]. He uses  $f(x) \equiv x^2 \pmod{n}$  where also here  $n$  is chosen to be a specific composite number for each user. Using the same methods we get:

**APPLICATION 2.** *Sending linearly related messages using the Rabin encryption function is insecure. If three such messages are sent it is possible to retrieve the message in polynomial time.*

Broder and Dolev proposed a protocol for flipping a coin in a distributed system [4]. Two of their essential ingredients were Shamir's method of sharing a secret [11] and the use of a deterministic PKC. The secret they use is the constant coefficient of a polynomial of degree  $t$  over a finite field. The secret is distributed by evaluating the polynomial at a given set of points. It is easy to see that  $t+1$  pieces, each consisting of the value of the polynomial at a point, are enough to get the secret back while  $t$  pieces are not sufficient to determine the polynomial. Broder and Dolev claim that  $t$  pieces are insufficient to find the secret, even in the presence of the encryption of other pieces. This is not correct if the cryptosystem used is RSA with small exponent. This is because when knowing  $t$  pieces the secret enters linearly in the remaining pieces and hence we can use Application 1.

**APPLICATION 3.** *The protocol by Broder and Dolev is insecure if RSA with small exponent is used.*

Of course if other cryptosystems are used then this attack does not work. A different type of attack on the Broder-Dolev protocol has been proposed by Benny Chor [6]. This attack just relies on the protocol and not on the cryptosystem. A provably secure protocol has been designed by Awerbuch et al. [2]. For further discussion of coin-flipping protocols see [2] and [4].

Finally we remark that there does not seem to be any way to extend the above attack to RSA with large exponent. The reason is that the integers involved are too big even to write down. There is still a large amount of structure present and it would be interesting to investigate whether this structure could be exploited to yield a successful cryptanalytic attack on RSA with large exponent.

**Acknowledgments.** I would like to thank Silvio Micali, Shafi Goldwasser and Benny Chor for suggesting the problem, listening to early solutions and suggesting improvements and simplifications. They also pointed out the incorrectness in the proof of Broder and Dolev. I am also very grateful to Ron Rivest for greatly simplifying the proof and finally I would like to thank Jeff Lagarias for many helpful comments.

#### REFERENCES

- [1] W. ALEXI, B. CHOR, O. GOLDREICH AND C. P. SCHNORR, *RSA/Rabin bits are  $\frac{1}{2} + 1/\text{poly}(\log N)$  secure*, Proc. 25th Annual IEEE Symposium on the Foundations of Computer Science, 1984, pp. 449-457.
- [2] B. AWERBUCH, B. CHOR, M. BLUM, S. GOLDWASSER AND S. MICALI, *Fair coin flip in a Byzantine environment*, in preparation.
- [3] M. BLUM AND S. GOLDWASSER, *An efficient probabilistic public key encryption scheme which hides all partial information*, in Advances in Cryptology: Proceedings of CRYPTO 84, G. R. Blakeley and D. Chaum, eds., Lecture Notes in Computer Science 196, Springer-Verlag, New York, Berlin, 1985, pp. 289-299.

- [4] A. Z. BRODER AND D. DOLEV, *Flipping coins in many pockets*, Proc. 25th Annual IEEE Symposium on the Foundations of Computer Science, 1984, pp. 157–170.
- [5] J. W. S. CASSELS, *An Introduction to the Geometry of Numbers*, Springer-Verlag, Heidelberg, 1971.
- [6] B. CHOR, *personal communication*, 1985.
- [7] S. GOLDWASSER AND S. MICALI, *Probabilistic encryption*, J. Comput. Syst. Sci., 28 (1984), pp. 270–299.
- [8] A. K. LENSTRA, H. W. LENSTRA AND L. LOVÁSZ, *Factoring polynomials with integer coefficients*, Mathematische Annalen, 261 (1982), pp. 513–534.
- [9] M. O. RABIN, *Digital signatures and public key functions as intractable as factorization*, Massachusetts Institute of Technology, Library of Congress Series, TR-212, 1979.
- [10] R. L. RIVEST, A. SHAMIR AND L. ADLEMAN, *A method for obtaining digital signatures and public key cryptosystems*, Comm. ACM, 21-2 (1978).
- [11] A. SHAMIR, *How to share a secret*, Comm. ACM, 22 (1979), pp. 612–613.

## UNIQUE EXTRAPOLATION OF POLYNOMIAL RECURRENCES\*

JEFFREY C. LAGARIAS† AND JAMES A. REEDS†

**Abstract.** Let a sequence of  $k$ -dimensional vectors  $\mathbf{x}_0, \mathbf{x}_1, \dots$  (over a ring  $A$ ) be determined by a polynomial recurrence of form  $\mathbf{x}_n = T(\mathbf{x}_{n-1})$ , where  $T: A^k \rightarrow A^k$  itself is known to be a polynomial map in  $k$  variables of degree at most  $d$  but is otherwise unknown. We show that there is a finite  $N$  such that the entire sequence  $\{\mathbf{x}_n : n \geq 0\}$  can be deduced from the first  $N+1$  terms  $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_N$  alone. The number  $N = \phi(d, k, A)$  depends on  $d$  and  $k$  and the ring  $A$  but not on  $T$ .

Let  $\phi^*(d, k)$  denote the maximum of  $\phi(d, k, A)$  over all commutative rings with unit. Then we show that  $\phi^*(d, k) < \infty$ . In particular,  $\phi^*(d, 1) = d + 1$  and  $\phi^*(1, k) = k + 1$ . In the general case  $\phi^*(d, k) \cong \binom{k+d}{k}$  and equality does not always hold because  $\phi^*(2, 2) \cong 7$ . In addition, we show that for each  $k$  that  $\max \{\phi(d, k, \mathbb{F}) : \mathbb{F} \text{ a field}\}$  is bounded by a polynomial in  $d$ .

These results are applied to the problem of correctly extrapolating the values  $\{x_i : i \geq 0\}$  of an unknown polynomial recurrence (mod  $M$ ) in  $k$  variables of degree at most  $d$ , where  $d$  and  $k$  are known and  $M$  is not known. A polynomial-time algorithm is given which computes a value  $\hat{x}_{n+1}$  given the values  $\{x_i : 0 \leq i \leq n\}$  of such a recurrence as input, and it is shown that  $\hat{x}_{n+1} \neq x_{n+1}$  for at most

$$1 + \phi^*(d, k) + \log(M^{dN} N^{\frac{1}{2}N})$$

values of  $n$ , where  $N = 1 + k \binom{k+d}{k}$ .

**Key words.** recurrences, pseudorandom numbers, cryptanalysis

**AMS(MOS) subject classifications.** 11K45, 11B37, 11T71

**1. Introduction.** One of the general methods of pseudorandom number generation is to iterate a function  $T$  starting with a "seed"  $x_0$  to obtain the  $T$ -orbit of  $x_0$ , namely the sequence  $x_0, x_1, x_2, \dots$  with  $x_n = T(x_{n-1})$  for  $n \geq 1$ . For example, the linear congruential generator [K] with *modulus*  $m$ , *multiplier*  $a$  and *increment*  $b$  arises from iteration of the linear function  $T(x) = ax + b$  in the finite ring  $\mathbb{Z}/m\mathbb{Z}$ .

In this paper we study the possibility of correctly predicting (or extrapolating) an entire orbit  $\{x_n : n \geq 0\}$  given only a finite segment of initial values  $x_0, x_1, \dots, x_N$  and given only minimal qualitative information about the function  $T$ .

Sometimes it is possible to deduce the whole orbit from a finite segment even if it is not possible to deduce the transformation  $T$ . For example, if  $N = 2$  and  $x_0 = x_1$  we know the whole orbit but know nothing about  $T$  except its value at a single point:  $T(x_0) = x_0$ .

The following result, conjectured by J. Boyar [B2], is a special case of those proved in this paper: Suppose the sequence  $\{x_n : n \geq 0\}$  is determined by a polynomial recurrence modulo  $m$ :

$$x_n \equiv P(x_{n-1}) \pmod{m}$$

where the degree of the polynomial  $P$  is  $d$  or less. Then the whole sequence  $\{x_n : n \geq 0\}$  is determined by the initial segment  $x_0, x_1, \dots, x_{d+1}$ . In other words, if  $\{y_n : n \geq 0\}$  is another such sequence, determined by a polynomial recurrence

$$y_n \equiv Q(y_{n-1}) \pmod{m}$$

\* Received by the editors March 12, 1986; accepted for publication (in revised form) June 11, 1987.

† AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

where the degree of  $Q$  is also less than or equal to  $d$ , and if for  $0 \leq n \leq d + 1$

$$x_n \equiv y_n \pmod{m},$$

then  $x_n \equiv y_n \pmod{m}$  for all  $n \geq 0$ .

This work arises from cryptography. A number of simple cryptosystems are based on masking a plaintext using a sequence of pseudorandom bits. A subproblem in the cryptanalysis of such systems is the correct prediction of the sequence of pseudorandom bits from a short initial segment of bits. Linear congruential generators are one of the simplest and most well studied pseudorandom number generators, and the problem of extrapolating the output of such generators was studied by J. Boyar [B1], [B2], [B3]. She showed that linear congruential generators are cryptographically insecure in the sense that even without knowledge of  $a, b, m$  there are polynomial-time algorithms which successfully extrapolate  $x_{n+1}$  given  $\{x_i : 0 \leq i \leq n\}$  for  $n = 1, 2, 3, \dots$  and which make at most  $2 + \log m$  incorrect extrapolations in the process. In the case that the modulus  $m$  is known but  $a, b$  are unknown, knowledge of the first three iterates suffices to permit correct extrapolation for all  $n$ . One might hope that polynomial recurrences give more cryptographic security than linear recurrences. This paper shows, in a qualitative sense, that this is not the case.

Motivated by the case of polynomial congruential generators we study the unique extrapolation properties of the orbits of multivariate polynomial recurrences over general commutative rings with unit. Throughout this paper “ring” will mean “commutative ring with unit.” Let  $A$  be a ring and let  $A^k$  be the  $k$ -tuples of elements of  $A$ . We consider polynomial maps  $T: A^k \rightarrow A^k$  defined by

$$T(X_1, \dots, X_k) = (T_1(X_1, \dots, X_k), \dots, T_k(X_1, \dots, X_k))$$

where each  $T_i(X_1, \dots, X_k) \in A[X_1, \dots, X_k]$  is a polynomial with coefficients in  $A$ . We define the degree of  $T$  by

$$\text{deg}(T) = \max \{ \text{total degree } T_i(X_1, \dots, X_k) : 1 \leq i \leq k \}.$$

A polynomial map  $T$  and an initial vector or seed  $\mathbf{x}_0 \in A^k$  determine a sequence of elements  $\mathbf{x}_n = T(\mathbf{x}_{n-1})$  of  $A^k$ , the  $T$ -orbit of  $\mathbf{x}_0$ . We let  $T^{(n)}$  denote the  $n$ -fold composition of  $T$  so that  $\mathbf{x}_n = T^{(n)}(\mathbf{x}_0)$ . We study conditions which guarantee that two orbits coincide. We say that a collection  $\Sigma$  of sequences of elements of  $A^k$  is  $m$ -prefix distinguishable if whenever  $\{\mathbf{x}_n : n \geq 0\}$  and  $\{\mathbf{y}_n : n \geq 0\}$  are two sequences in  $\Sigma$  for which  $\mathbf{x}_n = \mathbf{y}_n$  for  $0 \leq n \leq m$ , it follows that  $\mathbf{x}_n = \mathbf{y}_n$  for all  $n \geq 0$ . An arbitrary collection of sequences need not be  $m$ -prefix distinguishable for any finite  $m$ .

Let  $S(d, k, A)$  denote the set of all orbits of all polynomial maps  $T: A^k \rightarrow A^k$  of degree at most  $d$ . Let  $\phi(d, k, A)$  be the smallest  $m$  such that  $S(d, k, A)$  is  $m$ -prefix distinguishable and let  $\phi^*(d, k)$  be the supremum of  $\phi(d, k, A)$  over all rings  $A$ . If  $A$  is a subring of  $B$ , then  $\phi(d, k, A) \leq \phi(d, k, B)$ . Our first result is as follows.

**THEOREM 1.1.**  $\phi^*(d, k) < \infty$ .

Note that it is not a priori obvious that  $\phi(d, k, A) < \infty$  for any given ring  $A$ , let alone that the  $S(d, k, A)$  for different rings  $A$  should be  $m$ -prefix distinguishable for the same value of  $m$  at once. Our proof of this result is nonconstructive and uses Hilbert’s basis theorem.

We next obtain explicit bounds for  $\phi^*(d, k)$  in special cases. The simplest case is when the equations are linear, i.e., when  $d = 1$ .

**THEOREM 1.2.** For all dimensions  $k \geq 1$ ,

$$\phi^*(1, k) = k + 1.$$

In fact,  $\phi(1, k, \mathbf{Q}) = k + 1$ .



The main result of this paper concerns the one-dimensional case.

**THEOREM 1.3.** *For all degrees  $d \geq 1$ ,*

$$\phi^*(d, 1) = d + 1.$$

*In fact,  $\phi(d, 1, \mathbf{Q}) = d + 1$ .*

Boyar’s conjecture follows from this result, since

$$\phi(d, 1, \mathbf{Z}/m\mathbf{Z}) \leq \phi^*(d, 1) = d + 1.$$

The situation where the dimension  $k$  and the degree  $d$  are both at least two seems more complicated. We obtain a lower bound for  $\phi(d, k, \mathbf{F})$  when  $\mathbf{F}$  is an algebraically closed field using elementary methods from algebraic geometry.

**THEOREM 1.4.** *For any algebraically closed field  $\mathbf{F}$ ,  $\phi(d, k, \mathbf{F}) \geq \binom{d+k}{d}$ . Hence*

$$\phi^*(d, k) \geq \binom{d+k}{d}.$$

This result holds because a “generic” polynomial map  $T$  of degree  $d$  is uniquely determined by its values at  $\binom{k+d}{d}$  points (Theorem 4.1). Theorems 1.2 and 1.3 show this inequality is sharp when  $k = 1$  or  $d = 1$ . However, for  $k \geq 2$  and  $d \geq 2$  the quantity  $\phi(d, k, \mathbf{F})$  is apparently determined by “nongeneric” recurrences, and an example due to James Butler shows that  $\phi(2, 2, \mathbf{Q}) \geq 7$  (Proposition 4.2). (The “generic” bound is  $\binom{2+2}{2} = 6$ .)

We have not obtained any effective upper bounds for  $\phi^*(d, k)$  when  $d \geq 2$  and  $k \geq 2$ . In § 5 we obtain effective upper bounds for  $\phi(d, k, \mathbf{F})$  when  $\mathbf{F}$  is a field, using methods from algebraic geometry related to Bezout’s inequality (Theorem 5.7). In particular, we prove that

$$(1.1) \quad \phi(d, 2, \mathbf{F}) \leq d^4 + d,$$

and our results imply the asymptotic bound

$$\phi(d, k, \mathbf{F}) \leq d^{(2^{k-1}-1)k+2^{k-1}} \left( 1 + \mathcal{O}\left(\frac{1}{d}\right) \right),$$

as the degree  $d \rightarrow \infty$  with  $k$  fixed. By comparison the “generic” lower bound of Theorem 1.4 implies that

$$\phi(d, k, \mathbf{F}) \geq \frac{1}{k!} d^k + \mathcal{O}(d^{k-1})$$

as  $d \rightarrow \infty$  with  $k$  fixed. For the case  $k = d = 2$  we can show that  $\phi(2, 2, \mathbf{F}) \leq 15$  by more detailed arguments. These upper bounds are probably far from the best possible.

To summarize, if either  $d = 1$  or  $k = 1$  there are two kinds of recurrences: generic recurrences and exceptionally good recurrences. The generic recurrences can be uniquely extrapolated from an initial sequence  $x_0, x_1, \dots, x_n$ , with  $n = \binom{k+d}{d}$ , and the exceptionally good recurrences can be uniquely extrapolated from a shorter initial segment. If, however, both  $d > 1$  and  $k > 1$  then a new class occurs, the exceptionally bad recurrences for which initial segments of length more than  $1 + \binom{k+d}{d}$  are needed for unique extrapolation. Both kinds of exceptional recurrences are sparse: the generic recurrences form a dense open subset (in the Zariski topology) of the set of all recurrences of given degree and dimension. But now the worst case for  $d \geq 2$  and  $k \geq 2$  is not the generic case.

These results on unique extrapolation of recurrences can be applied to the problem of extrapolating the value  $\mathbf{x}_{n+1}$  of an unknown polynomial recurrence (mod  $M$ ) in  $k$  variables which is known to be of degree at most  $d$ , given  $\{\mathbf{x}_i; 0 \leq i \leq n\}$  as data, where the modulus  $M$  is itself unknown. The data  $\{\mathbf{x}_i; 0 \leq i \leq n\}$  are assumed to be given with integer entries that are least-positive residues (mod  $M$ ). In § 6 we show that there is a polynomial-time algorithm that produces an extrapolation  $\hat{\mathbf{x}}_{n+1}$  given  $\{\mathbf{x}_i; 0 \leq i \leq n\}$  as input, and that  $\hat{\mathbf{x}}_{n+1} \neq \mathbf{x}_{n+1}$  for at most

$$\phi^*(d, k) + 1 + \left[ 1 + k \binom{k+d}{d} \right] \left\{ d \log_2 M + \frac{1}{2} \log_2 \left( 1 + k \binom{k+d}{d} \right) \right\}$$

values of  $n$ . The proofs of this section essentially follow the approach of Boyar [B1], [B2], [B3].

There remain a number of open problems. The most important is that of finding an exact formula (or failing that, an effective upper bound) for  $\phi^*(d, k)$  when  $d \geq 2$  and  $k \geq 2$ . In another direction, Theorems 1.1 and 1.2 are proved by exhibiting certain “universal” polynomial identities. The identity used to prove  $\phi^*(1, k) = k + 1$  is just the Cayley–Hamilton identity stating that a matrix  $M$  satisfies its characteristic equation  $\chi_M(M) = 0$ . Do the identities used to show  $\phi^*(d, 1) = d + 1$  in Theorem 1.2 also have an interesting algebraic interpretation? (C. Mallows [Mal] has obtained a nice answer to this question.) A third problem area concerns the nature of exceptionally bad recurrences. Can the set of exceptionally bad recurrences be characterized in a simple way? Do exceptionally bad recurrences have interesting algebraic, combinatorial or cryptographic properties?

Other results concerning the cryptographic and pseudorandom properties of linear recurrences may be found in [K], [Mar], [FHKLS], [FKL], [HS].

**2. Unique extrapolation property for polynomial recurrences.** We derive Theorem 1.1 as a corollary of the following general result.

**THEOREM 2.1.** *For any dimension  $k$  and any degree bound  $d$  there is a finite integer  $f(d, k)$  with the following property: For any ring  $A$  (commutative, with unit), and any polynomial maps*

$$T: A^k \rightarrow A^k,$$

$$S: A^k \rightarrow A$$

*both having degree at most  $d$ , and any initial value  $\mathbf{x}_0 \in A^k$ , if*

$$\mathbf{x}_i = T^{(i)}(\mathbf{x}_0)$$

*and*

$$S(\mathbf{x}_i) = 0 \quad \text{for } 0 \leq i \leq f(d, k),$$

*then*

$$S(\mathbf{x}_i) = 0 \quad \text{for } 0 \leq i < \infty.$$

Theorem 1.1 is an easy consequence of Theorem 2.1.

*Proof of Theorem 1.1.* We show that  $f(d, k) + 1$  is an upper bound for  $\phi(d, k, A)$  for all  $A$ . Then

$$(2.1) \quad \phi^*(d, k) \leq f(d, k) + 1.$$

To see this, suppose  $P: A^k \rightarrow A^k$  and  $Q: A^k \rightarrow A^k$  are two polynomial maps of degree  $\leq d$ , and let

$$\mathbf{x}_i = P(\mathbf{x}_{i-1}) = P^{(i)}(\mathbf{x}_0)$$

and

$$\mathbf{y}_i = Q(\mathbf{y}_{i-1}) = Q^{(i)}(\mathbf{y}_0)$$

and suppose that

$$\mathbf{x}_i = \mathbf{y}_i \quad \text{for } 0 \leq i \leq f(d, k) + 1.$$

It suffices to prove that  $\mathbf{x}_i = \mathbf{y}_i$  for all  $i \geq 0$ . Let  $S_j(X_1, \dots, X_k) = P_j(X_1, \dots, X_k) - Q_j(X_1, \dots, X_k)$  be the difference of the  $j$ th components of  $P = (P_1, \dots, P_k)$  and  $Q = (Q_1, \dots, Q_k)$ . Then  $S_j$  is a polynomial of degree  $\leq d$ , and since  $S_j(\mathbf{x}_i) = 0$  for  $0 \leq i \leq f(d, k)$ , by Theorem 2.1 applied with  $T$  equal to  $P$  we know that  $S_j(\mathbf{x}_i) = 0$  for all  $i \geq 0$ . Repeating for all  $j$  we see that  $P(\mathbf{x}_i) = Q(\mathbf{x}_i)$  for all  $i$ .  $\square$

We can in fact derive a stronger result than Theorem 1.1 from Theorem 2.1.

**COROLLARY 2.2.** *Let  $\mathbf{P}(d, k, A)$  denote the set of all sequences  $\{z_n\}$  in  $A$  arising from polynomial maps*

$$\begin{aligned} T: A^k &\rightarrow A^k, \\ U: A^k &\rightarrow A \end{aligned}$$

via

$$\begin{aligned} \mathbf{x}_n &= T(\mathbf{x}_{n-1}), \\ z_n &= U(\mathbf{x}_n) \end{aligned}$$

where both  $T$  and  $U$  have degrees at most  $d$ . Then  $\mathbf{P}(d, k, A)$  is  $f(d, 2k)$ -prefix distinguishable.

*Proof of Corollary 2.2.* Suppose  $T_1, T_2: A^k \rightarrow A^k$  and  $U_1, U_2: A^k \rightarrow A$  are such that

$$\begin{aligned} \mathbf{x}_n &= T_1(\mathbf{x}_{n-1}), & \hat{\mathbf{x}}_n &= T_2(\hat{\mathbf{x}}_{n-1}), \\ z_n &= U_1(\mathbf{x}_n), & \hat{z}_n &= U_2(\hat{\mathbf{x}}_n) \end{aligned}$$

and that

$$(2.2) \quad z_i = \hat{z}_i \quad \text{for } 0 \leq i \leq f(d, 2k).$$

We want to show that  $z_i = \hat{z}_i$  for all  $i \geq 0$ .

Define mappings

$$\begin{aligned} T^*: A^{2k} &\rightarrow A^{2k}, \\ S: A^{2k} &\rightarrow A \end{aligned}$$

by  $T^* = T_1 \times T_2$  so that

$$T^*(X_1, \dots, X_{2k}) = (T_1(X_1, \dots, X_k), T_2(X_{k+1}, \dots, X_{2k}))$$

and

$$S(X_1, \dots, X_{2k}) = U_1(X_1, \dots, X_k) - U_2(X_{k+1}, \dots, X_{2k}).$$

The hypothesis (2.2) asserts that for

$$\mathbf{x}_n^* = T^*(\mathbf{x}_{n-1}^*) = (\mathbf{x}_n, \hat{\mathbf{x}}_n)$$

one has

$$S(\mathbf{x}_i^*) = 0 \quad \text{for } 0 \leq i \leq f(d, 2k).$$

Hence, by Theorem 2.1,

$$S(\mathbf{x}_i^*) = U_1(\mathbf{x}_i) - U_2(\hat{\mathbf{x}}_i) = 0 \quad \text{for all } i \geq 0;$$

then  $z_i = \hat{z}_i$  for all  $i \geq 0$  as desired.  $\square$

We note in passing that Corollary 2.2 can be applied to “sliding window” functions of nonlinear feedback shift registers. Suppose the sequence  $\{y_n : n \geq 0\}$  of elements of  $A$  is determined by a recurrence of form  $y_n = \psi(y_{n-1}, y_{n-2}, \dots, y_{n-k})$  where  $\psi$  is a polynomial of degree  $\leq d$ , and that the sequence  $z_n$  is determined by  $z_n = U(y_n, y_{n-1}, \dots, y_{n-k+1})$ , where  $U$  is another polynomial of degree  $\leq d$ . Then  $\{z_n\}$  is a sequence in  $\mathbf{P}(d, k, A)$  and Corollary 2.2 applies, as can be seen using the standard “state vector” trick of letting  $\mathbf{x}_n = (y_n, y_{n-1}, \dots, y_{n-k+1})$ .

Now we prove Theorem 2.1 using the Hilbert basis theorem.

*Proof of Theorem 2.1.* We prove the result first for a *specific* pair of “universal maps”

$$\begin{aligned} \hat{T} : \mathbf{A}^k &\rightarrow \mathbf{A}^k, \\ \hat{S} : \mathbf{A}^k &\rightarrow \mathbf{A} \end{aligned}$$

in a “universal ring”  $\mathbf{A} = \mathbf{A}_{dk}$ . The ring  $\mathbf{A}_{dk}$  is a polynomial ring  $\mathbf{Z}[\mathbf{X}, \mathbf{T}, \mathbf{S}]$  in  $(k+1) \times \binom{k+d}{d} + k$  intermediates over  $\mathbf{Z}$ . Here  $\binom{k+d}{d}$  indeterminates are

$$\mathbf{S} = \{S_{\mathbf{i}} : \mathbf{i} = (i_1, \dots, i_k) \text{ with } i_1 + \dots + i_k \leq d \text{ and all } i_j \geq 0\}$$

and  $k \binom{k+d}{d}$  indeterminates are

$$\mathbf{T} = \{T_{j\mathbf{i}} : 1 \leq j \leq k \text{ and } \mathbf{i} = (i_1, \dots, i_k) \text{ with } i_1 + \dots + i_k \leq d \text{ and all } i_j \geq 0\}$$

and  $k$  more indeterminates are

$$\mathbf{X} = \{X_i : 1 \leq i \leq k\}.$$

We think of the indeterminates  $\mathbf{X}$  as being *symbolic* or *generic* versions of the coordinates of the initial vector  $\mathbf{x}_0$  of the recurrence, and the  $\mathbf{T}$  and  $\mathbf{S}$  indeterminates as generic versions of the coefficients of the polynomial maps  $T$  and  $S$ . The universal polynomial map

$$\hat{T} = (\hat{T}_1, \dots, \hat{T}_k) : \mathbf{A}^k \rightarrow \mathbf{A}^k$$

of degree  $d$  is defined by

$$\hat{T}_j(Z_1, \dots, Z_k) = \sum_{\substack{\mathbf{i} = (i_1, \dots, i_k) \\ i_1 + \dots + i_k \leq d}} T_{j\mathbf{i}} Z_1^{i_1} \cdots Z_k^{i_k}$$

for  $1 \leq j \leq k$ , and the universal polynomial map

$$\hat{S} : \mathbf{A}^k \rightarrow \mathbf{A}$$

by

$$\hat{S}(Z_1, \dots, Z_k) = \sum_{\substack{\mathbf{i} = (i_1, \dots, i_k) \\ i_1 + \dots + i_k \leq d}} S_{\mathbf{i}} Z_1^{i_1} \cdots Z_k^{i_k}.$$

Now we define a series  $\{P_i\}$  of elements of  $\mathbf{A}$  by

$$\begin{aligned} P_0 &= \hat{S}((X_1, \dots, X_k)), \\ P_1 &= \hat{S}(\hat{T}(X_1, \dots, X_k)), \\ P_2 &= \hat{S}(\hat{T}(\hat{T}(X_1, \dots, X_k))), \\ &\dots \end{aligned} \tag{2.3}$$

Here we use the fact that the functional composition of polynomials is itself a polynomial in the coefficients of the original polynomials. We use these to define an ascending chain  $I_0 \subseteq I_1 \subseteq \dots$  of ideals in  $\mathbf{A}$  by

$$\begin{aligned} I_0 &= (P_0), \\ I_1 &= (P_0, P_1), \\ I_2 &= (P_0, P_1, P_2), \\ &\dots \end{aligned}$$

By Hilbert’s basis theorem [L, pp.144-145] the polynomial ring  $\mathbf{A} = \mathbf{Z}[\mathbf{X}, \mathbf{T}, \mathbf{S}]$  is Noetherian, so this ascending chain of ideals must eventually stabilize: For some finite  $n = f(d, k)$ , we have that

$$I_n = I_{n+1} = I_{n+2} = \dots$$

The relation  $I_n = I_{n+1}$  is equivalent to

$$P_{n+1} \in (P_0, P_1, \dots, P_n),$$

which is

$$(2.4) \quad \hat{S}(\hat{T}^{(n+1)}(\mathbf{X})) = \sum_{j=0}^n A_j(\mathbf{X}, \mathbf{T}, \mathbf{S}) \hat{S}(\hat{T}^{(j)}(\mathbf{X}))$$

for some polynomials  $A_j(\mathbf{X}, \mathbf{T}, \mathbf{S}) \in \mathbf{A}$ . This is a formal identity in the polynomial ring  $\mathbf{A}$ . Note that (2.3) gives

$$P_{j+1}(\mathbf{X}, \mathbf{T}, \mathbf{S}) = P_j(\hat{T}(\mathbf{X}), \mathbf{T}, \mathbf{S})$$

so that (2.4) implies that

$$\hat{S}(\hat{T}^{n+m+1}(\mathbf{X})) = \sum_{j=0}^n A_j(\hat{T}^{(m)}(\mathbf{X}), \mathbf{T}, \mathbf{S}) \hat{S}(\hat{T}^{(m+j)}(\mathbf{X})).$$

Also, since the  $P_i(\mathbf{X}, \mathbf{T}, \mathbf{S}) = \hat{S}(\hat{T}^{(i)}(\mathbf{X}))$  are homogeneous of degree one in the variables  $\{S_i\}$  by (2.3), it follows that (2.4) still holds if we drop all terms in  $A_j(\mathbf{X}, \mathbf{T}, \mathbf{S})$  depending on the variables  $\{S_i\}$ , so that we have the existence of a *universal identity* of the form

$$(2.5) \quad \hat{S}(\hat{T}^{(n+1)}(\mathbf{X})) = \sum_{j=0}^n A_j(\mathbf{X}, \mathbf{T}) \hat{S}(\hat{T}^{(j)}(\mathbf{X}))$$

valid in the ring  $\mathbf{A}_{dk} = \mathbf{Z}[\mathbf{X}, \mathbf{T}, \mathbf{S}]$ , where

$$A_j(\mathbf{X}, \mathbf{T}) \in \mathbf{Z}[\mathbf{X}, \mathbf{T}].$$

Now suppose we are given a ring  $A$  and polynomial maps  $T: A^k \rightarrow A^k$  and  $S: A^k \rightarrow A$  of degree  $\leq d$ . We apply (2.5) as follows. Define a homomorphism  $\rho: \mathbf{A}_{dk} \rightarrow A$  by mapping

- $\rho(1) = 1_A$  the identity element of  $A$ ,
- $\rho(T_{ji}) = t_{ji}$  the corresponding coefficient of  $T_j$  where  $T = (T_1, \dots, T_k)$ ,
- $\rho(S_i) = s_i$  the corresponding coefficient of  $S$ ,
- $\rho(X_i) = x_{0i} = i$ th coordinate of initial value  $\mathbf{x}_0$ .

Under this homomorphism

$$\rho(\hat{S}(\hat{T}^{(j)}(\mathbf{X}))) = S(T^{(j)}(\mathbf{x}_0)) = S(\mathbf{x}_j).$$

Set  $\mathbf{t} = \{t_{ji} : 0 \leq j \leq d \text{ and } \mathbf{i} = (i_1, \dots, i_k) \text{ with } i_1 + \dots + i_k \leq d \text{ and all } i_k \geq 0\}$ . Applying  $\rho$  to (2.5) gives the identity in  $A$  that

$$S(\mathbf{x}_{n+1}) = \sum_{j=0}^n A_j(\mathbf{x}_0, \mathbf{t})S(\mathbf{x}_j).$$

Now since  $S(\mathbf{x}_j) = 0$  for  $0 \leq j \leq n = f(d, k)$  by hypothesis, this equation implies

$$S(\mathbf{x}_{n+1}) = 0.$$

By induction on  $j$  it follows that

$$S(\mathbf{x}_{n+j}) = 0, \quad j = 1, 2, 3, \dots$$

using the hypothesis that  $\{S(\mathbf{x}_{n+j-i-1}) = 0 : 0 \leq i \leq f(d, k)\}$ .  $\square$

**3. Exact bounds for  $\phi^*(d, k)$  in the cases  $d = 1$  and  $k = 1$ .** We can obtain upper bounds for  $\phi^*(d, k)$  via upper bounds for  $f(d, k)$ , from

$$(3.1) \quad \phi^*(d, k) \leq f(d, k) + 1$$

obtained in the course of proving Theorem 1.1. To get upper bounds for  $f(d, k)$  we observe that any identity of the form

$$(3.2) \quad \hat{S}(\hat{T}^{(n+1)}(\mathbf{X})) = \sum_{j=0}^n A_j(\mathbf{X}, \mathbf{T})\hat{S}(\hat{T}^{(j)}(\mathbf{X}))$$

in the polynomial ring  $\mathbf{A}_{dk} = \mathbf{Z}[\mathbf{X}, \mathbf{T}, \mathbf{S}]$  implies that  $f(d, k) \leq n$ . Hence finding an explicit identity of form (3.2) yields an upper bound for  $f(d, k)$ .

We can obtain lower bounds for  $\phi^*(d, k)$  by explicit construction. If we find a ring  $A$  and two orbits in  $S(d, k, A)$  that agree for  $n$  consecutive values and then disagree, then we may conclude that  $\phi^*(d, k) \geq \phi(d, k, A) \geq n$ .

These methods can determine  $\phi^*(d, k)$  exactly only for those values of  $d$  and  $k$  for which the equality

$$(3.3) \quad \phi^*(d, k) = f(d, k) + 1$$

holds. The proof of Theorem 1.1 leaves open the possibility that  $\phi^*(d, k) < f(d, k) + 1$  can occur for some values of  $k$  and  $d$ . This is so because the universal identities found in Theorem 2.1 actually give the smallest value  $f(d, k)$  such that every test polynomial  $S$  of degree of at most  $d$  has the property that if  $S(\mathbf{x}_i) = 0$  for  $0 \leq i \leq f(d, k)$  for the iterates of a recurrence of degree  $\leq d$ , then  $S(\mathbf{x}_i) = 0$  for all  $i \geq 0$ , while the bound for  $\phi^*(d, k)$  used in the proof of Theorem 1.1 only requires that this property hold for those particular test polynomials  $S$  of the form  $S = T_1 - T_2$  where  $T_1$  and  $T_2$  produce the given set of iterates observed. For a particular set of iterates the set of such  $S$  will in general be of much lower dimension than the set of all  $S$ .

We prove that (3.3) does hold in the cases  $d = 1$  and  $k = 1$ , and we obtain exact formulae for  $\phi^*(d, k)$  in these cases. We begin with the case  $d = 1$ , and prove the following result, which implies Theorem 1.2.

**THEOREM 3.1.** For all dimensions  $k \geq 1$ ,

$$\phi^*(1, k) = f(1, k) + 1 = k + 1.$$

*Proof. Upper bound.* We first show that  $f(1, k) \leq k$  by explicit construction of a suitable identity (3.2), which turns out to be the Cayley–Hamilton theorem.

In the case  $d = 1$ , we have that

$$\hat{T}(\mathbf{x}) = (\hat{T}_1(\mathbf{x}), \dots, \hat{T}_k(\mathbf{x}))$$

where

$$\hat{T}_i(x_1, \dots, x_k) = t_{i0} + \sum_{j=1}^k t_{ij}x_j$$

for  $1 \leq i \leq k$ . We let  $\mathbf{x} = (1, x_1, \dots, x_k)^T$  and observe that

$$(3.4) \quad (1, \hat{T}(\mathbf{x}))^T = L\mathbf{x}$$

where  $L$  is the  $(k+1) \times (k+1)$  matrix with rows

$$\begin{aligned} \mathbf{L}_0 &= (1, 0, \dots, 0), \\ \mathbf{L}_i &= (t_{i0}, t_{i1}, \dots, t_{ik}), \quad 1 \leq i \leq k. \end{aligned}$$

For an arbitrary  $k+1$  by  $k+1$  matrix  $M$  over any commutative ring with unit  $A$ , the characteristic polynomial  $\chi_M(z)$  of  $M$  is

$$(3.5) \quad \chi_M(z) = \det(zI - M) = \sum_{j=0}^{k+1} c_j(M)z^j$$

where the  $c_j(M)$  are polynomial functions of the entries of  $M$ , with integer coefficients, and  $c_{k+1}(M) = 1$ . The Cayley-Hamilton theorem [L, p. 400] asserts that the matrix  $\chi_M(M)$  is the zero matrix, i.e.,

$$(3.6) \quad M^{k+1} = -\sum_{j=0}^k c_j(M)M^j.$$

In particular, we obtain

$$(3.7) \quad L^{k+1}\mathbf{x} = -\sum_{j=0}^k c_j(L)L^j\mathbf{x}.$$

Finally, let

$$(3.8) \quad \hat{S}(x_1, \dots, x_k) = s_0 + \sum_{i=1}^k s_i x_i = \mathbf{s}\mathbf{x}$$

where  $\mathbf{s} = (s_0, s_1, \dots, s_k)$  is a row vector of indeterminates. Then (3.7) gives the identity

$$(3.9) \quad \mathbf{s}L^{k+1}\mathbf{x} = -\sum_{j=0}^k c_j(L)\mathbf{s}L^j\mathbf{x}.$$

This is exactly an identity of the form (3.2) that we are looking for in the ring

$$\mathbf{A}_{1,k} = \mathbf{Z}[x_1, \dots, x_k; s_0, \dots, s_k; t_0, \dots, t_k]$$

since

$$\hat{\mathbf{S}}(\hat{T}^{(i)}(x_1, \dots, x_k)) = \mathbf{S}L^i\mathbf{x}$$

using (3.4) and (3.8), where

$$A_j(\mathbf{x}, \mathbf{T}) = A_j(\mathbf{T}) = -c_j(L).$$

The identity (3.9) implies that

$$f(1, k) \leq k,$$

as required.

*Lower bound.* For the bound  $\phi^*(1, k) \geq k + 1$  we take  $A = \mathbf{Q}$  and the transformations

$$T(x_1, \dots, x_k) = (x_k + 1, x_1, x_2, \dots, x_{k-1})$$

and

$$U(x_1, \dots, x_k) = (2x_k + 1, x_1, x_2, \dots, x_{k-1}).$$

Then for  $\mathbf{x}_0 = (0, 0, \dots, 0)$  and  $0 \leq i \leq k$ , we have that

$$T^{(i)}(\mathbf{x}_0) = U^{(i)}(\mathbf{x}_0) = (1, 1, \dots, 1, 0, \dots, 0)$$

with the 1's in the first  $i$  places. But  $T^{(k+1)}(\mathbf{x}_0) = (2, 1, 1, \dots, 1)$  and  $U^{(k+1)}(\mathbf{x}_0) = (3, 1, 1, \dots, 1)$ , so that  $\phi^*(1, k) \geq k + 1$ .  $\square$

We derive Theorem 1.3 from the following result.

**THEOREM 3.2.** *For all degrees  $d \geq 1$ ,*

$$\phi^*(d, 1) = f(d, 1) + 1 = d + 1.$$

*Proof. Upper bound.* To obtain the upper bound  $f(d, 1) \leq d$  we will show that in the polynomial ring  $\mathbf{A}_{d1} = \mathbf{Z}[X, T_0, T_1, \dots, T_d, S_0, S_1, \dots, S_d]$  there exists an identity of the form

$$(3.10) \quad \hat{S}(\hat{T}^{(d+1)}(X)) = \sum_{i=0}^d A_i(X, T) \hat{S}(\hat{T}^{(i)}(X))$$

with  $A_i(X, T) \in \mathbf{Z}[X, T_0, \dots, T_d]$ . Recall that the universal maps  $\hat{T}$  and  $\hat{S}$  are given by

$$\hat{T}(X) = \sum_{j=0}^d T_j X^j, \quad \hat{S}(X) = \sum_{j=0}^d S_j X^j;$$

hence,

$$\hat{S}(\hat{T}^{(i)}(X)) = \sum_{j=0}^d S_j [\hat{T}^{(i)}(X)]^j.$$

By the Lagrange interpolation formula

$$\hat{S}(\hat{T}^{(d+1)}(X)) = \sum_{i=0}^d A_i(X, T) \hat{S}(\hat{T}^{(i)}(X))^j$$

where

$$A_i(X, T) = \prod_{\substack{j=0 \\ j \neq i}}^d \left( \frac{\hat{T}^{(d+1)}(X) - \hat{T}^{(j)}(X)}{\hat{T}^{(i)}(X) - \hat{T}^{(j)}(X)} \right).$$

To establish the upper bound it suffices to show that the rational functions  $A_i(X, T)$  are actually polynomials in the ring  $\mathbf{Z}[X, T]$ . To prove this we work with certain non-Archimedean norms on  $\mathbf{Z}[X, T]$  and on its quotient field  $\mathbf{F} = \mathbf{Q}(X, T)$ . Since  $\mathbf{Z}$  is a unique factorization domain, so is  $\mathbf{Z}[X, T]$ . Let  $\mathbf{P}$  be a set of representatives of the irreducible elements of  $\mathbf{Z}[X, T]$  modulo the units of  $\mathbf{Z}[X, T]$ . Then any element  $f \in \mathbf{F}$  can be written uniquely as  $f = u \prod_{p \in \mathbf{P}} p^{\nu_p(f)}$  where  $u$  is a unit in  $\mathbf{Z}[X, T]$ . Define the  $p$ -norm of  $f$  by  $|f|_p = e^{-\nu_p(f)}$ . Then

$$\mathbf{Z}[X, T] = \{f \in \mathbf{F} : |f|_p \leq 1 \text{ for all } p \in \mathbf{P}\}.$$

(See [ZS, Ex. 2, p. 38]). Our method for showing that  $A_j(X, T) \in \mathbf{Z}[X, T]$  is to show that  $|A_j(X, T)|_p \leq 1$  for all  $p \in \mathbf{P}$ .



This follows from a general result about non-Archimedean norms, which contains the main idea of the proof, presented as Proposition 3.3 below. First we show that the sequence of iterates  $\hat{T}^{(0)}(X), \hat{T}^{(1)}(X), \dots$  is a contracting sequence for the norms  $|\cdot|_p$ , where a sequence  $x_0, x_1, \dots$  is said to be a *contracting sequence* if  $|x_{i+1} - x_{j+1}| \leq |x_i - x_j|$  for all  $i$  and  $j$ . We then apply the following proposition.

PROPOSITION 3.3. *Let  $\mathbf{F}$  be a field with non-Archimedean norm  $|\cdot|$ , and let  $x_0, x_1, \dots$  be a contracting sequence. Assume that the  $x_i$  are distinct.*

(1) *Let*

$$\theta_i(x_0, \dots, x_{d+1}) = \prod_{\substack{j=0 \\ j \neq i}}^d \left( \frac{x_{d+1} - x_j}{x_i - x_j} \right)$$

for  $0 \leq i \leq d$ . Then  $|\theta_i(x_0, \dots, x_{d+1})| \leq 1$ .

(2) *If  $S \in \mathbf{F}[X]$  has degree  $\leq d$ , then*

$$|S(x_{d+1})| \leq \max_{0 \leq i \leq d} |S(x_i)|.$$

(The proof of Proposition 3.3 is deferred until after Theorem 3.2 is proved.) Since  $A_j(X, \mathbf{T}) = \theta_j(\hat{T}^{(0)}(X), \dots, \hat{T}^{(d+1)}(X))$  we see that  $|A_j(X, \mathbf{T})|_p \leq 1$  as desired, and hence  $A_j(X, \mathbf{T}) \in \mathbf{Z}[X, \mathbf{T}]$ .

So to complete the proof of the upper bound it suffices to check that  $\{\hat{T}^{(i)}(X)\}$  is a contracting sequence for each  $|\cdot|_p$ . To see this we start from the polynomial identity

$$\begin{aligned} \hat{T}^{(1)}(V) &= \sum_{j=0}^d T_j V^j = \sum_{j=0}^d T_j (U + (V - U))^j \\ &= \hat{T}^{(1)}(U) + g(\mathbf{T}, U, V)(V - U) \end{aligned}$$

where  $U$  and  $V$  are indeterminates and  $g$  is a polynomial with integer coefficients. Substituting  $U = \hat{T}^{(i)}(X)$  and  $V = \hat{T}^{(j)}(X)$ , we have that

$$\hat{T}^{(i+1)}(X) - \hat{T}^{(j+1)}(X) = g(\mathbf{T}, \hat{T}^{(i)}(X), \hat{T}^{(j)}(X))(\hat{T}^{(i)}(X) - \hat{T}^{(j)}(X))$$

so that

$$|\hat{T}^{(i+1)}(X) - \hat{T}^{(j+1)}(X)|_p = |g(\mathbf{T}, \hat{T}^{(i)}(X), \hat{T}^{(j)}(X))|_p |\hat{T}^{(i)}(X) - \hat{T}^{(j)}(X)|_p.$$

Since  $\hat{T}^{(i)}(X)$  and  $\hat{T}^{(j)}(X)$  are in  $\mathbf{Z}[X, \mathbf{T}]$ , and since  $g$  has integer coefficients, the norm of the  $g$  factor is bounded by 1 and the contracting sequence property is verified.

Then by Proposition 3.3 we know that  $|A_j(X, \mathbf{T})|_p \leq 1$ . Repeating this argument for all irreducibles  $p$ , we know that  $A_j(X, \mathbf{T})$  is in  $\{f \in \mathbf{F}: |f|_p \leq 1 \text{ for all } p\}$  which we have already seen is  $\mathbf{Z}[X, \mathbf{T}]$ . This establishes the upper bound.

*Lower bound.* We let  $A = \mathbf{Q}$ . Take  $T(X) = X + 1$  and starting value  $x_0 = 0$ . Let

$$U(X) = 1 + X + \prod_{j=0}^{d-1} \left( \frac{X - j}{d - j} \right),$$

so that  $U(l) = l + 1$  for  $0 \leq l \leq d - 1$  and  $U(d) = d + 2$ . Then  $T^{(j)}(0) = U^{(j)}(0) = j$  for  $0 \leq j \leq d$ , but  $T^{(d+1)}(0) \neq U^{(d+1)}(0)$ . Hence  $\phi^*(d, 1) \geq d + 1$ .  $\square$

It remains to prove Proposition 3.3.

*Proof of Proposition 3.3.* Statement (2) follows from (1) by the Lagrange Interpolation Theorem and from the ultrametric inequality.

To prove (1), we first note that

$$(3.11) \quad |x_{i+k} - x_{j+k}| \leq |x_i - x_j|$$

holds for all  $i, j, k \geq 0$ . Now we write

$$|\theta_i(x_0, \dots, x_{d+1})| = A_{di}(x_0, \dots, x_{d+1})B_{di}(x_0, \dots, x_{d+1})$$

where

$$A_{di}(x_0, \dots, x_{d+1}) = \prod_{j=i+1}^k \left| \frac{x_{d+1} - x_j}{x_i - x_j} \right|,$$

$$B_{di}(x_0, \dots, x_{d+1}) = \prod_{j=0}^{i-1} \left| \frac{x_{d+1} - x_j}{x_i - x_j} \right|,$$

where in both cases empty products are taken to be one. In fact each of  $A_{di}(x_0, \dots, x_{d+1})$  and  $B_{di}(x_0, \dots, x_{d+1})$  are individually less than 1. We show  $A_{di}(x_0, \dots, x_{d+1}) \leq 1$  directly, and  $B_{di}(x_0, \dots, x_{d+1}) \leq 1$  by induction on  $d$ .

First we show that  $A_{di}(x_0, \dots, x_{d+1}) \leq 1$ . We have, on multiplying out and on permuting the factors in the denominator,

$$\begin{aligned} A_{di}(x_0, \dots, x_{d+1}) &= \prod_{j=i+1}^d \left| \frac{x_{d+1} - x_j}{x_i - x_j} \right| \\ &= \left| \frac{x_{d+1} - x_{i+1}}{x_d - x_i} \right| \left| \frac{x_{d+1} - x_{i+2}}{x_{d-1} - x_i} \right| \dots \left| \frac{x_{d+1} - x_d}{x_{i+1} - x_i} \right| \\ &\leq 1, \end{aligned}$$

where each of the  $d - i$  factors has norm less than or equal to one by (3.11).

Now we go to work on  $B_{di}(x_0, \dots, x_{d+1})$ . Suppose, as the induction hypothesis, that for all  $0 \leq i \leq d - 1$  and for all contracting sequences of distinct elements  $y_0, y_1, \dots$  we have  $B_{d-1,i}(y_0, \dots, y_d) \leq 1$ . By the ultrametric inequality either  $|x_{d+1} - x_0|$  is less than or equal to  $|x_i - x_0|$  or is less than or equal to  $|x_{d+1} - x_i|$  or both. In the first case

$$\begin{aligned} B_{di}(x_0, \dots, x_{d+1}) &\leq \prod_{j=1}^{i-1} \left| \frac{x_{d+1} - x_j}{x_i - x_j} \right| \\ &= B_{d-1,i-1}(x_1, \dots, x_{d+1}) \leq 1 \end{aligned}$$

by the induction hypothesis.

In the second case

$$\begin{aligned} B_{di}(x_0, \dots, x_{d+1}) &= \left| \frac{x_{d+1} - x_0}{x_i - x_0} \right| \left| \frac{x_{d+1} - x_1}{x_i - x_1} \right| \dots \left| \frac{x_{d+1} - x_{i-1}}{x_i - x_{i-1}} \right| \\ &\leq \left| \frac{x_{d+1} - x_i}{x_i - x_0} \right| \left| \frac{x_{d+1} - x_1}{x_i - x_1} \right| \dots \left| \frac{x_{d+1} - x_{i-1}}{x_i - x_{i-1}} \right|. \end{aligned}$$

By cyclicly permuting the factors of the numerator, we obtain

$$\begin{aligned} B_{di}(x_0, \dots, x_{d+1}) &\leq \left| \frac{x_{d+1} - x_1}{x_i - x_0} \right| \left| \frac{x_{d+1} - x_2}{x_i - x_1} \right| \dots \left| \frac{x_{d+1} - x_i}{x_i - x_{i-1}} \right| \\ &\leq \left| \frac{x_d - x_0}{x_i - x_0} \right| \left| \frac{x_d - x_1}{x_i - x_1} \right| \dots \left| \frac{x_d - x_{i-1}}{x_i - x_{i-1}} \right| \\ &= B_{d-1,i}(x_0, \dots, x_d) \leq 1, \end{aligned}$$

using the contraction property and the induction hypothesis.

It is easy to check that the induction can be started at say,  $d = 0$  or at  $d = 1$ .  $\square$

**4. Lower bounds for  $\phi^*(d, k)$ .** Let  $\mathbf{M}(d, k, l, A)$  denote the set of all polynomial maps  $T: A^k \rightarrow A^l$  of degree at most  $d$ . In the case that  $\mathbf{F}$  is an algebraically closed field, we will obtain the lower bound  $\phi(d, k, \mathbf{F}) \cong \binom{d+k}{k}$  by showing that most recurrences  $T$  in  $\mathbf{M}(d, k, k, \mathbf{F})$  are determined uniquely by their iterates  $\{T^{(i)}(x_0): 0 \leq i \leq \binom{d+k}{k}\}$  and by no fewer than this number of iterates. We do this by relating the extrapolation problem to the problem of interpolation of polynomials. A polynomial  $P(\mathbf{X})$  of degree  $\leq d$  in  $k$  variables has  $\binom{d+k}{k}$  coefficients, so that at least  $\binom{d+k}{k}$  distinct interpolation points are needed to determine it uniquely. For dimension  $k = 1$ , unique interpolation is possible for any set of  $d + 1$  distinct points by Lagrange’s interpolation formula, but for higher dimensions this is no longer true. We set  $N = \binom{d+k}{k}$  and call a set

$$S = \{\mathbf{x}_i \in \mathbf{F}^k: 1 \leq i \leq N\}$$

a *unique interpolation set* for  $\mathbf{M}(d, k, l, \mathbf{F})$  if for every sequence  $\mathbf{z}_i \in \mathbf{F}^l, 1 \leq i \leq N$  there is a unique polynomial map  $P(\mathbf{X})$  in  $\mathbf{M}(d, k, l, \mathbf{F})$  for which

$$(4.1) \quad P(\mathbf{x}_i) = \mathbf{z}_i \quad \text{for } 1 \leq i \leq N.$$

Note that  $S$  is a unique interpolation set for  $\mathbf{M}(d, k, l, \mathbf{F})$  if and only if it is a unique interpolation set for  $\mathbf{M}(d, k, 1, \mathbf{F})$ .

**THEOREM 4.1.** *Let  $\mathbf{F}$  be an algebraically closed field, and let  $N = \binom{d+k}{k}$ .*

(1) *The set  $Z_{dk} = \{S: S \text{ is a unique interpolation set for } \mathbf{M}(d, k, k, \mathbf{F})\}$  is an open dense subset of  $\mathbf{F}^{kN}$  in the Zariski topology.*

(2) *The sets of values  $S = \{\mathbf{x}_i \in \mathbf{F}^k: 0 \leq i \leq N\}$  in  $\mathbf{F}^{kN+k}$  such that there is a unique recurrence  $T \in \mathbf{M}(d, k, k, \mathbf{F})$  with*

$$(4.2) \quad T(\mathbf{x}_i) = \mathbf{x}_{i+1} \quad \text{for } 0 \leq i \leq N - 1$$

*contains the Zariski open subset  $Z_{dk} \times \mathbf{F}^k$  of  $\mathbf{F}^{kN+k}$ .*

(3)  $\phi(d, k, \mathbf{F}) \cong N$ .

*Proof.* (1) A point fails to be in  $Z_{dk}$  only if the linear equations (4.1) that the coefficients of  $P$  obey have deficient rank, i.e., a certain determinant vanishes. The coefficients of the system (4.1) are polynomial functions of the  $\mathbf{x}_i$ , and hence  $Z_{dk}$  is the complement of a closed set.

(2) We claim that if  $S = \{\mathbf{x}_i: 0 \leq i \leq N - 1\}$  is a unique interpolation set, then for each  $\mathbf{x}_N \in \mathbf{F}^k$  a unique recurrence  $T$  in  $\mathbf{M}(d, k, k, \mathbf{F})$  exists satisfying (4.2). Write  $T = (T_1, \dots, T_k)$  and observe that (4.2) is equivalent to

$$(4.3) \quad T_j(x_{1i}, \dots, x_{ki}) = x_{j,i+1}, \quad 0 \leq i \leq N - 1,$$

for  $1 \leq j \leq k$ . If  $S$  is a unique interpolation set, then by definition there exists a unique  $T_j$  satisfying (4.3), for each  $j$ . Hence  $T$  exists and is unique, so that  $Z_{dk} \times \mathbf{F}^k$  is contained in the desired set of values.

(3) Choose a unique interpolation set  $S = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{N-1})$  with  $N = \binom{d+k}{k}$ . Then by (2) we may choose two different vectors  $\mathbf{x}_N \neq \mathbf{x}_N^*$  in  $\mathbf{F}^k$  and find unique recurrences  $T, T^*$  in  $\mathbf{M}(d, k, k, \mathbf{F})$  with

$$T(\mathbf{x}_i) = T^*(\mathbf{x}_i) = \mathbf{x}_{i+1} \quad \text{for } 0 \leq i \leq N - 1,$$

while

$$T(\mathbf{x}_{N-1}) = \mathbf{x}_N \neq \mathbf{x}_N^* = T^*(\mathbf{x}_{N-1}).$$

Hence  $\phi(d, k, \mathbf{F}) \cong N = \binom{d+k}{k}$ .  $\square$

The problem of unique extrapolation of polynomial recurrences is in fact more complicated than the “generic” behavior of polynomial recurrences given by Theorem

4.1 suggests. The worst cases appear to be attained by orbits of recurrences that do not form unique interpolation sets. We are indebted to Dr. James Butler for the following example with  $d = 2$  and  $k = 2$ , where the generic bound is  $\binom{2+2}{2} = 6$ .

PROPOSITION 4.2.  $\phi(2, 2, \mathbf{Q}) \geq 7$ .

*Proof.* We consider the collection of transformations

$$T_{\lambda\mu}(x, y) = (-xy + y^2 + x - 2y + \lambda(x^2 - 1), -1/2xy + 1/2y^2 + x + \mu(x^2 - 1)),$$

parametrized by the constants  $\lambda, \mu$  in  $\mathbf{Q}$ . All of these transformations agree on the set

$$V = \{(x, y) : x = \pm 1\} = \{(x, y) : x^2 - 1 = 0\}.$$

It is easy to check that

$$T_{\lambda\mu}(\mathbf{x}_i) = \mathbf{x}_{i+1} \quad \text{for } 0 \leq i \leq 5,$$

with  $\mathbf{x}_0 = (1, 0)$ ,  $\mathbf{x}_1 = (1, 1)$ ,  $\mathbf{x}_2 = (-1, 1)$ ,  $\mathbf{x}_3 = (-1, 0)$ ,  $\mathbf{x}_4 = (-1, -1)$ ,  $\mathbf{x}_5 = (1, -1)$ ,  $\mathbf{x}_6 = (5, 2)$ , because the points  $\mathbf{x}_i$  for  $0 \leq i \leq 5$  all lie on the set  $V$ . However,

$$T_{\lambda\mu}(\mathbf{x}_6) = (-5 + 24\lambda, 2 + 24\mu)$$

depends on  $\lambda$  and  $\mu$ . Hence the  $T_{\lambda\mu}$  agree for seven consecutive values and then disagree.  $\square$

In this case the set  $\{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$  is not a unique interpolation set in  $\mathbf{M}(2, 2, 1, \mathbf{Q})$ .

**5. Upper bounds for  $\phi(d, k, \mathbf{F})$  for general  $d$  and  $k$ .** We apply methods from algebraic geometry to obtain upper bounds for  $\phi(d, k, \mathbf{F})$  when  $\mathbf{F}$  is a field. We assume without loss of generality that  $\mathbf{F}$  is algebraically closed. Let  $\mathbf{F}^k$  be affine  $k$ -space over  $\mathbf{F}$ , equipped with the Zariski topology. (A collection of basic open sets for the Zariski topology consists of those sets of points where a polynomial  $p(x_1, \dots, x_k) \neq 0$ .) We freely use the concepts of dimension and irredundant decomposition into irreducible closed sets, as expounded by [Fu], [H] and [Sh]. The (Zariski) closure of a set  $S \subseteq \mathbf{F}^k$  is denoted by  $\bar{S}$ . The cardinality of  $S$  is denoted by  $|S|$ .

The *degree*  $\deg(W)$  of a closed set is the sum of the degrees of its irreducible components, which in turn are defined as follows. If  $V$  is irreducible, its degree is the maximum cardinality of a finite intersection with a linear closed set:

$$\deg(V) = \max \{|V \cap L| : L \text{ linear and } |V \cap L| < \infty\}.$$

It can be shown that every closed set has finite degree, and for “most” linear closed sets  $L$  of dimension  $k - \dim(V)$ ,  $|V \cap L| = \deg V$ . The degree of a linear closed set is one. The degree of a singleton point is one. The degree of a hypersurface defined by a single polynomial equation of degree  $d$  is at most  $d$ . See [H, Chap. I, § 7] or [S] for details.

Our main tool is a generalization of Bezout’s theorem.

THEOREM 5.1 (Bezout’s Inequality). *Let  $V, W \subseteq \mathbf{F}^k$  be closed sets. Then*

$$\deg(V \cap W) \leq \deg(V) \deg(W).$$

This result is due to Fulton (see [FL]).

Let  $T : \mathbf{F}^k \rightarrow \mathbf{F}^k$  be a polynomial map. We need a series of elementary results relating the degrees of  $T, V$  and  $T(V)$  for closed sets  $V$ .

LEMMA 5.2. *Let the graph of  $T$  be*

$$\Gamma_T = \{(\mathbf{x}, T(\mathbf{x})) \in \mathbf{F}^{2k} : \mathbf{x} \in \mathbf{F}^k\} \subseteq \mathbf{F}^{2k}.$$

*Then  $\deg(\Gamma_T) \leq (\deg T)^k$ .*

*Proof.*  $\Gamma_T = \bigcap_{i=1}^k W_i$  where  $W_i$  is the hypersurface in  $\mathbf{F}^{2k}$  defined by the equation  $T_i(X_1, \dots, X_k) = X_{k+i}$ . So  $W_i$  has degree at most  $\deg(T)$ , and by Bezout's inequality,  $\deg(\bigcap_{i=1}^k W_i) \leq (\deg(T))^k$ .  $\square$

LEMMA 5.3. *Let  $S_1, S_2 \subseteq \mathbf{F}^k$  be closed. Then*

$$\deg(S_1 \times S_2) \leq \deg(S_1) \deg(S_2).$$

*Proof.* We may suppose that  $S_2 = \mathbf{F}^k$  for  $S_1 \times S_2 = (S_1 \times \mathbf{F}^k) \cap (\mathbf{F}^k \times S_2)$  and so by Bezout's inequality  $\deg(S_1 \times S_2) \leq \deg(S_1 \times \mathbf{F}^k) \deg(\mathbf{F}^k \times S_2)$ . Let  $D = \deg(S_1 \times \mathbf{F}^k)$ , let  $L$  be a closed linear set such that  $|L \cap (S_1 \times \mathbf{F}^k)| = D$ , and for  $s \in S_1$  let  $n(s) = |L \cap \{s\} \times \mathbf{F}^k|$  so that  $D = \sum_{s \in S_1} n(s)$ . Since  $\mathbf{F}$  is algebraically closed it is infinite and, by linearity, if  $n(s) > 1$ , then  $n(s) = \infty$ . But  $D < \infty$ , so  $n(s) \leq 1$  for all  $s \in S_1$ . Then  $n(s) = |L' \cap \{s\}|$  where  $L' = \{x \in \mathbf{F}^k : (x, y) \in L\}$  is a linear subspace of  $\mathbf{F}^k$ , so  $D = \sum n(s) = |L' \cap S_1| \leq \deg(S_1)$ .  $\square$

LEMMA 5.4. *Let  $p: \mathbf{F}^{2k} \rightarrow \mathbf{F}^k$  be the projection onto the first  $k$  coordinates. If  $S \subseteq \mathbf{F}^{2k}$  is closed, then*

$$\deg(\overline{p(S)}) \leq \deg(S).$$

*Proof.* Using irredundant decomposition into irreducibles, it suffices to prove this for irreducible  $S$ . Let  $V = \overline{p(S)}$ . Let  $L$  be a linear closed set such that  $|L \cap V| = \deg(V)$ . Then by the definition of the projection  $p$

$$(p^{-1}(L \cap V)) \cap S = (L \times \mathbf{F}^k) \cap S.$$

Since  $L \cap V$  is finite,  $p^{-1}(L \cap V)$  consists of  $\deg(V)$  many linear closed sets, each one of which has nonempty intersection with  $S$ . Thus  $\deg((L \times \mathbf{F}^k) \cap S)$  is the sum of  $\deg(V)$  positive integers so  $\deg(V) \leq \deg((L \times \mathbf{F}^k) \cap S)$ . By Bezout's inequality  $\deg((L \times \mathbf{F}^k) \cap S) \leq \deg(L \times \mathbf{F}^k) \cdot \deg(S) = \deg(S)$ . Hence  $\deg(V) \leq \deg(S)$ .  $\square$

LEMMA 5.5.  $\deg(\overline{T(V)}) \leq \deg(\Gamma_T) \deg(V) \leq (\deg T)^k \deg(V)$ .

*Proof.* Use  $\overline{T(V)} = \overline{p(W)}$  where  $W = \Gamma_T \cap (V \times \mathbf{F}^k)$ . The result then follows from Lemmas 5.2, 5.3 and 5.4.  $\square$

LEMMA 5.6. *Let  $V$  be a closed set. If  $V \subseteq \overline{T(V)}$  then*

$$T(V) \subseteq V = \overline{T(V)}.$$

*Proof.* Let  $d = \dim(V)$ . We use induction on increasing values of  $d$ . The case  $d = 0$  is trivial. For  $d > 0$ , let  $V_d$  be the union of those irreducible components of  $V$  that have dimension  $d$ ; let  $W_d$  be the union of the remaining components of  $V$ . We have  $\dim(W_d) < d$  and  $\dim \overline{T(W_d)} < d$ . Since each component of  $V_d$  has dimension  $d$  it must be contained in  $\overline{T(V_d)}$ , and hence  $V_d \subseteq \overline{T(V_d)}$ . Indeed, the irreducible components of  $\overline{T(V_d)}$  are the irreducible components of  $V_d$ , in possibly permuted order, and so  $V_d = \overline{T(V_d)}$ . By irredundance, none of the components of  $W_d$  are contained in any of the components of  $V_d = \overline{T(V_d)}$ . Since  $W_d \subseteq \overline{T(V_d)} \cup \overline{T(W_d)}$  we see that  $W_d \subseteq \overline{T(W_d)}$ . But  $\dim(W_d) < d$  so the induction hypothesis implies  $W_d = \overline{T(W_d)}$ . Hence

$$V = V_d \cup W_d = T(V_d) \cup \overline{T(W_d)} = \overline{T(V)}. \quad \square$$

With these preliminaries out of the way we derive a bound for  $\phi(d, k, \mathbf{F})$  expressed in terms of  $g(d, k)$  defined recursively in terms of a function  $h(i, d, k)$  by

$$h(1, d, k) = d, \quad h(i, d, k) = d^k (h(i-1, d, k))^2 + d$$

for  $1 \leq i \leq k$ , and

$$g(d, k) = h(k, d, k).$$

A computation shows that

$$g(d, 2) = d^4 + d, \quad g(d, 3) = d^{13} + 2d^9 + d^5 + d,$$

and that

$$g(d, k) = d^{(2^{k-1}-1)k+2^{k-1}} \left( 1 + O\left(\frac{1}{d}\right) \right)$$

as  $d \rightarrow \infty$  with  $k$  fixed.

**THEOREM 5.7.** *Let  $\mathbf{F}$  be an algebraically closed field and let  $T: \mathbf{F}^k \rightarrow \mathbf{F}^k$  be a polynomial map of degree at most  $d$ . Let  $C$  be a Zariski closed set of degree  $\text{deg}(C) \leq d$ . If*

$$T^{(j)}(\mathbf{x}) \in C \quad \text{for } 0 \leq j \leq g(d, k),$$

then

$$T^{(j)}(\mathbf{x}) \in C \quad \text{for } 0 \leq j < \infty.$$

Hence, taking  $C = \{\mathbf{x}: S(\mathbf{x}) = 0\}$ , we obtain

$$\phi(d, k, \mathbf{F}) \leq f(d, k, \mathbf{F}) + 1 \leq g(d, k) + 1.$$

*Proof.* We define a descending sequence of closed sets  $C_0 \supseteq C_1 \supseteq C_2 \supseteq \dots$  by  $C_0 = C$  and

$$(5.1) \quad C_j = C_{j-1} \cap \overline{T(C_{j-1})}.$$

Note that  $\bigcap_{i=0}^j T^{(i)}(C) \subseteq C_j$ . We have the following claim.

**CLAIM 1.** If  $T^{(i)}(\mathbf{x}) \in C$  for  $0 \leq i \leq n$ , then  $T^{(i)}(\mathbf{x}) \in C_i$  for  $0 \leq i \leq n$ .

In other words, successive iterates are more and more constrained by the condition that they be in  $C$ . The claim is obvious: If  $0 \leq i \leq n$ , then  $T^{(i)}(\mathbf{x}) = T^{(j)}(T^{(i-j)}(\mathbf{x}))$  for all  $0 \leq j \leq i$ . Since  $T^{(i-j)}(\mathbf{x}) \in C$  for such  $j$ , we have  $T^{(i)}(\mathbf{x}) \in T^{(j)}(C)$  and indeed  $T^{(i)}(\mathbf{x}) \in \bigcap_{j=0}^i T^{(j)}(C) \subseteq C_i$ .

Returning to the sequence  $C_0 \supseteq C_1 \supseteq \dots$ , we observe that because the Zariski topology is Noetherian this decreasing sequence of closed sets must eventually stabilize, i.e.,  $C_n = C_{n+1}$  for some  $n$ .

**CLAIM 2.** If  $C_n = C_{n+1}$ , then  $T(C_n) \subseteq C_n$ , and in fact  $\overline{T(C_n)} = C_n$ .

This follows directly from Lemma 5.6:  $C_{n+1} = C_n \cap \overline{T(C_n)}$ . If  $C_n = C_{n+1}$ , then  $C_n \subseteq \overline{T(C_n)}$ , and Lemma 5.6 applies.

An immediate consequence of Claim 2 is that if  $\mathbf{x} \in C_n$  with  $C_n = C_{n+1}$ , then  $T^{(j)}(\mathbf{x}) \in C_n$  for all  $j \geq 0$ . Using Claim 1 we then see that if  $T^{(j)}(\mathbf{x}) \in C$  for  $0 \leq j \leq n$ , where  $C_n = C_{n+1}$ , then  $T^{(j)}(\mathbf{x}) \in C_j \subseteq C$  for all  $j \leq 0$ . Hence the theorem follows from an upper bound on how soon the sequence  $\{C_j\}$  stabilizes.

**CLAIM 3.** The sequence  $\{C_j\}$  stabilizes after at most  $g(d, k)$  steps, i.e.,

$$C_n = C_{n+1}$$

for some  $n \leq g(d, k)$ .

The idea of the proof of Claim 3 is to bound the degrees and dimensions of the various irreducible components in the closed sets  $C_j$ . If  $C_{j+1} \neq C_j$  then, in going from  $C_j$  to  $C_{j+1}$  one or more irreducible components  $V$  “disappears” and is replaced by subvarieties of lower dimension, but of possibly higher degree. As a result  $C_{j+1}$  might have higher degree than  $C_j$  but only in higher codimension components than the ones that “disappear.” Meanwhile  $\text{deg}(V)$  of  $C_j$  has “leaked out” from dimension  $d = \dim(V)$  and is not in  $C_{j+1}$ .

Now define  $D(i, d, k)$  to be the maximum sum of the degrees of all components having dimension  $\geq i$  that ever occur in any element  $C_n$  of the descending sequence  $\{C_j\}$ . Then the sequence  $\{C_j\}$  must stabilize after at most  $D(0, d, k)$  steps, since at each step a component of degree at least one “disappears.” We develop recursive bounds for  $D(i, d, k)$ . First

$$(5.2) \quad D(k-1, d, k) \leq \deg C_0 \leq d$$

because this is an upper bound on the degree of all varieties in  $C_0$ , and because no new irreducible varieties of dimension  $k-1$  can arise during the intersection process (5.1). Next we prove inductively that

$$(5.3) \quad D(i, d, k) \leq d + d^k (D(i+1, d, k))^2.$$

Indeed each variety  $V$  of dimension  $\geq i$  is either in  $C_0$  or else arises as a component of an intersection  $\overline{T(V_1)} \cap V_2$  of two irreducible varieties  $V_1, V_2$  of dimension  $\geq i+1$  occurring during the intersection process  $\{C_j\}$ . Hence, using Theorem 5.1 and Lemma 5.5, we obtain

$$\begin{aligned} D(i, d, k) &\leq d + \sum_{\substack{V_1, V_2 \\ \dim(V_1), \dim(V_2) \geq i+1}} \deg(\overline{T(V_1)} \cap V_2), \\ &\leq d + \left( \sum_{\substack{V_1 \\ \dim(V_1) \geq i+1}} \deg(\overline{T(V_1)}) \right) \left( \sum_{\substack{V_2 \\ \dim(V_2) \geq i+1}} \deg(V_2) \right), \\ &\leq d + d^k \left[ \sum_{\substack{V_1 \\ \dim(V_1) \geq i+1}} \deg(V_1) \right]^2 \end{aligned}$$

which yields (5.3). The recursions (5.2), (5.3) immediately imply by induction on  $i$  that

$$D(k-i, d, k) \leq h(i, d, k)$$

for  $1 \leq i \leq k$ . In particular,

$$f(d, k, \mathbf{F}) \leq D(0, d, k) \leq h(k, d, k) = g(d, k).$$

Claim 3 is proved.  $\square$

We remark that the inequality (5.3) for  $D(i, d, k)$  is not sharp. In fact with more care one can show that

$$(5.4) \quad D(i, d, k) \leq d + d^k \binom{D(i+1, d, k) + 1}{2}$$

holds. (This bound is essentially the sharpest attainable using Bezout’s inequality alone.) The recursion (5.4) improves the bound of Theorem 5.5 asymptotically to

$$f(d, k, \mathbf{F}) \leq 2^{1-2^{k-1}} d^{(2^{k-1}-1)(k+2^{k-1})} \left( 1 + O\left(\frac{1}{d}\right) \right).$$

Also for  $k=2$  the bound (5.4) implies that

$$\phi(d, 2, \mathbf{F}) \leq f(d, 2, \mathbf{F}) + 1 \leq \frac{1}{2}(d^4 + d^3) + d + 1$$

so that  $\phi(2, 2, \mathbf{F}) \leq 15$ .

**6. Extrapolation of modular polynomial recurrences with unknown modulus  $M$ .** As an application we consider the problem of predicting the values  $\{\mathbf{x}_n : n = 1, 2, \dots\}$  generated by a polynomial recurrence of degree at most  $d$  in  $k$  variables defined over

a ring  $\mathbf{Z}/M\mathbf{Z}$  where  $d$  and  $k$  are known but the modulus  $M$  is *unknown*. Assume that the observed values of the  $\mathbf{x}_i$  are given with integer entries which are least positive residues modulo  $M$ . The problem of finding a good extrapolation  $\hat{\mathbf{x}}_{n+1}$  given  $\mathbf{x}_0, \dots, \mathbf{x}_n$  is a generalization of those studied by Boyar [B1], [B2], [B3]. Our algorithm, which we call Algorithm  $\mathbf{B}_{dk}$ , is based on Boyar's ideas. At stage  $n$  it takes as input the vectors  $\mathbf{x}_0, \dots, \mathbf{x}_n$  and produces output  $(T_n, m_n, \hat{\mathbf{x}}_{n+1})$  as its estimate of the unknown transformation, modulus and next value  $(T, M, \mathbf{x}_{n+1})$ . Here the  $T_n$  is a set of  $\mathbf{Z}/m_n\mathbf{Z}$ -valued coefficients for the transformation  $T$ , with the property that the equations  $\mathbf{x}_i \equiv T_n(\mathbf{x}_{i-1})$  hold in  $\mathbf{Z}/m_n\mathbf{Z}$  for  $i=1, 2, \dots, n$ , where we adopt the notational convention that  $\mathbf{Z}/m\mathbf{Z} = \mathbf{Q}$  if  $m = +\infty$ . The prediction  $\hat{\mathbf{x}}_{n+1}$  at step  $n$  is simply  $\hat{\mathbf{x}}_{n+1} \equiv T_n(\mathbf{x}_n)$  (modulo  $m_n$ ).

Let  $\Sigma_n$  denote the system of inhomogeneous linear equations with integer coefficients that the coefficients of  $T$  must satisfy if they are to explain the data  $\mathbf{x}_0, \dots, \mathbf{x}_n$ . That is to say,  $\Sigma_n$  is the system of  $nk$  equations  $\mathbf{x}_i = T(\mathbf{x}_{i-1})$  for  $1 \leq i \leq n$  in the  $k \binom{k+d}{k}$  coefficients of  $T$ . The general strategy of the algorithm is to keep track of the set of moduli  $m$  for which it is possible to solve  $\Sigma_n$  modulo  $m$ . (Of course  $\Sigma_n$  are always solvable modulo  $M$ , but not necessarily for other moduli.) For any system  $\Sigma$  of linear inhomogeneous equations with integer coefficients let  $m^*(\Sigma)$  denote the largest modulus  $m$  for which  $\Sigma$  are solvable in  $\mathbf{Z}/m\mathbf{Z}$ , the case  $m = +\infty$  not excluded. (See Lemma 6.3 below for more information about  $m^*(\Sigma)$ .)

With this notation we may state our algorithm  $\mathbf{B}_{dk}$ :

- $\mathbf{B}_{dk}(0)$  Start at step  $n = 0$ .
- $\mathbf{B}_{dk}(1)$  Attempt to solve  $\Sigma_n$  over  $\mathbf{Q}$ . If there is no solution go to  $\mathbf{B}_{dk}(2)$ . If there is no solution call it  $T_n$ , set  $m_n = +\infty$  and set  $\hat{\mathbf{x}}_{n+1} = T_n(\mathbf{x}_n)$ . Deliver  $(T_n, m_n, \hat{\mathbf{x}}_{n+1})$  and go on to the next  $n$ .
- $\mathbf{B}_{dk}(2)$  If  $\Sigma_n$  has no solution over  $\mathbf{Q}$  let  $m_n = m^*(\Sigma_n)$  as described in Lemma 6.3. Let  $T_n$  solve  $\Sigma_n$  modulo  $m_n$  and set  $\hat{\mathbf{x}}_{n+1} = T_n(\mathbf{x}_n)$  reduced modulo  $m_n$ . Deliver  $(T_n, m_n, \hat{\mathbf{x}}_{n+1})$  and go on to the next  $n$ .

The linear equation solving can be done in polynomial time using standard algorithms [CC], [KB]. Some shortcuts are possible in the above algorithm. For example, if  $m_{n-1} < \infty$ , then we know  $\Sigma_n$  has no solution over  $\mathbf{Q}$  and that statement  $\mathbf{B}_{dk}(2)$  will have to be used. In this case, then, step  $\mathbf{B}_{dk}(1)$  may safely be omitted. A less obvious observation is due to Boyar [B1]: if  $n > \phi^*(d, k)$  and  $m_{n-1} < \infty$ , then  $m_n$  and  $T_n$  may be very cheaply computed by using the following fact:  $m_n$  is the greatest common divisor of  $m_{n-1}$  and the entries of  $\mathbf{x}_n - \hat{\mathbf{x}}_n$ , and one may take  $T_n = T_{n-1}$  since  $T_{n-1}$  solves  $\Sigma_n$  modulo  $m_n$ . This is proven below as Proposition 6.5.

The correctness of algorithm  $\mathbf{B}_{dk}$  is summarized in the following theorem, which depends on the unique extrapolation results of the earlier part of the paper.

**THEOREM 6.1.** *If the sequence  $\mathbf{x}_0, \mathbf{x}_1, \dots$  obeys  $\mathbf{x}_n \equiv T(\mathbf{x}_{n-1})$  (modulo  $M$ ) for  $n = 1, 2, \dots$ , then:*

- (1) *For all  $1 \leq i \leq n$ ,  $\mathbf{x}_i \equiv T_n(\mathbf{x}_{i-1}) \pmod{m_n}$ .*
- (2) *If  $m_n < \infty$ , then  $M \mid m_n$ .*
- (3) *If  $m_n < \infty$ , then  $m_{n+1} \mid m_n$ .*
- (4) *If  $n \geq \phi^*(d, k)$  and  $\hat{\mathbf{x}}_{n+1} \neq \mathbf{x}_{n+1}$ , then  $m_{n+1} \neq m_n$ .*

The performance of the algorithm depends essentially on the magnitude of the quantities  $m^*(\Sigma_n)$  used in calculating  $m_n$  in  $\mathbf{B}_{dk}(2)$ :

**THEOREM 6.2.** *Let  $N = 1 + k \binom{k+d}{k}$ .*

- (1) *If  $m_n < \infty$ , then  $m_n \leq N^{N/2} M^{dN}$ .*
- (2)  *$\hat{\mathbf{x}}_n \neq \mathbf{x}_n$  for at most  $\phi^*(d, k) + 1 + \log_2(N^{N/2} M^{dN})$  values of  $n$ .*



The proofs of Theorems 6.1 and 6.2 are given later in this section. The following two lemmas give the details of the computation in  $\mathbf{B}_{ak}(2)$  promised above.

LEMMA 6.3. *Let the  $e$  by  $n$  matrix  $\mathbf{A}$  and the  $e$  vector  $\mathbf{b}$  have integer entries. Suppose the system  $\Sigma$  of inhomogeneous linear equations*

$$\mathbf{Ax} = \mathbf{b}$$

has no solution over  $\mathbf{Q}$ .

(1) *Then there is a unique modulus  $m^*(\Sigma)$  such that  $\Sigma$  is solvable modulo  $m$  if and only if  $m$  divides  $m^*(\Sigma)$ .*

(2) *If all the entries  $\mathbf{A}$  and  $\mathbf{b}$  are bounded by  $\lambda$  in absolute value, then*

$$m^*(\Sigma) \leq (n + 1)^{(n+1)/2} \lambda^{n+1}.$$

(3) *There is a polynomial-time algorithm which solves  $\Sigma$  over the rationals if possible, and otherwise determines  $m^*(\Sigma)$  and solves  $\Sigma$  modulo  $m^*(\Sigma)$ .*

*Proof.* Claim (1) can be seen by reducing  $\Sigma$  over  $\mathbf{Z}$  using the Smith normal form for  $\mathbf{A}$  (see [KB]) into a new system of the form

$$\lambda_1 z_1 = \mu_1,$$

$$\lambda_2 z_2 = \mu_2,$$

...

$$\lambda_r z_r = \mu_r,$$

$$0 = \mu_{r+1},$$

$$0 = \mu_{r+2},$$

...

$$0 = \mu_e.$$

(Here  $r$  is the rank of  $\mathbf{A}$ .) If  $r = e$  or if  $\mu_{r+1} = \mu_{r+2} = \dots = \mu_e = 0$ , the system has a solution over the rationals:  $z_i = \mu_i / \lambda_i$ . Otherwise only a mod  $m$  solution can exist, where  $m$  must divide each of  $\mu_{r+1}, \mu_{r+2}, \dots, \mu_e$ . Thus the set  $\mathbf{M} = \{m : \Sigma \text{ is solvable modulo } m\}$  is finite. Certainly  $\mathbf{M}$  contains 1. If  $m$  is in  $\mathbf{M}$  and  $m'$  divides  $m$ , then  $m'$  is also in  $\mathbf{M}$ . Suppose  $m_1$  and  $m_2$  are in  $\mathbf{M}$ , so  $m_2 / \gcd(m_1, m_2)$  is also in  $\mathbf{M}$ . By the Chinese Remainder Theorem the product  $m_1 m_2 / \gcd(m_1, m_2) = \text{lcm}(m_1, m_2)$  is also in  $\mathbf{M}$ . Hence  $\mathbf{M}$  contains the l.c.m. of all its elements, which is the promised  $m^*(\Sigma)$ . The value of  $m^*(\Sigma)$  can be computed from the  $\lambda_i$  and  $\mu_i$  by repeated application of the recipe of Lemma 6.4 below. This value of  $m^*(\Sigma)$  is used as  $m_n$  in  $\mathbf{B}_{ak}(2)$  above.

Claim (3), that all the indicated computations can be done in polynomial time (polynomial in the number of bits to specify  $\mathbf{A}$  and  $\mathbf{b}$  in binary), follows when the algorithms of [KB] and especially [CC] are used in the above sketch.

It remains to prove the bound of claim (2). Suppose that  $\mathbf{x}$  solves  $\Sigma$  modulo  $m$ . Consider the augmented coefficient matrix  $\mathbf{A}'$  obtained by adjoining the column  $\mathbf{b}$  to the right of  $\mathbf{A}$ , and let  $\mathbf{x}'$  be obtained by appending the entry  $-1$  to the end of  $\mathbf{x}$ . Then there is an integer vector  $\mathbf{v}$  such that

$$\mathbf{A}'\mathbf{x}' = m\mathbf{v}$$

(over the integers). Now reduce  $\mathbf{A}'$  to Smith normal form:  $\mathbf{A}' = \mathbf{K}_1 \mathbf{S} \mathbf{K}_2$  where  $\mathbf{K}_1$  is in  $SL(e, \mathbf{Z})$  and  $\mathbf{K}_2$  is in  $SL(n, \mathbf{Z})$ , and  $\mathbf{S}$  has nonzero entries  $s_1, s_2, \dots, s_r$  only on the main diagonal, all of the  $s_i$  dividing  $s_r$ , where  $r$  is the rank of  $\mathbf{A}'$ . Let  $\mathbf{y} = \mathbf{K}_2 \mathbf{x}'$  and let  $\mathbf{w} = \mathbf{K}_1^{-1} \mathbf{v}$ . Then

$$\mathbf{S}\mathbf{y} = m\mathbf{w},$$

so  $m \mid s_i y_i$  for all  $i = 1, 2, \dots, r$ , and hence  $m \mid s_r y_i$  for all  $i$ . Since the greatest common divisor of the entries of  $\mathbf{x}'$  is one and since  $\mathbf{K}_2 \in SL(n, \mathbf{Z})$ , the greatest common divisor of the entries of  $\mathbf{y}$  is also one and there exist integers  $k_i$  such that  $\sum_{i=1}^{n+1} k_i y_i = 1$ . Hence  $m$  divides  $\sum_{i=1}^{n+1} k_i s_r y_i = s_r$ . But  $s_r$  in turn is a divisor of the greatest common divisor of all  $r$  by  $r$  minors of  $\mathbf{A}'$ , each one of which, by Hadamard's inequality, is bounded by  $r^{r/2} \lambda^r \leq (n+1)^{(n+1)/2} \lambda^{n+1}$ .  $\square$

LEMMA 6.4. For given  $(m, a, b)$  the set of divisors  $\mu$  of  $m$  such that the congruence

$$ax \equiv b \pmod{\mu}$$

is solvable is precisely the set of divisors of  $f(m, a, b)$  where

$$f(m, a, b) = m / \gcd \left( \left( \frac{a}{\gcd(a, b)} \right)^j, \frac{m}{\gcd(m, a, b)} \right),$$

where  $j = \lceil \log_2 m \rceil$ .

*Proof.* (This is essentially Lemma 3 of Boyar [B1], who presents a recursive algorithm for computing  $f(m, a, b)$ .) For each prime  $p$ , if  $p^{m_p} \parallel m$ ,  $p^{a_p} \parallel a$  and  $p^{b_p} \parallel b$ , then  $p^{\mu_p} \parallel \mu$  where  $\mu_p \leq \min(m_p, b_p)$  if  $a_p > b_p$  and  $\mu_p \leq m_p$  otherwise. By a routine calculation we check that the maximal such  $\mu$  is  $f(m, a, b)$ .  $\square$

Now we proceed to the proofs of Theorems 6.1 and 6.2.

*Proof of Theorem 6.1.* Claim (1) is obvious from the construction of  $m_n$  and  $T_n$  in  $\mathbf{B}_{dk}$ . To see claim (2) note that the true  $T$  solves  $\Sigma_n$  modulo  $M$ , so Lemma 6.3 tells us that  $M \mid m_n$ . Claim (3) is also easy: the system  $\Sigma_n$  is a subset of  $\Sigma_{n+1}$  so  $m_{n+1} < \infty$  and  $T_{n+1}$  solves  $\Sigma_n$  modulo  $m_{n+1}$ . By Lemma 6.3 this means  $m_{n+1} \mid m_n$ . Finally, to prove claim (4), suppose at stage  $n$  the algorithm produces  $(T_n, m_n, \hat{\mathbf{x}}_{n+1})$  satisfying  $\Sigma_n$  modulo  $m_n$  and at stage  $n+1$  the algorithm produces  $(T_{n+1}, m_{n+1})$  satisfying  $\Sigma_{n+1}$  modulo  $m_{n+1}$ . If  $m_n = m_{n+1}$ , then by the unique extrapolation property for the ring  $\mathbf{Z}/m_n\mathbf{Z}$  we must have  $T_n(\mathbf{x}_n) = T_{n+1}(\mathbf{x}_n)$ , i.e.,  $\hat{\mathbf{x}}_{n+1} = \mathbf{x}_{n+1}$ .  $\square$

*Proof of Theorem 6.2.* Claim (1) follows directly from claim (2) of Lemma 6.3. There are  $k \binom{k+d}{k}$  coefficients in  $T$ , and hence that many variables in the equations to be solved in  $\mathbf{B}_{dk}(2)$ . The entries in the coefficient matrix are monomials of total degree  $\leq d$  in the data vector components, and hence are bounded by  $M^d$ . Claim (2) follows from this observation and from claims (3) and (4) of Theorem 6.1: the number of distinct elements in a chain of divisors

$$M \mid \dots \mid m_{n+1} \mid m_n$$

is bounded by  $1 + \log_2 m_n$ .  $\square$

Finally, we show that the "speed up" for  $\mathbf{B}_{dk}(2)$  mentioned above is correct.

PROPOSITION 6.5. Suppose  $n > \phi^*(d, k)$  and  $m_{n-1} < \infty$ . Let  $m$  denote the greatest common divisor of  $m_{n-1}$  and the entries of  $\mathbf{x}_n - \hat{\mathbf{x}}_n$ . Then  $m_n = m$  and  $T_{n-1}$  solves  $\Sigma_n$  modulo  $m_n$ .

*Proof.* By Theorem 6.1 we know  $m_n \mid m_{n-1}$  and that both  $T_{n-1}$  and  $T_n$  solve  $\Sigma_{n-1}$  modulo  $m_n$ . Use both  $T_{n-1}$  and  $T_n$  to predict  $\mathbf{x}_n$  modulo  $m_n$ . By the unique extrapolation property for the ring  $\mathbf{Z}/m_n\mathbf{Z}$  both predictions agree, so that  $T_{n-1}(\mathbf{x}_{n-1}) = T_n(\mathbf{x}_{n-1})$  modulo  $m_n$ . This means that  $\hat{\mathbf{x}}_n = \mathbf{x}_n$  modulo  $m_n$  and hence  $m_n$  is a divisor of all the coefficients of  $\hat{\mathbf{x}}_n - \mathbf{x}_n$  as well as of  $m_{n-1}$ . Since  $m$  is defined as the greatest such divisor,  $m_n \mid m$ . On the other hand,  $T_{n-1}$  solves  $\Sigma_n$  modulo  $m$  so by Lemma 6.3 we know  $m \mid m_n$ .  $\square$

*Remarks.* (1) Theorem 6.2 bounds the number of mistakes that the extrapolation algorithm may make, but it does not bound the number of values seen before the last mistake occurs. As an example, Algorithm  $\mathbf{B}_{11}$  takes  $M+1$  steps to distinguish the

simple recurrence  $y_0 = 0$  and  $y_n = y_{n-1} + 1$  over  $\mathbf{Z}$  from the recurrence  $y_0 = 0$  and  $y_n \equiv y_{n-1} + 1 \pmod{M}$ . This number of steps is exponential in  $\log M$ , which is the number of bits needed to specify the modulus  $M$ .

(2) The extrapolation method of this section does not apply to the more general problem of extrapolating a sequence  $\{z_i : i \leq 0\}$  generated by polynomial maps  $T : (\mathbf{Z}/M\mathbf{Z})^k \rightarrow \mathbf{Z}/M\mathbf{Z}$  and  $S : (\mathbf{Z}/M\mathbf{Z})^k \rightarrow \mathbf{Z}/M\mathbf{Z}$ , where

$$\mathbf{x}_n = T(\mathbf{x}_{n-1}), \quad z_n = S(\mathbf{x}_n),$$

even when *both the maps  $T$  and  $S$  and the modulus  $M$  are known*. In this case the problem of reconstructing a seed  $\mathbf{x}_0$  that produces the observed data  $\{z_i : 0 \leq i \leq n\}$  is nonlinear, and we do not know of a polynomial-time algorithm to solve it. The problem of extrapolating the  $\{z_i : 0 \leq i \leq n\}$  seems equally difficult.

**Acknowledgments.** We are indebted to J. H. Conway for suggesting that our preliminary results should hold over arbitrary commutative rings with unit, to C. P. Schnorr for references to a multidimensional Bezout's theorem, to J. P. Butler for Proposition 4.2 and to L. A. Shepp for helpful comments.

#### REFERENCES

- [B1] J. BOYAR (PLUMSTEAD), *Inferring a sequence generated by a linear congruence*, Proc. 23rd Annual IEEE Conference on the Foundations of Computer Science, 1982, pp. 153–159.
- [B2] ——— (PLUMSTEAD), *Inferring sequences produced by pseudo-random number generators*, Ph.D. thesis, Computer Science Dept., Univ. of Calif. Berkeley, CA, 1983.
- [B3] ———, *Inferring sequences produced by pseudo-random number generators*, preprint.
- [CC] T. J. CHOU AND G. E. COLLINS, *Algorithms for the solution of linear diophantine equations*, this Journal, 11 (1982), pp. 687–708.
- [FHKLS] A. FRIEZE, J. HASTAD, R. KANNAN, J. LAGARIAS AND A. SHAMIR, *Reconstructing truncated integer variables satisfying linear congruences*, this Journal, 17 (1988) pp. 262–280.
- [FKL] A. FRIEZE, R. KANNAN AND J. C. LAGARIAS, *Linear congruential generators do not produce random sequences*, Proc. 25th Annual IEEE Conference on the Foundations of Computer Science, 1984, pp. 480–484.
- [FL] W. FULTON AND R. LAZARFELD, *Positivity and excess intersection*, in *Enumerative Geometry and Classical Algebraic Geometry*, Birkhäuser, Boston, 1982.
- [Fu] W. FULTON, *Algebraic Curves*, W. A. Benjamin, New York, 1969.
- [H] R. HARTSHORNE, *Algebraic Geometry*, Springer-Verlag, New York, 1977.
- [HS] J. HASTAD AND A. SHAMIR, *The cryptographic security of truncated linearly related variables*, Proc. 17th Annual ACM Symposium on the Theory of Computing, 1985, pp. 356–362.
- [KB] R. KANNAN AND A. BACHEM, *Polynomial algorithms for computing the Hermite and Smith normal forms of an integer matrix*, this Journal, 8 (1979), pp. 499–507.
- [K] D. KNUTH, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1970.
- [L] S. LANG, *Algebra*, Addison-Wesley, Reading, MA, 1970.
- [Mal] C. MALLOWS, *Identities satisfied by iterated polynomials and  $Q(x)$ -binomial coefficients*, preprint.
- [Mar] G. MARSAGLIA, *Random numbers fall mainly in the planes*, Proc. Nat. Acad. Sci. U.S.A., 61 (1968), pp. 25–28.
- [N] H. NIEDERREITER, *Quasi-Monte Carlo methods and pseudo-random numbers*, Bull. Amer. Math. Soc., 84 (1978), pp. 957–1041.
- [S] C. SCHNORR, *An extension of Strassen's degree bound*, this Journal, 10 (1981), pp. 371–382.
- [Sh] I. SHAFAREVICH, *Basic Algebraic Geometry*, Springer-Verlag, New York, 1977.
- [ZS] O. ZARISKI AND P. SAMUEL, *Commutative Algebra*, vol. 2, Van Nostrand, Princeton, NJ, 1960.

## THE DISCRETE LOGARITHM HIDES $O(\log n)$ BITS\*

DOUGLAS L. LONG† AND AVI WIGDERSON‡

**Abstract.** The main result of this paper is that obtaining any information about the  $O(\log |p|)$  “most significant” bits of  $x$ , given  $g^x \pmod{p}$ , even with a tiny advantage over guessing, is equivalent to computing discrete logarithms mod  $p$ .

**Key words.** discrete logarithm, one-way functions, secure transactions, bit security, pseudorandom bit generator, public-key cryptography

**AMS(MOS) subject classifications.** 68P25, 68Q15

**1. Introduction.** A function  $f: [1, N] \rightarrow [1, N]$  is said to be one-way if computing  $f(x)$  from  $x$  is easy, but computing  $x$  from  $f(x)$  is hard. One-way functions are extremely attractive in cryptographic applications (see [5], [6], [16], [18], [19]). However, one should use them with care. The reason is that while  $x$  is hard to compute from  $f(x)$ , it may be possible to obtain *almost all* bits of  $x$  easily. It is clear, however, that some bits of  $x$  are hard to obtain.

In a key paper, Blum and Micali [5] introduced the notion of hiding a bit in a one-way function. A Boolean predicate  $B: [1, N] \rightarrow \{0, 1\}$  is said to be hard for  $f$  if an oracle for  $B(f(x))$  allows one to invert  $f$  easily. They prove that a certain Boolean predicate is hard for the discrete logarithm function, and show how to use that to construct good pseudorandom number generators.

Some natural questions arise in light of their work. Given a one-way function  $f$ :

- 1) What are its hard bits?
- 2) How many hard bits are there?
- 3) How many bits are hard simultaneously?

To explain what we mean by the last question, consider  $B$ , a hard Boolean predicate for  $f$ , and its complement,  $\bar{B}$ . Clearly,  $\bar{B}$  is also hard for  $f$ . However, some information about the pair of bits  $(B(f(x)), \bar{B}(f(x)))$  is available, namely the information that they are different. Therefore this two-bit predicate is not hard for  $f$ . We say that a  $k$ -bit predicate  $B^k: [1, N] \rightarrow \{0, 1\}^k$  is hard for  $f$  if for every Boolean predicate  $B: \{0, 1\}^k \rightarrow \{0, 1\}$ , an oracle for  $B(B^k(f(x)))$  will allow us to invert  $f$  easily. If such a  $B^k$  exists, we say that  $f$  hides  $k$  bits.

The fact that hiding only one bit suffices for many applications seems to reduce the motivation for the questions above. We would like to argue that this is not the case. From the practical point of view, the ability to hide  $k$  bits cuts the computation/communication per secure bit by a factor of  $k$ . One can flip  $k$  coins at a time or generate  $k$  bits at a time with a pseudorandom number generator without hurting security.

But what really motivates us is the theoretic point of view. Very little is known about one-way functions. In particular, a major open problem is whether they exist. We believe that discovering and studying the “hard core” (the collection of hard bits) of functions that are believed to be one-way may shed some light on this problem.

---

\* Received by the editors September 17, 1985; accepted for publication (in revised form) June 24, 1986.

† Computer Science Department, Wellesley College, Wellesley, Massachusetts 02181. The work of this author was supported in part by Army Research Office grant DAAG29-80-K-0090 and a Garden State Fellowship from Princeton University, Princeton, New Jersey 08544.

‡ Mathematical Sciences Research Institute, Berkeley, California 94720. The work of this author was supported in part by an IBM Graduate Fellowship from Princeton University, Princeton, New Jersey 08544.

In this paper we focus our attention on the discrete logarithm function, which is widely believed to be one-way. Our main result states that the discrete log function hides  $c \cdot \log |p|$  bits for any constant  $c$ , where  $p$  is the modulus and  $|p|$  is the number of bits in the binary representation of  $p$ . This paper subsumes the results of [11] where we proved the same result using an additional assumption (see § 5 for details). Similar results for the case of the RSA and Rabin encryption functions were subsequently obtained by Alexi, Chor, Goldreich and Schnorr [3] and by Vazirani and Vazirani [17].

Results similar to those of [11] were obtained by Long [12], using similar techniques but a different set of bits. Recently, using a different technique (similar in spirit to that used in [17]), Peralta [13] showed that the discrete log function can be used for a pseudorandom bit generator which outputs  $O(\log |p|)$  bits at time, a result which is equivalent to ours.

Section 2 states the main results of the paper. Section 3 contains some technical lemmas. In § 4 we extend the techniques of [5] to prove our main result for the case where the oracle is always correct. In § 5 we extend our results to the case where the oracle is more correct than incorrect. In § 6 we discuss an application of our work to pseudorandom bit generation. Section 7 concludes with an open problem.

**2. Hiding bits.** We start with some definitions. From this point on  $p$  will represent an  $n$ -bit prime,  $Z_p^*$  the group of units mod  $p$ ,  $g$  a generator of  $Z_p^*$ , and  $k$  an integer such that  $k \leq n$ .

**DEFINITION 2.1.** Suppose  $z \in Z_p^*$ . The **index of  $z$** ,  $\text{index}(z)$ , is the unique  $x \in Z_p^*$  such that  $z \equiv g^x \pmod{p}$ .  $x$  is also known as the **discrete logarithm of  $z$** .

Blum and Micali [5] showed how to use the discrete logarithm to hide a single bit. They hide their bit by dividing the range  $[1, p - 1]$  into two equal-size intervals called the left and the right. A “0” bit is encoded by choosing  $x$  from the left and computing  $g^x$ . A “1” bit is encoded by choosing  $x$  from the right. They prove that even a small advantage in determining from  $g^x$  if  $x$  is in the left or the right can be used to solve the discrete logarithm problem. In fact, it is only necessary to be able to do this for  $x$  that are in the beginning of the left or beginning of the right. We can state this formally as the

*t-Left-Right Problem.* Given  $g^x \in Z_p^*$ , such that  $x \in [1, (p-1)/t] \cup [(p-1)/2, (p-1)/2 + (p-1)/t]$ , determine if  $x \in [1, (p-1)/t]$  or  $x \in [(p-1)/2, (p-1)/2 + (p-1)/t]$ .

Blum and Micali’s result can be stated as the following:

**THEOREM 2.1** (Blum and Micali). *Suppose  $p$  is an  $n$ -bit prime and  $\delta < \frac{1}{2}$ . Suppose there exists an oracle which solves the  $t$ -left-right problem correctly with probability at least  $1 - \delta$ . Then there exists a Las Vegas algorithm for the discrete logarithm which makes calls to this oracle and has expected running time  $O(t \cdot n^3(1 - \delta)^{-n})$ .*

We generalize Blum and Micali’s method of hiding 1 bit to  $k$  bits by dividing the interval  $[1, p - 1]$  into  $2^k$  consecutive, roughly equal, intervals  $I_i^k = [L_i^k, U_i^k]$  for  $i$  from 0 to  $2^k - 1$ , where  $L_i^k = \lfloor ((p-1)/2^k)i \rfloor + 1$ , and  $U_i^k = \lfloor ((p-1)/2^k)(i+1) \rfloor$ . We will call this a partition at the  $k$ th level. For  $x \in [1, p - 1]$  let  $I^k(x)$  be the interval which contains  $x$ , i.e., the integer  $i$  such that  $x \in I_i^k$ . Let  $R^k = \lfloor (p-1)/2^k \rfloor$ . The lower indices of  $I_i^k$  are computed modulo  $2^k$ , i.e.,  $I_{r+s}^k$  means  $I_{r+s \pmod{2^k}}^k$ .

In order to hide the  $k$  bits representing the integer  $i$ ,  $0 \leq i < 2^k$ , choose a random  $x \in I_i^k$  and compute  $g^x \pmod{p}$ . An integer  $z \in Z_p^*$  hides the bits which represent the interval of its index.

We claim that these bits are securely hidden. In order to justify this claim we must show that any advantage in determining not only the bits, but any partial information

about the bits, can be used to solve the discrete logarithm problem. Determining partial information about the bits means evaluating some nontrivial Boolean predicate on  $k$  bits. We will call such a predicate a decision.

DEFINITION 2.2. A **decision on  $k$  bits** is a nonconstant function  $d : \{0, 1\}^k \rightarrow \{0, 1\}$ . If we consider  $d$  as a function from the integers  $0, \dots, 2^k - 1$  into  $\{0, 1\}$  then we can extend  $d$  to all integers by defining  $d(i) = d(i \pmod{2^k})$ .

We claim any advantage in evaluating any decision on  $k$  bits can be used to solve the discrete log problem. First we formalize what we mean by an advantage.

DEFINITION 2.3. Suppose  $d$  is a decision on  $k$  bits. Let  $f_d$  = the fraction of  $k$ -tuples  $(b_1, b_2, \dots, b_k)$  such that  $d(b_1, b_2, \dots, b_k) = 1$ .

Without loss of generality we assume that  $f_d \geq \frac{1}{2}$ .

DEFINITION 2.4. Suppose  $d$  is a decision on  $k$  bits. A function  $\theta_d : [1, p - 1] \rightarrow \{0, 1\}$  is an  $\alpha$ -**oracle** for  $d$  if  $\theta_d(g^x) = d(I^k(x))$  for the fraction  $\alpha$  of  $x \in [1, p - 1]$ . A 1-oracle is known as a **perfect oracle** (i.e., an oracle that never makes a mistake). An  $\alpha$ -oracle has an  $\varepsilon$ -**advantage** in evaluating  $d$  if  $\alpha = f_d + \varepsilon$ .

An oracle with an  $\varepsilon$ -advantage does  $\varepsilon$  better than an oracle that always guesses 1.

We can now state the main result of this paper. Any nontrivial advantage in evaluating any decision on  $k$  bits can be used to solve the discrete log problem. We state this formally as the following theorem and corollary.

THEOREM 2.2. For any  $k$ , let  $d$  be a decision on  $k$  bits. Let  $\theta_d$  be an  $\varepsilon$ -advantage oracle for  $d$ . Then, using  $\theta_d$ , it is possible to construct an algorithm that solves the  $s$ -left-right problem correctly with probability  $1 - \delta$ . The parameter  $s$  and the running time of the conversion algorithm are polynomial in  $\varepsilon^{-1}$ ,  $\delta^{-1}$ ,  $n$ , and  $2^k$ . (Note that  $\varepsilon^{-1}$  may be polynomial in  $n$ .)

COROLLARY 2.1. Suppose  $k = O(\log n)$ . Let  $d$  be a decision on  $k$  bits. If there exists an  $\varepsilon$ -advantage oracle for  $d$ , then there exists a Las Vegas algorithm for the discrete log that runs in time polynomial in  $n$  and  $\varepsilon^{-1}$ .

*Proof.* Follows from Theorem 2.1 and Theorem 2.2.  $\square$

**3. Properties of partitions.** In this section we present a series of technical lemmas outlining some of the properties of partitions. The reader may wish to skip the proofs on first reading.

The following lemma shows that the intervals  $I_i^k$  are of roughly the same size.

LEMMA 3.1. For  $0 \leq i \leq 2^k - 1$ ,  $|I_i^k| = R^k$  or  $R^k + 1$ .

*Proof.*

$$|I_i^k| = U_i^k - L_i^k + 1 = \left\lfloor \frac{p-1}{2^k} (i+1) \right\rfloor - \left\lfloor \frac{p-1}{2^k} \right\rfloor.$$

Let  $0 \leq r_i < 1$  and  $0 \leq s_i < 1$  be the fractional parts of the expressions in the floors. Then

$$|I_i^k| = \frac{p-1}{2^k} i + \frac{p-1}{2^k} - r_i - \left( \frac{p-1}{2^k} i - s_i \right) = \frac{p-1}{2^k} - r_i + s_i.$$

But  $|s_i - r_i| < 1$  and  $\lfloor (p-1)/2^k \rfloor \leq (p-1)/2^k < \lfloor (p-1)/2^k \rfloor + 1$  so  $|I_i^k|$  must be either  $\lfloor (p-1)/2^k \rfloor$  or  $\lfloor (p-1)/2^k \rfloor + 1$ .  $\square$

The following lemma shows that the partition at the  $k$ th level refines the partition at the  $(k-1)$ st level.

LEMMA 3.2.  $L_i^{k-1} = L_{2i}^k$ ,  $U_i^{k-1} = U_{2i+1}^k$ . Consequently  $I_i^{k-1} = I_{2i}^k \cup I_{2i+1}^k$ .

*Proof.* Follows immediately from the definition of  $I_i^k$ .  $\square$

Throughout this paper, given an element  $z \in \mathbb{Z}_p^*$ , we will apply the operations  $w \leftarrow zg^{iR^k}$ ,  $w \leftarrow \sqrt{z}$  ( $z$  is a quadratic residue and we pick the root with the smaller index), and  $w \leftarrow zg^{-1}$  ( $z$  is a nonresidue). The effect of these operations on the index of  $z$  is

to shift it by a multiple of  $R^k$ , to divide it by two (when even), and to subtract one from it (when odd), respectively. We will call these operations *index operations* because they change the index in a specific way. The following lemmas show into which intervals the new index may fall.

LEMMA 3.3. *Let  $x \in I_j^k$ . Then for every  $1 \leq i \leq 2^k - 1$*

- (1)  $x + iR^k \in I_{i+j}^k$ , or
- (2)  $x + tR^k = L_{t+j}^k$  for some  $0 \leq t < i$ .

*Proof.* By induction on  $i$ . If  $i = 1$  then from Lemma 3.1  $x + R^k \in I_j^k$  if and only if  $|I_j^k| = R^k + 1$  and  $x = L_j^k$ . If  $i > 1$  then let  $x' = x + (i - 1)R^k$ . If  $x' \notin I_{i+j-1}^k$  then case (2) holds for  $x'$  and therefore for  $x$ . Otherwise  $x' \in I_{i+j-1}^k$  so  $x + iR^k \equiv x' + R^k \pmod{p - 1}$  and the lemma follows by the same argument as for  $i = 1$ .  $\square$

LEMMA 3.4. *If  $2x \in I_j^{k-1}$ , then*

- (1)  $x \in I_j^k$  or  $x \in I_{j+2^{k-1}}^k$ , or
- (2)  $2x = L_j^{k-1}$  or  $2x = U_j^{k-1}$ .

*Proof.* Suppose  $x \leq (p - 1)/2$ . (The case  $x > (p - 1)/2$  is analogous.)

Suppose  $2x > L_{2j}^k = L_j^{k-1}$ . Then  $1 + \lfloor ((p - 1)/2^k)2j \rfloor < 2x$ . So  $((p - 1)/2^k)2j < 2x$  and  $\lfloor ((p - 1)/2^k)j \rfloor \leq ((p - 1)/2^k)j < x$ . Therefore  $L_j^k = 1 + \lfloor ((p - 1)/2^k)j \rfloor \leq x$ .

Now suppose  $2x < U_{2j+1}^k = U_j^{k-1}$ . Then  $2x < \lfloor ((p - 1)/2^k)((2j + 1) + 1) \rfloor \leq ((p - 1)/2^k)2(j + 1)$ . Thus  $x \leq \lfloor ((p - 1)/2^k)(j + 1) \rfloor = U_j^k$ .  $\square$

LEMMA 3.5. *If  $x \in I_j^k$  (and  $x$  is odd), then either*

- (1)  $(x - 1) \in I_j^k$ , or
- (2)  $x = L_j^k$ .

*Proof.* Follows immediately from the definition.  $\square$

The index operations given above are well behaved (in the sense that they change the index of  $z$  in a specified way) except when  $z = g^{L_j^k}$  or  $z = g^{U_j^k}$ . We call these points the *boundary points*. We can easily determine if  $z$  is a boundary point (and thereby compute  $\text{index}(z)$ ) by maintaining a sorted list of these values. There are  $O(2^k)$  values to maintain so sorting will cost  $O(k2^k)$  time and each query  $O(k)$  time. If  $z$  is not a boundary point then the index operations will result in the index of  $z$  falling in the intervals given in case (1) of Lemmas 3.3, 3.4 or 3.5. Henceforth, we will assume that whenever an index operation is performed on  $z$ , a test for case (2) of Lemmas 3.3, 3.4 or 3.5 is performed by checking to see if  $z$  is a boundary point.

**4. Oracles which are always correct.** This section considers the case of a perfect oracle. In the next section we will consider what happens if the oracle is not perfect.

We will prove the following

THEOREM 4.1. *Let  $k = O(\log n)$ . If there exists a decision  $d$  on  $k$  bits, and a perfect oracle for  $d$ ,  $\theta_a$ , then it is possible to solve the  $2^k$ -left-right problem. This can be done in time polynomial in  $n$ , including time for the boundary tests.*

We first note the following facts about decisions.

Decisions repeat themselves at intervals of  $2^k$ . Some decisions repeat themselves at smaller intervals.

DEFINITION 4.1. Let  $d$  be a decision on  $k$  bits. The **period of  $d$**  is the smallest  $r$  such that  $d(i) = d(i + r)$  for all  $i$ .

The period of a decision is always a power of two.

LEMMA 4.1. *Let  $d$  be a decision on  $k$  bits and let  $r$  be the period of  $d$ . Then  $r = 2^l$  where  $0 < l \leq k$ .*

*Proof.* Immediate from the definition of period.  $\square$

LEMMA 4.2. *Let  $d$  be a decision on  $k$  bits with period  $2^l$ . Then there exists an  $j < 2^{l-1}$  such that  $d(j) \neq d(j + 2^{l-1})$ . Moreover, such a  $j$  can be found in  $O(2^k)$  time by linear search.*

*Proof.* Immediate from the definition of period and the fact that a decision is nonconstant.  $\square$

Theorem 2.1 says that solving the  $t$ -left-right problem is as hard as inverting the discrete log. In the following we will show how to use the information provided by any decision on  $k$  bits to solve the  $t$ -left-right problem. For every  $j$ ,  $1 \leq j \leq k$ , consider the partition of the set of intervals  $\{I_i^j\}$  into two types, even and odd, depending on whether  $i$  is even or odd. The method is based on the following observation. The roots of quadratic residues in even (odd) intervals at level  $j$  fall in even (odd) intervals at level  $j+1$  (except at interval boundaries). For example, note that the roots of quadratic residues in  $I_0^1$  (the left half) are contained in  $I_0^2$  and  $I_2^2$ , while the roots of quadratic residues in  $I_1^1$  (the right-half) are contained in  $I_1^2$  and  $I_3^2$ .

A decision gives us information about the intervals at some level. To translate this information to information about the first level we repeatedly take square roots as shown by procedure REDUCE and the following lemmas.

For  $z \in [1, p-1]$  and  $0 \leq l \leq k$  define the procedure REDUCE  $(z, l)$ .

Procedure REDUCE  $(z, l)$

```

 $w \leftarrow z$ 
for  $i = 1$  to  $k-l$  do
    if  $w$  is a nonresidue then  $w \leftarrow wg^{-1}$ 
     $w \leftarrow \sqrt{w}$  (either root)
endfor
return  $(w)$ 
    
```

LEMMA 4.3. For any  $z \in [1, p-1]$  and  $0 \leq l \leq k$ , procedure REDUCE can be evaluated in time polynomial in  $n$ .

*Proof.* Since a quadratic nonresidue,  $g$ , is known, the extraction of square roots in the above procedure can be performed in time  $O(n^2)$  [1], [7].  $\square$

We proceed by showing how to use an oracle for a decision on  $k$  bits, with period  $2^l$ , and REDUCE to solve the  $2^k$ -left-right problem. This amounts to deciding, given  $g^x$ , if  $x \in I_0^k$  or  $x \in I_{2^{k-1}}^k$ . If the period,  $2^l$ , is less than  $2^k$  then it will always be the case that  $d(0) = d(2^{k-1})$  so an oracle for such a decision is not of direct use. However, it is possible to use REDUCE to get around this problem.

LEMMA 4.4. Let  $z \in [1, p-1]$  and let  $w = \text{REDUCE}(z, l)$ . Let  $x = \text{index}(z)$  and  $y = \text{index}(w)$ . If  $I^k(x) \equiv 0 \pmod{2^k}$  then  $I^k(y) \equiv 0 \pmod{2^l}$ . If  $I^k(x) \equiv 2^{k-1} \pmod{2^k}$  then  $I^k(y) \equiv 2^{l-1} \pmod{2^l}$ .

*Proof.* Suppose  $I^k(x) \equiv 0 \pmod{2^k}$ . (The other case is similar.)

The case  $l=k$  is clear. Assume the lemma is true for  $l+1$ . Let  $w' = \text{REDUCE}(z, l+1)$  and  $y' = \text{index}(w')$ . If  $y'$  is odd set  $w' \leftarrow w' \cdot g^{-1}$  and  $y' \leftarrow y' - 1$ . (This operation does not change  $I^k(y')$  by Lemma 3.5.) Let  $w = \sqrt{w'}$  and  $y = \text{index}(w)$ . (For example,  $w = \text{REDUCE}(z, l)$  and  $y' \equiv 2y \pmod{p-1}$ .) By the induction hypothesis  $I^k(y') \equiv 0 \pmod{2^{l+1}}$ . Let  $I^k(y') = 2j$ . Since  $2y \in I_{\beta j}^k \subset I_j^{k-1}$ , by Lemma 3.4  $y \in I_j^k$  or  $I_{j+2^{k-1}}^k$ . In other words,  $I^k(y) = I^k(y')/2$  or  $(I^k(y')/2) + 2^{k-1}$ . Therefore  $I^k(y) \equiv 0 \pmod{2^l}$ .  $\square$

Now we are ready to prove Theorem 4.1.

*Proof of Theorem 4.1.* Suppose there exists a perfect oracle  $\theta_d$  for some decision  $d$  on  $k$  bits. Let the period of  $d$  be  $2^l$ . Choose a  $j$  such that  $d(j) \neq d(j+2^{l-1})$ .

We will construct an algorithm for the  $2^k$ -left-right problem. Suppose  $z \in Z_p^*$  such that  $x = \text{index}(z) \in I_0^k \cup I_{2^{k-1}}^k$ . Let  $w = \text{REDUCE}(z, l)$  and  $y = \text{index}(w)$ . By Lemma 4.4  $x \in I_0^k$  if  $I^k(y) \equiv 0 \pmod{2^l}$ . Otherwise,  $x \in I_{2^{k-1}}^k$ . Let  $w' = w \cdot g^{jR^k}$  and  $y' = \text{index}(w')$ .



Then  $d(I^k(y')) = d(I^k(y + jR^k)) = d(I^k(y) + j)$ . If  $I^k(y) \equiv 0 \pmod{2^l}$  then  $d(I^k(y')) = d(j)$ . Otherwise  $d(I^k(y')) = d(j + 2^{l-1})$ . Since  $d(j) \neq d(j + 2^{l-1})$  the query  $\theta_a(w')$  will determine  $I^k(y) \pmod{2^l}$  and thus  $I^k(x) \pmod{2^k}$ .  $\square$

**5. Oracles which make mistakes.** We now consider the case of a nonperfect oracle. For example, there exists an oracle that gives some partial information about the  $k$  bits, but it does not always give the correct answer.

Blum and Micali handled the case of a nonperfect oracle by using a random sampling technique which they called ‘‘Concentrating the Stochastic Advantage.’’ They were able to use this technique to correctly determine the value of their predicate with high probability because they were able to sample over the entire range  $[1, p - 1]$ . In order to extend these results to  $k$  bits using an oracle with an  $\epsilon$ -advantage and the techniques developed in the previous section, it is necessary to be able to evaluate the value of the predicate correctly with high probability on a single interval  $I_i^k$ . Unfortunately, within an interval nothing is known about the proportion of  $x$  for which  $\theta_a(g^x)$  gives the correct value. In [11], we assumed that the error in every interval was bounded. Here we make no such assumptions. It might be the case that  $\theta_a$  gives the wrong answer for every  $x$  in some interval  $I_i^k$ . However, if this occurs there must be some other interval (or intervals) that have a higher percentage of correct answers. Many intervals can be sampled to obtain the necessary information.

As explained below, here we are faced with a continuous analogue of the discrete problem of § 4. In the previous section we used the Boolean values of a decision to extract information required to solve the left-right problem. We can interpret the values of a decision as a probability. For example, if  $d(I^k(x)) = 1$  then the probability that  $\theta_a(g^y) = 1$  for any  $y \in I^k(x)$  is 1 since a perfect oracle always gives the same answer for all values within an interval. When the oracle makes mistakes the probability that the oracle answers 1 on a random value in an interval is no longer either 0 or 1, but could be any value in between. In the following we develop the techniques to extract the information needed to solve the left-right problem from this sequence of probabilities.

DEFINITION 5.1. For  $0 \leq i < 2^k$ , let  $q(i) =$  the fraction of  $y \in I_i^k$  such that  $\theta_a(g^y) = 1$ .

If  $x$  is chosen randomly from  $I_i^k$ , then  $q(i)$  is the probability that  $\theta_a(g^x) = 1$ . We will postpone discussion of how to measure the  $q(i)$  and assume for the moment that given  $g^x$  we can measure  $q(I^k(x))$ .

We will also need the following definition.

DEFINITION 5.2. For  $0 \leq i < 2^k$ , let  $r(i) =$  the fraction of  $y \in I_i^k$  such that  $\theta_a(g^y) = d(i)$ .

If  $x$  is chosen randomly from  $I_i^k$ , then  $r(i)$  is the probability that  $\theta_a(g^x) = d(i)$ . Note that if  $d(i) = 1$  then  $r(i) = q(i)$  and otherwise  $r(i) = 1 - q(i)$ .

We are interested in certain families of subsequences of the  $q(i)$ . The indices of these subsequences are given below.

DEFINITION 5.3. Suppose  $0 \leq m \leq k$  and  $0 \leq j < 2^m$ . Let

$$Q_k(j, m) = (j, j + 2^m, j + 2 \cdot 2^m, \dots, j + (2^{k-m} - 1) \cdot 2^m).$$

The sequence  $Q_k(0, 0)$  is denoted by  $Q_k$ . Also note that  $i \in Q_k(j, m)$  if and only if  $j \equiv i \pmod{2^m}$ .

DEFINITION 5.4. The average value of the probabilities whose indices are given in each sequence  $Q_k(j, m)$  is given by

$$EQ_k(j, m) = \frac{1}{|Q_k(j, m)|} \sum_{i=0}^{2^{k-m}-1} q(j + i \cdot 2^m).$$

The average  $EQ_k(0, 0)$  is denoted by  $EQ_k$ .

In the discrete case a decision had to be nonconstant in order to be able to extract information from it. In the following recursive definition we formalize the continuous analogue.

**DEFINITION 5.5.** Suppose  $0 \leq \gamma \leq 1$ ,  $0 \leq j < 2^k$ , and  $0 \leq m \leq k$ . For  $m = k$  (i.e.,  $|Q_k(j, m)| = 1$ ), then  $Q_k(j, m)$  is  $\gamma$ -constant. For  $m < k$ ,  $Q_k(j, m)$  is  $\gamma$ -constant if and only if  $|EQ_k(j, m+1) - EQ_k(j+2^m, m+1)| \leq \gamma$  and  $Q_k(j, m+1)$  and  $Q_k(j+2^m, m+1)$  are each  $\gamma$ -constant.

The following lemma shows that in a  $\gamma$ -constant subsequence all values are close to the average.

**LEMMA 5.1.** *If  $Q_k(j, l)$  is  $\gamma$ -constant then for all  $i \in Q_k(j, l)$*

$$|EQ_k(j, l) - q(i)| \leq \frac{k-l}{2} \gamma.$$

*Proof.* The proof is by backwards induction on  $l$ .

If  $l = k$  then  $Q_k(j, k) = (j)$  and so  $EQ_k(j, k) = q(j)$ .

Assume the lemma holds for  $l+1$ . Suppose  $Q_k(j, l)$  is  $\gamma$ -constant. Then  $|EQ_k(j, l+1) - EQ_k(j+2^l, l+1)| \leq \gamma$  and  $Q_k(j, l+1)$  and  $Q_k(j+2^l, l+1)$  are both  $\gamma$ -constant. By the induction hypothesis, for all  $i \in Q_k(j, l+1)$

$$|EQ_k(j, l+1) - q(i)| \leq \frac{k-(l+1)}{2} \gamma$$

and for all  $i \in Q_k(j+2^l, l+1)$

$$|EQ_k(j+2^l, l+1) - q(i)| \leq \frac{k-(l+1)}{2} \gamma.$$

But

$$EQ_k(j, l) = \frac{EQ_k(j, l+1) + EQ_k(j+2^l, l+1)}{2}.$$

Therefore, for all  $i \in Q_k(j, l)$

$$|EQ_k(j, l) - q(i)| \leq \frac{\gamma}{2} + \frac{k-(l+1)}{2} \gamma = \frac{k-l}{2} \gamma. \quad \square$$

In § 4 we were able to obtain information from a nonconstant decision. In this section the analogous situation will be that the sequence  $Q_k$  must be nonconstant in the following sense.

**LEMMA 5.2.** *If  $\theta_d$  is an  $\varepsilon$ -advantage oracle for a decision  $d$  then  $Q_k$  is not  $(\varepsilon/k)$ -constant.*

*Proof.* Suppose  $Q_k$  is  $(\varepsilon/k)$ -constant. By Lemma 5.1, for  $0 \leq i < 2^k$ ,  $|q(i) - EQ_k| \leq (\varepsilon/2)$ . Thus  $q(i) \leq EQ_k + (\varepsilon/2)$  and  $1 - q(i) \leq 1 - EQ_k + (\varepsilon/2)$ . Let  $t = 2^k \cdot f_d$  = the number of  $k$ -tuples  $(b_1, b_2, \dots, b_k)$  such that  $d(b_1, b_2, \dots, b_k) = 1$ . Recall that if  $d(i) = 1$  then  $r(i) = q(i)$  and if  $d(i) = 0$  then  $r(i) = 1 - q(i)$ . Hence, there are  $2^k - t$  disjoint pairs  $(i_1, i_2)$  such that  $r(i_1) = q(i_1)$  and  $r(i_2) = 1 - q(i_2)$ . For these pairs  $r(i_1) + r(i_2) \leq EQ_k + (\varepsilon/2) + 1 - EQ_k + (\varepsilon/2) = 1 + \varepsilon$ . For each of the other  $2 \cdot t - 2^k$  numbers  $i$  which are not among these pairs,  $r(i) \leq 1$ . Since the oracle gives the correct answer for the

fraction  $(1/2^k) \sum r(i)$  we have

$$\begin{aligned} f_d + \varepsilon &= \frac{1}{2^k} \sum_{i=0}^{2^k-1} r(i) \leq \frac{1}{2^k} [(2^k - t)(1 + \varepsilon) + 2 \cdot t - 2^k] \\ &= \frac{1}{2^k} [t - t\varepsilon + 2^k\varepsilon] < \frac{1}{2^k} [t + 2^k\varepsilon] = \frac{t}{2^k} + \varepsilon = f_d + \varepsilon \end{aligned}$$

which is a contradiction. Therefore,  $Q_k$  is not  $(\varepsilon/k)$ -constant.  $\square$

Lemma 4.2 showed that a nonconstant decision must have a pair of distinguishable values, i.e., two values that differ by a power of two where the decision is different. The following corollary is the analogue of Lemma 4.2. It shows that there exists two indices that differ by a power of two and distinguish between two subsequences of the sequence  $Q_k$ .

**COROLLARY 5.1.** *If  $\theta_d$  is an  $\varepsilon$ -advantage oracle for a decision  $d$  then there exists numbers  $j$  and  $l$  such that  $0 < l \leq k, j < 2^{l-1}$  and*

$$|EQ_k(j, l) - EQ_k(j + 2^{l-1}, l)| > \frac{\varepsilon}{k}.$$

*Proof.* Follows immediately from Lemma 5.2 and Definition 5.5.  $\square$

We still must answer the questions: How do we measure  $q(i)$  given  $i$  and how do we measure  $q(I^k(x))$  given  $g^x$ ? The answer to these questions is that it is not possible to measure these values exactly. However we can make high precision estimates of the values by using the technique of ‘‘Concentrating the Stochastic Advantage’’ developed by Blum and Micali [5].

This technique is based on the weak law of large numbers. We summarize it here as the

**SAMPLING LEMMA.** *Suppose  $0 \leq \phi < \frac{1}{2}$  and  $0 \leq \delta < \frac{1}{2}$ . Let trials  $(\phi, \delta) = 9/16\delta\phi^2$  and  $M = \text{trials}(\phi, \delta)$ . Let  $Y_i$  be the outcome of an independent random trial that has outcome 1 with probability  $\alpha$  and 0 otherwise. Suppose  $M$  trials are made and the results are recorded as follows. With probability  $1 - 1/M$  let  $X_i = Y_i$  and with probability  $1/M$  let the value of  $X_i$  be set by an adversary. This latter case is known as a defective trial. Let  $S_M = \sum_{i=1}^M X_i$ . Then with probability at least  $1 - \delta$*

$$\left| \frac{S_M + 1}{M} - \alpha \right| \leq \phi.$$

*Note that if none of the trials are defective then  $S_M/M$  is an even better estimate of  $\alpha$ .*

We can use the Sampling Lemma to measure  $q(i)$ .

**LEMMA 5.3.** *Suppose  $\phi > 0$  and  $\psi > 0$ . Given  $i$ ,  $q(i)$  can be measured in time polynomial in  $\phi^{-1}$  and  $\psi^{-1}$  with error at most  $\phi$  with probability at least  $1 - \psi$ .*

*Proof.* Let  $z = g^{L_i^k}$  and  $M = \text{trials}(\phi, \psi)$ . Choose  $M$  random values  $r_1, r_2, \dots, r_M$  with uniform probability such that  $0 \leq r_i \leq R^k$ . Let  $z_i = z \cdot g^{r_i}$ . The indices of the  $z_i$  will be uniformly distributed in  $I_i^k$ . Let  $X_i = \theta_d(z_i)$ . The probability that  $X_i = 1$  is  $q(i)$ . Let  $\bar{q}(i) = (1/M) \sum_{i=1}^M X_i$ . Then by the Sampling Lemma

$$|\bar{q}(i) - q(i)| \leq \phi$$

with probability at least  $1 - \psi$ .  $\square$

Unfortunately, if we are given  $g^x$  and wish to compute  $q(I^k(x))$  the above sampling technique fails. However, under some circumstances the above technique will work.

**LEMMA 5.4.** *Suppose  $\phi > 0$  and  $\psi > 0$ . Suppose  $M = \text{trials}(\phi, \psi)$ . Given  $g^x$  such that  $L_i^k \leq x \leq L_i^k + (R^k/M)$ ,  $q(I^k(x))$  can be measured with error at most  $\phi$  with probability at least  $1 - \psi$ .*

*Proof.* For any  $r_i$  such that  $0 \leq r_i \leq ((M-1)/M)R^k$ ,  $x+r_i \in I_i^k$ . If  $((M-1)/M)R^k < r_i < R^k$  then  $x+r_i$  may not be in  $I_i^k$ . In this latter case the trial may be defective. This occurs with probability  $1/M$ . If we define  $\tilde{q}(i) = (1/M)[1 + \sum_{i=1}^M X_i]$  then by the Sampling Lemma

$$|\tilde{q}(i) - q(i)| \leq \phi$$

with probability at least  $1 - \psi$ .  $\square$

We can use the above sampling techniques to distinguish between  $EQ_k(j, l)$  and  $EQ_k(j + 2^{k-l}, l)$  with high probability.

LEMMA 5.5. *Suppose  $j$  and  $l$  are as in Corollary 5.1. Suppose  $\delta > 0$  and  $\phi > 0$  and let  $M = \text{trials}(\phi, \delta/2^{k-l})$ . Let  $m = I^k(y)$ . Suppose  $m \equiv j \pmod{2^{l-1}}$  and  $L_i^k \leq y \leq L_m^k + (R^k/M)$ . Then given  $g^y$  it is possible to distinguish  $EQ_k(j, l)$  from  $EQ_k(j + 2^{l-1}, l)$  with probability at least  $1 - \delta$ .*

*Proof.* For each interval  $m, m + 2^l, m + 2 \cdot 2^l, \dots, m + (2^{k-l} - 1)2^l$  use Lemma 5.4 to estimate the corresponding probability  $q(m + i \cdot 2^l)$  and let  $\tilde{E} = (1/2^{k-l}) \times \sum_{i=0}^{2^{k-l}-1} \tilde{q}(m + i \cdot 2^l)$ . Let  $E = (1/2^{k-l}) \sum_{i=0}^{2^{k-l}-1} q(m + i \cdot 2^l)$ . We must compare  $\tilde{E}$  with

$$\overline{EQ}_k(j, l) = \frac{1}{|Q_k(j, l)|} \sum_{i=0}^{2^{k-l}-1} \tilde{q}(j + i \cdot 2^l)$$

and  $\overline{EQ}_k(j + 2^{l-1}, l)$  which is defined similarly. These values are estimated using Lemma 5.3. The probability that

$$|\overline{EQ}_k(j, l) - EQ_k(j, l)| \leq 2^{k-l}\phi$$

is at least  $(1 - (\delta/2^{k-l}))^{2^{k-l}} > 1 - \delta$ . Similarly the probability that

$$|\tilde{E} - E| \leq 2^{k-l}\phi$$

is at least  $1 - \delta$ . If  $\phi = (1/2^{k-l})(\epsilon/8k)$  then with probability at least  $1 - \delta$  either  $|\tilde{E} - \overline{EQ}_k(j, l)| \leq \epsilon/4k$  or  $|\tilde{E} - \overline{EQ}_k(j + 2^{l-1}, l)| \leq \epsilon/4k$ . By Corollary 5.1 only one of these will hold with probability at least  $1 - \delta$ .  $\square$

We are now ready to prove Theorem 2.2.

*Proof of Theorem 2.2.* Let  $s = \text{trials}((1/2^{k-l})(\epsilon/8k), \delta/2^{k-l})$ . We will construct an oracle for the  $s$ -left-right problem. The construction is the same as in the proof of Theorem 4.1 except that Lemma 5.5 is used to distinguish  $EQ_k(j, l)$  from  $EQ_k(j + 2^{l-1}, l)$ . The number of samples required to estimate  $\tilde{q}(i)$  and  $\tilde{q}(i)$  is polynomial in  $\delta^{-1}$ ,  $\epsilon^{-1}$ , and  $2^k$ . The number of values estimated is  $O(2^k)$  and all other calculations are polynomial in  $n$  so the time bound follows.  $\square$

**6. Pseudorandom bit generation.** We keep this section informal. It can be easily formalized using the notation of [5] and [17].

It is natural at this point to extend the Blum-Micali pseudorandom bit generator to output  $k$  bits per step. If the current value in the generator is  $g^x$ , then it will output the  $k$  bits hidden by it, namely  $I^k(x) = b_1 b_2 \dots b_k$ . The security of this generator is ensured if for every  $i$ , given  $b_1 b_2 \dots b_i$ , guessing  $b_{i+1}$  with an  $\epsilon$ -advantage is as hard as computing discrete logarithms.

This definition of "hard bits" seems to differ from the one we give in § 2. We are concerned with computing any Boolean function of the  $b_i$ 's. However, a connection between the two definitions was established in [17], at least when  $k = O(\log n)$ . Their result holds for a wide class of generators, but we state it here only for the one above.

THEOREM 6.1 (Vazirani and Vazirani). *Assume  $k = O(\log n)$ . The generator above is secure if and only if for every  $I \subset \{1, 2, \dots, k\}$ , obtaining any partial information about the exclusive or of  $b_i, i \in I$ , is equivalent to computing the discrete log.*

As Theorem 2.2 states that any decision on  $k$  bits, including all these exclusive or's, are hard, it implies the security of this generator.

**7. Beyond  $O(\log n)$  bits.** The problem of showing that the discrete log (or any other function) hides asymptotically more than  $O(\log n)$  bits seems to us a fundamental question. In our proof of Theorem 2.2 the factor  $2^k$  comes into the complexity analysis in three different places, and we have no way of circumventing even one of them. A similar situation occurs in [3], [13] and [17]. We strongly believe that any improvement will involve totally new techniques.

## REFERENCES

- [1] L. ADLEMAN, K. MANDERS AND G. MILLER, *On taking roots in finite fields*, Proc. 18th IEEE Symposium on the Foundations of Computer Science, 1977, pp. 175-178.
- [2] L. ADLEMAN, *A subexponential algorithm for the discrete logarithm problem with applications to cryptography*, Proc. 20th IEEE Symposium on the Foundations of Computer Science, 1979, pp. 55-60.
- [3] W. ALEXI, B. CHOR, O. GOLDREICH AND C. P. SCHNORR, *RSA/Rabin bits are  $\frac{1}{2} + 1/\text{poly}(\log n)$  secure*, Proc. 25th IEEE Symposium on the Foundations of Computer Science, 1984, pp. 449-457.
- [4] E. BERLEKAMP, *Factoring polynomials over large finite fields*, Math. Comput., 24, (1970), pp. 713-735.
- [5] M. BLUM AND S. MICALI, *How to generate cryptographically strong sequences of pseudorandom bits*, Proc. 23rd IEEE Symposium on the Foundations of Computer Science, 1982, pp. 112-117.
- [6] W. DIFFE AND M. HELLMANN, *New directions in cryptography*, IEEE Trans. Inform. Theory IT-22, 6 (1976), pp. 644-654.
- [7] J. FINN, *Probabilistic methods in number-theoretic algorithms and digital signature schemes*, Ph.D. dissertation, Princeton Univ., Princeton, NJ, June 1982.
- [8] S. GOLDWASSER AND S. MICALI, *Probabilistic encryption and how to play mental poker keeping secret all partial information*, Proc. 14th ACM Symposium on the Theory of Computing, 1982, pp. 365-377.
- [9] K. IRELAND AND M. ROSEN, *Elements of Number Theory*, Bogden and Quigley, New York, 1972.
- [10] R. LIPTON, *How to cheat at mental poker*, 1979, unpublished manuscript.
- [11] D. LONG AND A. WIGDERSON, *How discrete is the discrete log?*, Proc. 15th ACM Symposium on the Theory of Computing, 1983, pp. 413-420.
- [12] D. LONG, *The security of bits in the discrete logarithm*, Ph.D. dissertation, Princeton Univ., Princeton, NJ, October 1983.
- [13] R. PERALTA, *Simultaneous security of bits in the discrete log*, 1984, unpublished manuscript.
- [14] S. POHLIG AND M. HELLMAN, *An improved algorithm for computing logarithms over  $GF(p)$  and its cryptographic significance*, IEEE Trans. Inform. Theory IT-24, 1 (1978), pp. 106-110.
- [15] M. RABIN, *Probabilistic algorithms in finite fields*, this Journal, 9 (1980), pp. 273-280.
- [16] A. SHAMIR, R. RIVEST AND L. ADLEMAN, *Mental Poker*, MIT Technical Report, February 1979.
- [17] U. VAZIRANI AND V. VAZIRANI, *Efficient and secure pseudorandom number generation*, Proc. 25th Symposium on the Foundations of Computer Science, 1984, pp. 458-463.
- [18] A. YAO, *Theory and applications of trapdoor functions*, Proc. 23rd Symposium on the Foundations of Computer Science, 1982, pp. 80-91.
- [19] ———, *Protocols for secure computations*, Proc. 23rd Symposium on the Foundations of Computer Science, 1982, pp. 160-164.

## HOW TO CONSTRUCT PSEUDORANDOM PERMUTATIONS FROM PSEUDORANDOM FUNCTIONS\*

MICHAEL LUBY† AND CHARLES RACKOFF‡

**Abstract.** We show how to efficiently construct a pseudorandom invertible permutation generator from a pseudorandom function generator. Goldreich, Goldwasser and Micali ["How to construct random functions," Proc. 25th Annual Symposium on Foundations of Computer Science, October 24–26, 1984.] introduce the notion of a pseudorandom function generator and show how to efficiently construct a pseudorandom function generator from a pseudorandom bit generator. We use some of the ideas behind the design of the Data Encryption Standard for our construction. A practical implication of our result is that any pseudorandom bit generator can be used to construct a block private key cryptosystem which is secure against chosen plaintext attack, which is one of the strongest known attacks against a cryptosystem.

**Key words.** cryptography, pseudorandom, Data Encryption Standard, security

**AMS(MOS) subject classifications.** 68P25, 68Q15, 68Q25

**1. Introduction.** The main result of this paper is a method for efficiently constructing a pseudorandom invertible permutation generator from a pseudorandom function generator. The question of whether pseudorandom permutation generators exist was first posed by Goldreich et al. [GGM]. A practical application of our result is that a pseudorandom bit generator can be used to efficiently construct a block private key cryptosystem which is provably secure against chosen plaintext attack, which is one of the strongest possible attacks known against a cryptosystem. We expect that there will be other applications of pseudorandom invertible permutation generators in cryptographic protocols. Before describing in detail our results, we discuss the basic questions which motivated our work and the partial answers we can give to these questions.

The field of cryptography has changed dramatically over the past few years. Several years ago, cryptography was an art more than a science. Cryptosystem design was based on clever ad hoc ideas. The security of the cryptosystem rested solely on the cleverness of the designer; a cryptosystem was deemed secure until it was broken (in many cases the cryptographer only knew it was broken long after the breaker). Certain design rules were recognized as playing a crucial role in improving security. Eventually, the apparent increase in security obtained by using these design rules became part of the folklore, although there were no formal proofs that security was increased. The foundations of cryptography were not yet established, there was no established formal framework for talking about a cryptographic protocol, security, etc. Thus, there was no formal way to either confirm or dispute the folklore.

More recently, researchers have formalized the notions of cryptographic protocols and security and today cryptography is an integral part of computational complexity. Many cryptosystems whose security is based on mathematical assumptions have come out of this research. However, the security of cryptosystems typically used in practice is still based on the old folklore. This paper uses the new cryptographic rigor to formalize and to analyze some of the previously unexamined folklore.

---

\* Received by the editors November 22, 1985; accepted for publication (in revised form) March 2, 1987.

† Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A4. The work of this author was supported by Natural Sciences and Engineering Research Council of Canada grant A8092 and University of Toronto grant 3-370-126-40.

‡ Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A4. The work of this author was supported by Natural Sciences and Engineering Research Council of Canada grant A3611.

Our primary motivation when we started this work was to investigate the soundness of some of the design rules used in the development of the Data Encryption Standard (hereafter called DES). DES, which was designed to be used as a block private key cryptosystem, was developed by IBM for the U.S. National Bureau of Standards, with an undisclosed amount of highly classified kibbitzing by the super-secret National Security Agency. The reader can consult [De] for more information about DES and [Ba] for more information about the National Security Agency and its role in the development of DES.

DES has features which make it very practical to use. However, one of the properties that DES does not have (as far as we know) is that it is provably secure, even assuming some reasonable mathematical hypothesis. DES certainly has several very clever features, but there is no formal justification that these features help achieve security. We abstract and formalize what we think are some of the more interesting design features of DES to see if they can be used to achieve security or if they are inherently flawed.

The construction of a pseudorandom invertible permutation generator from a pseudorandom function generator is based on one of the main design features of DES. We view this as partial justification for the use of this design rule in DES. What our result intuitively says is that if DES used pseudorandom function generators in its construction then it would be secure when used as a block private key cryptosystem. However, this is a weak justification because the function generators used in DES are not at all pseudorandom. Our result is more of a justification of the use of the design rule than it is a statement about the security of the entire DES system.

Section 2 gives terminology used throughout the remainder of the paper. In § 3, we discuss block private key cryptosystems, describe how DES is used as a block private key cryptosystem, make the connection between secure block private key cryptosystems and pseudorandom invertible permutation generators, and present the design rule of DES which we use to construct a pseudorandom invertible permutation generator from a pseudorandom function generator. In § 4, we give formal definitions of pseudorandom bit generators, pseudorandom function generators and pseudorandom permutation generators and state our main results. In § 5, we give the construction for an invertible permutation generator from a function generator and prove that if the function generator is pseudorandom then the invertible permutation generator is pseudorandom. A preliminary version of these results appears in [LR].

**2. Terminology.** A **string** is a bit string. Let  $\{0, 1\}^n$  be the set of all  $2^n$  strings of length  $n$ . Let  $a$  and  $b$  be two equal length strings. Define  $a \oplus b$  to be the bit-by-bit **exclusive or** of  $a$  and  $b$ , where the resulting string has the same length as  $a$ . Let  $a \bullet b$  denote the **concatenation** of the two strings  $a$  and  $b$ .

Let  $F^n$  be the set of all  $2^{n^2}$  functions mapping  $\{0, 1\}^n$  into  $\{0, 1\}^n$ . Let  $f_1$  and  $f_2$  be functions in  $F^n$ . We use  $f_1 \circ f_2$  to denote the function in  $F^n$  which is the composition of  $f_1$  and  $f_2$ . Let  $P^n \subset F^n$  be the set of such functions that are permutations, i.e., they are 1-1 onto functions.

In all cases, a random choice of an object from a set of objects is such that each object is equally likely to be chosen. For example, a random choice of a string from  $\{0, 1\}^n$  will choose each such string with probability  $1/2^n$ . As another example, a random choice of a function from  $F^n$  will choose each such function with probability  $1/2^{n^2}$ .

**3. Connections between cryptosystems and invertible permutation generators.** Suppose agent  $A$  wants to send plaintext to agent  $B$ .  $A$  wants to do this securely, i.e., in

a way such that any agent  $L$ , who is able to read all the information sent from  $A$  to  $B$ , has no significant idea about the content of the plaintext. A very practical way to achieve this goal is for  $A$  and  $B$  to use a secure block private key cryptosystem. Both  $A$  and  $B$ , but not  $L$ , have the same randomly chosen private key  $k$ . When  $A$  sends plaintext  $M$  to  $B$ ,  $M$  is partitioned into equal length plaintext blocks.  $A$  encrypts each plaintext block into a ciphertext block of the same length using  $k$ .  $B$ , upon receiving a ciphertext block, decrypts it back into the plaintext block using  $k$ .

There are two very attractive features of block private key cryptosystems. First, the total number of bits sent from  $A$  to  $B$  is the minimum number possible, because the number of encrypted bits sent from  $A$  to  $B$  is exactly the same as the number of plaintext bits. Second, since the plaintext is encrypted in blocks, if the encryption of one block is lost in transmission then the other blocks can still be decrypted by  $B$ , whether or not  $B$  knows that a block was lost. Other cryptosystems tend to send many more encrypted bits than there are plaintext bits and/or are not robust to transmission losses.

There is one inherent weakness in any block private key cryptosystem, which is that any listener  $L$  can always tell if exactly the same plaintext block is repeated from the encryption. In our definition of a secure block private key cryptosystem, this is essentially the only insecurity of the system. There are encryption systems which avoid this problem, but these systems tend to be less efficient than a block private key cryptosystem, in certain respects, as explained above.

As far as we know, there is no provably secure block private key cryptosystem. One of the main implications of the results given in this paper is that, under the assumption that there is a pseudorandom bit generator, there is a provably secure block private key cryptosystem.

One of the main motivations of this work is to study the security of DES, when it is used as a block private key cryptosystem. In this context, DES works as follows. Both  $A$  and  $B$ , but not  $L$ , have the same 56-bit private key  $k$ . Suppose  $A$  wants to send plaintext  $M$  to  $B$ .  $A$  first partitions  $M$  into plaintext blocks of 64 bits each.  $A$  encrypts each plaintext block, using the DES encryption algorithm with key  $k$ , into a 64-bit ciphertext block and sends the ciphertext blocks to  $B$ .  $B$  decrypts the ciphertext blocks, using the DES decryption algorithm with key  $k$ , to recover the plaintext blocks. One important property of DES is that, given  $k$ , it is easy to encrypt and decrypt. An important implication of this is that, given  $k$ , each string of length 64 has a unique encryption and a unique decryption.

The encryption algorithm for DES can be thought of as a family of  $2^{56}$  permutations  $h^{64} = \{h_k^{64} : k \in \{0, 1\}^{56}\}$ , where each permutation is a member of  $P^{64}$  indexed by a key  $k$ . Similarly, the decryption algorithm for DES can be thought of as a family of  $2^{56}$  permutations  $\bar{h}^{64} = \{\bar{h}_k^{64} : k \in \{0, 1\}^{56}\}$ , where each permutation is a member of  $P^{64}$  indexed by a key  $k$ . Furthermore,  $h_k^{64} \circ \bar{h}_k^{64}$  and  $\bar{h}_k^{64} \circ h_k^{64}$  are both the identity permutation. DES has the property that, given  $k$  and  $\alpha$ , both  $h_k^{64}(\alpha)$  and  $\bar{h}_k^{64}(\alpha)$  can be computed very efficiently.

One of the strongest attacks known against a cryptosystem is a chosen plaintext attack. We would deem DES secure if it was secure against chosen plaintext attack (which is a stronger notion of security than that given in the first paragraph of this section). Intuitively, in a chosen plaintext attack an agent  $L$ , who does not know the key  $k$ , is nevertheless able to trick  $A$  into sending to  $B$  encryptions of plaintext blocks chosen by  $L$ .  $L$  is allowed to choose a "reasonable" number of plaintext blocks  $M_1, \dots, M_i$ .  $A$  encrypts these plaintext blocks and  $L$  sees the encryptions of these plaintext blocks  $h_k^{64}(M_1), \dots, h_k^{64}(M_i)$ . During this process,  $L$  interactively chooses



the next plaintext block for  $A$  to encrypt based on all previous plaintext blocks and their encryptions. Let  $M_{i+1}$  be a plaintext block chosen by  $A$  which is different from all plaintext blocks chosen by  $L$  during the attack. Intuitively, the cryptosystem is secure against  $L$  if  $L$  cannot predict  $M_{i+1}$  given the information gained from the attack and given the encryption  $h_k^{64}(M_{i+1})$  of  $M_{i+1}$  (but not  $M_{i+1}$ ) “significantly” better than if  $L$  had not received the information obtained from the attack and from seeing  $h_k^{64}(M_{i+1})$ . The cryptosystem is secure against chosen plaintext attack if it is secure against all such agents  $L$ .

The apparent security of DES when it is used as a block private key cryptosystem rests on the fact that DES seems to pass the black box test, which was informally suggested by Turing [Hod]. The black box test is the following:

Say that we have two black boxes, one of which computes a fixed randomly chosen function from  $F^{64}$  and the other computes  $h_k^{64}$  for a fixed randomly chosen  $k$ . Then no algorithm which examines the boxes by feeding inputs to them and looking at the outputs can obtain, in a “reasonable” time, any “significant” idea about which box is which.

If DES passes the black box test then it is secure against a chosen plaintext attack when used as a block private key cryptosystem. We do not give a formal definition of security for a block private key cryptosystem. A rigorous definition in a somewhat different setting appears in [Ra].

The black box test, and therefore security, is very informally stated, i.e., the terms “reasonable” and “significant” are not well defined. The tools of mathematics and computer science are designed to analyze asymptotic behavior; to utilize these tools we introduce an asymptotic version of DES and define security in terms of asymptotic security. We introduce a collection of private key block cryptosystems, one for each possible plaintext block length  $n$ . Let  $h = \{h^n : n \in \mathbb{N}\}$  where, for each  $n$ ,  $h^n$  is the generalization of  $h^{64}$  defined for DES, i.e.,  $h^n$  is used to encrypt plaintext blocks of length  $n$ . Similarly, let  $\bar{h} = \{\bar{h}^n : n \in \mathbb{N}\}$ , where for each  $n$ ,  $\bar{h}^n$  is the generalization of  $\bar{h}^{64}$  defined for DES, i.e.,  $\bar{h}^n$  is used to decrypt plaintext blocks of length  $n$ . Thus, both  $h^n$  and  $\bar{h}^n$  specify for each key  $k$  of a given length a permutation,  $h_k^n \in P^n$ ,  $\bar{h}_k^n \in P^n$ , respectively, where  $h_k^n \circ \bar{h}_k^n$  is the identity permutation. We require that, given  $\alpha \in \{0, 1\}^n$  and a key  $k$  of a given length, both  $h_k^n(\alpha)$  and  $\bar{h}_k^n(\alpha)$  can be computed in time polynomial in  $n$ . In the terminology of § 4, both  $h$  and  $\bar{h}$  are invertible permutation generators. A very similar notion, a function generator, was first formally defined in [GGM]. We give the definition of a function generator in § 4, but intuitively it is the same as the definition of an invertible permutation generator except that each function specified by  $n$  and  $k$  is not necessarily 1-1 onto function, i.e., not necessarily a permutation. Thus, an invertible permutation generator is a special case of a function generator.

It is enough that  $h$  be pseudorandom for  $h$  to be secure against chosen plaintext attack when used as a block private key cryptosystem. The concept of a pseudorandom function generator was introduced in [GGM]. An invertible permutation generator is pseudorandom if it is a pseudorandom function generator. Informally, a pseudorandom function generator is a function generator which passes the black box test for all sufficiently large  $n$ , where  $n$  replaces all occurrences of 64 in the description of the black box test, “reasonable” time corresponds to time polynomial in  $n$  and “significant” corresponds to probability polynomial in  $1/n$ . We give a formal definition of a pseudorandom function generator in § 4.

Goldreich et al. [GGM] show how to construct a pseudorandom function generator

from a pseudorandom bit generator. Pseudorandom function generators have many applications in cryptography, but, unless they are also invertible permutation generators, they cannot be used directly in a block private key cryptosystem as described above. A natural question to ask is can we construct a pseudorandom invertible permutation generator from a pseudorandom function generator. In § 5, we give an efficient construction of an invertible permutation generator based on a function generator and we prove that if the function generator is pseudorandom then so is the invertible permutation generator. This construction uses some of the ideas behind the design of DES. Below, we describe the design details of DES relevant to our construction together with some comments about DES.

DES begins with  $f^{32}$ , where  $f^{32}$  specifies for each key  $k$  of length 48 a function  $f_k^{32} \in F^{32}$ . Alternatively,  $f^{32}$  can be thought of as an algorithm which, given  $k$  and an input  $\alpha$ , computes  $f_k^{32}(\alpha)$ . Let  $L, R \in \{0, 1\}^{32}$ . Define  $g^{64}$  such that for each key  $k$  of length 48,  $g_k^{64}(L \cdot R) = R \cdot [L \oplus f_k^{32}(R)]$ . It is easy to see that  $g_k^{64}$  is 1-1 onto and easy to invert if  $k$  is known. Let  $h^{64}$  be  $g^{64}$  composed with itself 16 times, so that the key length function for  $h^{64}$  is  $768 = 48 \cdot 16$ . Given a 768-bit key  $k$ ,  $h_k^{64}$  is 1-1 onto and easily invertible. We refer to  $h^{64}$  as MDDES for modified DES. MDDES differs from DES in certain inconsequential ways, but also in one way that might be very important: DES only has a 56-bit key, which is used to generate (in a very simple way) the  $16 \cdot 48$ -bit key to be used in MDDES. It is not clear if this makes DES more or less secure than MDDES, but most observers feel that MDDES would be much more secure than DES. In any case, no one has yet succeeded (as far as we know) in breaking DES.

The  $f^{32}$  used in DES is not by any stretch of the imagination pseudorandom. The creators of DES apparently feel that  $f^{32}$  does enough "mixing up" so that, together with the rest of the construction, DES as a whole is secure. Our construction replaces  $f^{32}$  with a pseudorandom function generator  $f$ . Invertible permutation generator  $g$  is constructed from  $f$  as described above. The invertible permutation generator  $h$  is formed by composing  $g$  three times with itself (instead of 16 times as in MDDES). We prove in § 5 that  $h$  formed in this way is a pseudorandom invertible permutation generator if  $f$  is a pseudorandom function generator.

**4. Formal definitions and statement of results.** In this section we present some necessary formal definitions and state the main results of the paper.

**4.1. Pseudorandom bit generators.** The original definitions for pseudorandom bit generators are due to Blum and Micali [BM] and are generalized to those given here by Yao [Yao]. A **bit generator** is a collection of functions  $f = \{f^n: n \in \mathbb{N}\}$  such that  $f^n$  maps  $\{0, 1\}^n$  into  $\{0, 1\}^{t(n)}$ , where  $t(n) \geq n + 1$  (for example,  $t(n) = n^3$ ) and, given  $n$  and  $\alpha$ ,  $f^n(\alpha)$  can be computed in time polynomial in  $n$  (this implies that  $t(n)$  is polynomial in  $n$ ). Informally,  $f$  is pseudorandom if there is no (probabilistic) polynomial in  $n$  time algorithm which, for infinitely many  $n$ , can significantly distinguish a string  $\beta = f^n(\alpha)$ , where  $\alpha$  is a randomly chosen from  $\{0, 1\}^n$ , from a string randomly chosen from  $\{0, 1\}^{t(n)}$ . Formally,  $f$  is **pseudorandom** if there is no distinguishing circuit family for  $f$ . A distinguishing circuit family for  $f$  is an infinite family of Boolean circuits consisting of and/or/not unbounded fan-out gates (for readers unfamiliar with this model of computation, replace all occurrences of a family of Boolean circuits with a probabilistic polynomial time algorithm and all the results are the same)  $C = \{C_{n_1}, C_{n_2}, \dots\}$ , where  $n_1 < n_2 < \dots$ , such that for some pair of constants  $s$  and  $c$ , for each  $n$  for which there is a circuit  $C_n$ :

(1) The size of  $C_n$  is less than or equal to  $n^s$ . The size of a circuit is the total number of wires plus the number of gates in the circuit.

(2) The input to  $C_n$  is a string of length  $t(n)$  and the output is a single bit.

(3) Let  $r_n$  be the probability that the output bit of  $C_n$  is one when the input to  $C_n$  is a string randomly chosen from  $\{0, 1\}^{t(n)}$ . Let  $p_n$  be the probability that the output bit of  $C_n$  is 1 when a string  $\alpha$  is randomly chosen from  $\{0, 1\}^n$  and the input is  $f^n(\alpha)$ . Let  $d_n = |p_n - r_n|$  be the distinguishing probability for  $C_n$ . Then,  $d_n \geq 1/n^c$ .

It is important to note that if  $f$  is a pseudorandom bit generator, then  $f$  is a pseudorandom bit generator even if we allow distinguishing circuit families for  $f$  to be probabilistic. A probabilistic circuit  $C_n$ , besides the inputs described above, has an additional number of input bits whose values are randomly chosen. It is not hard to see that there is some way of fixing these additional bits to  $C_n$ , forming a deterministic circuit  $C'_n$ , such that the distinguishing probability for  $C'_n$  is at least as great as the distinguishing probability for  $C_n$ , i.e.,  $d'_n \geq d_n$ . Hence, if  $f$  has a distinguishing probabilistic circuit family then  $f$  has a distinguishing deterministic circuit family, which implies that if  $f$  has no distinguishing deterministic circuit family then  $f$  has no distinguishing probabilistic circuit family.

Whether or not pseudorandom bit generators exist is an open question. Blum and Micali [BM] introduce conditions which are sufficient for constructing pseudorandom bit generators. They show how to construct a pseudorandom bit generator based on the assumption that the Discrete Log problem is hard. Yao [Yao] generalizes these conditions and shows how to construct a pseudorandom bit generator based on the assumption that the factoring problem is hard. A series of results has improved the efficiency of the Yao construction [ACGS], [BCS], [GMT], and [VV]. Levin [Le] introduces weaker conditions which are sufficient to construct a pseudorandom bit generator.

**4.2. Pseudorandom function generators.** The concept of a pseudorandom function generator is defined in [GGM]. They give a construction for a pseudorandom function generator using a pseudorandom bit generator. We give the definition of a pseudorandom function generator in this section. Let  $l(n)$  be polynomial in  $n$ . A **function generator** with key length function  $l(n)$  is a collection  $f = \{f^n : n \in \mathbb{N}\}$ , where  $f^n$  specifies for each key  $k$  of length  $l(n)$  a function  $f_k^n \in F^n$ . It is required that, given a key  $k \in \{0, 1\}^{l(n)}$ , and a string  $\alpha \in \{0, 1\}^n$ ,  $f_k^n(\alpha)$  can be computed in time polynomial in  $n$ .

Informally,  $f$  is pseudorandom if there is no polynomial time in  $n$  algorithm which, for infinitely many  $n$ , is able to even slightly distinguish whether a function was randomly chosen from  $f^n$  or from  $F^n$  after seeing polynomial in  $n$  input/output pairs of the function, even when the algorithm is allowed to choose the next input based on all previously seen input/output pairs. Formally,  $f$  is **pseudorandom** if there is no distinguishing circuit family for  $f$ . A distinguishing circuit family for  $f$  is an infinite family of circuits  $\{C_{n_1}, C_{n_2}, \dots\}$ , where  $n_1 < n_2 < \dots$ , such that for some pair of constants  $s$  and  $c$ , for each  $n$  for which there is a circuit  $C_n$ :

(1)  $C_n$  is an acyclic circuit which contains Boolean gates of type **and**, **or** and **not** (these gates are interpreted in the usual way, i.e., the **and** gate computes the “and” of the two inputs). In addition there are constant gates of type **zero** and **one**. Each constant gate has no inputs, and the output is 0 if it is a **zero** gate and 1 if it is a **one** gate.  $C_n$  also contains **oracle gates**. Each oracle gate has an input and an output which are both strings of length  $n$ . Each oracle gate is to be evaluated using some function from  $F^n$ , which for now is left unspecified and is to be thought of as a variable of the circuit. All gates can fan-out their output bits to an unbounded number of other gates. The output of  $C_n$  is a single bit. Such a circuit is called an **oracle circuit**.

(2) The size of  $C_n$  is less than or equal to  $n^s$ . The size of  $C_n$  is the total number

of connections between gates, Boolean gates, constant gates and oracle gates.

(3) We let  $\Pr[C_n(F^n)]$  be the probability that the output bit of  $C_n$  is one when a function is randomly chosen from  $F^n$  and used to evaluate the oracle gates. We let  $\Pr[C_n(f^n)]$  be the probability that the output bit of  $C_n$  is one when a key  $k$  of length  $l(n)$  is randomly chosen and  $f_k^n$  is used to evaluate the oracle gates. The distinguishing probability for  $C_n$ ,  $|\Pr[C_n(F^n)] - \Pr[C_n(f^n)]|$ , is greater than or equal to  $1/n^c$ .

**THEOREM [GGM].** *If there is a pseudorandom bit generator then there is a pseudorandom function generator.*

In fact, Goldreich et al. [GGM] give an explicit construction for a function generator  $f$  based on a bit generator  $g$  and prove that if  $g$  is a pseudorandom bit generator then  $f$  is a pseudorandom function generator. The converse of this theorem is easily seen to be true.

**4.3. Pseudorandom invertible permutation generators and block private key cryptosystems.** A **permutation generator**  $f$  is a function generator such that each function  $f_k^n$  is 1-1 onto. Let  $\bar{f} = \{\bar{f}^n : n \in \mathbb{N}\}$ , where  $\bar{f}^n = \{\bar{f}_k^n : k \in \{0, 1\}^{l(n)}\}$ , where  $\bar{f}_k^n$  is the inverse function of  $f_k^n$ . We say  $f$  is **invertible** if  $\bar{f}$  is also a permutation generator. In this case,  $\bar{f}$  is the inverse permutation generator for  $f$  (of course,  $f$  is the inverse permutation generator for  $\bar{f}$  also). We say  $f$  is **pseudorandom** if it is pseudorandom as a function generator as defined in the previous section. (The definition of pseudorandom is provably no different if we replace  $F^n$  with  $P^n$  in part 3 of the definition of an oracle circuit.) In § 5, we give a construction for a pseudorandom invertible permutation generator based on a pseudorandom function generator.

**4.4. Super pseudorandom invertible permutation generators.** There is even a stronger notion of pseudorandom for invertible permutation generators which is very natural. Let  $f$  be an invertible permutation generator. We say that  $f$  is a **super pseudorandom** if there is no super distinguishing circuit family for  $f$ . A super distinguishing circuit family for  $f$  is an infinite family of circuits  $C = \{C_{n_1}, C_{n_2}, \dots\}$  where  $n_1 < n_2 < \dots$ , where each circuit is an oracle circuit containing two types of oracle gates, **normal** and **inverse**. For some pair of constants  $s$  and  $c$ , for each  $n$  for which there is a circuit  $C_n$ :

(1) The size of  $C_n$  is less than or equal to  $n^s$ .

(2) We let  $\Pr[C_n(P^n)]$  be the probability that the output bit of  $C_n$  is one when a permutation  $\sigma$  is randomly chosen from  $P^n$  and  $\sigma$  and  $\bar{\sigma}$  are used to evaluate normal and inverse gates in  $C_n$ , respectively, where  $\bar{\sigma}$  is the inverse permutation of  $\sigma$ . We let  $\Pr[C_n(f^n)]$  be the probability that the output bit of  $C_n$  is one when a key  $k$  of length  $l(n)$  is randomly chosen and  $f_k^n$  and  $\bar{f}_k^n$  are used to evaluate the normal and inverse oracle gates in  $C_n$ , respectively. The distinguishing probability for  $C_n$ ,  $|\Pr[C_n(P^n)] - \Pr[C_n(f^n)]|$ , is greater than or equal to  $1/n^c$ .

We state in § 5 that there is a super pseudorandom invertible permutation generator if there is a pseudorandom function generator. Although every super pseudorandom invertible generator is also a pseudorandom invertible generator, we show in § 5 that the converse is not necessarily true.

Let  $f$  be a super pseudorandom invertible permutation generator. We can use  $f$  as described above in a block private key cryptosystem. This cryptosystem is secure against chosen plaintext/ciphertext attack, which is an even stronger attack than chosen plaintext attack. In chosen plaintext/ciphertext attack,  $L$  can interactively choose plaintext blocks and see their encryptions and choose encryptions and see their corresponding plaintext blocks. Thus,  $L$  is allowed to attack the cryptosystem from

“both ends.” Let  $M$  be a plaintext block which is different than all plaintext blocks seen in the attack. Intuitively, the cryptosystem is secure against chosen plaintext/ciphertext attack if for all agents  $L$ , given the information  $L$  gained from the attack and given the encryption  $f_k^n(M)$  of  $M$  (but not  $M$ ),  $L$  cannot predict the value of  $M$  any better than if  $L$  had no information about  $M$ . Symmetrically, let  $E$  be the encryption of a plaintext block which is different than all encryptions of plaintext blocks seen in the attack: The cryptosystem is secure against chosen plaintext/ciphertext attack if for all agents  $L$ , given the information  $L$  gained from the attack and given the decryption  $\tilde{f}_k^n(E)$  of  $E$  (but not  $E$ ),  $L$  cannot predict the value of  $E$  any better than if  $L$  had no information about  $E$ .

**4.5. Definition of cryptographic composition.** In this section, we formally define the composition of function generators. Let  $g$  and  $h$  be two function generators, where  $l_1(n)$  and  $l_2(n)$  are the key length functions for  $g$  and  $h$ , respectively (we allow the possibility that  $g = h$ ). Then  $f = g \circ h$  is a function generator defined as follows.

- (1) The key length function for  $f$  is  $l(n) = l_1(n) + l_2(n)$ .
- (2)  $f_{k_2 \cdot k_1}^n = h_{k_2}^n \circ g_{k_1}^n$ , where  $k_1$  is of length  $l_1(n)$  and  $k_2$  is of length  $l_2(n)$ .

Note that if both  $g$  and  $h$  are (invertible) permutation generators then so is  $f$ .

**5. Generating permutations from functions.** In this section we show how to construct a pseudorandom invertible permutation generator from a pseudorandom function generator (see § 4 for definitions). A natural thing to try is an abstraction of MDDES. Let  $f = \{f^n\}$  be a pseudorandom function generator where the key length function is  $l(n)$ . Define a function generator  $g = \{g^n\}$  in terms of  $f$  as follows. Let  $k$  be a string of length  $l(n)$ , let  $k'$  be a string of length  $l(n + 1)$ , let  $L, R$  and  $L'$  be strings of length  $n$  and let  $R'$  be a string of length  $n + 1$ . Then

$$g_k^{2n}(L \cdot R) = R \cdot [L \oplus f_k^n(R)],$$

$$g_{k'}^{2n+1}(L' \cdot R') = R' \cdot [L' \oplus \text{first } n \text{ bits of } f_{k'}^{n+1}(R')].$$

Note that  $g$  is an invertible permutation generator. The inverse permutation generator for  $g, \bar{g}$ , is computed as follows. Let  $\alpha, \beta$  and  $\beta'$  be strings of length  $n$  and let  $\alpha'$  be a string of length  $n + 1$ . Then

$$\bar{g}_k^{2n}(\alpha \cdot \beta) = [\beta \oplus f_k^n(\alpha)] \cdot \alpha,$$

$$\bar{g}_{k'}^{2n+1}(\alpha' \cdot \beta') = [\beta' \oplus \text{first } n \text{ bits of } f_{k'}^{n+1}(\alpha')] \cdot \alpha'.$$

Thus,

$$\bar{g}_k^{2n}(g_k^{2n}(L \cdot R)) = L \cdot R,$$

$$\bar{g}_{k'}^{2n+1}(g_{k'}^{2n+1}(L' \cdot R')) = L' \cdot R'.$$

Let  $h = g \circ g \circ g$ . Since  $g$  is an invertible permutation generator,  $h$  is also an invertible permutation generator. Theorem 1 shows that  $h$  is pseudorandom if  $f$  is pseudorandom. Before proving this theorem, we show that  $h = g \circ g$  is not at all pseudorandom, and then we state and prove the main lemma used in the proof of Theorem 1. The proof of the main lemma, which is a combinatorial lemma not based on any unproven assumptions, is the interesting and difficult portion of the proof of the theorem.

We show that  $h = g \circ g$  is *not* pseudorandom as follows. Let  $k_1$  and  $k_2$  be keys of length  $l(n)$ . For all strings  $L_1, L_2$  and  $R$  of length  $n$ , the  $\oplus$  of the first  $n$  bits of  $h_{k_2 \cdot k_1}^{2n}(L_1 \cdot R)$  and  $h_{k_2 \cdot k_1}^{2n}(L_2 \cdot R)$  is equal to  $L_1 \oplus L_2$ . Thus, a distinguishing circuit  $C_{2n}$  for  $h^{2n}$  can be described as follows. Choose  $L_1$  and  $L_2$  such that  $L_1 \neq L_2$  and let  $R$  be

any string of length  $n$ , e.g., a string of  $n$  zeros.  $C_{2n}$  has two oracle gates: the input to the first oracle gate is  $L_1 \bullet R$  and the input to the second oracle gate is  $L_2 \bullet R$ . The output of  $C_{2n}$  is 1 iff the  $\oplus$  of the first  $n$  bits of the outputs from the two oracle gates is equal to  $L_1 \oplus L_2$ . Thus, the output of  $C_{2n}$  is always 1 when the oracle gates are evaluated using a function in  $h^{2n}$ . On the other hand, if the oracle gates are evaluated using a function randomly chosen from  $F^{2n}$ , the output of  $C_{2n}$  is 1 with probability  $1/2^n$ .

The following definitions are used in the main lemma and in Theorem 1. We define the operator  $H : F^n \times F^n \times F^n \rightarrow P^{2n}$  as follows. Let  $f_0, f_1$  and  $f_2$  be three functions from  $F^n$ . Let  $L$  and  $R$  be strings of length  $n$ . Define

$$\alpha = f_0(R), \quad \beta = f_1(L \oplus \alpha), \quad \gamma = f_2(R \oplus \beta),$$

$$H(f_2, f_1, f_0)(L \bullet R) = [R \oplus \beta] \bullet [L \oplus \alpha \oplus \gamma].$$

$H$  is derived from three applications of the DES design rule described in § 3. The main lemma states that any Boolean circuit with  $m$  oracle gates, where each oracle gate has an input and an output which are both strings of length  $2n$ , can distinguish with probability at most  $m^2/2^n$  between the case when the oracle gates are evaluated using a randomly chosen function from  $F^{2n}$  and the case when the oracle gates are evaluated using  $H(f_2, f_1, f_0)$  and  $f_2, f_1$  and  $f_0$  are randomly chosen from  $F^n$ . This result is surprising in the sense that it uses no unproven assumptions and that the number of distinct permutations generated by  $H$  is at most  $2^{3n}$ , which is a very small fraction of the  $2^{2n}$  functions in  $F^{2n}$ . On the other hand, it is not so surprising given that we allow the distinguishing circuit to examine only a very small portion of the domain of the function (there are only  $m$  oracle gates).

**PROPOSITION.** *Let  $D_n$  be an oracle circuit with  $m$  oracle gates and size  $s$ , where the input and output of each oracle gate is a string of length  $n$  and where  $m \leq 2^n$ .  $D_n$  can be easily modified to another circuit  $D'_n$  which never repeats an input value to an oracle gate such that: (1) For each function  $f_0 \in F^n$ , the output bit of  $D'_n$  when  $f_0$  is used to evaluate the oracle gates is exactly the same as the output bit of  $D_n$  when  $f_0$  is used to evaluate the oracle gates; (2) The number of oracle gates in  $D'_n$  is  $m$ ; (3) The size of  $D'_n$  is at most  $s^3$ .*

*Proof.* Omitted but easy.  $\square$

**MAIN LEMMA.** *Let  $C_{2n}$  be an oracle circuit with  $m$  oracle gates such that no input value is repeated to an oracle gate as in the above proposition. Then,  $|\Pr [C_{2n}(F^{2n})] - \Pr [C_{2n}(H(F^n, F^n, F^n))]| \leq m^2/2^n$ .*

*Proof.* There are really two experiments occurring in different probability spaces in the statement of the Main Lemma. For  $\Pr [C_{2n}(F^{2n})]$  the sample space is  $F^{2n}$  with the uniform probability distribution, and for  $\Pr [C_{2n}(H(F^n, F^n, F^n))]$  the sample space is  $F^n \times F^n \times F^n$  with the uniform probability distribution. To be able to analyze the behavior of  $C_{2n}$  with respect to these two probability distributions we introduce a new probability space with sample space  $\Omega = \{0, 1\}^{3nm}$  and the uniform probability distribution, i.e., for all  $\omega \in \Omega$ ,  $\Pr [\omega] = \frac{1}{2^{3nm}}$ . We define two random variables  $A, B$  on the new probability space with the following properties:

(1)  $A : \Omega \rightarrow \{0, 1\}, B : \Omega \rightarrow \{0, 1\}$ .

(2)  $E[A] = \Pr [C_{2n}(F^{2n})]$ . (The left-hand side is with respect to the new probability space, the right-hand side is with respect to the probability space for the original experiment.)

(3)  $E[B] = \Pr [C_{2n}(H(F^n, F^n, F^n))]$ . (Same remarks as for (2).)

(4)  $|E[A] - E[B]| \leq m^2/2^n$ .

Note that  $|E[A] - E[B]| = |\sum_{\omega \in \Omega} (A(\omega) - B(\omega))|/2^{3nm}$ . Our objective is to define  $A$  and  $B$  with properties (1), (2) and (3) such that for most sample points  $A$  and  $B$  take on the same value, so that property (4) is also satisfied. A sample point in  $\Omega$  is a string  $\omega = \omega_1, \dots, \omega_{3nm}$ . For  $1 \leq i \leq m$  we define the random variables  $X_i: \Omega \rightarrow \{0, 1\}^n$ ,  $Y_i: \Omega \rightarrow \{0, 1\}^n$ ,  $Z_i: \Omega \rightarrow \{0, 1\}^n$  as follows:

$$X_i(\omega) = \omega_{(i-1)n+1} \bullet \dots \bullet \omega_{(i-1)n+n},$$

$$Y_i(\omega) = \omega_{mn+(i-1)n+1} \bullet \dots \bullet \omega_{mn+(i-1)n+n},$$

$$Z_i(\omega) = \omega_{2mn+(i-1)n+1} \bullet \dots \bullet \omega_{2mn+(i-1)n+n}.$$

We define the vector  $X(\omega) = \langle X_1(\omega), \dots, X_m(\omega) \rangle$ .  $Y(\omega)$  and  $Z(\omega)$  are defined similarly.

At sample point  $\omega \in \Omega$ ,  $B(\omega)$  is defined as the output bit of  $C_{2n}$  when the oracle gates of  $C_{2n}$  are computed as described below. This description also defines  $L(\omega)$  and  $R(\omega)$ , where  $L$  and  $R$  are vectors of  $m$  random variables corresponding to the left half of the inputs to the oracle gates and the right half of the inputs to the oracle gates, respectively. In addition,  $\alpha'(\omega)$ ,  $\beta'(\omega)$  and  $\gamma'(\omega)$  are defined, where  $\alpha'$ ,  $\beta'$  and  $\gamma'$  are each vectors of  $m$  random variables defined by the following circuit. The  $i$ th oracle gate of  $C_{2n}$  is computed as follows:

**$B$ -gate $_i$ :**

The input is  $L_i(\omega) \bullet R_i(\omega)$ ,  
 $l \leftarrow \min \{j: 1 \leq j \leq i \text{ and } R_i(\omega) = R_j(\omega)\}$ ,  
 $\alpha'_i(\omega) \leftarrow L_i(\omega) \oplus X_i(\omega)$ ,  
 $l \leftarrow \min \{j: 1 \leq j \leq i \text{ and } \alpha'_i(\omega) = \alpha'_j(\omega)\}$ ,  
 $\beta'_i(\omega) \leftarrow R_i(\omega) \oplus Y_i(\omega)$ ,  
 $l \leftarrow \min \{j: 1 \leq j \leq i \text{ and } \beta'_i(\omega) = \beta'_j(\omega)\}$ ,  
 $\gamma'_i(\omega) \leftarrow \alpha'_i(\omega) \oplus Z_i(\omega)$ ,  
 The output is  $\beta'_i(\omega) \bullet \gamma'_i(\omega)$ .

The circuit defining  $B$  can be thought of as a circuit with Boolean **and**, **or** and **not** gates and constant **zero** and **one** gates. The input to the entire circuit can be thought of as  $X(\omega)$ ,  $Y(\omega)$  and  $Z(\omega)$ .

We now describe a random variable  $B'$  which is exactly the same as  $B$  except that it is the output bit of  $C_{2n}$  when the  $i$ th oracle gate is computed as follows:

**$B'$ -gate $_i$ :**

The input is  $L_i(\omega) \bullet R_i(\omega)$ ,  
 $l \leftarrow \min \{j: 1 \leq j \leq i \text{ and } R_i(\omega) = R_j(\omega)\}$ ,  
 $\alpha'_i(\omega) \leftarrow L_i(\omega) \oplus X_i(\omega)$ ,  
 $l \leftarrow \min \{j: 1 \leq j \leq i \text{ and } \alpha'_i(\omega) = \alpha'_j(\omega)\}$ ,  
 $Y'_i(\omega) \leftarrow Y_i(\omega) \oplus R_i(\omega)$ ,  
 $\beta'_i(\omega) \leftarrow R_i(\omega) \oplus Y'_i(\omega)$ ,  
 $l \leftarrow \min \{j: 1 \leq j \leq i \text{ and } \beta'_i(\omega) = \beta'_j(\omega)\}$ ,  
 $Z'_i(\omega) \leftarrow Z_i(\omega) \oplus \alpha'_i(\omega)$ ,  
 $\gamma'_i(\omega) \leftarrow \alpha'_i(\omega) \oplus Z'_i(\omega)$ ,  
 The output is  $\beta'_i(\omega) \bullet \gamma'_i(\omega)$ .

$B$  and  $B'$  are exactly the same except that  $Y'_i$  and  $Z'_i$  are computed from  $Y_i$  and  $Z_i$

and used in place of  $Y_i$  and  $Z_i$  in the oracle gate computation. The definition of  $B'$  also redefines the vectors of random variables  $L, R, \alpha', \beta', \gamma'$ .

FACT.  $E[B'] = E[B]$ . This follows because the random variables  $X_i, Y_i$  and  $Z_i$  are identically and uniformly distributed and because  $R_i$  and  $\alpha'_i$  do not depend on  $Y_i$  and  $Z_i$ , respectively. Thus,  $Y'_i$  and  $Z'_i$  are independently and randomly chosen strings from  $\{0, 1\}^n$ .

We now describe the computation of  $B'$  in an equivalent way. This alternative description is used in all further references to  $B'$ .  $B'$  is the output of  $C_{2n}$  when the  $i$ th oracle gate in  $C_{2n}$  is evaluated as follows:

**$B'$ -gate $_i$ :**

The input is  $L_i(\omega) \bullet R_i(\omega)$ ,  
 $u_i(\omega) \leftarrow \min \{j : 1 \leq j \leq i \text{ and } R_i(\omega) = R_j(\omega)\}$ ,  
 $\alpha'_i(\omega) \leftarrow L_i(\omega) \oplus X_{u_i(\omega)}(\omega)$ ,  
 $v_i(\omega) \leftarrow \min \{j : 1 \leq j \leq i \text{ and } \alpha'_i(\omega) = \alpha'_j(\omega)\}$ ,  
 $\beta'_i(\omega) \leftarrow R_i(\omega) \oplus R_{v_i(\omega)}(\omega) \oplus Y_{v_i(\omega)}(\omega)$ ,  
 $w_i(\omega) \leftarrow \min \{j : 1 \leq j \leq i \text{ and } \beta'_i(\omega) = \beta'_j(\omega)\}$ ,  
 $\gamma'_i(\omega) \leftarrow \alpha'_i(\omega) \oplus \alpha'_{w_i(\omega)}(\omega) \oplus Z_{w_i(\omega)}(\omega)$ ,  
 The output is  $\beta'_i(\omega) \bullet \gamma'_i(\omega)$ .

From the above discussion, it is clear that  $E[B'] = \Pr [C_{2n}(H(F^n, F^n, F^n))]$ . In addition to defining  $L, R, \alpha', \beta'$  and  $\gamma'$  in terms of  $X, Y$  and  $Z$ , this also defines the vectors  $u, v$  and  $w$  in terms of  $X, Y$  and  $Z$ .

Let  $A$  be the random variable which is defined to be the output of  $C_{2n}$  when the oracle gates are evaluated exactly the same way as in the definition of  $B'$  except that the output of the  $i$ th oracle gate is  $Y_i(\omega) \bullet Z_i(\omega)$ . This defines different vectors of random variables  $L, R, \alpha', \beta', \gamma', u, v$  and  $w$  in terms of  $X, Y$  and  $Z$  then those defined for  $B'$ . In the following, the vectors of random variables we are considering are explicitly mentioned as needed. Because  $A$  is determined by  $C_{2n}$  when the output values from each oracle gate are independently and identically distributed random variables and because  $C_{2n}$  never repeats an input value to an oracle gate,  $E[A] = \Pr [C_{2n}(F^{2n})]$ . Our goal now is to show that  $|E[A] - E[B']| \leq m^2/2^n$ .

DEFINITION.  $\omega \in \Omega$  is **preserving** if for  $1 \leq i \leq m$ ,  $v_i(\omega) = i$  and  $w_i(\omega) = i$ , where  $v_i$  and  $w_i$  are the random variables defined with respect to  $A$ .

CLAIM. For all  $\omega \in \Omega$ , if  $\omega$  is preserving then  $A(\omega) = B'(\omega)$ .

Justification. If for  $1 \leq i \leq m$ ,  $v_i(\omega) = i$  and  $w_i(\omega) = i$ , then for  $1 \leq i \leq m$ ,  $\beta'_i(\omega) = Y_i(\omega)$  and  $\gamma'_i(\omega) = Z_i(\omega)$ , where all these random variables are those defined with respect to  $A$ . Thus, all the outputs of the oracle gates are the same as though they were calculated as in  $B'$ , and thus all the variables defined with respect to  $A$  have exactly the same values as the corresponding variables defined with respect to  $B'$ .

For all  $\omega \in \Omega$ , if  $\omega$  is preserving then  $A(\omega) - B'(\omega) = 0$  and even if  $\omega$  is not preserving then  $|A(\omega) - B'(\omega)| \leq 1$ . Thus,  $|E[A] - E[B']| \leq \Pr [\omega \text{ is not preserving}]$ . In the following discussion, all variables are defined with respect to  $A$ .

DEFINITION. For all  $\omega \in \Omega$ ,  $Y(\omega)$  is **bad** if there is  $i, j, 1 \leq j < i \leq m$ , such that  $Y_i(\omega) = Y_j(\omega)$ . It is not hard to verify that  $\Pr [Y \text{ is bad}] \leq m^2/2^{n+1}$ .

DEFINITION. For all  $\omega \in \Omega$ ,  $X(\omega)$  is **bad** if there is  $i, j, 1 \leq j < i \leq m$ , such that  $\alpha'_i(\omega) = \alpha'_j(\omega)$ .

CLAIM.  $\Pr [X \text{ is bad}] \leq m^2/2^{n+1}$ .

Proof. For  $1 \leq i \leq m$ , let  $\tilde{y}_i$  and  $\tilde{z}_i$  be strings of length  $n$ , let  $\tilde{y} = \langle \tilde{y}_1, \dots, \tilde{y}_m \rangle$  and let  $\tilde{z} = \langle \tilde{z}_1, \dots, \tilde{z}_m \rangle$ . Let  $\Omega_{\tilde{y}, \tilde{z}} = \{\omega \in \Omega : Y(\omega) = \tilde{y} \text{ and } Z(\omega) = \tilde{z}\}$ . We show that for each



$\tilde{y}, \tilde{z}$ ,  $\Pr[X(\omega) \text{ is bad} \mid \Omega_{\tilde{y}, \tilde{z}}] \leq m^2/2^{n+1}$  which implies that  $\Pr[X \text{ is bad}] \leq m^2/2^{n+1}$ .  $Y, Z, L, R$  and  $u$  are constants  $\tilde{y}, \tilde{z}, \tilde{l}, \tilde{r}$  and  $\tilde{u}$  on  $\Omega_{\tilde{y}, \tilde{z}}$ , respectively. This is by definition for  $Y$  and  $Z$ , and it is true for  $L$  and  $R$  because the outputs from all oracle gates are constant on  $\Omega_{\tilde{y}, \tilde{z}}$ , which implies that the inputs to all oracle gates are also constant.  $u$  is constant because it depends only on  $R$ . On the other hand,  $X$  is a vector of  $m$  independently and identically distributed random variables on the probability space restricted to  $\Omega_{\tilde{y}, \tilde{z}}$ , and each such variable has uniform distribution on  $\{0, 1\}^n$ . Let  $eqn(i, j)$  be the equation  $\alpha'_j(\omega) = \alpha'_i(\omega)$ . On  $\Omega_{\tilde{y}, \tilde{z}}$ , this equation reduces to

$$\tilde{l}_j \oplus X_{\tilde{u}_j}(\omega) = \tilde{l}_i \oplus X_{\tilde{u}_i}(\omega).$$

When  $\tilde{u}_j = \tilde{u}_i$ ,  $\Pr[eqn(i, j) \text{ is satisfied} \mid \Omega_{\tilde{y}, \tilde{z}}] = 0$  because  $\tilde{u}_j = \tilde{u}_i$  implies that  $\tilde{r}_j = \tilde{r}_i$  which implies that  $\tilde{l}_j \neq \tilde{l}_i$  (because  $C_{2n}$  does not repeat an input value to an oracle gate) which implies that  $eqn(i, j)$  is not satisfied (because  $eqn(i, j)$  simplifies to  $\tilde{l}_j = \tilde{l}_i$  in this case). When  $\tilde{u}_j \neq \tilde{u}_i$ ,  $\Pr[eqn(i, j) \text{ is satisfied} \mid \Omega_{\tilde{y}, \tilde{z}}] = 1/2^n$  because  $\tilde{u}_j \neq \tilde{u}_i$  together with the fact that  $X$  is a vector of independent random variables on  $\Omega_{\tilde{y}, \tilde{z}}$  implies that for fixed  $X_{\tilde{u}_i}$ , the probability that a randomly chosen  $X_{\tilde{u}_j}$  satisfies the equation is  $1/2^n$ . Thus, summing these probabilities over all  $m(m-1)/2$  equations yields  $\Pr[X(\omega) \text{ is bad} \mid \Omega_{\tilde{y}, \tilde{z}}] \leq m^2/2^{n+1}$ .

CLAIM. For all  $\omega \in \Omega$ ,  $X(\omega)$  is not bad and  $Y(\omega)$  is not bad implies that  $\omega$  is preserving.

*Proof.*  $X(\omega)$  is not bad implies that for  $1 \leq j < i \leq m$ ,  $\alpha'_j(\omega) \neq \alpha'_i(\omega)$  which implies that for  $1 \leq i \leq m$ ,  $v_i(\omega) = i$  which implies that for  $1 \leq i \leq m$ ,  $\beta'_i(\omega) = Y_i(\omega)$ .  $Y(\omega)$  is not bad and for  $1 \leq i \leq m$ ,  $\beta'_i(\omega) = Y_i(\omega)$  implies that for  $1 \leq j < i \leq m$ ,  $\beta'_j(\omega) \neq \beta'_i(\omega)$  which implies that for  $1 \leq i \leq m$ ,  $w_i(\omega) = i$ .

CLAIM.  $\Pr[\omega \text{ is not preserving}] \leq \Pr[X \text{ is bad or } Y \text{ is bad}] \leq m^2/2^n$ . Thus,  $|E[A] - E[B']| \leq m^2/2^n$  and the Main Lemma follows.  $\square$

THEOREM 1.  $h = g \circ g \circ g$  is a pseudorandom invertible permutation generator.

*Proof.* What must be shown is that  $h$  is pseudorandom, i.e., there is no distinguishing circuit family for  $h$ . The proof is by contradiction. Assume that there is a distinguishing circuit family  $C = \{C_{2n_1}, C_{2n_2}, \dots\}$  where  $n_1 < n_2 < \dots$ , for  $h$ . We show this implies there is a distinguishing circuit family  $D = \{D_{n_i} : i \in \mathbb{N}\}$  for  $f$ , contradicting the fact that  $f$  is a pseudorandom function generator. In particular, for a fixed  $n$  we show that if  $C_{2n}$  distinguishes  $h^{2n}$  from  $F^{2n}$  with probability at least  $1/n^c$  then there is a circuit  $D_n$  (where  $D_n$  is not much bigger than  $C_{2n}$ ) which distinguishes  $f^n$  from  $F^n$  with probability at least  $1/4n^c$ . (The proof for  $C_{2n+1}$  is similar). Let us first state the definition of  $h^{2n}$  in a different way. If  $k_0, k_1$  and  $k_2$  are strings of length  $l(n)$  then

$$h_{k_2 \cdot k_1 \cdot k_0}^{2n} = H(f_{k_2}^n, f_{k_1}^n, f_{k_0}^n).$$

Let  $p_0^H, p_1^H, p_2^H, p_3^H$  be  $\Pr[C_{2n}(H(f^n, f^n, f^n))]$ ,  $\Pr[C_{2n}(H(f^n, f^n, F^n))]$ ,  $\Pr[C_{2n}(H(f^n, F^n, F^n))]$ ,  $\Pr[C_{2n}(H(F^n, F^n, F^n))]$ , respectively. Let  $p^R = \Pr[C_{2n}(F^{2n})]$ . The main lemma shows that  $|p^R - p_3^H| \leq m^2/2^n$ , where  $m$  is the number of oracle gates in  $C_{2n}$ . Since  $m$  is less than or equal to  $n^s$  for some constant  $s$ ,  $m^2/2^n \leq 1/4n^c$  for sufficiently large  $n$ . Since

$$1/n^c \leq |p^R - p_0^H| \leq |p^R - p_3^H| + |p_3^H - p_2^H| + |p_2^H - p_1^H| + |p_1^H - p_0^H|,$$

there is an  $i \in \{0, 1, 2\}$  such that  $|p_{i+1}^H - p_i^H| \geq 1/4n^c$ . The cases when  $i = 0$  and  $i = 2$  are similar to the case when  $i = 1$  and are omitted. For the case  $i = 1$ , we use  $C_{2n}$  to

construct a probabilistic circuit  $D_n$  not much larger than  $C_{2n}$  which will distinguish  $f^n$  from  $F^n$  with probability at least  $1/4n^c$ .  $D_n$  is the same as  $C_{2n}$  except that it first randomly chooses  $k_2 \in \{0, 1\}^{l(n)}$  and  $f_0 \in F^n$ , and then each oracle gate  $X$  in  $C_{2n}$  is replaced with a subcircuit which computes  $H(f_{k_2}, X', f_0)$ , where  $X'$  is an oracle gate with an input and output which are both strings of length  $n$ . The main problem with constructing  $D_n$  is how to randomly choose  $f_0$  and  $F^n$ . In fact,  $D_n$  does not specify  $f_0$  at all (since a complete description would involve  $n2^n$  bits). Instead,  $D_n$  remembers all inputs and outputs to  $f_0$ . When  $D_n$  is to compute  $f_0(\alpha)$  for some string  $\alpha$  of length  $n$ , if  $\alpha$  is the same as a previous input then the same output as before is given, otherwise  $\alpha$  is a new input and then  $f_0(\alpha)$  is a randomly chosen string from  $\{0, 1\}^n$ .

When the oracle gates in  $D_n$  are evaluated using  $f_{k_1}^n$ , where  $k_1$  is a randomly chosen key of length  $l(n)$ , then  $p_1^H = \Pr$  [The output of  $D_n$  is 1]. When the oracle gates in  $D_n$  are evaluated using  $f_1$ , where  $f_1$  is randomly chosen from  $F^n$ , then  $p_2^H = \Pr$  [The output of  $D_n$  is 1]. Since  $|p_2^H - p_1^H| \geq 1/4n^c$ ,  $D_n$  distinguishes  $F^n$  from  $f^n$  with probability greater than or equal to  $1/4n^c$ .  $\square$

**THEOREM 2.** *Let  $g$  be defined in terms of pseudorandom function generator  $f$  as described in the beginning of this section. Then  $h = g \circ g \circ g \circ g$  is a super pseudorandom invertible permutation generator.*

*Proof.* The proof of this theorem is very similar to the proof of Theorem 1 and is omitted.  $\square$

It is interesting to note that  $h = g \circ g \circ g$  is *not* super pseudorandom, although Theorem 1 shows that it is pseudorandom. This can be seen as follows. Let  $k_1, k_2$  and  $k_3$  be keys of length  $l(n)$ . A distinguishing circuit  $C_{2n}$  for  $h^{2n}$  can be described as follows. Let  $L_1, L_2$  and  $R$  be strings of length  $n$  such that  $L_1 \neq L_2$ .  $C_{2n}$  has two normal oracle gates: the input to the first normal oracle gate is  $L_1 \bullet R$  and the input to the second normal oracle gate is  $L_2 \bullet R$ . Let  $S_1 \bullet T_1$  and  $S_2 \bullet T_2$  be the outputs of these two normal oracle gates respectively.  $C_{2n}$  also has an inverse oracle gate with input  $S_2 \bullet [T_2 \oplus L_1 \oplus L_2]$ . The output of  $C_{2n}$  is 1 if and only if the last  $n$  bits of the output from this inverse oracle gate is equal to  $R \oplus S_1 \oplus S_2$ . It can be verified that the output of  $C_{2n}$  is always 1 when the normal oracle gates and inverse oracle gates are computed by first choosing a key  $k$  at random and then using  $h_k^{2n}$  for the normal oracle gates and  $\bar{h}_k^{2n}$  to compute the inverse oracle gates. On the other hand, if the oracle gates are computed using a permutation randomly chosen from  $P^{2n}$ , the output of  $C_{2n}$  is 1 with probability  $1/2^n$ .

**6. Conclusions.** Assuming the existence of a pseudorandom function generator, we have proven the existence of a super pseudorandom invertible permutation generator. This is of interest by itself, but the construction and the proof can be viewed as a partial justification of some of the methodology used in the design of DES.

Another methodology which is important both in DES and in the cryptographic literature is using cryptographic composition of permutation generators to **increase** security. In [LR] we define a **measure** of security for permutation generators, and prove that if one composes two permutation generators which are slightly secure the result is more secure than either one alone.

**Acknowledgments.** We thank Oded Goldreich for suggesting the notion of super pseudorandom and for finding a problem with an initial construction we had for a pseudorandom invertible permutation generator. We thank Johan Hastad for simplifying the proof of the Main Lemma. We thank Paul Beame for numerous suggestions which kept us on the right track.

## REFERENCES

- [ACGS] W. ALEXI, B. CHOR, O. GOLDREICH AND C. P. SCHNORR, *RSA and Rabin functions: Certain parts are as hard as the whole*, SIAM J. Comput., 17 (1988), pp. 194–209; preliminary version in Proceedings of the 25th IEEE Symposium on Foundations of Computer Science, New York, 1984, pp. 449–457.
- [Ba] J. BAMFORD, *The Puzzle Palace: A Report on America's Most Secret Agency*, Penguin, 1983.
- [BCS] M. BEN-OR, B. CHOR AND A. SHAMIR, *On the cryptographic security of single RSA bits*, in Proceedings of the 15th ACM Symposium on Theory of Computing, Boston, 1983, pp. 421–430.
- [BM] M. BLUM AND S. MICALI, *How to generate cryptographically strong sequences of pseudo-random bits*, in Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, November 3–5, 1982, pp. 112–117; preliminary version in SIAM J. Comput., 13 (1984), 850–886.
- [De] D. DENNING, *Cryptography and Data Security*, Addison-Wesley, Reading, MA, January 1983.
- [GGM] O. GOLDREICH, S. GOLDWASSER AND S. MICALI, *How to construct random functions*, in Proceedings of the 25th Annual Symposium on Foundations of Computer Science, October 24–26, 1984.
- [GMT] S. GOLDWASSER, S. MICALI AND P. TONG, *Why and how to establish a private code on a public network*, in Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, New York, 1982, pp. 134–144.
- [H] A. HODGES, *Alan Turing: The Enigma of Intelligence*, Unwin Paperbacks, 1985.
- [Le] L. A. LEVIN, *One-way functions and pseudorandom generators*, in Proceedings of the 17th ACM Symposium on Theory of Computing, Providence, RI, 1985, pp. 363–365.
- [LR] M. LUBY AND C. RACKOFF, *Pseudorandom permutation generator and cryptographic composition*, Proc. 18th Annual Symposium on Theory of Computing, May 28–30, 1986.
- [Ra] C. RACKOFF, *Class Notes on Cryptography*, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, 1985.
- [VV] U. V. VAZIRANI AND V. V. VAZIRANI, *Efficient and secure pseudo-random number generation*, in Proceedings of the 25th IEEE Symposium on Foundations of Computer Science, 1984, pp. 458–463.
- [Yao] A. C. YAO, *Theory and applications of trapdoor functions*, 23rd Annual Symposium on Foundations of Computer Science, November 3–5, 1982, pp. 80–91.

## A PIPELINE ARCHITECTURE FOR FACTORING LARGE INTEGERS WITH THE QUADRATIC SIEVE ALGORITHM\*

CARL POMERANCE†, J. W. SMITH‡ AND RANDY TULER‡

**Abstract.** We describe the quadratic sieve factoring algorithm and a pipeline architecture on which it could be efficiently implemented. Such a device would be of moderate cost to build and would be able to factor 100-digit numbers in less than a month. This represents an order of magnitude speed-up over current implementations on supercomputers. Using a distributed network of many such devices, it is predicted much larger numbers could be practically factored.

**Key words.** pipeline architecture, factoring algorithms, quadratic sieve

**AMS(MOS) subject classifications.** 11Y05, 68M05

**1. Introduction.** The problem of efficiently factoring large composite numbers has been of interest for centuries. It shares with many other basic problems in the sciences the twin attributes of being easy to state, yet (so far) difficult to solve. In recent years, it has also become an applied science. In fact, several new public-key cryptosystems and signature schemes, including the RSA public-key cryptosystem [10], base their security on the supposed intractability of the factoring problem.

Although there is no known polynomial time algorithm for factoring, we do have subexponential algorithms. Over the last few years there has developed a remarkable six-way tie for the asymptotically fastest factoring algorithms. These methods all have the common running time

$$(1) \quad L(N) = \exp \{ (1 + o(1)) \sqrt{\ln N \ln \ln N} \}$$

to factor  $N$ . It might be tempting to conjecture that  $L(N)$  is in fact the true complexity of factoring, but no one seems to have any idea how to obtain even heuristic lower bounds for factoring. The six methods are as follows:

- (i) The elliptic curve algorithm of Lenstra [6];
- (ii) The class-group algorithm of Schnorr-Lenstra [11];
- (iii) The linear sieve algorithm of Schroepfel (see [1] and [8]);
- (iv) The quadratic sieve algorithm of Pomerance [8], [9];
- (v) The residue list sieve algorithm of Coppersmith, Odlyzko and Schroepfel [1];
- (vi) The continued fraction algorithm of Morrison-Brillhart [7].

It might be pointed out that none of these methods have actually been proved to have the running time  $L(N)$ , but rather there are heuristic arguments that give this function as the common running time. The heuristic analyses of the latter four algorithms use a new (rigorous) elimination algorithm of Wiedemann [13] for sparse matrices over finite fields. (This method may in fact be a practical tool; this is discussed further below.)

It should also be pointed out that to achieve the running time  $L(N)$ , method (vi) uses a weak form of the elliptic curve method as a subroutine.

As a last comment, the elliptic curve method has  $L(N)$  as a worst-case running time, while the other five methods have  $L(N)$  as a typical running time. The worst

---

\* Received by the editors December 1, 1985; accepted for publication March 2, 1987. This research was partially supported by a National Science Foundation grant.

† Department of Mathematics, University of Georgia, Athens, Georgia 30602.

‡ Department of Computer Science, University of Georgia, Athens, Georgia 30602.

case for the elliptic curve method is when  $N$  is the product of two primes of about the same length. Of course, this is precisely the case of interest for cryptography.

A proof that  $L(N)$  is the correct asymptotic complexity for factoring would of course be sensational. One corollary would be that  $P \neq NP$ . It seems to us, however, that of equal importance to practicing cryptographers is a complexity bound valid for finite values of  $N$ . Of course, such a bound necessarily enters into the amount of resources one is willing to invest in the problem.

To be specific, what is the largest number of decimal digits such that any number with this many or fewer digits can be factored in a year using equipment that would cost \$10,000,000 to replace? The dollar amount is not completely arbitrary; it is the order of magnitude of the cost of a new supercomputer. An answer to this question is not a fundamental and frozen constant, but rather a dynamic figure that reflects the state of the art at a given point in time. Nevertheless, it is an answer to precisely this kind of question that practicing cryptographers need.

Given an actual experiment where the running time is less than a year, performance for a full year can be extrapolated using the expression (1) (but ignoring the " $o(1)$ "). The factoring algorithms listed above are all "divide and conquer," that is, with  $k$  identical computers assigned to the problem, the number will be factored about  $k$  times as fast. Thus if an actual experiment involves equipment costing less than \$10,000,000, again one can extrapolate.

Here are two examples. In 1984, the Sandia National Laboratories team of Davis, Holdridge and Simmons [3] factored a 71-digit number in 9.5 hours on a Cray XMP computer. Moreover, the algorithm they used, the quadratic sieve, has running time  $L(N)$ . Since

$$\frac{1 \text{ year}}{9.5 \text{ hours}} \approx \frac{L(10^{101})}{L(10^{71})},$$

we thus predict that the Sandia group could factor any 101-digit number in a year-long run on a Cray XMP.

The other example comes from some recent work of Silverman who has implemented the quadratic sieve algorithm on a distributed network of 9 SUN 3 workstations. Although at retail a SUN 3 is fairly expensive, it may be fairer for these purposes to use a wholesale price for a stripped down version, say \$5000 each. With this system, which we value at \$45,000, Silverman was able to factor an 81-digit number in one week. Since

$$\frac{1 \text{ year}}{1 \text{ week}} \cdot \frac{10,000,000}{45,000} \approx \frac{L(10^{126})}{L(10^{81})},$$

we predict that he would be able to factor any 126-digit number in a year-long run with 2000 SUN 3's.

In this paper we shall describe a machine which should cost about \$50,000 to build and which should be able to factor 100-digit numbers in two weeks. This custom-designed processor will implement the quadratic sieve algorithm. Since

$$\frac{1 \text{ year}}{2 \text{ weeks}} \cdot \frac{10,000,000}{50,000} \approx \frac{L(10^{144})}{L(10^{100})},$$

we predict that with \$10,000,000 any 144-digit number could be factored in a year. The cost to factor a 200-digit number in a year with this strategy would be about  $10^{11}$  dollars—or only 5 percent of the current U.S. national debt!

**2. Combination of congruences factorization algorithms.** To properly describe the factoring project, it is necessary to begin with a description of the quadratic sieve (qs)

as it will be implemented. This is in fact a fairly complex algorithm with many stages. Some of these stages are common to other factoring algorithms, other stages are specific to qs.

The qs algorithm belongs to a family of factoring algorithms known as combination of congruences. With these, the basic goal in factoring  $N$  is to find two squares  $X^2, Y^2$  with  $X^2 \equiv Y^2 \pmod N$ . If these squares are found in a random or pseudorandom fashion and if  $N$  is composite, then with probability at least  $\frac{1}{2}$ , the greatest common factor  $(X - Y, N)$  will give a nontrivial factorization of  $N$ . This greatest common factor can be computed very rapidly using Euclid's algorithm.

The two congruent squares  $X^2$  and  $Y^2$  are constructed from auxiliary congruences of the form

$$(2) \quad u_i^2 \equiv v_i^2 w_i \pmod N, \quad u_i^2 \neq v_i^2 w_i.$$

If some nonempty set of indices  $I$  can be found such that  $\prod_{i \in I} w_i$  is a square, we can let

$$X \equiv \prod_{i \in I} u_i \pmod N,$$

$$Y \equiv \left( \prod_{i \in I} v_i \right) \left( \prod_{i \in I} w_i \right)^{1/2} \pmod N.$$

This special set of indices  $I$  can be found using the techniques of linear algebra. In fact, if the number  $w_i$  appearing in (2) has the prime factorization

$$w_i = (-1)^{\alpha_{0,i}} \prod_{j=1}^{\infty} p_j^{\alpha_{j,i}},$$

where  $p_j$  denotes the  $j$ th prime and there are only finitely many  $j$  for which  $\alpha_{j,i} > 0$ , then let  $\vec{\alpha}_i$  denote the vector  $(\alpha_{0,i}, \alpha_{1,i}, \dots)$ . Then the following two statements are equivalent for finite sets of indices  $I$ :

- (i)  $\prod_{i \in I} w_i$  is a square;
- (ii)  $\sum_{i \in I} \vec{\alpha}_i \equiv \vec{0} \pmod 2$ .

Thus the algorithmic problem of finding  $I$  (when given a collection of integers  $w_i$  and their prime factorizations) is reduced to the problem of finding a nontrivial linear dependency among the vectors  $\vec{\alpha}_i$  over the finite field with two elements.

For this method to work, we need the prime factorizations of the numbers  $w_i$  appearing in (2). It is the difficulty in finding enough completely factored  $w_i$  so that a linear dependency may be found among the  $\vec{\alpha}_i$  that is the major bottleneck in the combination of congruences family of algorithms.

This bottleneck is ameliorated by discarding those  $w_i$  that do not completely factor with small primes. This is a good plan for two reasons. First, those  $w_i$  with a large prime factor probably will not be involved in a dependency. Second, those  $w_i$  that can be factored completely over the small primes can be seen as having this property with a not unreasonable amount of work. The set of small primes used is fixed beforehand and is called the "factor base."

**3. The quadratic sieve algorithm.** To make the above factoring scheme into an algorithm, we need a systematic method of generating congruences of the shape (2) and a systematic method of recognizing which  $w_i$  can be factored completely over the factor base. In the qs algorithm, we use parametric solutions of (2). That is, we exhibit three polynomials  $u(x), v(x), w(x)$  such that for each integral  $x$  we obtain a solution

of (2) with  $u(x) = u_i, v(x) = v_i, w(x) = w_i$ . Moreover, those  $x$  for which  $w(x)$  completely factors over the factor base can be quickly found by a sieve procedure described below.

In the original description of the qs algorithm [8], only one triple of polynomials  $u(x), v(x), w(x)$  was used, namely

$$u(x) = [\sqrt{N}] + x, \quad v(x) = 1, \quad w(x) = ([\sqrt{N}] + x)^2 - N.$$

It was later found more advantageous to work with a family of triples of polynomials. The Sandia implementation of qs used a scheme of Davis and Holdridge [2]: If  $p$  is a large prime and  $w(x_0) \equiv 0 \pmod p, 0 \leq x_0 < p$ , then let

$$u_p(x) = u(x_0 + xp), \quad v_p(x) = 1, \quad w_p(x) = w(x_0 + xp).$$

A further refinement of this idea in [9] that has never been implemented is to choose  $x_1$  with  $w(x_1) \equiv 0 \pmod{p^2}, 0 \leq x_1 < p^2$ , and

$$u_{p^2}(x) = u(x_1 + xp^2), \quad v_{p^2}(x) = p, \quad w_{p^2}(x) = \frac{1}{p^2} w(x_1 + xp^2).$$

A different and somewhat better scheme for choosing multiple polynomials was suggested by Peter Montgomery (see [9] and [12]). This method has been implemented by Silverman and is the method that we too shall use. Suppose we know beforehand that we will only be dealing with a polynomial  $w(x)$  for values of  $x$  in  $[-M, M]$ , where  $M$  is some large, but fixed integer. Then we choose quadratic polynomials  $w(x)$  that “fit” this interval well. That is,  $w(M) \approx w(-M) \approx -w(0)$  and these approximately common values are as small as possible.

This task is done as follows. First choose an integer  $a$  with

$$(3) \quad a^2 \approx \sqrt{2N}/M$$

and such that  $b^2 \equiv N \pmod{a^2}$  is solvable. Let  $b, c$  be integers with

$$(4) \quad b^2 - N = a^2c, \quad |b| < a^2/2.$$

Then we let

$$(5) \quad u(x) = a^2x + b, \quad v(x) = a, \quad w(x) = a^2x^2 + 2bx + c.$$

How do we determine which values of  $x$  in  $[-M, M]$  give a number  $w(x)$  that completely factors over the factor base? Fix the factor base as the primes  $p \leq B$  for which

$$(6) \quad t^2 \equiv N \pmod p$$

is solvable. The parameter  $B$  is fixed at the beginning of the program. We shall only recognize those values of  $w(x)$  not divisible by any prime power greater than  $B$ . Presumably, if  $B$  is large enough compared with typical values of  $|w(x)|$  (typical values are about  $\sqrt{N} M$ ), most values that factor completely with the primes up to  $B$  will, in fact, not be divisible by any prime power greater than  $B$ .

For each prime power  $q = p^\alpha \leq B$  where  $p$  is in the factor base, solve the congruence

$$(7) \quad w(x) \equiv 0 \pmod q$$

and list the solutions  $A_q^1, A_q^2, \dots, A_q^{k(q)}$ . The number of solutions  $k(q)$  is either 1, 2 or 4. (Almost always we have  $k(q) = 2$ .)

Next we compute integral approximations to the numbers  $\log |w(x)|$  for  $x \in [-M, M]$ . Because of the relationships of  $a, b, c, M, N$  to each other given by (3) and (4), the values of  $[\log |w(x)|]$  tend to stay constant on long subintervals of  $[-M, M]$ . For example, it is about  $[\log (M\sqrt{N}/2)]$  for  $|x| < \sqrt{1/4} M$  and  $\sqrt{3/4} M < |x| < M$ . It is about  $[\log (M\sqrt{N}/2)] - 1$  for  $\sqrt{1/4} M < |x| < \sqrt{3/8} M$  and  $\sqrt{5/8} M < |x| < \sqrt{3/4} M$ , etc. Thus not only is this an easy computation for one choice of polynomial, but the results are virtually the same for each polynomial.

If  $q = p^\alpha \leq B$  where  $p$  is a prime in the factor base, let  $\Lambda(q) = \log p$ . We compute single precision values for the  $\Lambda(q)$ . This computation can be done once; it is independent of the polynomial used. For other integers  $m$  let  $\Lambda(m) = 0$ . Then every prime power factor of  $w(x)$  is at most  $B$  if and only if  $\log |w(x)| = \sum_{m|w(x)} \Lambda(m)$ .

We are now ready to sieve. A memory addressed by integers  $x$  in  $[-M, M)$  is initialized to zero. For each power  $q \leq B$  of a prime in the factor base and each  $j \leq k(q)$ , we retrieve the numbers in those memory locations addressed by integers  $x \equiv A_q^j \pmod q$ , add  $\Lambda(q)$  to the number there, and put this result back in the same place. This is what we call sieving and it is of central importance to the qs algorithm. In pseudocode it may be described as follows:

For each power  $q \leq B$  of a prime in the factor base and

$j \in \{1, \dots, k(q)\}$  do:

Let  $A = A_q^j + \lceil (-M - A_q^j)/q \rceil q$  (thus  $A$  is the first number in  $[-M, M)$  which is  $\equiv A_q^j \pmod q$ )

While  $A < M$  do:

$D \leftarrow S(A)$

$S(A) \leftarrow D + \Lambda(q)$ ;  $A \leftarrow A + q$ .

After sieving, we scan the  $2M$  memory locations for values near  $\log |w(x)|$ . Such a location  $x$  most likely corresponds to a value of  $w(x)$ , all of whose prime power factors are at most  $B$ . It is possible that there could be some false reports or some locations that should have been reported that are missed. This error, which is introduced from our approximate logarithms, should be negligible in practice.

**4. Fine points.** In §§ 2 and 3 we described the basic qs algorithm using Montgomery's polynomials. In this section we shall give some adornments to the basic algorithm which should speed things up.

**Use of a multiplier.** We may wish to replace  $N$  in (3), (4) and (6) with  $kN$  where  $k$  is some small, fixed, positive, square-free integer. This trick, which goes back to Kraitchik [5, p. 208], can sometimes speed up implementation by a factor of 2 or 3. The idea is to skew the factor base towards smaller primes. However, there is a penalty in that the values  $|w(x)|$  are larger by a factor of  $\sqrt{k}$ . We balance these opposing forces with the function

$$f(k, N) = -\frac{1}{2} \log k + \sum_{p \leq B} E_p^{(k)},$$

where the sum is over primes  $p \leq B$ ,  $E_p^{(k)} = 0$  if  $t^2 \equiv kN \pmod p$  is not solvable, and

$$E_p^{(k)} = \begin{cases} \frac{2 \log p}{p-1}, & p \text{ odd and } p \nmid k, \\ \frac{\log p}{p}, & p \text{ odd and } p \mid k, \\ \frac{1}{2} \log 2, & p = 2 \text{ and } kN \equiv 2 \text{ or } 3 \pmod 4, \\ \log 2, & p = 2 \text{ and } kN \equiv 5 \pmod 8, \\ 2 \log 2, & p = 2 \text{ and } kN \equiv 1 \pmod 8, \end{cases}$$

if  $t^2 \equiv kN \pmod p$  is solvable (see [9]).

When presented with a number  $N$  to factor, we first find the value of  $k$  which maximizes  $f(k, N)$ . In practice one can assume  $k < 100$ . Also in practice, we may replace  $B$  in the definition of  $f(k, N)$  with a smaller number, say 1000.



**Small prime variation.** If  $q \leq B$  is a power of a prime in the factor base, then the time it takes to sieve with one  $A_q^i$  is proportional to  $M/q$ . Thus we spend more time with a smaller  $q$ , less with a larger  $q$ . However, small  $q$ 's do not contribute very much. For example,

$$\sum_{q < 30} \Lambda(q) < 42,$$

which is usually small compared to  $\log |w(x)|$ . In addition, it is usually not the case that every prime less than 30 is in the factor base and it usually is the case that a number which factors over the factor base is not divisible by all the factor base primes below 30. Thus only a small error is introduced by forgetting to sieve with the prime powers below 30. By lowering the threshold which causes a value  $w(x)$  to be reported, no fully factored values need be lost. The only penalty is possibly a few more false reports. In fact, even this should not occur (see [9]). The small prime variation might save 20 percent of the running time.

**Large prime variation.** If  $x$  is such that

$$\log |w(x)| - \sum_{m|w(x)} \Lambda(m) < 2 \log B,$$

then either  $w(x)$  completely factors with the primes in the factor base, or there is some large prime  $p$ ,

$$B < p < B^2,$$

such that  $w(x)/p$  completely factors with the primes in the factor base. Thus, by again lowering the threshold for reports, we can catch these values of  $w(x)$  as well. For such a value to be eventually part of a linear dependency (see § 2), there must be at least one other report  $w_1(x_1)$  using the same large prime  $p$ . The birthday paradox suggests that duplication of large primes  $p$  should not be so uncommon. In practice we shall probably only try to use those  $w(x)$  which factor with a large prime  $p < 100 B$ . The large prime variation (also used by Silverman [12]) speeds up the algorithm by about a factor of 2 or 3. However, the larger we take  $B$ , for a fixed  $N$ , the less useful will be this variation. It should also be noted that the large prime variation has been implemented in other combinations of congruences algorithms as well.

**Generation of polynomials and sieve initialization data.** Both experience [3], [12] and theory [9] suggest that it is advantageous to change polynomials fairly often. The reason for this is as follows. On  $[-M, M)$ , the largest values of  $|w(x)|$  are about  $M\sqrt{N}/2$ . Moreover, more than half of the values are at least half this big. However, the larger is  $|w(x)|$ , the less likely it will factor completely over the factor base. Thus it would be advantageous to choose a rather small  $M$ . But  $M$  is directly proportional to the time spent sieving  $w(x)$ , so a smaller  $M$  translates to less time spent per polynomial.

Since we will want to change polynomials often, we should learn to do this efficiently. For each polynomial  $w(x) = a^2x^2 + 2bx + c$  given by (5) we shall need to find  $a, b, c$  satisfying (3) and (4) and we shall need to solve all of the congruences (7). One last criterion is that  $a$  should be divisible either by a prime greater than the large prime bound or by two primes greater than  $B$  so that we do not get duplicate solutions of (2) from different polynomials.

One possibility, suggested in [9] and implemented in [12], is to choose  $a$  as a prime  $\approx (\sqrt{2N}/M)^{1/2}$  with  $t^2 \equiv N \pmod{a^2}$  solvable. Then we may let the solution for  $t$  least in absolute value be  $b$  and choose  $c = (b^2 - N)/a^2$ . Since  $a$  is a prime, the

quadratic congruence is easily solved. This then gives a legal polynomial  $w(x)$  and it remains to solve the congruences (7) for all the powers  $q \leq B$  of a prime in the factor base. By the quadratic formula, these roots (for  $q$  coprime to  $2a$ ) are given by the expression

$$(8) \quad (-b \pm \sqrt{N})a^{-2} \pmod{q},$$

where  $\sqrt{N}$  is interpreted as a residue  $t_q \pmod{q}$  with  $t_q^2 \equiv N \pmod{q}$  and where  $a^{-2}$  is the multiplicative inverse of  $a^2 \pmod{q}$ . The numbers  $t_q$  may be stored of course, since they are used in each polynomial. However, since each polynomial uses a new value of  $a$  in this scheme, it appears that an inversion mod  $q$  (using the extended g.c.d. algorithm) is necessary for each  $q$  and for each polynomial.

Thanks to a suggestion from Peter Montgomery, sieve initialization can in fact be accomplished with much less computation. The idea is to choose  $a$  as a product of  $l$  primes  $g \approx (\sqrt{2N}/M)^{1/(2l)}$  with  $t^2 \equiv N \pmod{g^2}$  solvable. (The value of  $l$  here is quite small, we plan to use  $l=3$  or  $4$ .) Say  $a = g_1 \cdots g_l$ . Using known solutions  $\pm b_i$  of the congruences  $t^2 \equiv N \pmod{g_i^2}$  for  $i = 1, \dots, l$ , we may assemble via the Chinese Remainder Theorem  $2^l$  different values of  $b \pmod{a^2}$  which satisfy  $b^2 \equiv N \pmod{a^2}$ . Since  $b$  and  $-b$  will give essentially the same polynomial  $w(x)$ , we will get  $2^{l-1}$  different polynomials for the one value of  $a$ .

Suppose now we use  $r$  primes  $g_1, \dots, g_r$  and we form different values of  $a$  by choosing  $l$  of these primes. Thus there are  $\binom{r}{l}$  values of  $a$  and thus  $2^{l-1}\binom{r}{l}$  different polynomials. If  $a = g_{i_1} \cdots g_{i_l}$ , then

$$(9) \quad a^{-2} \pmod{q} \equiv g_{i_1}^{-2} \cdots g_{i_l}^{-2} \pmod{q}.$$

Thus if the numbers  $g_i^{-2} \pmod{q}$  are precomputed and stored for each  $i = 1, \dots, r$  and for each  $q$ , the computation of  $a^{-2} \pmod{q}$  need not require any more inversions. It appears as if  $l-1$  multiplications mod  $q$  are necessary when (9) is used to compute  $a^{-2} \pmod{q}$ . But if we form a new  $l$ -tuple of  $g$ 's by trading just one  $g$  for a new one and saving an intermediate calculation, it takes only one multiplication mod  $q$ .

Here is another idea for sieve initialization that might be practical. Let  $p_1, \dots, p_l$  be a set of factor base primes with each  $p_i \approx 500$  and with  $l$  as large as possible so that  $K = p_1 \cdots p_l$  is still small compared with  $\sqrt{N}/M$ . Let  $f(x) = ax^2 + 2bx + c$  be a polynomial with  $b^2 - ac = N$ ,  $|b| < a/2$ ,  $a \approx \sqrt{2N}/KM$ . Consider the solutions  $u$  of  $f(x) \equiv 0 \pmod{K}$ . For each solution  $u$ , let

$$g_u(x) = \frac{1}{K} f(u + xK).$$

There are  $2^l$  choices of  $u$  and for each choice we obtain a polynomial  $g_u(x)$  which may be sieved for  $x$  in  $[-M, M)$ . Suppressing the details, it turns out that with a small amount of precomputation, the sieve initialization data for each polynomial  $g_u$  may be computed with 2 additions mod  $q$  for each  $q$ . For  $N \approx 10^{100}$ , we may be able to take  $l$  as large as 14 or 15, so that  $2^{14}$  or  $2^{15}$  different polynomials may be generated in this way for the one value of  $K$ .

**5. Implementation.** Our implementation of the qs algorithm will have 5 stages: (1) preprocessing, (2) sieve initialization, (3) pipe i/o, (4) pipe, (5) postprocessing. Stages (2), (3), (4) occur simultaneously on three different devices that interact frequently. As their names suggest, stage (1) is completed before the other stages are begun and stage (5) is done only after all other stages have ended their work.

**Stage (1): Preprocessing.** This relatively minor stage involves the creation of various constants that are used in later stages of the algorithm. These include (i) choice of a multiplier, (ii) creation of the factor base, (iii) solution of the congruences  $t^2 \equiv N \pmod q$  for each power  $q \leq B$  of a prime in the factor base, (iv) construction of a list of primes  $g_1, \dots, g_r$  and solutions of the congruences  $t^2 \equiv N \pmod{g_i^2}$  as discussed in § 4, (v) computation of  $g_i^{-2} \pmod q$  for each  $i = 1, \dots, r$  and each power  $q \leq B$  of a prime in the factor base.

The necessary inputs from which all of these numbers are created are  $N$  (the number to be factored),  $B$  (the bound on the factor base),  $M$  (half the length of the interval on which a polynomial is sieved), and  $r$  (where  $4\binom{r}{3}$  is a sufficient number of polynomials to complete the factorization of  $N$ ).

Preprocessing can be completed on virtually any computer with a large memory. For example, a SUN 3 workstation would be sufficient, even for very large numbers.

**Stage (2): Sieve initialization.** In this stage a three-element subset  $i, j, k$  is selected from  $\{1, \dots, r\}$  and from this triple a polynomial  $w(x)$  is constructed together with sieve initialization data. Indeed, from the preprocessed data, the sieve initializer lets  $a = g_i g_j g_k$  and chooses  $b, c$  to satisfy (4). This then defines a polynomial  $w(x) = a^2 x^2 + 2bx + c$ . Next the sieve initializer computes  $A_q^j$  for each power  $q \leq B$  of a prime in the factor base via the formulas (8) and (9).

The actual sieving of  $w(x)$  is performed in the next two stages. The sieve initializer has a direct link to the pipe i/o which controls the sieving. As soon as the sieve initialization data has been prepared, the pipe i/o and pipe cease their work on the previous polynomial and the pipe i/o receives the data for the next polynomial.

This configuration of tasks shows how the parameters  $B$  and  $M$  are related. The time for the sieve initializer to prepare a polynomial and sieve initialization data depends only on  $B$ , while the time for the sieving units to sieve the polynomial on  $[-M, M)$  with the powers  $q \leq B$  of the primes in the factor base depends on both  $M$  and  $B$ . We choose the parameters  $B, M$  so that these two times are equal. In practice, we shall choose  $B$  first and then determine empirically the value of  $M$  that works.

Since the sieve initializer will be working as long as we are sieving, it would be desirable for it to be a dedicated piece of hardware. It is also desirable, but not crucial for the sieve initializer to be fast. A 50 percent speed-up of the sieve initializer may yield only a 15 percent speed-up in the total factorization time for a 100-digit number. While not negligible, this shows that resources might be more profitably allocated elsewhere. We are planning on dedicating a SUN 3 workstation to sieve initialization. It is likely we could build a custom processor for sieve initialization with the same performance as the SUN for less than \$5000. Although we currently do not anticipate building this custom processor, we nevertheless use the figure \$5000 for the cost of a sieve initializer in our estimate of the cost of the entire project.

**Stages (3) and (4): Pipe i/o and pipe.** It makes the best sense to give a joint overview of these two stages since the two units work in tandem to sieve a polynomial  $w(x)$  on the interval  $[-M, M)$  with the powers  $q \leq B$  of the primes in the factor base. These units form the heart of our factorization project and will be described in detail in §§ 5 and 6.

Since the number  $M$  will be relatively large in our implementation (for example, we may choose  $M \approx 10^8$ ) it would take a large memory to sieve these  $2M$  values all at once. It would be a more efficient use of resources to use a somewhat smaller memory; denote its length by  $I$ . (In one configuration of our machine we have chosen  $I = 2^{20}$ .) After the first  $I$  values are sieved, we then sieve the next  $I$  values and so on.

Thus the “polynomial interval”  $[-M, M)$  is broken into a number of subintervals of length  $I$ .

The pipe then is a unit that can sieve  $I$  consecutive values of a polynomial  $w(x)$ . The pipe i/o is a unit that (i) receives from the sieve initializer the sieving data for a polynomial, (ii) initializes the pipe, (iii) sends out the sieving data ( $A \bmod q, \Lambda(q)$ ) one arithmetic progression at a time into one end of the pipe, (iv) receives processed sieving data from the other end of the pipe and readies it for the next subinterval, (v) receives reported “successes” (locations  $x$  where  $w(x)$  has been completely factored) from the pipe and sends them out to a host computer, probably the sieve initializer.

We will custom build the pipe i/o and pipe units. It is with these units, which will be specifically designed to sieve quickly, that we hope to achieve gains over previous implementations of the qs algorithm. The pipeline architecture is particularly well suited to sieving with “adjustable stride” which the qs algorithm demands. The usual problem of pipeline architectures, that of having software that keeps the pipe filled, is met here by custom tailoring of the hardware and software in the same project.

The pipe i/o and pipe will not need any diagnostic circuitry. Errors caused by hardware fault will either be detected during reporting of successes (either too few or too many reports will signal an error) or during post processing (a false report is detected here). It also should be noted that it is not necessary to design special checkpoint/restart capability since the operation of the algorithm involves sieving a new polynomial every five to ten seconds. Unlike primality testing where one glitch can throw out an entire primality proof, the quadratic sieve is a robust algorithm where local errors will not propagate.

**Stage (5): Post-processing.** Each reported success consists of four integers  $a, b, c, x$  such that

$$(a^2x + b)^2 \equiv a^2(a^2x^2 + 2bx + c) \pmod{N}$$

and such that  $w(x) = a^2x^2 + 2bx + c$  completely factors over the factor base except possibly for one larger prime (see § 4 for a description of the large prime variation). The first step in post-processing is to compute the actual prime factorizations of the various successful numbers  $w(x)$ . This can be found by trial division. However, it is possible for the pipe i/o and pipe to immediately resieve in a special mode any subinterval in which a success is found. This special mode reports the prime powers which “hit,” that is, divide, the number  $w(x)$ . If this is done then the prime factorization of  $w(x)$  will be nearly complete and the postprocessor will have little work for this step. (Thanks are due to R. Schroepel and S. S. Wagstaff, Jr. for this idea.) Without this resieving mode, as many as  $10^{11}$  multiprecision divides would be necessary in post-processing (assuming a factor base of  $10^5$  and  $10^6$  reports). By resieving, we would have instead about the same number of low precision additions performed on specially tailored hardware.

Corresponding to each factorization of a  $w(x)$  we have a (sparse) 0, 1 vector of exponents on the primes in the factor base reduced mod 2 (see § 2). The second step in post-processing is to find several linear dependencies mod 2 among these vectors. The third step in post-processing is to use a dependency to assemble two integers  $X, Y$  with  $X^2 \equiv Y^2 \pmod{N}$ , as discussed in § 2. The fourth and final step in post-processing is to compute  $(X - Y, N)$  by Euclid’s algorithm. If this gives only a trivial divisor of  $N$ , another dependency is used to assemble another pair  $X', Y'$ , etc.

The most complex of these steps is the linear algebra required to find the dependencies. The length of the vectors depends on how large a value of  $B$  is chosen. The optimal choice of  $B$  for sieving may well be larger than  $10^6$ . This would lead to vectors

of length about 39,000 or more (even ignoring the problem of encoding a large prime involved in a factorization in the same 0, 1 format). To factor very large numbers we may even wish to use vectors of length 100,000. Note that we need about as many vectors as their length. A  $10^5$  square matrix is probably too big to store in virtual memory on any commercially available computer (with the possible exception of a Cray 2), even given that each matrix entry is and will remain a single bit.

Here are several options that are available for the storage and processing of such a huge matrix. The problem of the large primes, ignored above, can be very easily solved using a sparse encoding of the vectors and quickly eliminating large primes by Gaussian elimination. This is quick and there is little fill-in since any factored  $w(x)$  has at most one large prime in the interval  $(B, B^2)$ .

This Gaussian elimination might be continued further, but now fill-in will begin to occur. It may be possible to then switch to the 0, 1 encoding and continue with Gaussian elimination on this smaller, but no longer sparse, problem.

A promising option is to use a sparse encoding and Wiedemann's elimination algorithm [13] for sparse matrices over a finite field (after the large primes have been eliminated as described above).

An unimaginative but possible plan is to use Gaussian elimination and the 0, 1 encoding (after the large primes are eliminated), but process only two slim slices of the matrix at any given time. This would involve a certain amount of i/o between the central memory and disc storage.

The matrix portion of post-processing will be performed on a large mainframe computer, perhaps the Cyber 205 at the University of Georgia. The other stages of post-processing will be performed on the same computer or perhaps on our SUN 3. In all, post-processing should not be time-critical for factoring; its difficulties will be solved in software on conventional computers.

**6. The pipe.** As mentioned above, the pipe is a unit capable of sieving  $I$  consecutive values of a polynomial  $w(x)$  with the powers  $q \leq B$  of the primes in the factor base. We now describe details of its organization.

**Block processors.** The pipe is segmented with each segment consisting of a section of store and some arithmetic capability used in sieving. We call the section of store a *block*, and the arithmetic capability together with the store a *block processor* (BP). The size of the storage on each BP is denoted  $|BP|$ . It is necessary to choose  $|BP|$  a power of 2; our working figure is  $|BP| = 2^{16}$  which we shall assume in the following. The word length of this store is nominally eight bits, since this will provide the resolution required for the approximate logarithms in the sieving process. The number of BP's in the pipe ( $\#BP$ ) is also a power of 2; our working figure is 16. The total store in the pipe is  $\#BP \times |BP| = 2^{20} = I$ , the length of a subinterval that we sieve at a given time.

**Subinterval processing.** First, the pipe must be initialized for the subinterval. This involves setting the contents of the BP store to a constant (we use 0), and setting the threshold value  $T$  at which each BP will detect a result. The pipe i/o will direct these operations.

Next, the sieve must be run. The pipe i/o will send out the progression records. These are entered into the pipe as rapidly as possible. Then when they exit the pipe they are stored in the pipe i/o for use in successive subintervals. During sieving, the BP will add  $\Lambda(q)$  to each location at which the progression  $A \bmod q$  hits. Some locations may then reach the threshold level  $T$ .

**Reports.** If a BP detects that a location has exceeded the threshold value, it will immediately report this to the pipe i/o. At the end of sieving, the address(es) that exceeded the value is (are) reported for use in post-processing.

One refinement which increases the utility of the reports is to report not only the address, but each of the prime powers that hit at that location. This will reduce the work required in post-processing. We can accomplish this goal by running the sieve “backwards” from the new set of  $A$ 's we have calculated, observing those that hit the marked location. (There is no special need to “mark” locations, however. By setting each  $\Lambda(q) = 0$  for the re-sieving, only locations already exceeding the report tolerance from before cause a report to be made.) The pipe is wired so that information “flows” only in one direction, so sieving “backwards” must be simulated by reversing the order of the store in the pipe.

When a reportable result is found, the BP will raise a flag called “report request,” which activates a daisy chain protocol. This signal will stop all the BP's simultaneously in the middle of the next cycle. If the pipe i/o has been in sieve mode, it reads from the BP the address that caused the threshold to be surpassed and then continues to sieve normally the rest of the subinterval. The pipe i/o then enters the re-sieve mode. This begins with reversing the order of the store in the pipe. (In fact, it is only necessary to reverse the order of the store of the reported address or addresses. In practice it may be simpler to just re-initialize the key location(s) with some preset value that we know will be above the threshold.) Next each BP is set to the re-sieve mode; this entails subtracting, rather than adding in the address register. Recall that each  $\Lambda(q)$  has been set to 0. Thus there will be report requests at only those marked locations that have been preset with above threshold values. Now when there is a report request, the pipe i/o transfers the prime power from the BP, not the address. When the subinterval has been completely re-sieved the pipe i/o returns to sieve mode. When the polynomial has been completely sieved, the reports are transferred to the host.

**Pipe arithmetic.** The arithmetic capability which is on the BP is concentrated into address and data arithmetic units which do the following operations:

$$\begin{aligned} A &\leftarrow A + q, \\ D &\leftarrow S(A), \\ S(A) &\leftarrow D + \Lambda(q). \end{aligned}$$

Since there are two separate arithmetic units for address and data arithmetic, these arithmetic operations can proceed in parallel. This is an important consideration in the performance of the BP.

This arithmetic capability, while simple, allows a BP to perform several different functions:

```

Initializing:  While (A < |BP|){
                S(A) ← 0;
                A ← A + 1;
            }

Sieving:      While (A < |BP|){
                S(A) ← S(A) + Λ(q);
                If (S(A) > T) {report A}
                A ← A + q;
            }
  
```

```

Reporting:  While ( $A > 0$ ){
               $S(A) \leftarrow S(A) + 0$ ;
              If ( $S(A) > T$ ) {report  $q$ }
               $A \leftarrow A - q$ ;
            }

```

**Interconnects.** Each BP is connected to both of its neighbor BP's (predecessor and successor). The predecessor of the first BP and successor of the last BP is the pipe i/o unit. In addition, each BP has a bus connection to the pipe i/o unit. From the predecessor, each BP receives the data items  $A$ ,  $q$ ,  $\Lambda(q)$  which are the progression record for sieving and sends the signal "BUSY," which while on tells the predecessor not to send anything. To the successor, each BP sends the data items  $A$ ,  $q$ ,  $\Lambda(q)$  and receives the signal "BUSY."

From the pipe i/o bus, the BP receives initialization instructions and a report tolerance  $T$  (valid for this BP in this subinterval run). To the pipe i/o bus, the BP sends the signal "SUCCESS" if some  $S(A) > T$ . In this case, the  $A$  and  $q$  report values are sent to the pipe i/o unit over this bus. The BP control modes are broadcast to the pipe from the pipe i/o unit over this bus as well.

**Performance.** The fundamental performance parameter of the BP is the cycle time of the BP store,  $C$ . All the other performance values can be stated in terms of this value.

The time of a prime power  $q$  in a BP is at most  $\lceil |\text{BP}|/q \rceil \times 2C$ . That is, each sieve step is accomplished in two cycles. To see how this can be done we consider the worst possible case, namely when several arithmetic progressions each successively hit exactly once in a particular BP. Thus in two cycles, the BP needs to receive the sieving data, recognize that there is a hit in this BP, do the sieving, see if there is a SUCCESS, recognize that the arithmetic progression does not hit a second time, and send out the altered sieving data. Also we shall see in the next section that the sieving data is not sent all in one cycle, but the  $A$  value is sent in one cycle and the  $q$ ,  $\Lambda(q)$  values are sent in the next cycle. This worst case is outlined in Table 1 which shows what happens to two consecutive sieving records  $A_i$ ,  $q_i$ ,  $\Lambda(q_i)$  for  $i = 0, 1$ .

Thus during an even-numbered cycle in this worst-case scenario, the BP performs the five operations listed for cycle 2 above. During an odd-numbered cycle, the BP

TABLE 1

Cycle	Progression 0	Progression 1
0	In [ $A_0$ ]: BPID [=] $A \leftarrow \text{IN}[A_0]$	
1	$D \leftarrow S(A)$ $A_{in} \leftarrow A + q_{in}$ ; $A_{in}$ : BPID [ $\neq$ ] $q, \Lambda(q) \leftarrow \text{IN}[q_0, \Lambda(q_0)]$	
2	OUT [ $A'_0$ ] $\leftarrow A_{in}$ $S(A) \leftarrow D + \Lambda(q)$ ; $S(A)$ : $T$ [ $<$ ]	IN [ $A_1$ ]: BPID [=] $A \leftarrow \text{IN}[A_1]$
3	OUT [ $q_0, \Lambda(q_0)$ ] $\leftarrow q, \Lambda(q)$	$D \leftarrow S(A)$ $A_{in} \leftarrow A + q_{in}$ ; $A_{in}$ : BPID [ $\neq$ ] $q, \Lambda(q) \leftarrow \text{IN}[q_1, \Lambda(q_1)]$
4		OUT [ $A'_1$ ] $\leftarrow A_{in}$ $S(A) \leftarrow D + \Lambda(q)$ ; $S(A)$ : $T$ [ $<$ ]
5		OUT [ $q_1, \Lambda(q_1)$ ] $\leftarrow q, \Lambda(q)$

performs the five operations listed in cycle 3. Although an arithmetic progression involves the BP for 4 cycles, it can travel down the pipe spending 2 cycles in each BP.

If the first compare with BPID is [ $\neq$ ], this means the arithmetic progression does not hit in this BP and it can be sent on through to the next BP. If there is a hit (as shown in the pseudocode above) and the second compare with BPID is [ $=$ ], then this means that the arithmetic progression hits a second time in this BP in which case it is, of course, not sent out right away.

**Collision avoidance.** To avoid collisions a BP will register “BUSY” during the time a prime power occupies the BP. We classify prime powers  $q$  in three categories. A value  $q$  is “small” if  $q < |\text{BP}|$ , so that  $q$  has the potential to hit more than one location in a BP. It is “moderate” if  $|\text{BP}| \leq q \leq I$ , so that  $q$  hits at most once in any BP, but will definitely hit at least one BP in the subinterval. Finally,  $q$  is “big” if  $I < q \leq B$ . Big prime powers hit at most one BP in a subinterval. The progressions are sent through a subinterval in the following order: (i) big prime powers that actually hit some BP in the subinterval; (ii) moderate prime powers ordered by decreasing size; (iii) small prime powers ordered by decreasing size. This organization of the progressions keeps delays in the sieve caused by “BUSY” signals to a minimum. In fact there are no delays at all with big and moderate prime powers.

**7. The pipe i/o unit.** The pipe i/o unit is an interface, storage, and control mechanism for the sieving process. The pipe i/o unit interfaces to the sieve initializer to receive progression records for each polynomial, and to send reports of the sieve’s successes. It stores the progression records, since the length  $I$  of the subinterval that is sieved at one time is considerably less than the length  $2M$  of the polynomial interval. The pipe i/o sends the progression records in a proper order to the pipe, then receives the modified records from the pipe and stores them for the next subinterval. The pipe i/o contains the control and sequencing logic that controls the pipe and its modes of operation.

**Interfaces.** The pipe i/o unit has a data path to the sieve initializer (SI). When a polynomial change is to occur (determined by the SI), the operation of the pipe is halted. The storage on the pipe i/o unit is then loaded with the progression records for the next polynomial interval. When this operation is completed, the pipe i/o begins the sieving sequence for the new polynomial by sending out the progressions in order. The time required to load the pipe i/o store is a function of the amount of data to be transferred, the width of the data path between the sieve initializer and the pipe i/o, and the bandwidth of the respective memories. (We assume that the loading will be a direct memory-to-memory transfer.)

The other operational use of the data path from SI to pipe i/o is for the reporting of results. When the sieve has a result to report, it will send polynomial coefficients, a polynomial argument, and a list of powers of primes from the factor base that divide the polynomial at the argument. The sieve initializer will receive these reports and retain them for the post-processing step.

**Store.** The storage of the pipe i/o must be large enough to accommodate usually two progression records for each power  $q \leq B$  of a prime in the factor base. At the same time, it must be fast enough to feed the pipe at “full guzzle” while sieving by moderate and big prime powers. This storage unit is one of the major challenges of the qs processor.



Each progression record consists of  $A$ ,  $q$ ,  $\Lambda(q)$  for the sieving, and a link field which is used by the pipe i/o to send out only those progressions which will hit in at least one BP in the subinterval. The length of a progression record is

$$\max |A| + \max |q| + \max |\Lambda(q)| + \max |L| = 32 + 24 + 8 + 20 = 84 \text{ bits,}$$

nominally. Since the  $q$ ,  $\Lambda(q)$  are identical for usually two progressions, these fields may be shared between the two progressions, reducing the nominal size to 68 bits. There must be a progression record for every solution of (7) for each power  $q \leq B$  of a prime in the factor base. If we choose  $B = 2,750,000$ , say, there will be about 200,000 progressions requiring  $200,000 \times 68 \text{ bits} = 1.7 \text{ MB}$  store.

**Speed.** The pipe i/o store must be able to keep up with the pipe. For this reason, we wish to be able to feed out a progression record in time  $2C$ , since that is the rate at which the pipe can accept progressions with modulus a moderate or big prime power. We intend to fetch the progression record in parallel (all 84 bits) from the pipe i/o store, store it in a buffer register, then send out the  $A$  in one cycle,  $q$ ,  $\Lambda(q)$  in the next (recall that the  $L$  field is for the use of the pipe i/o only).

This makes it appear that the cycle time of the pipe i/o store can be double that of the pipe's store. However, this is not the case. Once the transaction record has made it through the pipe, it will be received by the pipe i/o and must be stored for use in the next subinterval. This means that it must be stored in the pipe i/o store. Since receive transactions are occurring while send transactions are still going on, we must be able to WRITE the pipe i/o store once and READ it once in the time  $2C$ . This means that the cycle time of the pipe i/o store must be the same as that of the pipe.

**Partitioning.** The pipe i/o requires a large, fast store. More than this, since the pipe i/o store is about twice the size of the factor base, we must partition the store horizontally if at all possible so that the factor base size is not a hard, "designed in" limit on the system.

**Operation.** Once the pipe i/o store is filled with progression records, operation of the sieve begins. The pipe i/o controls the sieving operation, which consists of a cycle in which the events:

initialize the pipe  
sieve by progressions which hit in subinterval  $i$   
report any successes in subinterval  $i$   
 $i \leftarrow i + 1$

are repeated until the sieve initializer has the next polynomial ready.

**Sending progressions.** Since a big prime power may not hit in subinterval  $i$ , we keep a linked list of those we know will hit in the subinterval. Thus we only dispatch those big prime powers that will hit in the subinterval. The pipe i/o manages the linked lists on a per-subinterval basis, so that each subinterval has a list of the prime powers that will hit in that subinterval. This list might have the big prime powers out of numerical order, but since they hit only once in  $i$  anyway, it will not result in a pipe collision. As mentioned before, progressions corresponding to moderate and small prime powers are sent out in reverse numerical order of modulus. Thus the linked list mechanism is not necessary for these progressions.

**Receiving progressions.** When a progression record  $A$ ,  $q$ ,  $\Lambda(q)$  is sent into the pipe, it is processed there to generate the next progression record by modification of the  $A$  value. The pipe will add  $q$  to the  $A$  value until the new  $A$  no longer falls in the

subinterval. Then the progression record will exit from the pipe. At the exit from the pipe is the pipe i/o receiver which will store the record for use in the progression's next subinterval.

Recall that  $|BP|$  and  $I$  are both powers of 2. This means that one field of  $A$  can be regarded as the subinterval number  $i$ . This field of  $A$  is important in the processing of the received record upon exit from the pipe; the value of  $i$  will indicate the next subinterval in which the progression will fall. For moderate and small prime powers this is always the next subinterval. For big prime powers, up to  $n - 1$  subintervals can be skipped over where  $n = \lceil B/I \rceil$ . There are  $n$  linked lists maintained in the pipe i/o, and the progression will be added to the appropriate list upon exit from the pipe.

Each progression has a home location in pipe i/o store where it resides. When the progression is sent into the pipe, its home address is entered into a FIFO store. Since the pipe preserves the time order of progressions, the home address can be retrieved from this store when the progression is received. The new  $A$  will then be stored at the home address location. Additionally, the progression is added to the proper interval list by the simple procedure of placing the list pointer in the link field of the progression, then placing the progression address in the list pointer.

Receiving this progression information must compete for pipe i/o store with the sending process. Only  $A$  and  $L$  must be stored, and this will require one cycle of store. Since the  $A$  and  $L$  are buffered and since the sending process uses only one cycle from every two, storing the progression uses the other cycle.

**8. Performance on 100 digit numbers.** In this section we give some indication how performance on 100 digit numbers can be estimated. In the sequel, we assume  $N \approx 10^{100}$ , where  $N$  is the number to be factored.

**Factor base.** We shall assume the multiplier is one (see § 4). We shall consider a factor base of 100,000 primes so that  $B$ , the bound for the largest prime in the factor base, is about 2,750,000. We estimate that sieve initialization for this choice of  $B$  will take five seconds. The pipe i/o unit will require about 1.7MB of store.

**Time to sieve a subinterval.** We shall assume that each BP has size  $2^{16}$  and that the pipe consists of  $2^4$  BP's. This gives the value  $2^{20}$  to  $I$ , the subinterval length. We assume the cycle time in the pipe i/o and pipe is 70 nanoseconds. The time to sieve a subinterval has several components (measured in milliseconds):

- (i) Initialize the pipe—4.59;
- (ii) Small prime powers—11.30;
- (iii) Moderate prime powers—10.51;
- (iv) Big prime powers—9.87;
- (v) Empty the pipe with least progression—1.47.

We shall account for reporting time later. Thus the total time to sieve a subinterval of length  $I = 2^{20}$  is 37.74 milliseconds.

With one polynomial, we sieve for 5 seconds, the sieve initialization time. Thus in this time we shall sieve about  $1.38 \times 10^8$  values, so that  $2M \approx 1.38 \times 10^8$ .

**Success rate.** The largest values of a polynomial will have size  $M\sqrt{N/2}$ . However, the smaller values will give a disproportionately high success rate. Thus we shall assume the "typical" value is

$$\frac{1}{3}M\sqrt{N/2} \approx 1.63 \times 10^{57}.$$

The probability that a random number of this size completely factors with the primes up to  $2 \times 10^6$  can be estimated from the table for " $\rho_1(\alpha)$ " given in Knuth and Trabb

Pardo [4]. We take

$$\alpha = \frac{\log(1.63 \times 10^{57})}{\log(2750000)} = 8.8848.$$

We geometrically interpolate in the table to get

$$\rho_1(\alpha) \approx 1.513 \times 10^{-9}.$$

This means that we have to sieve about  $1/\rho_1(\alpha) \approx 6.61 \times 10^8$  numbers to find one success. That is, we have one success every 23.9 seconds on average.

**Large prime variation.** Multiplying 23.9 seconds by 100,000, the nominal number of successes we shall need, we obtain a sieving time of 27.7 days. However, by also reporting polynomial values which factor completely over the factor base except for one large prime in the interval  $(B, 100B)$ , we estimate a speed-up factor of 0.415. This estimate is obtained by splitting  $(B, 100B)$  into smaller intervals, using the Knuth-Trabb Pardo table to estimate success rates for large primes in the smaller intervals, and then using a "birthday paradox" analysis to estimate the usefulness of these large prime factorizations. Thus the 27.7 day sieving time is reduced to 11.5 days.

**Enforced sieve down time.** The sieve must be down when the pipe i/o is being loaded from the sieve initializer and when it is in the reporting mode. We assume that loading time per polynomial is 0.2 seconds. This must be done for every five-second polynomial run, or for a total of 0.5 days during the factorization.

With the large prime variation, there will be about 700,000 reports during the run. We shall assume, however, there are  $10^6$  reports since some of these will be "false." We also assume that a report takes 75 milliseconds (about twice the time to sieve a subinterval) for recall that reporting involves re-sieving a subinterval in a new mode. Thus total reporting time is about 0.9 days.

**Total running time.** We shall assume that preprocessing and post-processing together take at most 0.5 days of computing time. Thus our total running time can now be estimated from the following:

- (i) Sieving time—11.5 days;
- (ii) Loading time—0.5 days;
- (iii) Reporting time—0.9 days;
- (iv) Pre- and post-processing—0.5 days;

or 13.4 days.

**Estimated cost of processor.** Pre- and post-processing are performed on conventional hardware. The most critical step is the matrix processing discussed in § 5. It is assumed that this will not be an important bottleneck. We do not include the cost of buying computer time for these stages since we are considering here only a relatively insignificant portion of the factoring project. In extrapolations to very large numbers, it might be fair to set aside 5–10 percent of monetary resources for pre- and post-processing.

Sieve initialization will be performed on a dedicated SUN 3. It would be possible to build a dedicated processor with equal performance for sieve initialization tasks for \$5,000. Even though a SUN 3 costs about ten times as much, we nevertheless use the figure \$5,000 for the cost of a sieve initializer.

We estimate the cost of parts for a pipe i/o unit with 1.7MB of store at \$10,000. Finally we estimate the cost of a pipe consisting of 16 BP's, each of size  $64k \times 8 = 64kb$ , together with a power supply, at \$10,000.

Thus we estimate a total cost of \$25,000 for parts. To figure in development costs, overhead, and a margin for error, we use a multiplier of 2, thus bringing our estimate to \$50,000.

**9. Summary.** We have described the quadratic sieve factorization algorithm and an inexpensive processor on which it can be efficiently run. If it runs as quickly and is as inexpensive as we think, then 144-digit numbers can be factored in a year with a budget of \$10,000,000.

**Acknowledgment.** The authors acknowledge the constructive criticisms and helpful suggestions of Peter Montgomery, Andrew Odlyzko, Richard Schroepel, Sam Wagstaff and the referees.

#### REFERENCES

- [1] D. COPPERSMITH, A. M. ODLYZKO AND R. SCHROEPEL, *Discrete logarithms in  $GF(p)$* , *Algorithmica*, 1 (1986), pp. 1-15.
- [2] J. A. DAVIS AND D. B. HOLDRIDGE, *Factorization using the quadratic sieve algorithm*, Tech. Rpt. SAND 83-1346, Sandia National Laboratories, Albuquerque, NM, December, 1983.
- [3] J. A. DAVIS, D. B. HOLDRIDGE AND G. J. SIMMONS, *Status report on factoring* (at the Sandia National Laboratories), in *Advances in Cryptology, Lecture Notes in Computer Science 209*, 1985, pp. 183-215.
- [4] D. E. KNUTH AND L. TRABB PARDO, *Analysis of a simple factorization algorithm*, *Theoret. Comput. Sci.*, 3 (1976), pp. 321-348.
- [5] M. KRAITCHIK, *Théorie des Nombres*, Tome II, Gauthier-Villars, Paris, 1926.
- [6] H. W. LENSTRA, JR., *Factoring integers with elliptic curves*, *Ann. of Math.*, to appear.
- [7] M. A. MORRISON AND J. BRILLHART, *A method of factoring and the factorization of  $F_7$* , *Math. Comp.*, 29 (1975), pp. 183-205.
- [8] C. POMERANCE, *Analysis and comparison of some integer factoring algorithms*, in *Computational Methods in Number Theory*, H. W. Lenstra, Jr. and R. Tijdeman, eds., *Math. Centrum Tract 154*, 1982, pp. 89-139.
- [9] ———, *The quadratic sieve factoring algorithm*, in *Advances in Cryptology, Lecture Notes in Computer Science 209*, 1985, pp. 169-182.
- [10] R. RIVEST, A. SHAMIR AND L. ADLEMAN, *A method for obtaining digital signatures and public-key cryptosystems*, *Commun. ACM*, 21 (1978), pp. 120-126.
- [11] C. P. SCHNORR AND H. W. LENSTRA, JR., *A Monte Carlo factoring algorithm with linear storage*, *Math. Comp.*, 43 (1984), pp. 289-311.
- [12] R. D. SILVERMAN, *The multiple polynomial quadratic sieve*, *Math. Comp.*, 48 (1987), pp. 329-339.
- [13] D. WIEDEMANN, *Solving sparse linear equations over finite fields*, *IEEE Trans. Inform. Theory*, 32 (1986), pp. 54-62.

## EFFICIENT PARALLEL PSEUDORANDOM NUMBER GENERATION\*

J. H. REIF† AND J. D. TYGAR‡

**Abstract.** We present a parallel algorithm for pseudorandom number generation. Given a seed of  $n^\epsilon$  truly random bits for any  $\epsilon > 0$ , our algorithm generates  $n^c$  pseudorandom bits for any  $c > 1$ . This takes poly-log time using  $n^{\epsilon'}$  processors where  $\epsilon' = k\epsilon$  for some fixed small constant  $k > 1$ . We show that the pseudorandom bits output by our algorithm cannot be distinguished from truly random bits in parallel poly-log time using a polynomial number of processors with probability  $\frac{1}{2} + 1/n^{O(1)}$  if the Multiplicative Inverse Problem almost always cannot be solved in RNC. The proof is interesting and is quite different from previous proofs for sequential pseudorandom number generators.

Our generator is fast and its output is provably as effective for RNC algorithms as truly random bits. Our generator passes all the statistical tests in Knuth [14].

Moreover, the existence of our generator has a number of central consequences for complexity theory. Given a randomized parallel algorithm  $\mathcal{A}$  (over a wide class of machine models such as parallel RAMs and fixed connection networks) with time bound  $T(n)$  and processor bound  $P(n)$ , we show that  $\mathcal{A}$  can be simulated by a parallel algorithm with time bound  $T(n) + O((\log n)(\log \log n))$ , processor bound  $P(n)n^\epsilon$ , and only using  $n^\epsilon$  truly random bits for any  $\epsilon > 0$ .

Also, we show that if the Multiplicative Inverse Problem is almost always not in RNC, the RNC is within the class of languages accepted by uniform poly-log depth circuits with unbounded fan-in and strictly subexponential size  $\bigcap_{\epsilon > 0} 2^{n^\epsilon}$ .

**Key words.** cryptography, parallel algorithms, pseudorandom, number generators

**AMS(MOS) subject classifications.** 5.24, 5.27.

**1. Introduction.** A number of parallel randomized algorithms have appeared recently. These algorithms typically use a large number of random bits which must be generated in a small amount of time. Nonetheless, the area of parallel random bit generation remains unexplored.

In reality, our computers are deterministic and unable to generate truly random values. But we can give algorithms which will give pseudorandom bits on input of a random seed  $s_0$ . These pseudorandom bits satisfy conditions which suggest that for algorithmic purposes they are as effective as truly random bits.

What conditions should a pseudorandom bit sequence satisfy?

Improving on an idea by Shamir [16], Blum and Micali [6] argue that the notion of “cryptographic strength” captures the important facets of random sequences. To demonstrate cryptographic strength they follow this schema:

- (1) Upper bound the computational resources by *Resources A*.
- (2) Assume that *Problem B* cannot be solved within the limits of *Resources A*.
- (3) Produce a *Pseudorandom Bit Generator G*.
- (4) Argue that if an opponent sees the first  $m_0$  bits generated by *Pseudorandom Bit Generator G* and can utilize *Resources A* to predict the remaining bits with an

\* Received by the editors February 14, 1985; accepted for publication April 14, 1987.

† This work was done at the Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. Present address, Computer Science Department, Duke University, Durham, North Carolina 27706. The work of this author was supported in part by National Science Foundation grant NSF-MCS-79-21024 and Office of Naval Research contract N0014-80-C-0674.

‡ This work was done at the Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts 02138. Present address, Department of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213. The work of this author was supported in part by a National Science Foundation graduate fellowship and National Science Foundation grant MCS-81-21431.

accuracy rate of  $\frac{1}{2} + \varepsilon(m)$  (where  $m$  is the size of the seed and  $\varepsilon$  is a fixed function satisfying  $\lim_{m \rightarrow \infty} \varepsilon(m) = 0$ ), then the opponent will be able to solve *Problem B* limited to *Resources A* by consulting the bit-guessing oracle, a contradiction.

Several cryptographically-strong pseudorandom bit generators have been proposed (Blum, Blum and Shub [5], Blum and Micali [6],) and many applications have been discussed (Alexi, Chor, Goldreich and Schnorr [3], Goldreich, Goldwasser and Micali [9], Goldwasser, Micali and Tong [10], Vazirani and Vazirani [19], Yao [21].) These generators are all inherently sequential, require polynomial time, and their cryptographic strength relies on some unproven cryptographic assumption.

*Notation.* When we say a class of circuit is *uniform*, we mean that it is constructible in logarithmic space by a deterministic Turing Machine.

**NC** (**NC<sub>U</sub>**) is the class of languages accepted by (uniform, respectively) deterministic circuits constructible in log-space with poly-log depth and polynomial size.

**RNC** (**RNC<sub>U</sub>**) is the class of languages accepted by (uniform, respectively) randomized circuits constructible in log-space with two-sided error, poly-log depth, polynomial size, and acceptance probability greater than  $\frac{1}{2}$ .

We give more precise definitions of these terms in § 4.

*Our result.* We present a new cryptographically-strong pseudorandom bit generator which runs in **NC<sub>U</sub>** but which is secure against attacks taking parallel poly-log time if the Multiplicative Inverse Problem almost always is not in **RNC**. While we use the schema described above for demonstrating the cryptographic strength of our random number generator, because of the inherent parallel nature of our generator, the technical details of our proof are quite different from those of previous proofs for sequential pseudorandom number generators. In particular, we prove that if the bits output by our pseudorandom bit generator can be predicted in **RNC**, then we can solve the multiplicative inverse problem in **RNC** almost always and this requires that we construct an interesting, nontrivial, parallel algorithm for that problem. (See § 3.)

*About the assumption.* While our assumption has not been proved, it is quite interesting to observe that it is *testable* in the following sense: If an **RNC** algorithm takes more than poly-log time using our pseudorandom bits instead of truly random bits then we can observe this event by timing. Thus one of two scenarios is possible: either every application of our generator to an **RNC** algorithm yields a poly-log algorithm using only a small number of random bits, or some application of our generator is discovered to exceed its poly-log time bounds and we can immediately derive a **NC** algorithm for multiplicative inverse.

*About the measure of randomness.* Valiant, Skyum, Berkowitz and Rackoff [18] show that an **NC**-machine can evaluate any straight-line program which computes a multivariate polynomial which has degree polynomial in the length of the program. Thus if our assumption is correct, our pseudorandom bit generator is secure against any statistical test which can be so formulated as a straight-line program. This includes most standard statistical tests for random number generators (Knuth [14]).

*Applications.* Our method for parallel pseudorandom bit generation is actually very practical. It requires, for any  $\varepsilon > 0$ , only  $O(\log n(\log \log(n)))$  added depth and a factor of  $n^\varepsilon$  for a bounded fan-in circuit. Here is an example: Karp and Wigderson [12] give a deterministic algorithm for the maximal independent set problem in  $O((\log n)^4)$  time using  $O(n^3/(\log n)^3)$  processors. They also give a uniform randomized algorithm for the same problem running in  $O((\log n)^3)$  expected time with  $O(n^2)$  processors using  $O(n^2)$  random bits. Our results immediately yield a uniform algorithm with  $O((\log n)^3)$  running time and  $O(n^{2+\varepsilon'})$  processors using only  $n^\varepsilon$  random bits, where  $\varepsilon, \varepsilon' > 0$  can be set arbitrarily small.

Recently, Karp, Upfal and Wigderson [13] have shown that finding a maximum graph matching is in  $\mathbf{RNC}_U$ , and Anderson and Mayr [2] have shown that finding a maximal path is in  $\mathbf{RNC}_U$ . Our results also immediately yield efficient randomized uniform algorithms for these problems, using only  $n^\epsilon$  bits for any  $\epsilon > 0$ .

*Implications.* An interesting theoretical application of our result is that  $\mathbf{RNC}_U$  is contained within the class of languages recognized by uniform deterministic circuits of unbounded fan-in with poly-log depth and  $2^{n^\epsilon}$  size for any  $\epsilon > 0$ . (Adleman [1] proved  $\mathbf{RNC}_U$  is contained in (nonuniform)  $\mathbf{NC}$ , but the previous best construction for bounding  $\mathbf{RNC}_U$  by deterministic uniform circuits of poly-log depth required  $2^{\Omega(n)}$  size.) This extends a result of Yao [21] for sequential polynomial time computations to poly-log time parallel computations.

**2. Definitions and results.**

*Notation.* We use the following notation throughout the paper:

$N$  A positive composite integer such that each prime factor of  $N$  is greater than  $N^c$  for a fixed  $c > 0$ .

$\mathbf{Z}_N^*$  The multiplicative group of positive integers less than and relatively prime to  $N$ . (Note that the fact that  $N$  has only large factors implies that a random positive integer less than  $N$  is an element of  $\mathbf{Z}_N^*$  with high probability.)

We will sometimes use  $x \bmod N$  to indicate the residue of  $x$  modulo  $N$ .

*Definitions.* An  $\mathbf{NC}$ -machine (Cook [8]) is a deterministic parallel algorithm which runs on  $n^{O(1)}$  parallel-RAM (P-RAM) processors in time  $(\log n)^{O(1)}$  for input of size  $n$ . (Note that  $\mathbf{NC}_U$  is the class of languages accepted by  $\mathbf{NC}$ -machines.)

An  $\mathbf{RNC}$ -machine is a randomized parallel algorithm which runs on  $n^{O(1)}$  P-RAM processors in time  $(\log n)^{O(1)}$  for input of size  $n$ . (Note that  $\mathbf{RNC}_U$  is the class of languages accepted by  $\mathbf{RNC}$ -machines.)

Given  $s_0 \in \mathbf{Z}_N^*$ , the *multiplicative inverse* of  $s_0$  modulo  $N$  is the  $s_0^{-1}$  such that  $s_0 s_0^{-1} = 1 \bmod N$ .

For a fixed  $N$ , given an arbitrary  $k \in \mathbf{Z}_N^*$ , the Multiplicative Inverse Problem is to find the multiplicative inverse of  $k$  modulo  $N$ . Note that the input size to the problem is  $n = \lceil \log N \rceil$ .

The problem of finding multiplicative inverses in poly-log depth has been studied extensively. (Cook [8], Kannan, Miller and Randolph [11], Reif [15] and von zur Gathen [20].) Based on the lack of significant positive results obtained so far we conjecture the following.

*Complexity assumption.* There exists an infinite sequence of numbers  $N_1, N_2, \dots$  constructible in  $\mathbf{NC}_U$  such that for each  $n = 1, 2, \dots$  we have  $n = \lceil \log N_n \rceil$  and that no  $\mathbf{RNC}$ -machine can solve the Multiplicative Inverse Problem for arbitrary elements of  $\mathbf{Z}_{N_n}^*$  for almost all values of  $n$ .

(Actually we could replace this complexity assumption with the weaker assumption that there exists a  $k$  such that for almost all  $n$  there exists an  $n'$  such that  $n < n' < n^k$  and no  $\mathbf{RNC}$ -machine can solve the Multiplicative Inverse Problem for arbitrary elements of  $\mathbf{Z}_{N_{n'}}^*$ . All the theorems in this paper would remain true under that weaker assumption.)

*Definitions.* A set  $S$  of bit sequences  $\sigma = (b_1, \dots, b_J)$  of length  $J = n^{O(1)}$  pseudorandom bits is  $\mathbf{RNC}$ -cryptographically strong if no  $\mathbf{RNC}$ -machine can, on a random input  $b_1, \dots, b_i \in S$  ( $i < J$ ) predict any one bit  $b_i, \dots, b_J$  with expected success of  $\frac{1}{2} + 1/n^{O(1)}$ . Informally, the bit sequences are  $\mathbf{RNC}$ -cryptographically strong if no  $\mathbf{RNC}$ -machine can predict untransmitted bits with an expected success rate significantly better than  $\frac{1}{2}$ .

**THEOREM.** *If no RNC-machine can solve the Multiplicative Inverse Problem for almost all  $n$ , then there exists a deterministic NC-machine  $\mathcal{G}$  which on an input seed of  $n$  bits outputs an RNC-cryptographically strong sequence of  $J = n^{O(1)}$  pseudorandom bits.  $\mathcal{G}$  can be computed by a bounded fan-in uniform Boolean circuit of depth  $O((\log n)(\log \log n))$  and size  $n^{O(1)}$ .*

This theorem is proved in § 3.

**Definition.** An RNC-statistical test is an RNC-machine which attempts to distinguish truly random bit sequences from pseudorandom bit sequences. A statistical test succeeds if it correctly distinguishes the pseudorandom bit sequences from truly random bit sequences with probability at least  $1/n^{O(1)}$ .

By a technique due to Yao [21] we can show that no RNC statistical test can succeed on RNC-cryptographically strong bit sequences. Hence we have the following.

**COROLLARY 1.** *If no RNC-machine can solve the Multiplicative Inverse Problem for almost all  $n$ , then no RNC-statistical test can succeed on our pseudorandom bit generator  $\mathcal{G}$ .*

**COROLLARY 2.** *If the  $N_n$  are constructible in depth  $h(n)$ , then given a randomized parallel algorithm  $\mathcal{A}$  (over a wide class of machine models such as parallel RAMS and fixed connection networks) with time bound  $T(n)$  and processor bound  $P(n)$  then  $\mathcal{A}$  can be simulated by a parallel algorithm with time bound  $T(n) + h(n) + O((\log n)(\log \log n))$ , processor bound  $P(n)n^{\varepsilon'}$ , and only using  $n^\varepsilon$  truly random bits for any  $\varepsilon > 0$ , where  $\varepsilon' = O(\varepsilon)$ .*

$\text{CIRCUIT}_U(D(n), S(n))$  is the class of languages accepted by uniform deterministic circuits with unbounded fan-in, depth  $D(n)$ , and size  $S(n)$ . (See § 4 for a precise definition of these complexity classes.)

**COROLLARY 3.** *If for almost all  $n$  the Multiplicative Inverse Problem is not in RNC then*

$$\text{RNC}_U \subseteq \bigcup_{c>0} \bigcap_{\varepsilon>0} \text{CIRCUIT}_U((\log n)^c, 2^{n^\varepsilon})$$

This corollary is proved in § 4.

**COROLLARY 4.** *There exists a cryptosystem where encryption and decryption can be done by an NC-machine on  $n^{O(1)}$  bits given a secret shared key exactly  $n$  bits long (here  $n$  is a security parameter). If no RNC-machine can solve the Multiplicative Inverse Problem, then no RNC-machine can decrypt ciphertext exchanged in this cryptosystem.*

We use the pseudorandom bits as a “one-time pad”—we take the sequential exclusive-or of the plaintext and the pseudorandom bits to produce the ciphertext and take the sequential exclusive-or of the ciphertext and the pseudorandom bits to obtain the plaintext again. Encryption and decryption both take parallel poly-log time but an opponent cannot decrypt the ciphertext with RNC-machine.

### 3. The proof of the main theorem.

**Properties.** We recall the following facts which we use implicitly (Beame, Cook and Hoover [4], Reif [15], and Shonhage and Strassen [17]):

- There exists an NC-machine for multiplication of two numbers in  $\mathbf{Z}_N^*$ .
- $2 \log p$  multiplications suffice to find the  $p$ th power of a number in  $\mathbf{Z}_N^*$ .
- If  $p < (\log N)^{O(1)}$  there exists an NC-machine for finding the  $p$ th power of a number in  $\mathbf{Z}_N^*$ .

Fix  $m = \lceil \log N \rceil$  throughout this section.

Let  $\mathcal{G}$  be the NC-machine which performs the following operations:

Input: random elements  $s_0, k \in \mathbf{Z}_N^*$ .



Output:  $b_1, \dots, b_J$  where  $J = m^{O(1)}$ .

Method: In parallel each processor  $P_i$  ( $i = 1, \dots, J$ ) calculates  $s_i = ks_0^i \bmod N$  and  $b_{J-i+1} = B(s_i)$  where

$$B(x) = \begin{cases} 0 & \text{if } x \leq N/2, \\ 1 & \text{if } x > N/2. \end{cases}$$

LEMMA. *If there exists an RNC-machine which can determine the value of  $b_j$  with probability 1 (i.e., no error) on input  $b_1, \dots, b_{j-1}$ , then there exists an RNC-machine which can solve the Multiplicative Inverse Problem for  $\mathbf{Z}_{N_n}^*$ .*

*Proof of the lemma.* Suppose that MB (for “magic box”) is an oracle which can determine the value of  $b_j$  with probability 1. Then given  $s_0 \in \mathbf{Z}_N^*$  we can find  $s_0^{-1} \bmod N$ . We can find this by running in parallel the following algorithm on each processor  $P_i$  for ( $0 \leq i \leq m$ ):

Set  $k \leftarrow 2^i$ . In parallel set  $b_i \leftarrow B(ks_0^{J-i-1})$  for  $1 \leq i \leq J-1$ . Note that  $b_J = B(2^i s_0^{-1})$ . Feed the sequence  $(b_1, \dots, b_{J-1})$  to MB to get  $b_J$ . Set the  $i$ th most significant bit of  $\delta$  to be  $B(2^i s_0^{-1})$ . Define

$$\phi(\delta) = \left\lceil \frac{\delta N}{2^m} \right\rceil.$$

Then  $\phi(\delta) = s_0^{-1} \bmod N$ .  $\square$

THEOREM. *If there exists an RNC-machine which can determine the value of  $b_j$  with probability at least  $\frac{1}{2} + 1/m^{O(1)}$  on input  $b_1, \dots, b_j$ , then there exists an RNC-machine  $\mathcal{H}$  which can solve the Multiplicative Inverse Problem for  $\mathbf{Z}_{N_n}^*$ .  $\mathcal{H}$  can be computed by a bounded fan-in Boolean circuit of depth  $O((\log n)(\log \log n))$  and size  $n^{O(1)}$ .*

*Proof of the theorem.* Assume that there exists an RNC-machine MB which can predict  $b_j$  with probability  $\frac{1}{2} + 2/m^c$ . Let  $H = 2(c+1) \lceil \log m \rceil$ . Let  $\delta$  and  $\phi$  be as in the proof of the lemma.

Let  $S = \{0, 1, \dots, 2^{H-1} - 1\}$ . For each  $0 \leq y < x \leq m$ , we will create, by randomized methods, two functions  $F_{x,y} : S \rightarrow \{0, 1\}^{x-y}$  and  $G_{x,y} : S \rightarrow S$ . Informally, values in  $S$  are guesses;  $F_{x,y}$  is a rule for transforming a guess  $j_x \in S$  into the  $x$ th to  $y$ th most significant bits of  $\delta$ ; and  $G_{x,y}$  is a rule for transforming the guess  $j_x \in S$  into the guess  $j_{x-1} \in S$ .

If an RNC-machine could find  $\delta$  for arbitrary  $s_0$ , we could solve the Multiplicative Inverse Problem. It will turn out that for some  $j_m \in S$ ,  $F_{m,0}(j_m) = \delta$  with probability  $\frac{1}{2}$ . We can verify this occurrence simply by checking whether  $s_0 \phi(\delta) = 1 \bmod N$ . If we do not immediately find  $s_0^{-1} \bmod N$ , we simply form a new  $F_{m,0}$  by randomized methods and continue testing until we do find  $s_0^{-1} \bmod N$ .

Suppose we can determine  $j_x$  such that we know  $(2^x s_0^{-1} \bmod N)$  belongs to one of the two intervals

$$\left\{ \left[ \left\lceil \frac{j_x N}{2^H} \right\rceil, \left\lfloor \frac{(j_x + 1)N}{2^H} \right\rfloor \right], \left[ \left\lceil \frac{(2^{H-1} + j_x)N}{2^H} \right\rceil, \left\lfloor \frac{(2^{H-1} + j_x + 1)N}{2^H} \right\rfloor \right] \right\}.$$

We can pick  $2^H$  random values  $\beta \in \mathbf{Z}_N^*$  and let  $v$  be MB’s prediction for

$$B \left( 2^x s_0^{-1} + \left\lfloor \frac{Nj_x}{2^H} \right\rfloor + \beta \bmod N \right).$$

When  $\beta$  lies in the interval

$$\left[ 0, \left\lfloor \frac{(2^{H-1} - 1)N}{2^H} \right\rfloor \right],$$

mark a vote for  $v$ , when  $\beta$  lies in the interval

$$\left[ \left\lceil \frac{N}{2} \right\rceil, \left\lfloor \frac{(2^H - 1)N}{2^H} \right\rfloor \right],$$

mark a vote the complement of  $v$ , and mark a *null* vote when  $\beta$  lies in other intervals. By assumption, MB predicts correctly with probability at least  $\frac{1}{2} + 2/m^c$ .

We can assign a processor to calculate MB's prediction for each of the  $2^H$  randomly chosen values of  $\beta \in \mathbf{Z}_N^*$ . This computation can be done in poly-log time for each  $\beta$ . The expected fraction of *null* votes is  $2^{1-H} < 1/m^c$ . Thus we have a bias of at least  $2/m^c - 1/m^c = 1/m^c$  between 0 and 1 votes. Set  $F_{x,x-1}(j_x)$  (our guess for  $B(2^x s_0^{-1} \bmod N)$ ) to be a value which got the most votes. If our guess for  $B(2^x s_0^{-1} \bmod N)$  is right, this immediately identifies which of the two intervals  $(2^x s_0^{-1} \bmod N)$  belongs to. By the argument in the Appendix,  $2^H$  tests are sufficient to make our guess correct with probability at least  $1 - 1/2^m$ . If our guess is right, that immediately determines the value of  $j_{x-1}$ ; that is, we can determine that  $(2^{x-1} s_0^{-1} \bmod N)$  lies in one of the two intervals

$$\left\{ \left[ \left[ \frac{j_{x-1}N}{2^H} \right], \left[ \frac{(j_{x-1}+1)N}{2^H} \right] \right], \left[ \left[ \frac{(2^{H-1}+j_{x-1})N}{2^H} \right], \left[ \frac{(2^{H-1}+j_{x-1}+1)N}{2^H} \right] \right] \right\},$$

namely

$$j_{x-1} = G_{x,x-1}(j_x) = \lfloor j_x/2 \rfloor + 2^{H-2}(F_{x,x-1}(j_x)).$$

We can calculate in parallel, for each  $m \cong x \cong 1$ , the functions  $F_{x,x-1}$  and  $G_{x,x-1}$ , since the domain is finite and of polynomial size. If  $x - y > 1$ , then  $F_{x,y}$  and  $G_{x,y}$  can be recursively defined as

$$F_{x,y}(j_x) = F_{z,y}(G_{x,z}(j_x))2^{x-z} + F_{x,z}(j_x)$$

and

$$G_{x,y}(j_x) = G_{z,y}(G_{x,z}(j_x))$$

where  $z = \lceil (x+y)/2 \rceil$ . For each  $x, y$  pair ( $0 \leq y < x \leq m$ ) and each  $j_x \in S$  we repeatedly calculate the appropriate compositions of these functions for all  $j_x$  in the domain of the functions. Thus we can compute  $F_{m,0}$  in  $\lceil \log m \rceil$  stages.

Some guess  $j_m$  is correct. Suppose that for all  $1 \leq i \leq m$ , that (1)  $G_{i,i-1}(j_i)$  is the correct value of  $j_{i-1}$ . Then (2)  $F_{m,0}(j_m)$  would be the correct value of  $\delta$ . For each  $i$ , the probability that (1) is true for a particular  $j_i$  is  $(1 - 2^{-m})$ , so the probability that (2) is true is  $(1 - 2^{-m})^{m-1} > 1 - (m-1)2^{-m} > \frac{1}{2}$ .

For some  $j_m \in S$ , it will be true that  $F_{m,0}(j_m) = \delta$  with probability  $\frac{1}{2}$ . We can try all possible  $j_m$  in parallel, and find out if we have a correct value by checking whether  $\phi(F_{m,0}(j_m))s_0 = 1 \bmod N$ . (Of course, it might happen that an incorrect guess for  $j_m$  might give a correct value for  $\delta$  but this can only speed the calculation.) In the event that we do not get the correct value for  $s_0^{-1} \bmod N$ , we simply form new  $F_{x,y}$  and  $G_{x,y}$  functions and continue until we do get the correct value.  $\square$

**4. Randomized and deterministic parallel complexity.** Let  $\mathcal{C}$  be a list of circuits  $(C_1, C_2, \dots)$  of unbounded fan-in where  $C_n$  and  $n$  inputs and size  $S(n)$ . We consider  $\mathcal{C}$  to be *uniform* if there exists a  $(\log S(n))$  space deterministic Turing machine which, given any  $n$ , outputs the circuit  $C_n$ . Let  $\text{CIRCUIT}(D(n), S(n))$  be the class of all languages accepted by deterministic Boolean circuits with unbounded fan-in, depth  $D(n)$ , and size  $S(n)$ . As usual we define

$$\text{NC} = \bigcup_{k_1 > 0, k_2 > 0} \text{CIRCUIT}((\log n)^{k_1}, n^{k_2}).$$

We allow a *randomized Boolean circuit*  $\mathbf{C}$  to have  $r$  special nodes, each of which are assigned independent random bits chosen from  $\{0, 1\}$  with equal probability.  $\mathbf{C}$  *accepts* an input  $\omega \in \{0, 1\}^n$  if  $\mathbf{C}$  outputs 1 with probability  $> \frac{1}{2}$ ; otherwise  $\mathbf{C}$  *rejects* the input. For simplicity, we consider only one-sided error randomized circuits which

never output a 1 on an input they have rejected. (The construction below can easily be extended to two-sided error randomized circuits which have an acceptance probability of at least  $\frac{1}{2} + 1/n^k$  for some  $k > 1$ .) Let RCIRCUIT  $(D(n), S(n))$  be the class of languages accepted by randomized circuits with unbounded fan-in, depth  $D(n)$ , and size  $S(n)$ . We define

$$\mathbf{RNC} = \bigcup_{k_1 > 0, k_2 > 0} \text{RCIRCUIT}((\log n)^{k_1}, n^{k_2}).$$

We define CIRCUIT<sub>U</sub>, NC<sub>U</sub>, RCIRCUIT<sub>U</sub>, and RNC<sub>U</sub> analogously—restricting the circuits to be uniform.

*Proof of Corollary 3.* Let C be a (one-sided error) uniform randomized Boolean circuit with  $n$  inputs, depth  $D(n) = (\log n)^{k_1}$ , and size  $S(n) = n^{k_2}$ . Fix any  $\epsilon > 0$ .

First suppose we had a source of  $b = \lceil n^{\epsilon/2} \rceil$  truly random bits. Observe that C uses at most  $S(n) = n^{k_2}$  random bits on each execution. Since  $S(n) \leq b^{\epsilon'}$  where  $\epsilon' = \lceil \epsilon/k_2 \rceil$  is constant, we can apply our parallel pseudorandom bit generator  $\mathcal{G}$  to produce  $S(n)$  pseudorandom bits in  $(\log n)^{O(1)}$  parallel time using  $n^{O(1)}$  processors and using the  $b$  truly random bits as the seed. We can view the execution of C on the given input  $\omega$  as a statistical test. By Corollary 2, given an input  $\omega \in \{0, 1\}^n$ , we need only execute C on  $\omega$  for each of the  $2^b$  possible pseudorandom bit sequences. We accept  $\omega$  if C ever outputs 1.

Furthermore, we can avoid the use of a truly random seed by simply (1) enumerating all  $b$ -bit numbers in parallel; (2) executing the parallel pseudorandom bit generator using each of the  $b$ -bit numbers as a seed; and (3) executing C in parallel on  $\omega$  on each of the resulting pseudorandom bit sequences. If C ever outputs 1 we accept  $\omega$ . The resulting uniform circuit requires size  $2^{b^2} \leq 2^{n^\epsilon}$  and depth  $(\log n)^{O(1)} + O(D(n)) = (\log n)^{O(1)}$ .  $\square$

Note that if we require that our simulation circuit have bounded fan-in, then to simulate a circuit accepting a language in RNC<sub>U</sub>, we require  $n^{O(1)}$  (rather than  $(\log n)^{O(1)}$  depth) and  $2^{n^\epsilon}$  size. This is an improvement over previous size bounds for RNC<sub>U</sub>.

**Appendix.** Let  $X$  be the binomial variable which is the sum of  $\tau$  independent Bernoulli trials each of which has a probability  $p = \frac{1}{2} + 1/m^c$  of giving a value 1 and probability  $1 - p = \frac{1}{2} - 1/m^c$  of giving a value of 0. We need to find  $\tau$  large enough so that

$$\Pr [X < \tau/2] < 1/2^n.$$

Using Chernoff bounds (Chernoff [7]) we recall that

$$\Pr [X < (1 - \delta)\tau p] < e^{-\delta^2 \tau p/2}.$$

Substituting  $p = \frac{1}{2} - 1/m^c$  and  $(1 - \delta)p = \frac{1}{2}$  we get

$$\Pr [X < \tau/2] < e^{-(1/m^{2c})\tau(1/2 - 1/m^c)(1/2)} < e^{-\tau/8m^{2c}}.$$

If we set  $\tau = m^{2c+1}$ , our initial condition is satisfied. Since  $2^H = \tau$ , setting

$$H = 2(c + 1) \lceil \log m \rceil = O(\log m)$$

will give us conditions sufficient to prove the main theorem.

**Acknowledgments.** We would like to thank Michael Rabin for being an inspiration to us in the fields of randomized algorithms and cryptography.

We are indebted to Silvio Micali for his many helpful and insightful comments on this manuscript.

Also, thanks to Benny Chor, Shafi Goldwasser, Johan Hastad, Brian O'Toole, Charles Rackoff, Les Valiant and Vijay Vazirani for their comments.

## REFERENCES

- [1] L. ADLEMAN, *Two theorems on random polynomial time*, Proc. 19th IEEE Symposium on Foundations of Computer Science, Ann Arbor, MI, October 1978, pp. 75–83.
- [2] R. ANDERSON, *A parallel algorithm for the maximal path problem*, Proc. 17th ACM Symposium on Theory of Computing, Providence, RI, May 1985, pp. 33–37.
- [3] W. ALEXI, B. CHOR, O. GOLDREICH AND C. SCHNORR, *RSA/Rabin bits are  $\frac{1}{2} + 1/\text{poly}(\log N)$  secure*, Proc. 25th IEEE Symposium on Foundations of Computer Science, Singer Island, FL, October 1984, pp. 449–457.
- [4] P. BEAME, S. COOK AND H. HOOVER, *Small depth circuits for integer products, powers, and division*, Proc. 25th IEEE Symposium on Foundations of Computer Science, Singer Island, FL, October 1984, pp. 1–6.
- [5] L. BLUM, M. BLUM AND M. SHUB, *A simple secure pseudo-random number generator*, Proc. CRYPTO-82, Santa Barbara, CA, September 1982, pp. 112–117.
- [6] M. BLUM AND S. MICALI, *How to generate cryptographically strong sequences of pseudo-random bits*, this Journal, 13 (1984), pp. 850–864.
- [7] H. CHERNOFF, *A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations*, Ann. Math. Statist., 23 (1952), pp. 493–507.
- [8] S. A. COOK, *The classification of problems which have fast parallel algorithms*, Lecture Notes in Computer Science, Vol. 158, M. Karpenski, ed., Springer-Verlag, New York, Berlin, 1983.
- [9] O. GOLDREICH, S. GOLDWASSER AND S. MICALI, *How to construct random functions*, Proc. 25th IEEE Symposium on Foundations of Computer Science, Singer Island, FL, October 1984, pp. 464–479.
- [10] S. GOLDWASSER, S. MICALI AND P. TONG, *Why and how to establish a private code on a public network*, Proc. 23rd IEEE Symposium on Foundations of Computer Science, Chicago, IL, October 1982, pp. 134–144.
- [11] R. KANNAN, G. MILLER AND L. RUDOLF, *Sublinear parallel algorithms for the greatest common divisor of two integers*, Proc. 25th IEEE Symposium on Foundations of Computer Science, Singer Island, FL, October 1984, pp. 7–11.
- [12] R. KARP AND A. WIGDERSON, *A fast parallel algorithm for the maximal independent set problem*, Proc. 16th ACM Symposium on Theory of Computation, Washington, DC, May 1984, pp. 266–272.
- [13] R. KARP, E. UPFAL AND A. WIGDERSON, *Constructing a perfect graph matching in Random NC*, Proc. 17th ACM Symposium on Theory of Computing, Providence, RI, May 1987, pp. 22–32.
- [14] D. KNUTH, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 2nd ed., Addison-Wesley, Reading, MA, 1981.
- [15] J. REIF, *Logarithmic depth circuits for algebraic functions*, Proc. 24th IEEE Symposium on Foundations of Computer Science, Tucson, AZ, October 1983, pp. 138–145. Revised in Technical Report TR-84-18, Center for Research in Computing Technology, Harvard University, Cambridge, MA, this Journal, to appear.
- [16] A. SHAMIR, *On the generation of cryptographically strong pseudo-random sequences*, ACM Trans. Comput. Sys., 1 (1983), pp. 38–44.
- [17] A. SHONHAGE AND V. STRASSEN, *Schnelle Multiplikation grosser Zahlen*, Computing, 7 (1974), pp. 281–292.
- [18] L. VALIANT, S. SKYUM, S. BERKOWITZ AND C. RACKOFF, *Fast parallel computation of polynomials using few processors*, this Journal, 12 (1983), pp. 641–644.
- [19] U. VAZIRANI AND V. VAZIRANI, *Trapdoor pseudorandom number generators with applications to protocol design*, Proc. 24th IEEE Symposium on Foundations of Computer Science, Tucson, AZ, October 1983, pp. 23–30.
- [20] J. VON ZUR GATHEN, *Parallel algorithms for algebraic problems*, Proc. 15th ACM Symposium on Theory of Computing, Boston, MA, April 1983, pp. 13–18.
- [21] A. YAO, *Theory and applications of trapdoor functions*, Proc. 23rd IEEE Symposium on Foundations of Computer Science, Chicago, IL, October 1982, pp. 80–91.

## THE NOTION OF SECURITY FOR PROBABILISTIC CRYPTOSYSTEMS\*

SILVIO MICALI<sup>†</sup>, CHARLES RACKOFF<sup>‡</sup> AND BOB SLOAN<sup>§</sup>

**Abstract.** Three very different formal definitions of security for public-key cryptosystems have been proposed—two by Goldwasser and Micali and one by Yao. We prove all of them to be equivalent. This equivalence provides evidence that the right formalization of the notion of security has been reached.

**Key words.** cryptosystem, public-key cryptography, probabilistic encryption

**AMS(MOS) subject classifications.** 94A60, 68Q99

**1. Introduction.** The key desideratum for any cryptosystem is that encrypted messages must be secure. Before one can discuss whether a cryptosystem has this property, however, one must first rigorously define what is meant by security. Three different rigorous notions of security have been proposed. Goldwasser and Micali [6] suggested two different definitions, polynomial security and semantic security, and proved that the first notion implies the second. Yao [11] proposed a third definition, one inspired by information theory, and suggested that it implies semantic security.

Not completely knowing the relative strength of these definitions is rather unpleasant. For instance, several protocols have been proved correct adopting the notion of polynomial security. Are these protocols secure with respect to a particular definition or are they secure protocols in a more general sense? In other words, a natural question arises: Which of the definitions is the “correct” one? Even better: How should we decide the “correctness” of a definition?

The best possible answer to these questions would be to find that the proposed definitions—each attempting to be as general as possible—are all equivalent. In this case, one obviously no longer has to decide which definition is best. Moreover, the equivalence suggests that one has indeed found a strong, natural definition.

In this paper, we show that these notions are *essentially* equivalent. The three originally proposed definitions were not equivalent. However, as we point out, this inequivalence was caused only by some minor technical choices. After rectifying these marginal choices, we succeed in proving the desired equivalences, keeping the spirit of the definitions intact. We believe this to be an essential step in developing theory in the field of cryptography.

**2. Public-key scenarios.** Let us briefly review what is meant by the notion of public-key cryptography, first proposed by Diffie and Hellman [4] in 1976. As with all cryptography, the goal is that A(lice), by using an encryption algorithm  $E$ , becomes able to securely send a message  $m$  to B(ob). What is meant by “securely” is that it is impossible for any party T who has tapped A and B’s line to figure out information

---

\* Received by the editors June 6, 1986; accepted for publication (in revised form) May 19, 1987.

<sup>†</sup> Massachusetts Institute of Technology, Laboratory for Computer Science, Cambridge, Massachusetts 02138. The work of this author was supported by Army Research Office grant DAAL03-86-K-0171, National Science Foundation grant DCR-8413577, and an IBM Faculty Development Award.

<sup>‡</sup> University of Toronto, Toronto, Ontario, Canada M5S 1A4.

<sup>§</sup> Massachusetts Institute of Technology, Laboratory for Computer Science, Cambridge, Massachusetts 02138. The work of this author was supported by a General Electric Foundation Fellowship and National Science Foundation grant DCR-8509905.

about  $m$  from  $E(m)$ . The distinguishing feature of public-key cryptography is that we require this security property to hold even when T knows the encryption algorithm  $E$ .

We believe that until now, the notion of public-key cryptography has not been fully understood. In fact, it is crucial to consider exactly how the communication between A and B establishes the algorithm  $E$ . Therefore, we introduce the fundamental notion of a *pass*. We will first explain what passes are, and then explain their implications for security.

**2.1. Passes.** Within the public-key model, A and B can alternate communicating back and forth as many times as they feel are necessary to achieve security. Call each alternation a *pass*.

Any number of passes are, of course, permissible. We concentrate on what we believe are the two most interesting and important cases, one and three passes. We do not consider more than three passes, because, if trapdoor permutations exist, a well-designed probabilistic encryption scheme can achieve as much security as is possible using only three passes.

**Three-pass systems.** The three-pass case is, perhaps, the most natural to think about. It corresponds to a telephone conversation. A has a message  $m$  that she wants to securely communicate to B. A calls up B and says, "I have a message I'd like to send to you." B, so alerted, proceeds to generate an encryption/decryption algorithm pair,  $(E, D)$ , and tells A, "Please use  $E$  to encrypt your message." A then uses  $E$  to encrypt her message and tells B " $E(m)$ ."

Notice the key property of a three-pass system: The message and the encryption algorithm are selected independently of one another. We are nevertheless in a public-key model, since anyone tapping the phone line gets to hear B tell  $E$  to A.

**One-pass systems.** A one-pass system corresponds to what is commonly called a public file system. In the one-pass model, A simply looks up B's public encryption algorithm,  $E$ , in a "phone book" and uses it to encrypt her message. (One pass is a slight misnomer. At some point, in what we may view as a preprocessing stage, B must have communicated his encryption algorithm, presumably by telling it to whomever publishes the phone book of encryption algorithms, and thus indirectly to A. "One-and-a-half passes" might be more accurate. "Half" refers to the preprocessing stage that needs to be performed only once.) In this case, the choice of message *can* depend on  $E$ .

**2.2. Passes and security.** The main result of this paper is:

GM-security, semantic security, and Y-security (all formally defined in section 3) are equivalent both for one-pass and three-pass cryptosystems.

Interestingly, the equivalence still holds in the one-pass scenario, but the notions of security vary between the one-pass and three-pass scenarios. This point has not been given the proper attention, because people frequently confuse the notion of one-pass public-key cryptography with public-key cryptography in general.

The distinction, however, is crucial for avoiding errors, particularly in cryptographic protocols. Let us informally state the two definitions of security that are achievable in the two scenarios if trapdoor permutations exist.

A three-pass cryptosystem is secure if, for every message  $m$  in the message space, it is impossible to efficiently distinguish an encryption of  $m$  from random noise.

A one-pass cryptosystem is secure if, for every message  $m$  that is efficiently computable on input the encryption algorithm alone, it is impossible to efficiently distinguish an encryption of  $m$  from random noise.

In other words, in the one-pass scenario one cannot just blithely write, “For all messages  $m$ .” For instance, if one closely analyzes all known public-key cryptosystems, it is *conceivable* that if  $(E, D)$  is an encryption/decryption pair, then  $D$  can be easily computed from  $E(D)$ . For instance, the constructive reduction of security to quadratic residuosity given by Goldwasser and Micali [6] for their cryptosystem would *vanish* if the encrypted message is allowed to be  $D$  itself.<sup>1</sup>

Such problems cannot arise in the three-pass scenario because the encryption algorithm  $E$  is selected after and independently of the message  $m$ .

In this paper we concentrate on providing all the details of the proofs for the three-pass case, and sketch the results for the one-pass case in the final section. The reason for this choice is that the definitions of security are much more easily stated for three-pass systems. It is much more convenient to say, “For all messages  $m$ ,” than “For all messages  $m$  that are efficiently computable given the encryption algorithm as an input.”

**3. Notions of security for three-pass systems.** In this section we will formally specify our cryptographic scenario, and define the three notions of security. These definitions are the same in spirit as those originally chosen by Goldwasser and Micali and Yao; therefore, we will use either the names they chose or their initials. We will point out explicitly at the end of this section the minor changes we needed to make to reach the right level of generality.

**3.1. Notation and conventions for probabilistic algorithms.** We introduce some generally useful notation and conventions for discussing probabilistic algorithms. (We make the natural assumption that all parties may make use of probabilistic methods.)

We emphasize the number of inputs received by an algorithm as follows. If algorithm  $A$  receives only one input we write “ $A(\cdot)$ ”, if it receives two inputs we write “ $A(\cdot, \cdot)$ ” and so on.

“PS” will stand for “probability space”; in this paper we only consider countable probability spaces. In fact, we deal almost exclusively with probability spaces arising from probabilistic algorithms.

If  $A(\cdot)$  is a probabilistic algorithm, then for any input  $i$ , the notation  $A(i)$  refers to the PS which assigns to the string  $\sigma$  the probability that  $A$ , on input  $i$ , outputs  $\sigma$ . Notice the special case where  $A$  takes no inputs; in this case the notation  $A$  refers to the algorithm itself, whereas the notation  $A()$  refers to the PS defined by running  $A$  with no input. If  $S$  is a PS, denote by  $\Pr_S(e)$  the probability that  $S$  associates with element  $e$ . Also, we denote by  $[S]$  the set of elements which  $S$  gives positive probability. In the case that  $[S]$  is a singleton set  $\{e\}$  we will use  $S$  to denote the value  $e$ ; this is in agreement with traditional notation. (For instance, if  $A(\cdot)$  is an algorithm that, on input  $i$ , outputs  $i^3$ , then we may write  $A(2) = 8$  instead of  $[A(2)] = \{8\}$ .)

<sup>1</sup> Notice that if Bob publishes an encryption algorithm  $E$  in the public file while keeping its associated decryption algorithm  $D$  secret, then any other user, being limited to efficient computation and ignorant of  $D$ , necessarily selects her message  $m$  efficiently from the input  $E$ —maybe without even looking at  $E$ —and perhaps other inputs altogether independent of  $(E, D)$ . However, in designing cryptographic protocols, one would often like to be able to transmit things like  $E(D)$ . For instance, if that type of message were allowed, one would have a trivial solution to the problem of verifiable secret sharing [3].

If  $f(\cdot)$  and  $g(\cdot, \dots)$  are probabilistic algorithms then  $f(g(\cdot, \dots))$  is the probabilistic algorithm obtained by composing  $f$  and  $g$  (i.e., running  $f$  on  $g$ 's output). For any inputs  $x, y, \dots$  the associated probability space is denoted  $f(g(x, y, \dots))$ .

If  $S$  is any PS, then  $x \leftarrow S$  denotes the algorithm which assigns to  $x$  an element randomly selected according to  $S$ ; that is,  $x$  is assigned the value  $e$  with probability  $\Pr_S(e)$ . Note that for the sake of compactness, we may write  $x \leftarrow S$  even in cases where  $S$  has several outputs (say  $x, y$ , and  $z$ ), but  $x$  is the only output in which we are interested at the moment. If  $F$  is a finite set, then the notation  $x \leftarrow F$  denotes the algorithm which assigns to  $x$  an element randomly selected from the PS which has sample space  $F$  and the uniform probability distribution on the sample points. Thus, in particular,  $x \leftarrow \{0, 1\}$  means  $x$  is assigned the result of a coin toss.

The notation  $\Pr(x \leftarrow S; y \leftarrow T; \dots : p(x, y, \dots))$  denotes the probability that the predicate  $p(x, y, \dots)$  will be true, after the ordered execution of the algorithms  $x \leftarrow S$ ,  $y \leftarrow T$ , etc. We use analogous notation for expected value— $\text{Ex}(x \leftarrow S; y \leftarrow T; \dots : f(x, y, \dots))$ —where now  $f$  is a function which takes numerical values.

By  $1^n$  we denote the unary representation of the integer  $n$ , i.e.,

$$\underbrace{11\dots 1}_n.$$

We recall that  $n^{-\omega(1)}$  represents any function of  $n$  that vanishes faster than  $n^{-k}$  for any constant  $k$ .

**3.2. Cryptographic scenario.** Here we specify those elements that are necessary for all public-key cryptography.

A *cryptographic scenario* consists of the following components:

- A *security parameter*  $n$  which is chosen by the user when he creates his encryption and decryption algorithms. The parameter  $n$  will determine a number of quantities (length of plaintext messages, overall security, etc.).
- A sequence of *message spaces*,  $M = \{M_n\}$  from which all plaintext messages will be drawn.  $M_n$  consists of all messages allowed to be sent if the security parameter has been set equal to  $n$ . In order to make our notation simpler (but without loss of generality), we will assume that  $M_n = \{0, 1\}^n$ .

There may also be a probability distribution on each message space,  $\Pr_n : M_n \rightarrow [0, 1]$  such that  $\sum_{m \in M_n} \Pr_n(m) = 1$ . It is interesting to note that one of our definitions, GM-security, makes no mention of the probability distribution, and that definition is nevertheless equivalent to the other two which explicitly depend on the probability distribution on the messages.

We allow probability distributions over message spaces because we are interested in defeating the strongest adversaries possible. Some adversaries may possess a priori knowledge of what plaintext is encrypted by the ciphertext they have intercepted. For instance, in the case where an embassy is under attack, somebody tapping the embassy's encrypted communications will know that the messages, "Should we burn all the documents?" and "Send in the Marines!" are both much more probable than "Hello. How are things back home?"

- A *public-key cryptosystem* is an expected polynomial time algorithm that on input  $1^n$  outputs the description of two polynomial-size circuits  $E$  and  $D$  such that:
  1.  $E$  has  $n$  inputs and  $l(n)$  outputs, and  $D$  has  $l(n)$  inputs and  $n$  outputs. ( $l$  is some polynomial that gives the length of the ciphertext.)



- 2.  $E$  is probabilistic;  $D$  may be as well.
- 3. For all  $m \in \{0, 1\}^n$ ,

$$\Pr((E, D) \leftarrow \mathcal{C}(1^n); \alpha \leftarrow E(m) : D(\alpha) = m) = 1 - n^{-\omega(1)}.^2$$

Notice that  $[E(m)]$  is a set which is typically quite large. Our notation requires us to write  $\alpha \in [E(m)]$  to refer to  $\alpha$ , a particular encryption of  $m$ . Nevertheless, we will sometimes sloppily write  $E(m)$  for a particular encryption of  $m$  when the meaning is clear.

**Two examples.** To illustrate our definition, we now present two examples of a cryptosystem.

First we show how the quadratic residuosity cryptosystem of Goldwasser and Micali [6] fits our definition.  $\mathcal{C}(1^n)$  picks two random distinct primes,  $p_1$  and  $p_2$ , each of length  $n$ , and sets  $N = p_1 p_2$ , and also picks a random quadratic nonresidue with Jacobi symbol  $+1, y \pmod N$ .

The encryption algorithm,  $E$ , to encrypt  $b = b_1 \cdots b_n$  is:

- for each  $b_i \in b$ ,
  - $E$  picks  $x \in \mathbf{Z}_N^*$  at random
  - if  $b_i = 1$   $E$  sets  $e_i = yx^2 \pmod N$
  - else  $E$  sets  $e_i = x^2 \pmod N$
- $E$ 's output is the  $n$ -tuple  $(e_1, \dots, e_n) = E(b)$ .

The decryption algorithm,  $D$ , to decrypt  $e = (e_1, \dots, e_n)$  is

- for each  $e_i \in e$ ,
  - $D$  sets  $b_i$  to be 1 if  $e_i$  is a quadratic residue mod  $n$ , and 0 otherwise.
  - (Note: The whole point of this scheme is that  $D$  gets the factorization of  $N$  from  $\mathcal{C}$  and thus can determine quadratic residuosity.)
- $D$ 's output is  $b_1 \cdots b_n$ .

Our second example illustrates that  $\mathcal{C}$  need not be a *good* cryptosystem merely to meet the definition of a cryptosystem. In fact, the identity function works:  $\mathcal{C}(1^n)$  can simply output  $E = D =$  the identity function on  $\{0, 1\}^n$ .

**3.3. GM-security (three-pass).** This definition is essentially what Goldwasser and Micali [6] called polynomial security.

A *line tapper* is a family of polynomial-size probabilistic circuits  $T = \{T_n\}$ . Each  $T_n$  takes four strings as input and outputs either 0 or 1. However, to make our next equation more readable, we will treat  $T_n$ 's output as being either its second or third input (representing 0 or 1, respectively).

DEFINITION. Let  $\mathcal{C}$  be a public-key cryptosystem.  $\mathcal{C}$  is *GM-secure* if for all line tappers  $T$  and  $c > 0$ , for all sufficiently large  $n$ , for every  $m_0, m_1 \in \{0, 1\}^n$

$$(1)\Pr(m \leftarrow \{m_0, m_1\}; E \leftarrow \mathcal{C}(1^n); \alpha \leftarrow E(m) : T_n(E, m_0, m_1, \alpha) = m) < \frac{1}{2} + n^{-c}.$$

*Remark.* In reading the above definition, one should pay close attention to our notation. Upon casual consideration of equation (1), one might conclude that there are

---

<sup>2</sup> We allow probability of error in the decrypting in order to reach a reasonable level of generality. For instance, before the recent work on primality of Goldwasser and Kilian [5] and Adleman and Huang [1] if one selected primes, to implement the quadratic residuosity cryptosystem described below, by a probabilistic algorithm (such as Solovay and Strassen [10] or Rabin [8]), then an exponentially vanishing fraction of the time one might have made a mistake in determining primality, and thus produced a bogus decryption algorithm.

not any GM-secure cryptosystems! After all, the definition says that the encryption  $E$  must be secure for *any*  $m_0$  and  $m_1$ , both of which are given as inputs to the line tapper. What happens if we put  $m_0 = D$ , a description of the decryption algorithm? The answer to this question is that our notation specifies that *first* we choose  $m$  from  $\{m_0, m_1\}$  (and thus  $m_0$  and  $m_1$  already had been set), and *then* we choose our encryption algorithm. If  $\mathcal{C}$  is GM-secure, then the probability that  $\mathcal{C}(1^n)$  assigns to any given output is quite small, say  $O(2^{-n})$ . Thus there is little worry that  $\mathcal{C}$  will just happen to output a decryption algorithm  $D = m_0$ . Notice how the above definition (via our notation) models the three-pass scenario.

**3.4. Semantic security (three-pass).** Again, this definition is essentially the same as in [6]. It can be viewed as a polynomial-time bounded version of Shannon's "perfect secrecy" [9]. Informally, let  $f$  be *any* function defined on a message space sequence.  $f(m)$  constitutes information about the message  $m$ . Intuitively,  $f$  should be thought of as some particular information about the plaintext that the adversary is going to try to compute from the ciphertext—say the first seventeen bits of the plaintext. A cryptosystem is semantically secure if no adversary, on input  $E(m)$  can compute  $f(m)$  more accurately than by random guessing (taking into account the probability distribution on the message space).

DEFINITION. Let  $\mathcal{C}$  be a public-key cryptosystem, and let  $M = \{M_n\}$  be a sequence of message spaces. Let  $\mathcal{F} = \{f_E : M_n \rightarrow \Sigma^* \mid E \in [\mathcal{C}(1^n)], n \in \mathbb{N}\}$  be any set of functions on the message spaces. For any value  $v \in \Sigma^*$ , we denote by  $f_E^{-1}(v)$  the inverse image of  $v$ ; that is, the set  $\{m \in M_n \mid f_E(m) = v\}$ . Then the probability of the most probable value for  $f_E(m)$  is  $p_E = \max \left\{ \sum_{m \in f_E^{-1}(v)} \Pr_n(m) \mid v \in \Sigma^* \right\}$ .  $p_E$  is the maximum probability with which one could guess  $f_E(m)$  knowing only the probability distribution from which  $m$  has been drawn.

$\mathcal{C}$  is *semantically secure* if for all message space sequences  $M$ , for all families of functions  $\mathcal{F}$ , for every family of polynomial-size probabilistic circuits  $A = \{A_n(\cdot, \cdot)\}$ , for all  $c > 0$ , and for all sufficiently large  $n$

$$(2) \quad \Pr(m \leftarrow M_n; E \leftarrow \mathcal{C}(1^n); \alpha \leftarrow E(m) : A_n(E, \alpha) = f_E(m)) < p_E + \frac{1}{n^c}.$$

Notice that  $p_E$  implicitly depends on  $n$ , because  $E$  depends on  $n$ . Notice also that we quantify over message spaces in order to take into account all possible probability distributions on the messages.

**3.5. Y-security (three-pass).** Yao's definition [11] is inspired by information theory, but its context differs from classical information theory in that the communicating agents, A(lice) and B(ob), are limited to probabilistic polynomial-time computations. Note that the goals and knowledge of A and B in information theory have no resemblance to those of their cryptographic cousins described above in section 2.

An intuitive explanation of Yao's definition is the following: A has a series of  $n^k$  messages, selected from a probability space known to both A and B, and an encryption of each message. She wishes to transmit enough bits to B so that he can (in polynomial time with very high probability) compute all the plaintexts. A cryptosystem is Y-secure if the average number of bits A must send B is essentially the same regardless of whether or not B possesses a copy of the ciphertexts.

We now make this notion precise, first by defining "Alice and Bob," and then by eventually defining Y-security itself.

Let  $M = \{M_n\}$  be a sequence of message spaces. Each  $M_n$  is  $\{0, 1\}^n$  with a fixed probability distribution. (Note that an information theorist would consider  $M$  to be a sequence of *sources*.)

For the sake of compactness of notation, the expression  $\vec{m}$  will denote a particular series of  $n^k$  messages. That is,  $\vec{m}$  stands for  $m_1, m_2, \dots, m_{n^k}$ .

Let  $f$  be any positive function such that  $f(n) \leq n$ . Intuitively,  $f(n)$  is the number of bits per message that A must transmit to B in order for B to recover the plaintexts. Recall that all the messages in  $M_n$  have length  $n$ .

**DEFINITION.** An  $f$  compressor/decompressor pair (hereinafter  $c/d$  pair) for  $M$  is a pair of families of probabilistic polynomial-size circuits,  $\{A_n\}$  and  $\{B_n\}$ , satisfying the following three properties for some constant  $k$  and all sufficiently large  $n$ :

1. " $B_n$  understands  $A_n$ ."

$$(3) \quad \Pr(m_1 \leftarrow M_n; \dots; m_{n^k} \leftarrow M_n; \beta \leftarrow A_n(\vec{m}); \\ y \leftarrow B_n(\beta) : \vec{m} = y) = 1 - n^{-\omega(1)}.$$

2. " $A_n$  transmits at most  $f(n)$  bits per message."

$$(4) \quad \text{Ex} \left[ m_1 \leftarrow M_n; \dots; m_{n^k} \leftarrow M_n; \beta \leftarrow A_n(\vec{m}) : \frac{|\beta|}{n^k} \right] \leq f(n).$$

3. "The output of  $A_n$  can be parsed."

For all polynomials  $Q$  there exists a probabilistic polynomial-time Turing machine  $S^Q$  such that  $S^Q$  takes as input  $n$  and a concatenated string of  $Q(n)$   $\beta$ 's, each of which is a good output from  $A_n$ , and separates them. That is, its input is  $\beta_1\beta_2 \dots \beta_{Q(n)}$  and its output is  $\beta_1\#\beta_2\#\dots\#\beta_{Q(n)}$ . We require that

$$(5) \quad \Pr(S^Q \text{ correctly splits } \beta_1\beta_2 \dots \beta_{Q(n)}) = 1 - n^{-\omega(1)}.$$

*Remark.* The requirement that  $S^Q$  exist is a technical requirement. It creates a finite analogue of classical information theory's requirement that messages be transmitted one bit at a time, in an infinite sequence of bits.

We say that the *cost of communicating  $M$  is less than or equal to  $f(n)$* , in symbols  $C(M) \leq f(n)$ , if there exists an  $f(n)$   $c/d$  pair for  $M$ .

We define  $C(M) > f(n)$  to be the negation of  $C(M) \leq f(n)$ —that is, *any* circuits "communicating  $M$ " must use at least  $f(n)$  bits. The definition of  $C(M) = f(n)$  is analogous.

Let  $\mathcal{C}$  be a cryptosystem. We define  $C(M | E_{\mathcal{C}}(M)) \leq f(n)$ , and  $C(M | E_{\mathcal{C}}(M)) > f(n)$ , *the cost of communicating  $M$  given encryptions from  $\mathcal{C}$  in a manner analogous to  $C(M)$* . The only difference is that now *both  $A_n$  and  $B_n$  also get  $E$  and the  $n^k$  values of some encryption function  $E \in [\mathcal{C}(1^n)]$  as inputs*. That is, for this definition we must rewrite equation (3) above to read:

$$(6) \quad \Pr(m_1 \leftarrow M_n; \dots; m_{n^k} \leftarrow M_n; E \leftarrow \mathcal{C}(1^n); \\ \alpha_1 \leftarrow E(m_1); \dots; \alpha_{n^k} \leftarrow E(m_{n^k}); \\ \beta \leftarrow A_n(E, \vec{m}, \vec{\alpha}); y \leftarrow B_n(E, \beta, \vec{\alpha}); \vec{m} = y) = 1 - n^{-\omega(1)}.$$

An analogous change must also be made to equation (4).

Notice that for this definition, the probabilities involved must be taken over the different choices of  $E$  from  $\mathcal{C}$  as well as everything else.

**DEFINITION.** Let  $\mathcal{C}$  be a public-key cryptosystem. Fix a sequence of message spaces  $M = \{M_n\}$  (and thus the probability distribution on each  $M_n$ ). We say that  $\mathcal{C}$  is *Y-secure with respect to  $M$*  if

$$(7) \quad C(M | E_{\mathcal{C}}(M)) \leq f(n) \Rightarrow C(M) \leq f(n) + n^{-\omega(1)}.$$

We say that  $\mathcal{C}$  is *Y-secure* if for all  $M$ ,  $\mathcal{C}$  is Y-secure with respect to  $M$ .

**3.6. The original definitions versus ours.** As we pointed out in the Introduction, we made minor changes in cryptographic scenario from [6] and [11]. Here we will spell out what those changes are and why they were made.

**Changes to Goldwasser and Micali's definition.** There are two ways a cryptosystem (the server that generates encryption/decryption algorithm pairs) can achieve security:

1. The cryptosystem gets a description of a message space  $M$  (and thus its probability distribution) as one of its inputs and will output an encryption/decryption algorithm pair to securely encrypt  $M$ .
2. The cryptosystem is told nothing about the message space. The encryption algorithms it outputs are supposed to be secure for every possible message space.

We will call the former cryptosystems *adaptive* and the latter *oblivious*.

Goldwasser and Micali consider adaptive cryptosystems for both of their definitions of security [6]; Yao does not make it clear which type of cryptosystem he is assuming for his definition of security [11]. We believe it makes more sense to consider oblivious cryptosystems, for both theoretical and applied reasons.

The theoretical reason for preferring oblivious cryptosystems is that all three definitions of security are equivalent. (See section 4.) This is a desirable property that fails to hold for adaptive cryptosystems, as we will show in the next section.

The practical reason for preferring oblivious cryptosystems is that, although it is certainly conceivable that having knowledge of the message space would allow one to design a better encryption algorithm, cryptographers have in fact normally tried to design cryptosystems that are secure for all message spaces. For example, consider the cryptosystem based on arbitrary trapdoor predicates proposed by Goldwasser and Micali [6]. Although they only considered security in the adaptive cryptosystem sense, their cryptosystem is in fact secure in the stronger, oblivious sense.

**Changes to Yao's definition.** In [11], Yao assumes deterministic private-key cryptography, but the definition is immediately extended to probabilistic public-key cryptography.

Yao defines the compressor  $A$  and decompressor  $B$  to be Turing machines, not circuits. We have switched to circuits because it is not clear that there are *any* secure cryptosystems with respect to probabilistic Turing machines. It might be that one can always achieve greater polynomial-time compression given the ciphertext simply because having a *shared* pseudorandom string (in this case the ciphertext!) helps. If it does help, however, having made the compressor and decompressor nonuniform circuits, we can always hardwire in a shared random string of bits.

**3.7. Inequivalence of the original definitions.** In this section, we point out that, for adaptive cryptosystems, GM-security is a notion stronger than either semantic security or Y-security. We do this in the following two claims, each supported by

an informal argument. These claims can be easily transformed to theorems after formalizing the discussed security notions in terms of adaptive cryptosystems, a tedious effort once we have realized that the adaptive setting is not the “right” one.

**CLAIM 1.** *If any GM-secure adaptive public-key cryptosystem exists, then there exist adaptive public-key cryptosystems that are semantically secure but not GM-secure.*

Let  $\mathcal{C}(\cdot, \cdot)$  be any GM-secure (and thus semantically secure) adaptive cryptosystem. We will construct a  $\mathcal{C}'(\cdot, \cdot)$  that is still semantically secure, but is not GM-secure.

$\mathcal{C}'$  behaves identically to  $\mathcal{C}$  for all message spaces, except for the message space  $\{0, 1\}^n$  with uniform probability distribution. In this case,  $\mathcal{C}'$  runs  $\mathcal{C}$  to compute an encryption algorithm  $E$ , and then outputs the algorithm  $E'$  defined by:

$$(8) \quad E'(x) = \begin{cases} 0^n & \text{if } x = 0^n, \\ E(x) \neq 0^n & \text{otherwise.} \end{cases}$$

$\mathcal{C}'$  is clearly not GM-secure, because, for the special message space described above, there is a message,  $0^n$ , which is easily distinguished from other messages by its encryption. However,  $\mathcal{C}'$  is still semantically secure. The message  $0^n$  has such a low probability weight that it will not give an adversary any significant advantage—on average—in computing a function of the plaintext on input the ciphertext.  $\square$

Note that the above argument would not go through if  $\mathcal{C}$  were an oblivious cryptosystem. An oblivious  $\mathcal{C}$  would not receive any description of the probability distribution on the messages, so it would be unable to alter its output depending on what that probability distribution is.

**CLAIM 2.** *If any GM-secure adaptive public-key cryptosystem exists, then there exist adaptive public-key cryptosystems that are Y-secure but not GM-secure.*

We construct exactly the same  $\mathcal{C}'$  as we did for the previous claim.  $\mathcal{C}'$  is of course not GM-secure. However, the “weak message” has such low probability that it basically does not affect the average number of bits necessary to communicate messages from the message space. Thus  $\mathcal{C}'$  is Y-secure.  $\square$

**4. Main results.** In this section we provide the proof of the equivalence of GM-security, semantic security, and Y-security. We choose to do these proofs by showing that GM-security is equivalent to Y-security and that GM-security is equivalent to semantic security. We present here only three of the four necessary implications. The proof that GM-security implies semantic security may be found in [6]. We will present the three proofs in order of increasing difficulty and technical complexity.

**4.1. Semantic security implies GM-security.** This proof is quite simple. If a cryptosystem is not GM-secure, then there exist two messages,  $m_1$  and  $m_2$ , which we can easily distinguish. If we make a new message space in which these are the only messages, then given only a ciphertext, one has a better than random chance of figuring out which of the two plaintext messages this ciphertext represents.

**THEOREM 1.** *Let  $\mathcal{C}$  be a public-key cryptosystem. If  $\mathcal{C}$  is semantically secure, then  $\mathcal{C}$  is GM-secure.*

*Proof.* We prove the contrapositive. Let  $\mathcal{C}$  be a public-key cryptosystem that is not GM-secure. We will prove that  $\mathcal{C}$  is not semantically secure.

Formally,  $\mathcal{C}$  is not GM-secure means that there exist a line tapper  $T$  and a  $c > 0$  such that for infinitely many  $n$  there are  $m_1^n, m_2^n \in M_n$  for which

$$(9) \Pr(m \leftarrow \{m_1^n, m_2^n\}; E \leftarrow \mathcal{C}(1^n); \alpha \leftarrow E(m) : T_n(E, m_1^n, m_2^n, \alpha) = m) \geq \frac{1}{2} + \frac{1}{n^c}.$$

We construct a new message space  $M_n$  as follows: For those  $n$  for which equation (9) holds,  $\Pr_n(m_1^n) = 1/2$  and  $\Pr_n(m_2^n) = 1/2$ .

We have set up the message space so one can simply guess the plaintext by seeing the ciphertext. More precisely, a circuit  $A_n$  can compute the function of the plaintext  $f_E(m) = m$  from the ciphertext by having the values  $m_1^n$  and  $m_2^n$  hardwired into it, and using  $T$  as a subroutine. By equation (9), such an  $A_n$  has a polynomial advantage over random guessing. On the other hand, without seeing the ciphertext, circuits with no input can only randomly guess the plaintext.  $\square$

**4.2. Y-security implies GM-security.** In the proof of the next theorem, we use a technical lemma, one of Hoeffding's Inequalities [7], that is a variation of Chernoff's bound [2].

LEMMA 1 (HOEFFDING). *Let  $X$  be a random variable having binomial distribution, with  $r$  trials and probability of success  $p$ . For  $0 \leq \alpha \leq 1/2 \leq p \leq 1$ , we have  $\Pr(X \leq \alpha r) \leq e^{-2(p-\alpha)^2 r}$ .*

THEOREM 2. *Let  $C$  be a cryptosystem. If  $C$  is Y-secure, then  $C$  is GM-secure.*

*Proof.* Again we will prove the contrapositive. Let  $C$  be a cryptosystem that is not GM-secure for some message space  $M$ . Then there exist a family of line tappers  $T = \{T_n\}$  and an infinite subset  $N' \subseteq \mathbb{N}$  such that for some constant  $j$ , and all  $n \in N'$ , there are  $m_0^n, m_1^n \in M_n$  such that

$$\Pr(m \leftarrow \{m_1^n, m_2^n\}; E \leftarrow C(1^n); \alpha \leftarrow E(m) : T_n(E, m_1^n, m_2^n, \alpha) = m) \geq \frac{1}{2} + \frac{1}{n^j}.$$

Consider now a new message space  $M'$  that, for  $n \in N'$ , has  $\Pr_n(m_0^n) = 1/2$ ,  $\Pr_n(m_1^n) = 1/2$ , and  $\Pr_n(m) = 0$  for all other  $m \in \{0, 1\}^n$ , and for  $n \in \overline{N'}$  has  $\Pr_n(0^n) = 1/2$ ,  $\Pr_n(1^n) = 1/2$ , and  $\Pr_n(m) = 0$  for all other  $m$ .

Clearly for all  $d > 0$ ,  $C(M') > 1 - n^{-d}$ : Any circuits not sharing ciphertext would need exactly one bit per message to perfectly communicate outputs from  $M'$ , and our definition in equation (3) allows an error of at most  $n^{-\omega(1)}$ .

On the other hand, we will now show that

$$C(M' | E_C(M')) \leq \begin{cases} 1 - 1/n^k & \text{if } n \in N', \\ 1 & \text{otherwise} \end{cases}$$

where  $k = 2j + 1$ . This value is achieved by a shared ciphertext  $c/d$  pair that transmits  $n^k$  messages at a time.

$A_n$  gets  $n^k$  messages in both cleartext and ciphertext as its input. Since there are only two messages in  $M'_n$ , each message can be considered to be a bit  $b$  and each ciphertext the encryption of a bit. That is to say,  $A_n$ 's input is  $b_1, b_2, \dots, b_{n^k}, \alpha_1, \alpha_2, \dots, \alpha_{n^k}$  where  $\alpha_i \in [E(b_i)]$ .  $A_n$  now XORs each adjacent pair of messages (bits). That is, put  $c_i = b_i \oplus b_{i+1}$  for  $i = 1, 2, \dots, n^k - 1$ . Put  $\beta = c_1 c_2 \dots c_{n^k - 1}$ . This  $\beta$  is the "hint" that  $A_n$  sends to  $B_n$ . Obviously,  $|\beta|/n^k = 1 - 1/n^k$ .

Now, can  $B_n$ , given  $\beta$  and the  $\alpha_i$ 's as its input, determine the plaintext with probability  $1 - n^{-\omega(1)}$ ? Yes. The "hint",  $\beta$ , constrains  $B_n$  to only two possible choices of values for the  $b_i$ . That is, if  $B_n$  decides that  $b_1 = 0$ , then it knows the value of all the bits—say  $v_1 v_2 \dots v_{n^k}$ . On the other hand, if  $B_n$  decides that  $b_1 = 1$ , then the whole series of messages must of have been  $\bar{v}_1 \bar{v}_2 \dots \bar{v}_{n^k}$  (where  $\bar{v}$  is the complement of  $v$ ).

$B_n$  also has a line tapper,  $T_n$ , that it can use to test the  $\alpha_i$ .  $B_n$  runs  $T_n$  on each  $\alpha_i$  and obtains  $T_n$ 's opinion as to what each bit was. Call this sequence  $t_1 t_2 \dots t_{n^k}$ .

Since  $\mathcal{C}$  is not GM-secure, each  $t_i$  is correct with probability  $p = \frac{1}{2} + 1/n^j$ , for some fixed  $j$ . By Lemma 1 (with  $\alpha = 1/2$  and  $r = n^k$ ), if we make  $k \geq 2j + 1$ , then the majority of  $t_i$ 's will be correct with probability  $1 - O(e^{-2n})$ .  $B_n$  compares the  $t_i$  to both the  $v_i$  and the  $\bar{v}_i$ , and decides either  $b_1 = 0$  if the majority of  $t_i$  coincide with the  $v_i$ , or  $b_1 = 1$  if the majority of the  $t_i$  coincide with the  $\bar{v}_i$ .

Because for infinitely many  $n$ ,  $C(M' | E_{\mathcal{C}}(M')) \leq 1 - 1/n^k$ , but for all  $n$ ,  $C(M') > 1 - 1/2n^k$ ,  $M'$  is *not* Y-secure.  $\square$

### 4.3. GM-security implies Y-security.

**THEOREM 3.** *Let  $\mathcal{C}$  be a public-key cryptosystem. If  $\mathcal{C}$  is GM-secure, then  $\mathcal{C}$  is Y-secure.*

*Proof.* We will prove the contrapositive. A bird's-eye view of our proof is as follows. Assuming that  $\mathcal{C}$  is not Y-secure, there exists a good shared ciphertext c/d pair that manages to communicate using "few" bits. This pair will allow us to test (for some special pair of messages  $m_1$  and  $m_2$ ) whether a particular  $\alpha$  is the encryption of either  $m_1$  or  $m_2$  thus violating the GM-security condition. Namely, if the pair works successfully on inputs  $\alpha$  and  $m_1$ , we declare  $\alpha$  to be an encryption of  $m_1$ ; otherwise we declare  $\alpha$  to be an encryption of  $m_2$ .

Let us proceed formally. Since  $\mathcal{C}$  is not Y-secure, there is a particular message space sequence,  $M = \{M_n\}$ , such that  $\mathcal{C}$  is not Y-secure for  $M$ . That is to say, there exist a shared ciphertext c/d pair  $(A, B)$  (which we will use as shorthand for  $(\{A_n\}, \{B_n\})$ ), a positive integer  $k$ , and a polynomial  $P$  such that

- ( $\star$ )  $A_n$  communicates  $n^k$  messages from  $M_n$  to  $B_n$  using "few" bits per message—on top of the ciphertext which they get to share for free.
- ( $\star\star$ ) Furthermore, for every c/d pair  $(A', B')$ , there exists an infinite subset  $N' \subseteq \mathbb{N}$ , such that for all  $n \in N'$ , on average  $(A', B')$  uses at least  $1/P(n)$  more bits per message than does  $(A, B)$ . (Intuitively, the point is that  $(A', B')$  is not a *shared ciphertext* c/d pair;  $A'$  and  $B'$  share nothing except that  $B'$  "knows"  $\beta$ .)

We are now going to run a series of experiments to see how  $(A, B)$  behaves on inputs that it does not "expect." We begin, however, by running a control experiment.

In experiment  $n\text{-EXP}_0$ , we pick  $n^k$  messages  $m_i$  at random from  $M_n$  and an  $E$  at random from  $[\mathcal{C}(1^n)]$ , and run  $A_n$  on input

$$\begin{array}{cccccc} m_1 & m_2 & m_3 & \dots & m_{n^k} \\ E(m_1) & E(m_2) & E(m_3) & \dots & E(m_{n^k}) \end{array}$$

(The output will be a string  $\beta$  such that  $B_n$ , on input  $\beta$  and  $E(m_1), \dots, E(m_{n^k})$  will output  $m_1, \dots, m_{n^k}$  with overwhelming probability.)

Now consider the following experiment,  $n\text{-EXP}_i$ : This time we again pick  $n^k$  messages and an  $E$  at random, but we also pick one more message,  $r$ , at random from  $M_n$ , and set  $\rho = E(r)$ . Now we run  $A_n$  with  $i$  copies of  $\rho$  replacing the first  $i$  ciphertexts in its input. A "picture" of  $A_n$ 's input is

$$\begin{array}{cccccc} m_1 & \dots & m_i & m_{i+1} & \dots & m_{n^k} \\ \rho & \dots & \rho & E(m_{i+1}) & \dots & E(m_{n^k}) \end{array}$$

We then run  $B_n$  with  $A_n$ 's output and  $\rho, \rho, \dots, E(m_{n^k})$  as its inputs.

**DEFINITION.** We define the *difference between  $n\text{-EXP}_i$  and  $n\text{-EXP}_j$* ,  $d_n(i, j)$  to be the maximum of the average difference between

1. the length of the  $\beta$ 's output by  $A_n$  in the two experiments, and

2. the frequency with which  $B_n$  recovers the correct plaintexts in the two experiments.

(This definition is unusual in that we are comparing a number of bits with a probability. Nevertheless, both are numbers, so one can indeed take a maximum, and, indeed, both are handled in a similar manner in this proof.)

CLAIM 3. *There exists a polynomial  $Q$  such that for an infinite subset  $N'' \subseteq \mathbb{N}$  and for all  $n \in N''$   $d_n(0, n^k) > 1/Q(n)$ .*

*Proof of Claim 3.* By contradiction. Assume that for all sufficiently large  $n$ ,  $n\text{-EXP}_0$  is indistinguishable from  $n\text{-EXP}_{n^k}$ . Then  $A_n$  and  $B_n$  still function successfully on input

$$\begin{array}{cccccc} m_1 & m_2 & m_3 & \dots & m_{n^k} \\ \rho & \rho & \rho & \dots & \rho \end{array}$$

where  $m_i, r, \rho$ , and  $E$  are as above. We now construct  $A'_n, B'_n$  to violate  $(\star\star)$ . We simply hardwire the encryption of some random string into a pair of circuits which are identical to  $A_n$  and  $B_n$  except that they do not share any ciphertext. By assumption, these circuits are a c/d pair violating  $(\star\star)$ .  $\square$

CLAIM 4. *For all  $n \in N''$ , there is a polynomial  $Q'$  and an  $i, 0 \leq i \leq n^k - 1$ , such that  $d_n(i, i + 1) > 1/Q'(n)$ .*

*Proof of Claim 4.* Fix  $n \in N''$ .  $d_n(0, n^k) \geq 1/Q(n)$ . Therefore, there must be an  $i$  such that  $d_n(i, i + 1) \geq \frac{1}{n^k Q(n)}$ .  $\square$

Let  $n \in N''$ . We will consider the case where  $i = 0$  in Claim 4, and  $d_n(0, 1)$  is due to a difference in the length of  $A_n$ 's output, rather than  $B_n$ 's success rate. The other case, where the difference is due to a difference in  $B_n$ 's success rate, is similar, but simpler.

Let us restate Claim 4 in a more convenient form. Consider the following *joint* experiment,  $n\text{-EXP}_{01}$ . Randomly draw  $r, m_1, \dots, m_{n^k}$  from  $M_n$  and set  $E \leftarrow \mathcal{C}(1^n)$ . Run both  $n\text{-EXP}_0$  and  $n\text{-EXP}_1$  on the *same* inputs. That is, run  $n\text{-EXP}_0$  on input

$$\begin{array}{cccccc} m_1 & m_2 & m_3 & \dots & m_{n^k} \\ E(m_1) & E(m_2) & E(m_3) & \dots & E(m_{n^k}) \end{array}$$

to compute  $A_n$ 's output  $\beta_0$  and run  $n\text{-EXP}_1$  on input

$$\begin{array}{cccccc} m_1 & m_2 & m_3 & \dots & m_{n^k} \\ E(r) & E(m_2) & E(m_3) & \dots & E(m_{n^k}) \end{array}$$

to compute  $A_n$ 's output on this input,  $\beta_1$ . The output of  $n\text{-EXP}_{01}$  is  $|\beta_1| - |\beta_0|$ . Then, by the linearity of the average we get that the expected value of the output of  $n\text{-EXP}_{01}$  is at least  $1/Q'(n)$ .

From this it immediately follows that

$(\star\star\star)$  there exist  $\bar{r}, \bar{m}_1, \bar{m}_2, \dots, \bar{m}_{n^k}$  in  $M_n$  such that the expected value of the output of  $n\text{-EXP}_{01}$  is still greater than  $1/Q'(n)$  when the average of the length of  $\beta$  is computed only over the choice of  $E \leftarrow \mathcal{C}(1^n)$  and of encryptions of messages.

Now for all  $n \in N''$  we can build a tapper  $T_n$  that will succeed in distinguishing two messages  $m_1^n$  and  $m_2^n$ , described below.

Fix  $\bar{r}$  and  $\bar{m}_1$  to be messages that fit the requirements of  $(\star\star\star)$ . We set  $T_n$ 's inputs:  $m_1^n = \bar{m}_1$  and  $m_2^n = \bar{r}$ .  $T_n$  gets as inputs  $E \in [\mathcal{C}(1^n)]$ ,  $m_1^n, m_2^n$ , and  $\alpha$ , where either  $\alpha \in [E(m_1^n)]$  or  $\alpha \in [E(m_2^n)]$ .



Unfortunately, merely because the *average* length of  $\beta$  output by  $A_n$  is shorter if its input includes  $E(\bar{m}_1)$  than if its input includes  $E(\bar{r})$  does not imply that a shorter  $\beta$  occurs *more frequently* with input  $E(\bar{m}_1)$ . Indeed, it might be the case that for most  $E \in [\mathcal{C}(1^n)]$ ,  $A_n$  outputs a longer  $\beta$  given  $E(\bar{m}_1)$  than given  $E(\bar{r})$ , but for a few  $E \in [\mathcal{C}(1^n)]$ ,  $A_n$  outputs a much shorter  $\beta$  given  $E(\bar{m}_1)$ .

What we can say for certain is only that

$$(10) \quad \text{Ex}[|\beta| \mid \alpha \in [E(m_2^n)]] - \text{Ex}[|\beta| \mid \alpha \in [E(m_1^n)]] > 1/Q'(n)$$

where the expectations are computed over the choice of  $E \leftarrow \mathcal{C}(1^n)$ . Now

$$(11) \quad \text{Ex}[|\beta|] = \sum_i^{R(n)} i \Pr(A_n \text{'s output has length } i)$$

where  $R(n)$  is a bound on the running time of  $A_n$ . Therefore, there must be some  $i$  such that if  $|\beta| = i$ , then it is at least  $1/[Q'(n)R(n)]$  more likely that  $\alpha$  is an encryption of  $m_2^n$ .

$T_n$  sets  $m = m_1^n$  and runs  $A_n$  on input

$$\begin{array}{cccccc} m & \bar{m}_2 & \bar{m}_3 & \dots & \bar{m}_{n^k} \\ \alpha & E(\bar{m}_2) & E(\bar{m}_3) & \dots & E(\bar{m}_{n^k}). \end{array}$$

If  $|\beta| \neq i$ , then  $T_n$  simply flips a coin. If  $|\beta| = i$ , the  $T_n$  guesses  $m_2^n$ .  $T_n$  is correct at least polynomially more than half the time.<sup>3</sup>  $\square$

Notice that at several points in the proof we took advantage of the fact that  $T_n$  is nonuniform.  $v$  is hardwired into  $T_n$ , as are  $\bar{r}, \bar{m}_1, \dots, \bar{m}_{n^k}$ . In fact, most of these uses of nonuniformity could be replaced by polynomial-size Monte Carlo experiments. However,  $T_n$  must be nonuniform since  $A_n$  and  $B_n$  are nonuniform.

**5. One-pass scenarios.** In this section we present the proper definitions of security for one-pass cryptography. These definitions are all considerably more complicated than the analogous ones for the three-pass scenario. They are equivalent to one another but *not* to the three-pass definitions. Instead of proving all implications, we only show how to extend the proof that semantic security implies GM-security from the three-pass to the one-pass scenario. Extending the other implication presents no additional difficulties.

**5.1. GM-security (one-pass).** As discussed above in section 2.2, for a one-pass cryptosystem, we must change from requiring security “for all messages  $m$ ,” to requiring security for every message  $m$  that is efficiently computable on input of the encryption algorithm alone. In order to do this, we introduce an adversary called a message finder.

A *message finder* is a family of polynomial-size probabilistic circuits  $F = \{F_n(\cdot)\}$  each of which takes the description of an encryption algorithm as its input and has two messages of length  $n$  as its output. Intuitively, on input  $E$ ,  $F_n$  tries to find  $m_0$  and  $m_1$  such that it is easy for a fellow adversary (a line tapper) to distinguish encryptions of  $m_0$  from encryptions of  $m_1$ .

<sup>3</sup> Notice that if we were working with a difference in  $B_n$ 's success rate instead of a difference in  $|\beta|$ , then a much simpler algorithm would work:  $T_n$  guesses  $\alpha = m_1^n$  if and only if  $B_n$  decodes correctly.

DEFINITION. Let  $\mathcal{C}$  be a public-key cryptosystem.  $\mathcal{C}$  is *GM-secure (one-pass)* if for all message finders  $F$ , line tappers  $T$ , and  $c > 0$ , for all sufficiently large  $n$ ,

$$(12) \quad \Pr(E \leftarrow \mathcal{C}(1^n); m_0, m_1 \leftarrow F_n(E); m \leftarrow \{m_0, m_1\}; \\ \alpha \leftarrow E(m) : T_n(E, m_0, m_1, \alpha) = m) \leq \frac{1}{2} + n^{-c}.$$

**5.2. Semantic security (one-pass).** To change the definition of semantic security to fit the one-pass scenario, we need to introduce something like the message finders of the previous section. For semantic security, however, we are concerned not with finding two “weak” messages, but rather with the probability distribution of the entire message space. Thus our second adversary will not pick out particular messages, but instead set the probability distribution of the message space. Furthermore, we now explicitly give the other adversary a description of that probability distribution.

A *message space enemy* is a family of polynomial-size probabilistic circuits  $B = \{B_n(\cdot)\}$ . Each  $B_n$  takes the description of an encryption algorithm as its input, and outputs the description of a probabilistic Turing machine  $N(\cdot)$ .  $N$  outputs elements of  $\{0, 1\}^n$  with some probability distribution.

As in the three-pass definition, we let  $V$  be any set and let  $\mathcal{F} = \{f_n^E : M_n \rightarrow V \mid E \in [\mathcal{C}(n)]\}$  be any set of functions. Again set  $p_N^E$  to be the probability of the most probable value for  $f(m)$ ; set  $p_n^E = \max\{\sum_{m \in f_n^{E^{-1}(v)}} \Pr_n(m) \mid v \in V\}$ .

DEFINITION. Let  $\mathcal{C}$  be a public-key cryptosystem.  $\mathcal{C}$  is *semantically secure* if for every message space enemy  $B$ , family of polynomial-size probabilistic circuits  $A = \{A_n(\cdot, \cdot, \cdot)\}$ , and  $c > 0$ , for all sufficiently large  $n$

$$(13) \quad \Pr(E \leftarrow \mathcal{C}(1^n); N \leftarrow B_n(E); m \leftarrow N()); \\ \alpha \leftarrow E(m) : A_n(E, N, \alpha) = f_n^E(m)) < p_n^E + \frac{1}{n^c}.$$

**5.3. Y-security (one-pass).** The changes that must be made to the definition of Y-security are completely analogous to the changes we made to the definition of semantic security.

**5.4. Equivalence.** The proofs that the three definitions of security are all equivalent are quite similar to the proofs for the three-pass case. Here we will redo only the proof of the easiest of the four implications, semantic security implies GM-security. This proof shows the additional details that must be taken into consideration when working with the one-pass scenario.

**THEOREM 4.** *GM-security (one-pass), semantic security (one-pass), and Y-security (one-pass) are all equivalent.*

*Proof that semantic security (one-pass) implies GM-security (one-pass).* We will, as usual, prove the contrapositive. Let  $\mathcal{C}$  be a public-key cryptosystem that is not GM-secure. We know that there exist a message finder family of circuits  $F = \{F_n\}$  and a line tapper family of circuits  $T = \{T_n\}$ . We will use the  $F_n$  as subroutines (circuit components to be precise) for building our message space enemy circuits and then use the  $T_n$  to do the distinguishing.

Our message space enemy,  $B_n$ , on input an encryption algorithm  $E \in [\mathcal{C}(1^n)]$ , runs  $F_n$  with input  $E$ .  $F_n$  outputs two messages,  $m_0, m_1 \in \{0, 1\}^n$ .  $B_n$  outputs the

design of a Turing machine  $N()$  such that

$$N \text{ outputs } \begin{cases} m_0 & \text{with probability } 1/2, \\ m_1 & \text{with probability } 1/2. \end{cases}$$

An adversary  $A$  who uses  $T_n$  as a subroutine can distinguish encryptions of  $m_0$  from encryptions of  $m_1$ . In other words, on the message space defined by the output of  $N()$ ,  $A$  can compute the function  $f(m) = m$  (with probability greater than at random) given only an encryption of  $m$ .  $A$  gets  $E$  and  $\alpha$  where either  $\alpha \in [E(m_0)]$  or  $\alpha \in [E(m_1)]$ . However,  $T_n$  also requires  $m_0$  and  $m_1$  as inputs.  $A$  can obtain that information for  $T_n$  simply by running  $N$  a few times.

Formally, since  $\mathcal{C}$  is not GM-secure we know that there exists a  $c > 0$  such that for infinitely many  $n$

$$(14) \quad \Pr(E \leftarrow \mathcal{C}(1^n); m_0, m_1 \leftarrow F_n(E); m \leftarrow \{m_0, m_1\}; \\ \alpha \leftarrow E(m) : T_n(E, m_0, m_1, \alpha) = m) > \frac{1}{2} + n^{-c}.$$

For those  $n$ , equation (14) in fact says that  $\Pr(A \text{ computes } f(m) = m \text{ correctly}) > \frac{1}{2} + n^{-c}$ .  $\square$

**Acknowledgments.** We wish to thank Shafi Goldwasser and Shimon Even for insight and helpful discussions about the notions of security, and general help in preparing this paper. We would like to thank Peter Elias for his help with information theory, and Ravi Boppana for his help with Chernoff bounds. We would also like to thank the referee for his extremely careful reading of our paper and the many improvements he suggested.

## REFERENCES

- [1] L. ADLEMAN AND M. HUANG, *Recognizing primes in random polynomial time*, in Proc. 19th Annual ACM Symposium on the Theory of Computing, 1987, pp. 462–469.
- [2] H. CHERNOFF, *A measure of asymptotic efficiency for tests of a hypothesis based of the sum of observations*, Annal. Math. Statist., 23 (1952), pp. 493–509.
- [3] B. CHOR, S. GOLDWASSER, S. MICALI, AND B. AWERBUCH, *Verifiable secret sharing and achieving simultaneity in the presence of faults*, in Proc. 26th Annual IEEE Symposium on the Foundations of Computer Science, 1985, pp. 383–395.
- [4] W. DIFFIE AND M. E. HELLMAN, *New direction in cryptography*, IEEE Trans. Inform. Theory, IT-22 (1976), pp. 644–654.
- [5] S. GOLDWASSER AND J. KILIAN, *Almost all primes can be quickly certified*, in Proc. 18th Annual ACM Symposium on the Theory of Computing, 1986, pp. 316–329.
- [6] S. GOLDWASSER AND S. MICALI, *Probabilistic encryption*, J. Comput. System Sci., 28 (1984), pp. 270–299.
- [7] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1963), pp. 13–30.
- [8] M. RABIN, *Probabilistic algorithms for testing primality*, J. Number Theory, 12 (1980), pp. 128–138.
- [9] C. E. SHANNON, *Communication theory of secrecy systems*, Bell System Tech. J., 28 (1949), pp. 656–749.
- [10] R. SOLOVAY AND V. STRASSEN, *A fast Monte Carlo test for primality*, this Journal, 6 (1977), pp. 84–85.
- [11] A. C. YAO, *Theory and applications of trapdoor functions (extended abstract)*, in Proc. 23rd Annual IEEE Symposium on the Foundations of Computer Science, 1982, pp. 80–91.

## A FUNCTIONAL APPROACH TO DATA STRUCTURES AND ITS USE IN MULTIDIMENSIONAL SEARCHING\*

BERNARD CHAZELLE†

**Abstract.** We establish new upper bounds on the complexity of multidimensional searching. Our results include, in particular, linear-size data structures for range and rectangle counting in two dimensions with logarithmic query time. More generally, we give improved data structures for *rectangle problems* in any dimension, in a static as well as a dynamic setting. Several of the algorithms we give are simple to implement and might be the solutions of choice in practice. Central to this paper is the nonstandard approach followed to achieve these results. At its root we find a redefinition of data structures in terms of functional specifications.

**Key words.** functional programming, data structures, concrete complexity, multidimensional search, computational geometry, pointer machine, range search, intersection search, rectangle problems

**CR Categories.** 5.25, 3.74, 5.39

**1. Introduction.** This paper has two main parts: in § 2, we discuss a method for transforming data structures using functional specifications; in the remaining sections, we use such transformations to solve a number of problems in multidimensional searching. To begin with, let us summarize the complexity results of this paper.

The generalization of the notion of rectangle in higher dimensions is called a *d-range*: it is defined as the Cartesian product of  $d$  closed intervals over the reals. Let  $V$  be a set of  $n$  points  $\mathfrak{R}^d$  and let  $v$  be a function mapping a point  $p$  to an element  $v(p)$  in a commutative semigroup  $(G, +)$ . Let  $W$  be a set of  $n$   $d$ -ranges.

- (1) **Range counting:** given a  $d$ -range  $q$ , compute the size of  $V \cap q$ .
- (2) **Range reporting:** given a  $d$ -range  $q$ , report each point of  $V \cap q$ .
- (3) **Semigroup range searching:** given a  $d$ -range  $q$ , compute  $\sum_{p \in V \cap q} v(p)$ .
- (4) **Range searching for maximum:** semigroup range searching with maximum as semigroup operation.
- (5) **Rectangle counting:** given a  $d$ -range  $q$ , compute the size of  $\{r \in W \mid q \cap r \neq \emptyset\}$ .
- (6) **Rectangle reporting:** given a  $d$ -range  $q$ , report each element of  $\{r \in W \mid q \cap r \neq \emptyset\}$ .

In each case,  $q$  represents a query to which we expect a fast response. The idea is to do some preprocessing to accommodate incoming queries in a repetitive fashion.

Note that range counting (resp., reporting) is a subcase of rectangle counting (resp. reporting). To clarify the exposition (and keep up with tradition), however, we prefer to treat these problems separately. Other well-known problems falling under the umbrella of rectangle searching include *point enclosure* and *orthogonal segment intersection*. The former involves computing the number of  $d$ -ranges enclosing a given query point, while the latter, set in two dimensions, calls for computing how many horizontal segments from a given collection intersect a query vertical segment. In both cases, the reduction to rectangle counting is immediate.

The thrust of our results is to demonstrate the existence of efficient solutions to these problems that use minimum storage. One of the key ideas is to redesign range

---

\* Received by the editors September 23, 1985; accepted for publication (in revised form) March 9, 1987. A preliminary version of this paper has appeared in the Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science, Portland, Oregon, October 1985, pp. 165-174.

† Department of Computer Science, Princeton University, Princeton, New Jersey 08544. This work was begun when the author was at Brown University, Providence, Rhode Island 02912. This work was supported in part by National Science Foundation grant MCS 83-03925.

trees (Bentley [B2]) and segment trees (Bentley [B1], Bentley and Wood [BW]) so as to require only linear storage. Since improvements in this area typically involve trimming off logarithmic factors, one must be clear about the models of computation to be used. It is all too easy to “improve” algorithms by encoding astronomical numbers in one computer word or allowing arbitrarily complex arithmetic. To guard us from such dubious tricks, we will assume that the registers and memory cells of our machines can only hold integers in the range  $[0, n]$ . In the following, the sign  $\times$  refers to multiplication,  $\div$  to division (truncated to the floor), and *shift* to the operation  $\text{shift}(k) = 2^k$ , defined for any  $k$  ( $0 \leq k \leq \lfloor \log n \rfloor$ ).<sup>1</sup> No operation is allowed if the result or any of the operands falls outside of the range  $[0, n]$ . This will make our model very weak and thus give all the more significance to our upper bounds.

*Remark.* As observed in Gabow et al. [GBT] the “orthogonal” nature of the problems listed above makes it possible to work in rank space, that is, to deal not with the coordinates themselves but with their ranks. This is precisely what we will be doing here. The conversion costs logarithmic time per coordinate, but it has the advantage of replacing real numbers by integers in the range  $[1, n]$ . It is important to keep in mind that although our data structures will use integers over  $O(\log n)$  bits internally, no such restriction will be placed on the input and query coordinates. On the contrary these will be allowed to assume any real values.

The models of computation we will consider include variants of *pointer machines* (Tarjan [T]). Recall that the main characteristic of these machines is to forbid any kind of address calculation. New memory cells can be obtained from a free list and are delivered along with pointers to them. A pointer is just a symbolic name, that is, an address whose particular representation is transparent to the machine and on which no arithmetic operation is defined. Only pointers provided by the free list can be used. For the time being, let us assume that the only operations allowed are  $=$  and  $<$  along with the standard Booleans. We introduce our models of computation in order of increasing power. Of course, we always assume that the semigroup operations which might be defined (problems 3–4) can be performed in constant time.

- (1) An *elementary pointer machine* (EPM) is a pointer machine endowed with  $+$ .
- (2) A *semi-arithmetic pointer machine* (SAPM) is a pointer machine endowed with  $+$ ,  $-$ ,  $\times$ ,  $\div$ .
- (3) An *arithmetic pointer machine* (APM) is a pointer machine endowed with  $+$ ,  $-$ ,  $\times$ ,  $\div$ , *shift*.
- (4) A *random access machine* (RAM) is endowed with comparisons, and  $+$ ,  $-$ ,  $\times$ ,  $\div$ . See Aho et al. [AHU] for details.

Our motivation for distinguishing between these models is twofold: one reason is to show that our basic techniques still work even on the barest machines. Another is to assess the sensitivity of the complexity of query-answering to the model of computation. Next, we briefly discuss these definitions. First of all, complexity is measured in all cases under the uniform cost criterion (Aho et al. [AHU]). Note that subtraction can be simulated in  $O(\log n)$  time and  $O(n)$  space on an EPM by binary search. Similarly, *shift* can be simulated on an SAPM in  $O(\log \log n)$  time by binary search in a tree of size  $O(\log n)$  (this can also be done with constant extra space by repeated squaring). For this reason, we will drop the SAPM model from consideration altogether. Any result mentioned in this paper with regard to an APM also holds on an SAPM, up to within a multiplicative factor of  $\log \log n$  in the time complexity. All

<sup>1</sup> All logarithms are taken to the base 2. Throughout this paper, we will use the notation  $\log^c n$  to designate  $(\log n)^c$ .

the operations mentioned above are in the instruction set of any modern computer, so our models are quite realistic. We have omitted *shift* from the RAM, because this operation can be simulated in constant time by table look-up. One final comment concerns the word-size. Suppose that for some application we need to use integers in the range  $[-n^c, n^c]$ , for some constant  $c > 0$ . Any of the machines described above will work just as well. Indeed, we can accommodate any polynomial range by considering virtual words made of a constant number of actual machine words. The simulation will degrade the time performance by only a constant factor. See Knuth [K2] for details on multiple-precision arithmetic.

*The complexity results.* We have summarized our results for  $\mathfrak{R}^2$  in Tables 1 and 2. The first concerns the static case. Each pair  $(x, y)$  indicates the storage  $O(x)$  required by the data structure and the time  $O(y)$  to answer a query; we use  $\epsilon$  to denote an arbitrary positive real. In the second table one will find our results for the dynamic case. We give successively the storage requirement, the query time, and the time for an insertion or a deletion. In both cases,  $k$  indicates the number of objects to be reported plus one (we add a 1 to treat the no-output case uniformly). The time to construct each of these data structures is  $O(n \log n)$ .

TABLE 1  
The static case.

Problem	RAM	APM	EPM
range/rectangle counting	$(n, \log n)$	$(n, \log n)$	$(n, \log^2 n)$
range/rectangle reporting	$\left( n, k \left( \log \frac{2n}{k} \right)^\epsilon + \log n \right)$ $\left( n \log \log n, k \log \log \frac{4n}{k} + \log n \right)$ $(n \log^\epsilon n, k + \log n)$	$\left( n, k \log \frac{2n}{k} \right)$	$\left( n, k \left( \log \frac{2n}{k} \right)^2 \right)$
range search for max	$(n, \log^{1+\epsilon} n)$ $(n \log \log n, \log n \log \log n)$ $(n \log^\epsilon n, \log n)$	$(n, \log^2 n)$	$(n, \log^3 n)$
semigroup range search	$(n, \log^{2+\epsilon} n)$ $(n \log \log n, \log^2 n \log \log n)$ $(n \log^\epsilon n, \log^2 n)$	$(n, \log^3 n)$	$(n, \log^4 n)$

TABLE 2  
The dynamic case on an EPM.

Problem	Storage	Query time	Update time
range/rectangle counting	$O(n)$	$O(\log^2 n)$	$O(\log^2 n)$
range reporting	$O(n)$	$O(k(\log 2n/k)^2)$	$O(\log^2 n)$
range search for max	$O(n)$	$O(\log^3 n \log \log n)$	$O(\log^3 n \log \log n)$
semigroup range search	$O(n)$	$O(\log^4 n)$	$O(\log^4 n)$
rectangle reporting	$O(n)$	$O(k(\log 2n/k)^2 + \log^3 n)$	$O(\log^2 n)$

In the dynamic case, we have restricted ourselves to the EPM model. Our objective was only to show that the techniques of this paper could be dynamized even in the weakest model. It is likely that many of these bounds can be significantly lowered if we are ready to use a more powerful model such as a RAM or an APM. We leave these improvements as open problems. In the remainder of this paper we will mention upper bounds only in connection with the *weakest* models in which they hold. This is quite harmless as long as the reader keeps in mind that any result relative to an EPM holds on an APM or a RAM. Similarly, anything one can do on an APM can be done just as well on a RAM. Also for the sake of exposition, we restrict ourselves to two dimensions ( $d = 2$ ), but we recall a classical technique (Bentley [B2]) that allows us to extend all our data structures to  $\mathfrak{R}^d$  ( $d > 2$ ). To obtain the complexity of the resulting algorithms, just multiply each expression in the original complexity by a factor of  $\log^{d-2} n$  (note: the terms involving  $k$  remain unchanged, but a term  $\log^{d-1} n$  is to be included in the query times).

Update times are best thought of as amortized bounds, that is, averaged out over a sequence of transactions. A general technique can be used in most cases, however, to turn these bounds into worst-case bounds (Willard and Lueker [WL]). Similarly, a method described in Overmars [O] can often be invoked to reduce deletion times by a factor of  $\log n$ . Finally, we can use a result of Mehlhorn [Me] and Willard [W1] to show that if we can afford an extra  $\log^\epsilon n$  factor in query time then the storage can often be reduced by a factor of  $\log \log n$  with each increment in dimension. We will not consider these variants here for at least two reasons. The first is that the techniques have been already thoroughly exposed and it would be tedious but elementary to apply them to our data structures. The second is that these variants are usually too complex to be practical. We will strive in this paper to present data structures that are easy to implement. We have not succeeded in all cases, but in some we believe that we have. For example, our solutions to range counting are short, simple, and very efficient in practice. To illustrate this point we have included in the paper the code of a Pascal implementation of one of the solutions.

*Comparison with previous work.* Roughly speaking, our results constitute improvements of a logarithmic factor in storage over previous methods. In particular, we present the first linear-size data structures for range and rectangle counting in two dimensions with logarithmic query times. For these two problems our data structures are essentially memory compressed versions of the range tree of Bentley [B2], using new implementations of the idea of a downpointer introduced by Willard [W2]. As regards range and rectangle reporting on a RAM, we improve a method in Chazelle [C1] from  $(n(\log n/\log \log n), k + \log n)$  to  $(n \log^\epsilon n, k + \log n)$ . Interestingly, we have shown in Chazelle [C2] that the  $(n(\log n/\log \log n), k + \log n)$  algorithm is optimal on a pointer machine. This constitutes a rare example (outside of hashing), where a pointer machine is provably less powerful than a RAM. Concerning range search for maximum, we improve over a data structure of Gabow et al. [GBT] from  $(n \log n, \log n)$  to  $(n \log^\epsilon n, \log n)$ . As regards semigroup range searching we present an improvement of a factor  $\log^{1-\epsilon} n$  space (again for any  $\epsilon > 0$ ) over the algorithm for the same problem in Willard [W1]. In the group model (the special case of semigroup range searching where an inverse operation exists) we could not find any obvious way of taking advantage of the inverse operation to improve on the results in the table (except, of course, for the case of range counting). As a result, our  $(n \log^\epsilon n, \log^2 n)$  algorithm may compare favorably with Willard's  $(n \log n, \log n)$  [W2] in storage requirement, but it is superseded in query time efficiency. Our other results for the static case represent tradeoffs and cannot be compared with previous work. In the dynamic case,

our upper bounds improve previous results by a factor of  $\log n$  space but, except for range and rectangle counting, they also entail extra polylogarithmic costs in query and update times. Also, what makes comparisons even more difficult is that in order to prove the generality of our space-reduction techniques we have purposely chosen a very weak model of computation, i.e., the EPM.

**2. Functional data structures.** In trying to assess whether a particular data structure is optimal or not, it is natural to ask oneself: why is the data stored where it is and not elsewhere? The answer is usually “to facilitate the computation of some functions implicitly associated with the records of the data structure.” For example, one will store keys in the nodes of a binary search tree to be able to branch left or right depending on the outcome of a comparison. An important observation is that nothing demands that a node should store its own key explicitly. All that is required is that whenever the function associated with that node is called, the node had better allow for its prompt evaluation. Having the key stored at the node at all times might be handy but it certainly is more than is strictly needed. There is a classical example of this fact: in their well-known data structure for planar point location [LP1], Lee and Preparata start out with a balanced tree whose nodes are associated with various lists. Then they make the key remark that many elements in these lists can be removed because whenever they are needed by the algorithm they will always have been encountered in other lists before. This simple transformation brings down the amount of storage required from quadratic to linear. However different from the previous one the resulting data structure might be (being much smaller, for one thing), it still has the same *functional* structure as before. In other words, the functions and arguments associated with each node, as well as their interconnections, have gone unchanged through the transformation. Only the assignment of data has been altered. This is no isolated case. Actually, many data structures have been discovered through a similar process. We propose to examine this phenomenon in all generality and see if some useful methodology can be derived from it.

There are several ways of looking at data structures from a design point of view. One might choose to treat them as structured mappings of data into memory. This compiler-level view addresses implementation issues and thus tends to be rigid and overspecifying in the early stage of the design process. Instead, one can take data structures a bit closer to the notation of abstract data type, and think of them as combinatorial structures that can be used and manipulated. For example, a data structure can be modeled as a graph with data stored at the nodes (Earley [Ea]). Semantic rules can be added to specify how the structure can be modified and constraints can be placed to enforce certain “shape” criteria (e.g., balance or degree conditions). If needed, formal definitions of data structures can be provided by means of grammars or operational specifications (Gonnet and Tompa [GoT]). Note that despite the added abstraction of this setting, a data structure is still far removed from an abstract data type (Guttag [G]). In particular, unlike the latter, the former specifies an *architecture* for communicating data and operating on it.

The framework above favors the treatment of data structures as combinatorial objects. It emphasizes *how* they are made and used rather than *why* they are the way they are. This is to be expected, of course, since a data structure may be used for many different purposes, and part of its interpretation is thus best left to the user. Balanced binary trees are a case in point: they can be used as search structures, priority queues, models of parallel architecture, etc. It is thus only natural to delay their interpretation so one can appreciate their versatility. This approach is sound, for it allows us to map



rich combinatorial constructions into useful data structures. It has one negative side-effect, however. Too much concern with the definition and construction aspects of a data structure makes one forget about why it is being used in the first place. What makes the problem all the more troublesome is that algorithms often run two distinct processes: one to build data structures, and another to use them. Whether these processes be distinct in time (e.g., as in a static search tree) or interleaved (e.g., as in dynamic or self-adjusting structures), they can most often be distinguished. Surprisingly, these processes often bear little relation to each other. Building or updating a range tree (Bentley [B2]), for example, and using it to answer a query are only remotely connected tasks. This makes our earlier question all the more difficult to answer: why is a particular structuring of data preferable to another?

This question goes to the heart of our discussion. Data structures are implementations of *paradigms*. The latter, unlike the former, tend to be few and far between, so most data structures appear as variants around certain themes. It is therefore important to facilitate the task of *data structure transformation*, since this is perhaps the most common method for discovering new data structures. To do so, it is useful to think of a data structure not only in terms of its own operational semantics but also of the semantics of the algorithms that use it, the *clients*. Unfortunately, the “combinatorial” view of data structures often lacks this flexibility. Their semantics often do reflect the clients’ needs, but too indirectly to be readily modified around them. This is not to say that modeling data structures after nice, elegant combinatorial objects is not the best route to good design: we believe that it is. Occasionally, however, useful transformations will be obscured or discouraged, because they seemingly get in the way of the combinatorial identity of the objects in question.

We propose a design discipline that involves looking at the *functionality* of data structures from a client’s viewpoint. Informally, we extend the notion of a data structure by associating with each node  $v$  of a graph, not a piece of data as is usually the case, but a function  $f_v$  (or several if needed) as well as a collection of data  $S(v)$ . To evaluate  $f_v$ , it is sufficient, yet not always necessary, to have available  $S(v)$  and some values of the functions associated with nodes adjacent to  $v$ . For expressiveness, we allow  $f_v$  to be a higher-order function (i.e., a function that includes other functions among its arguments). This definition is incomplete but gives the gist of what we will call a *functional data structure* (FDS). In essence, it is a communication scheme for routing transfers of information among a collection of functions. Allowing stepwise refinement is an essential feature of an FDS. Refining an FDS is to replace nodes by other FDS’s iteratively, and in the process, get closer and closer to a data structure in the traditional sense.

An important consequence of this setting is to release the data from memory assignment. Indeed, the pairing  $(v, S(v))$  is virtual and need not correspond to any physical reality. This should not be equated with, say, the independence of linked lists with regard to memory allocation. The abstraction provided by a functional data structure is much stronger. Indeed, it will not even be required that values be physically stored in the records with which they are associated. The only requirement is that these values should be available somehow whenever needed; in particular when the functions to which they are arguments must be evaluated. Whereas a data structure emphasizes data and communication between data, an FDS emphasizes functions and communication between parameters of these functions. It can be argued, of course, that an FDS can be emulated by almost any data structure, so of what good can it be? What makes the consideration of FDS’s worthwhile is that they hint at data structures without *imposing* them. They key benefit of this setting is to allow the designer to bring upon

storage the same effect that *call by need* and *lazy evaluation* are known to have on execution time (Henderson [H]); namely to ensure that only the minimum amount of information needed to evaluate a function in some given amount of time be provided. This leads to a useful *folding* transformation, which we will describe shortly. Before going any further, let us take a simple example to illustrate our discussion. Let  $f$  be a *search* function mapping a key  $q$  to some element  $f(S, q)$  of a set  $S$ :

$$f(S, q) \equiv \text{if } t(S) \text{ then } r(S, q) \\ \text{else if } g(S, q) \text{ then } f(p(S), q) \text{ else } f(S \setminus p(S), q).$$

If  $S$  is too small, i.e.,  $t(S)$  is true, then  $r(S, q)$  provides the answer directly. Otherwise the function recurs with respect to either a subset  $p(S)$  of  $S$  or its complement  $S \setminus p(S)$ , depending on the outcome of the predicate  $g(S, q)$ . This definition suggests an infinite binary tree  $F(S)$  as an appropriate FDS for  $f$ . Each node  $v$  is associated with some set  $S(v)$  obtained by applying to  $S$  compositions of  $\lambda(s)p(s)$  and  $\lambda(s)\{s \setminus p(s)\}$  intermixed in the obvious way.<sup>2</sup> For example, the root is associated with  $S$ , its left child to  $p(S)$ , and its right child to  $S \setminus p(S)$ . The tree can be made finite by ignoring all nodes associated with empty subsets. Next, one associates an FDS for  $\lambda(q)g(S(v), q)$  with each node  $v$  of the tree. As is often the case in multidimensional searching,  $g$  may have a definition similar to  $f$ , e.g.,

$$g(V, q) \equiv \text{if } t'(V) \text{ then } r'(V, q) \\ \text{else if } h(V, q) \text{ then } g(p'(V), q) \text{ else } g(V \setminus p'(V), q).$$

This leads to an FDS,  $G(V)$ , defined for  $V \subseteq S$  with respect to  $g$  similarly to  $F(S)$ . Putting things together, we refine  $F(S)$  by associating  $G(S(v))$  with each node  $v$  of  $F(S)$ . If  $p$  and  $p'$  are very different functions then it is not clear what benefits might be gained from this formalism. But suppose that it is possible to choose  $h$  so that  $p$  and  $p'$  are the same. Then  $f$  and  $g$  have identical "communication schemes." This allows us to perform a so-called *folding* transformation: the idea is to use the same FDS's for both  $f$  and  $g$  by taking each tree  $G(S(v))$  and *folding* it over the subtree of  $F(S)$  rooted at  $v$ . In effect, this is replacing  $G(S(v))$  by an FDS for  $\lambda(q)h(S(v), q)$ . Once  $F(S)$  has been refined in this manner, one will see a data structure with several layers of functional data structures superimposed on it. As a result, the storage used might be greatly inferior to what it would have been without folding the FDS. Indeed, each node  $v$  will thus only need an FDS for  $\lambda(q)h(S(v), q)$  as opposed to an FDS for each  $\lambda(q)h(V, q)$ , where  $V$  is obtained by applying to  $S(v)$  mixed compositions of  $\lambda(s)p'(s)$  and  $\lambda(s)\{s \setminus p'(s)\}$ . To put things in perspective, we must recall that folding might entail redefining  $h$ : all benefits will be lost if the new implementation of  $g$  becomes much more complicated as a result. Although functional data structures have no complexity per se, they *hint* at the ultimate complexity of the data structure. Suppose that  $g(S, q)$  can be evaluated very fast but at great cost in storage. Then folding offers the possibility of trade-off between space and time.

*Searching in the past of mergesort.* Here is a problem which, although a bit esoteric at first, is nevertheless fundamental. Run mergesort on a set of  $n$  distinct keys  $S = \{y_1, \dots, y_n\}$  and record the list produced after each merge. The resulting data structure can be modeled as a balanced binary tree  $T$ : the leaves are assigned the keys  $y_1, \dots, y_n$  from left to right, and each node  $v$  is associated with the sorted list  $R(v)$  consisting

<sup>2</sup> We will use  $\lambda$ -expressions in the following either for sheer convenience as above or in order to distinguish between multivariate functions, such as  $\lambda(x, y)f(x, y)$ , and restrictions to univariate functions with constants, as in  $\lambda(x)f(x, y)$ .

of the keys at the leaves descending from  $v$ . For any  $q$ , define  $s(v, q) = \min \{x \in R(v) \cup \{+\infty\} \mid q \leq x\}$ . What is an efficient data structure for computing  $s(v, q)$ , assuming that  $v$  is given by its inorder rank in the tree and that we are in a pointer machine model? Consider the data structure obtained by storing at  $v$  its inorder rank as well as a pointer to the root of a complete binary tree on  $R(v)$ . This is essentially Bentley's *range tree* [B2]: it allows us to compute  $s(v, q)$  in  $O(\log n)$  time and  $O(n \log n)$  space (search for  $v$  in  $T$ , and then search for  $q$  in  $R(v)$ ). Whether the storage can be reduced to linear and the query time kept polylogarithmic has been a long-standing open problem (Lee and Preparata [LP2]).

We settle this question by interpreting the range tree in functional terms. We have  $s(v, q) \equiv f(v, \text{root}, q)$ , with

$$f(v, z, q) \equiv \text{if } v = z \text{ then } g(R(v), q) \\ \text{else if } z < v \text{ then } f(v, z.r, q) \text{ else } f(v, z.l, q);$$

$z$  is an ancestor of  $v$  or  $v$  itself, and  $z.l$  (resp.,  $z.r$ ) denotes the left (resp., right) child of  $z$ . A natural FDS for  $f$  involves taking  $T$  and associating with each node  $v$  the function  $\lambda(q)g(R(v), q)$  and the set  $R(v)$ . Since  $g$  is a *search* function it can be decomposed as follows (ignoring termination for simplicity):

$$g(V, q) \equiv \text{if } h(V, q) \text{ then } g(p(V), q) \text{ else } g(V \setminus p(V), q).$$

We can now refine the FDS for  $f$  by replacing each node  $v$  by an FDS for  $\lambda(q)g(R(v), q)$ . It is natural to think of  $p$  as a halving function taking a sorted set of numbers  $\{p_1, \dots, p_t\}$  as input and returning  $\{p_1, \dots, p_{\lfloor (t+1)/2 \rfloor}\}$  as output. This makes the implementation of  $h$  quite simple (one comparison), but unfortunately gives two widely different partitioning functions for  $f$  and  $g$ : one is based on the indices of the  $y_i$ 's, the other on their values. So, one can try to make the two partitionings identical to allow *folding*, hoping that  $h$  will not become too complex as a result. To do that, we write  $p(R(v)) \equiv R(v.l)$  (the function  $p$  becomes partial as a result, but it is all right). Instead of an FDS for  $\lambda(q)g(R(v), q)$  at  $v$ , an FDS for  $\lambda(q)h(R(v), q)$  now suffices.

Although by doing so,  $h$  increases in complexity, we must hope that  $h$  remains simpler to compute than  $g$ , otherwise the transformation would be useless. Since  $h$  is a predicate, it can be succinctly encoded as a bit vector  $B(v)$ . We have  $B(v) = [b_0, \dots, b_{|R(v)|-1}]$ , where  $b_i = 0$  (resp., 1) if the element at position  $i$  in  $R(v)$  comes from the left (resp., right) child of  $v$ . Interestingly,  $B(v)$  represents the transcript of pointer motions during the merge at node  $v$ . Note that if  $v$  is a leaf then  $B(v)$  is not defined since that node, being associated with a single key, does not witness any merge. To compute  $h(R(v), q)$  we proceed by binary search in  $B(v)$ . Unfortunately, we cannot do so with the bits of  $B(v)$  alone, so we must find a way to produce the key  $y^v(i) \in R(v)$  corresponding to a given bit position  $i$  in  $B(v)$ . This crucial operation is called *identifying* the bit  $b_i$ . Note that each bit of  $B(v)$  has a distinct *identifier* in  $R(v)$ .

We can assume that  $v$  is not a leaf. If  $b_i = 0$  (resp.,  $b_i = 1$ ) then we have  $y^v(i) = y^{v.l}(j)$  (resp.,  $y^v(i) = y^{v.r}(k)$ ), for some  $j$  (resp.,  $k$ ). In other words, the key corresponding to  $b_i$  can be *traced* either at  $v.l$  or  $v.r$ , depending on the value of  $b_i$ . We easily see that  $j$  (resp.,  $k$ ) is the number of 0's (resp., 1's) in  $[b_0, \dots, b_{i-1}]$ . Suppose that

$$R(v.l) = [2, 3, 5, 7, 11, 13, 17, 19, 23, 27, 29, 31, 37, 41]$$

and

$$R(v.r) = [1, 6, 12, 14, 15, 20, 21, 24, 25, 26, 32, 33, 44, 46].$$

We have

$$R(v) = [1, 2, 3, 5, 6, 7, 11, 12, 13, 14, 15, 17, 19, 20, 21, 23, \\ 24, 25, 26, 27, 29, 31, 32, 33, 37, 41, 44, 46]$$

and

$$B(v) = [1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1]$$

so, for example, 23 is the identifier of bit #15 in  $B(v)$ . Since this bit is 0, the key 23 can be found in  $R(v.l)$ . Its position is #8, which is also the number of 0's among the first 15 bits of  $B(v)$ ,  $[1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1]$ .

Iterating on this process until we reach a leaf of  $T$  allows us to identify any bit of  $B(v)$ . For this we assume that each leaf keeps its corresponding identifier alongside. Once identification is possible, we can compute  $h(R(v), q)$  by binary search in  $B(v)$ . This will take  $O(t(n) \log^2 n)$  time, where  $t(n)$  is the time to find  $b_i$  and the sum  $c_i = \sum_{0 \leq k < i} b_k$ , given  $i$ . Computing  $g(R(v), q)$  can then be done by binary search in  $T$  (following the definition of  $g$  and the fact that its FDS has been folded). We can take a shortcut, however, by observing that the value of  $h(R(v), q)$  is given by a bit of  $B(v)$  whose identifier,  $g(R(v), q)$ , is explicitly determined in the course of evaluating  $h$ . The computation of  $g$  will then also take  $O(t(n) \log^2 n)$  operations.

Let us examine  $t(n)$ . To compute  $b_i$  and  $c_i$ , given  $i$  and  $v$ , we break up  $B(v)$  into computer words  $\beta_0, \dots, \beta_{m-1}$  (filling each word up to  $\lfloor \log n \rfloor$  bits) and we make the sequence  $\beta_0, \dots, \beta_{m-1}$  the leaves of a complete binary tree. At each internal node of the tree we indicate the number of 0's and the number of 1's among the bits of  $B(v)$  stored at the leaves descending from its left child. With this data structure, it is easy to find in  $O(\log n)$  time the word  $\beta_j$  that contains the bit  $b_i$  as well as the number of 1's or 0's among the bits of  $\beta_0, \dots, \beta_{j-1}$ . At that point, it remains to examine each bit of  $\beta_j$  to complete the task. Bit enumeration on an EPM can be done very simply by searching for the number  $\beta_j$  in a perfectly balanced binary search tree of  $2^{\lfloor \log n \rfloor}$  leaves and associating left turns with 0 and right turns with 1.

To conclude,  $O(\log^3 n)$  time suffices to compute  $s(v, q)$ . To see that only linear storage is required, we remark that besides the input the number of bits used is proportional to the number of steps taken by running mergesort on  $n$  keys. This quantity is also proportional to the number of bits needed to store the input. Therefore, without any reference to the size of a computer word, we can conclude to the linearity of the data structure (in the sense that a data structure for a Turing machine is linear). This result holds on a pointer machine whose only operations are comparisons and additions. Note that whether the keys in  $S$  are integers or real numbers is irrelevant to our analysis. Also, one should observe that the use of bits need not be explicit in the algorithm. It will actually not appear at all in the implementation which we give in the next section.

It is easy to modify the data structure to produce an  $O(n)$  space,  $O(\log^4 n)$  time solution for range counting in two dimensions. But one can do much better. We must stop at this point to notice that the transformations which we have used so far are quite crude. They treat the recursive steps of the query-answering process as a sequence of independent computations. To enable the algorithm to use at any time information gathered at previous steps we must modify the refinement of the underlying functional data structure. The idea is to exploit the fact that the operations performed in the four nested loops of an  $O(\log^4 n)$  time algorithm are unlikely to be independent. We feel that our point concerning the fruitfulness of functional data structures has been made, however, so we will not use the previous formalism any further. Rather, we will rely

on the intuition that it allowed us to develop and, in particular, we will keep the idea of folding in the back of our mind.

### 3. Range counting.

**3.1. Range counting on a random access machine.** We will describe a linear-size solution to range counting in two dimensions on a RAM with logarithmic query time. Although this result is not the strongest (the same holds on an APM as shown in the first table), it is very efficient in practice and illustrates most of the new ideas in simple terms. We have implemented the algorithm in Pascal (aiming for clarity rather than efficiency). The code is quite short, so we will give it in its entirety. For convenience, we assume that  $n \geq 2$  and that a word can store numbers not just in  $[0, n]$  as we said earlier but in the range  $[0, 4n \log^2 n]$ . Then, rather amazingly, the data structure consists *solely* of four arrays  $X, Y, B, C$ : array  $[0..n-1]$ . The arrays  $X$  and  $Y$  contain, respectively, the  $x$  and  $y$  coordinates of the points in increasing order;  $B$  and  $C$  are a bit more mysterious. Here is an example. If the input is

$$\{(34, 3), (12, 1), (28, 23), (63, 15), (2, 35), (5, 17), (52, 43), (22, 13)\},$$

then we have

$$B = [64, 97, 81, 66, 33, 82, 34, 83]$$

and

$$C = [17, 35, 69, 6, 7, 9, 10, 12].$$

How do these numbers come about? We begin with an informal description of the algorithm.

To shorten the code given in this paper we make a number of simplifying assumptions, every one of which can be easily satisfied with a bit of care: (1)  $n$  is a power of two (pad with points at infinity, if needed); (2) all coordinates are distinct (go into rank space using presorting, if needed); (3) the query does not share coordinates with the point set and falls inside the smallest 2-range containing the input points.

It is convenient to leave aside the function  $s(v, q)$  and use the related *ranking* function  $r(v, q) = |\{y \in R(v) | y < q\}|$ . To compute this function, we keep a similar data structure: each internal node  $v$  of  $T$  stores the bit vector  $B(v)$ , broken up into words  $\beta_0, \dots, \beta_{m-1}$ , as usual. We also have an array  $C(v)$ , whose cell #  $i$  contains the number of 1's in  $\beta_0, \dots, \beta_i$ . The problem is now to find the rank  $r$  of  $q$  in  $R(w)$ , assuming that we know its rank  $r'$  in  $R(v)$  (without loss of generality,  $w$  is the right child of  $v$ ). This rank is the number of 1's in  $B(v)$  at positions  $[0..r'-1]$ . To find it, we must locate the corresponding bit in  $B(v)$ : first we find the word  $\beta_j$  that contains it, which is done in  $O(1)$  time on a RAM, then we compute its relative position  $k$  in  $\beta_j$ . We also set  $r$  to the approximate count given by cell #  $(j-1)$  in  $C(v)$ . There now remains to shift  $\beta_j$  by  $k$  bits (done by division) and find the number of 1's in the remaining word by table look-up. Adding this number to  $r$  produces the desired result. So, after an initial binary search in the array  $Y = R(\text{root})$ , we find the rank of  $q$  in  $R(\text{root})$ , and percolate down to  $v$ . To carry out these computations, we need powers of two and tallies of ones for each integer in  $[0, n-1]$ . We will store all the  $B(v)$ 's and tallies in one array  $B[0..n-1]$ , and all the partial counts and powers of two in another array  $C[0..n-1]$ .

*The data structure.* The arrays  $X[0..n-1]$  and  $Y[0..n-1]$  give the  $x$  and  $y$  coordinates of the  $n$  points in increasing order. Let  $\lambda = \log n$ ,  $\mu = 2(1 + \lceil \log \lambda \rceil)$ , and  $M = 2^\mu$ . In the code below,  $n$ ,  $M$ , and  $\lambda$  (denoted lambda) will be global variables. Note that the word size  $w$  has been assumed to be at least as large as

$\lfloor \log n + 2 \log \log n + 2 \rfloor + 1 \geq \lambda + \mu$ . Let  $P, \Lambda_i, M_i$  be the sequence of bits in  $B[i]$ , from left to right.

- (1)  $P$  is a sequence of  $w - \lambda - \mu$  zeros (padding).
- (2)  $\Lambda_i$  is a sequence of  $\lambda$  bits, called the  $\lambda$ -part of  $B[i]$ . Since  $n$  is a power of two, we can nicely characterize each node of  $T$  by a pair  $(a, b)$ , where  $a$  is its height (0 for a leaf,  $\lambda$  for the root) and  $b$  is its rank at height  $a$ , counted from left to right starting at 0. Sort all the nodes in lexicographic order and concatenate into one string  $b[0..l]$ , in this order, all the bit vectors  $B(v)$  (recall that these vectors are defined only from height 1 up). The string of bits  $\Lambda_0 \cdot \Lambda_1 \cdot \dots \cdot \Lambda_{n-1}$  is set to  $b[0..l]$ . Note that the count of bits is the same in both cases ( $l = \lambda n - 1$ ) but that the boundaries of  $B(v)$ 's might not coincide at all with those of  $\lambda$ -parts. In the following, we will use the notation  $b[i]$  to indicate the  $(i+1)$ st bit of  $b[0..l]$ .
- (3)  $M_i$ , called the  $\mu$ -part of  $B[i]$ , contains the number of 1's in the binary representation of  $i$  (note that  $\mu$  bits are ample for that purpose).
- (4) The  $\lambda$ -part of  $C[i]$  is equal to  $2^i$  for  $i \in [0, \lambda - 1]$ . For each  $i \in [0, \lambda - 1]$ , let us (only in thought) turn to 0 each bit of the  $\lambda$ -part of  $C[i]$ . Then for each  $i \in [0, n - 1]$ ,  $C[i]$  is the total number of bits equal to 1 among the  $\lambda$ -parts of  $B[0..i]$ . Note that the powers of two stored in the  $\lambda$ -parts never conflict with the tallies of ones because the number of 1's among the first  $\lambda$   $\lambda$ -parts of  $B[0..\lambda - 1]$  can be encoded over  $\lfloor \log(\lambda^2) \rfloor + 1 \leq \mu$  bits.

We will illustrate these definitions with the first cells of  $B$  and  $C$ . The mergesort starts with the sequence of points  $x$ -sorted  $(2, 35), (5, 17), (12, 1), (22, 13), \dots$ , and merges  $\{35\}, \{17\}$  (which gives  $b[0] = 1$  and  $b[1] = 0$ ) and then  $\{1\}, \{13\}$  (which gives  $b[2] = 0$  and  $b[3] = 1$ ). As a result, the  $\lambda$ -part of  $B[0]$  is 100. Since  $\mu = 4$ , the  $\mu$ -part of  $B[0]$  is expressed over 4 bits. It is equal to the number of 1's in 0, that is, 0000. This gives  $B[0] = 1000000$  in binary, i.e., 64 in decimal. The  $\lambda$ -part of  $C[0]$  is equal to 1. This number, shifted by  $\mu$  bits, becomes 16, to which the number of 1's in  $B[0]$  ( $=1$ ) must now be added. This gives the final value  $C[0] = 17$ .

*The preprocessing.* We use a nonrecursive mergesort to fill the  $\lambda$ -parts of  $B$ . To do so, we declare a temporary array  $T[0..n]$ . Variables *step* and *l* form the lexicographic pair in the sort. A counter *index* maintains the current position in  $B$ . A bit is inserted to the right after multiplying the current word by two to make room for it. When a  $\lambda$ -part is full (“if lambda = fill then begin”) it is shifted by  $\mu$  bits to its final position, and the  $\mu$ -part is computed by enumerating the bits of *index*. The powers of two,  $\{2^0, 2^1, \dots, 2^{\lambda-1}\}$  stored in the  $\lambda$ -parts of  $C$  are computed in an extra pass (“for index := 0 to lambda - 1 do”). Initially,  $X$  contains the  $x$ -coordinates in increasing order and  $Y[i]$  is the  $y$ -coordinate corresponding to  $X[i]$ ;  $B$  and  $C$  are set to 0, and *maxint* stands for any integer larger than the coordinates. At the end  $Y$  contains the  $y$ -coordinates in increasing order.

**procedure** Preprocessing;

**var** fill, index,  $i, j, k, l, r, u, cur, tmp$ , step: integer;

**begin**

fill := 0; index := 0; step := 2;

**while** step <=  $n$  **do begin**

$l := 0$ ;

**while**  $l < n$  **do begin**

$r := l + \text{step} - 1$ ;  $u := (l + r) \text{ div } 2$ ;

**for**  $k := l$  **to**  $r$  **do**  $T[k] := Y[k]$ ;

```

T[r + 1] := maxint; i := l; j := u + 1; k := l - 1;
while (i <= u) or (j <= r) do begin
  B[index] := 2 * B[index]; k := k + 1;
  if (T[j] < T[i]) or (i > u) then
    begin
      Y[k] := T[j]; j := j + 1;
      B[index] := B[index] + 1;
      C[index] := C[index] + 1
    end
  else begin Y[k] := T[i]; i := i + 1 end;
  fill := fill + 1;
  if lambda = fill then begin
    cur := index; B[index] := B[index] * M;
    while cur > 0 do begin
      tmp := cur; cur := cur div 2;
      B[index] := B[index] + tmp - 2 * cur
    end;
    index := index + 1; fill := 0;
    if index < n then C[index] := C[index - 1]
  end
end;
l := l + step
end;
step := 2 * step
end;
fill := M;
for index := 0 to lamda - 1 do
  begin C[index] := C[index] + fill;
  fill := 2 * fill end
end;

```

*The query-answering algorithm.* Let  $[x_1, x_2] \times [y_1, y_2]$  be the query range. The function RangeCount (query) decomposes  $[x_1, x_2]$  into  $O(\log n)$  canonical pieces, each represented by a node of  $T$ . The count of points inside the query is obtained by summing up the values  $r(v, y_2) - r(v, y_1)$  for each  $v$  in the decomposition. Since we add up differences, we may redefine the function  $r(v, q)$  as follows. Let  $i$  be the number of elements in  $R(v)$  strictly less than  $q$ . By construction, the  $(i + 1)$ st bit in the vector  $B(v)$  (if defined) appears somewhere in  $b[0..l]$ , say as  $b[j]$ ; then set  $\rho(v, q) = j$ . Consistently, all the bits of the  $B(v)$ 's will be addressed from now on by their position in  $b[0..l]$ .

Next, we describe the functions needed in a bottom-up fashion. The function One (pos) returns the number of 1's in  $b[0, \text{pos} - 1]$ . First, we find the index  $i$  such that  $B[i]$  contains the bit  $b[\text{pos}]$ . To compute the number of 1's in  $B[i]$  left of the bit  $b[\text{pos}]$ , we shift and truncate  $B[i]$  accordingly to obtain the integer  $j$  (we use a little trick to avoid shifts by  $\lambda$ ). Then we use the  $\mu$ -parts of  $B$  to find  $z$ , the number of 1's in  $j$ . This number is added to  $C[i - 1]$  in order to get the final result. We need a corrective term if  $i - 1$  is less than or equal to  $\lambda - 1$  (because of the powers of two stored in the first cells of  $C$ ).

```

function One (pos: integer): integer;
  var i, j, z: integer;

```

```

begin
   $i := \text{pos} \text{ div } \text{lambda};$ 
   $j := B[i] \text{ div } (2 * M * (C[\text{lambda} * (1 + i) - \text{pos} - 1] \text{ div } M));$ 
   $z := B[j] - M * (B[j] \text{ div } M);$ 
  if  $0 < i$  then  $z := z + C[i - 1];$ 
  if  $(0 < i)$  and  $(i \leq \text{lambda})$  then  $z := z - M * (C[i - 1] \text{ div } M);$ 
   $\text{One} := z$ 
end;

```

The function `Newpos` (`dir`, `block`, `pos`, `width`) allows us to trace the position of a bit  $b[\text{pos}]$  from a node  $v$  to a child  $w$ . In other words, it computes the values  $\rho(w, q)$ , given  $\rho(v, q)$ . (The reader familiar with Willard [W2] might guess rightly that we are trying here to simulate the effect of *downpointers*). We allow the value of  $\rho(w, q)$  to be negative. Why is that so? The first cells of  $B$  store the history of the first pass of mergesort. Therefore, these do not correspond to leaves of  $T$  but to parents of leaves. From the lexicographic ordering in  $B$ , we can see that the leaves of  $T$ , if they were to be stored in  $B$ , would appear in positions  $-n, -n + 1, \dots, -1$ . We do not need to store these leaves because no identification is required for range counting. The arguments of `Newpos` (`dir`, `block`, `pos`, `width`) denote, respectively,

- (1) *dir*: which child of  $v$  is being considered (0 if left, 1 if right); we use an integer instead of a Boolean only for brevity.
- (2) *block*: the position in  $b[0..l]$  of the first bit of  $B(v)$ ,
- (3) *pos*: the position in  $b[0..l]$  of the bit of interest in  $B(v)$ ,
- (4) *width*: the number of leaves in the subtree rooted at  $v$ .

Because of the lexicographic ordering of the nodes of  $T$ , the starting position of  $B(w)$  in  $b[0..l]$  is “ $\text{block} - n$ ” (left child) or “ $\text{block} - n + \text{width}/2$ ” (right child).

```

function Newpos (dir, block, pos, width: integer): integer;
begin
  if dir = 0 then Newpos := pos - n + One(block) - One(pos)
  else Newpos := block - n + One(pos) - One(block) + (width div 2)
end;

```

The function `Path` ( $A, q$ ) searches for  $q$  in the array  $A$  of type “Tableau = array  $[0..n - 1]$  of integer”, and returns  $\min(\{i | q \cong A[i]\} \cup \{n - 1\})$ . It will be used for two purposes: first to locate  $y_1$  and  $y_2$  in  $Y$ , and then  $x_1$  and  $x_2$  in  $X$ .

```

function Path (A: Tableau; q: integer): integer;
  var l, k, r: integer;
begin
  l := 0; r := n - 1;
  while l < r do begin
    k := (l + r) div 2;
    if q <= A[k] then r := k end
    else l := k + 1 end;
  Path := l
end;

```

Let  $l_1$  and  $l_2$  be the two leaves of  $T$  returned by `Path` when applied, respectively, to  $x_1$  and  $x_2$  in  $X$ . Let  $l_0$  be the leaf preceding  $l_1$  in inorder (such a leaf is guaranteed to exist because of our assumption that the query falls entirely within the smallest box containing the input points). The nodes of the decomposition of  $[x_1, x_2]$  are precisely those adjacent to the path from  $l_0$  to  $l_2$ , but not on it, whose inorder ranks fall in



between the ranks of these two leaves. To find them, it suffices to determine the least common ancestor  $w$  of  $l_0$  and  $l_2$ , and then collect the nodes  $N_1$  (resp.,  $N_2$ ) hanging off the right (resp., left) of the path from  $l_0$  (resp.,  $l_2$ ) to  $w$ . The number of points inside the query range is

$$\sum_{v \in N_1} \rho(v, y_2) + \sum_{v \in N_2} \rho(v, y_2) - \sum_{v \in N_1} \rho(v, y_1) - \sum_{v \in N_2} \rho(v, y_1).$$

All four computations are carried out by calling the function `Cum (cut, init, path, dir)`. Let us take the case of  $\sum_{v \in N_1} \rho(v, y_2)$ , for example. Then “cut” is the number of leaves descending from  $w$ , “init” is equal to  $\rho(\text{root}, y_2)$ , “path” to  $l_0$ , and “dir” to 0 (it is 1 for  $N_2$ ). The bits of “path” indicate the sequence of turns from the root to  $l_0$ . Along with “cut,” this allows us to determine  $w$  and, hence, the nodes of  $N_1$ .

```

function Cum (cut, init, path, dir: integer): integer;
  var pos, z, bit, block, cur: integer;
begin
  pos := init; z := 0; block := (lambda - 1) * n; cut := n;
  while cur >= 2 do begin
    bit := (2 * path) div cur - 2 * (path div cur);
    if (cur < cut) and (bit = dir) then
      z := z + Newpos(1 - dir, block, pos, cur);
    pos := Newpos(bit, block, pos, cur);
    cur := cur div 2; block := block - n + bit * cur
  end;
  Cum := z
end;

```

For completeness, we also give the code of the mainline, which is quite straightforward.

```

function RangeCount (x1, x2, y1, y2: integer): integer;
  var left, right, low, high, cut, z: integer;
begin
  low := Path(Y, y1) + (lambda - 1) * n;
  high := Path(Y, y2) + (lambda - 1) * n;
  left := Path(X, x1) - 1; right := Path(X, x2); cut := n;
  while (2 * left) div cut = (2 * right) div cut do cut := cut div 2;
  z := Cum (cut, high, left, 0) + Cum (cut, high, right, 1);
  RangeCount := z - Cum (cut, low, left, 0) - Cum (cut, low, right, 1)
end;

```

The description of the algorithm is now complete. Again, observe that the coordinates can be assumed to be arbitrary reals, if needed. Note that the ratio between the storage needed and the input size is only 2. In the context of this algorithm, the  $\lambda$ -parts of  $B$  cannot be compressed. This is not true of  $C$ , however: this array can indeed be shrunken arbitrarily by skipping chunks at regular intervals. The blanks can be *made up* by further work in  $B$  at query-answering time. This simple remark shows that the ratio can be brought arbitrarily close to 1.5, while increasing the query time only by a constant factor.

One might wonder how much dependent on the RAM model the algorithm really is. We will see that, although it loses some of its simplicity in the process, the algorithm can still be ported to a pointer machine.

**3.2. Range counting on a pointer machine.** To avoid the need for address calculations, we implement the tree  $T$  with pointers. Each node  $v$  is associated with a list  $W(v)$  (assumed to be doubly-linked for convenience). Aside from pointers for the list

itself, each record of  $W(v)$  has eight fields,  $B, C, C', D, E, F, G, H$ , made of one word each. The key specification is that, concatenated together, the  $B$ -fields of  $W(v)$  form the bit vector  $B(v)$ . This statement must be clarified and refined a little. First of all, only the  $\lambda = \lfloor \log n \rfloor$  least significant bits of a  $B$ -field will contain bits of  $B(v)$ . These are called the *meaningful* bits of the  $B$ -field. We will ignore the others. With this scheme each  $B$ -field, except possibly the last one, has precisely  $\lambda$  meaningful bits. As regards the last one, we may assume that its meaningful bits are right-justified, and that an extra record is added to indicate how many bits of the  $B$ -field are indeed meaningful. Let the subscript  $k$  refer to the  $k$ th record of  $W(v)$ . Then,

- (1)  $C_k$  stores the number of 1's in  $B_0, \dots, B_k$  and  $C'_k$  the number of 0's.
- (2)  $D_k$  stores the identifier of the least significant bit of  $B_k$ . Recall that this is the value of  $R(v)$  in one-to-one correspondence with that bit.
- (3) Consider the smallest value of  $R(v)$  greater than or equal to integer  $q$  that appears as a  $D$ -field of  $W(v)$ ; we define  $n(v, q)$  as the address of the record containing this  $D$ -field if it exists, or as a null pointer if it does not. Let  $w$  (resp.,  $w'$ ) be the left (resp., right) child of  $v$ ; then  $E_k$  (resp.,  $F_k$ ) contains the address  $n(w, D_k)$  (resp.,  $n(w', D_k)$ ), or more precisely a pointer to the relevant record. These fields are left blank if  $v$  is a leaf.
- (4)  $G_k$  and  $H_k$  are not immediately needed and, for the sake of clarity, will be introduced later on.

The root of the tree receives a special treatment: we augment  $W(\text{root})$  with a balanced search tree for  $R(\text{root})$ . This will allow us to compute the rank of a  $y$ -coordinate and thus get the query-answering process started. All the fields should be well motivated from the previous discussion, except perhaps for  $E$  and  $F$ . These fields play a role somewhat similar to Willard's *downpointers* [W2]. They provide a mechanism to avoid repeated binary searches when examining sequences of lists  $W(v)$ . Unlike downpointers, however, they do not span the whole range of values in  $R(v)$ , but an evenly distributed sample. Before proceeding with a description of the query algorithm, we should make the obvious observation that, as before, the data structure is linear in size. Furthermore, it can be constructed in  $O(n)$  space and  $O(n \log n)$  time, using mergesort as a basis for the algorithm.

To answer a query, we simulate the process described in the previous section. We start with a binary search in  $R(\text{root})$  with respect to  $y_1$  and  $y_2$  and follow  $E$ - and/or  $F$ -fields appropriately. Let  $v$  be an internal node of  $T$  and  $w$  be one of its children (say, the left one, without loss of generality). We need to compute  $r(w, q)$ , given  $r(v, q)$ . We will show how to find the pair  $(n(w, q), r(w, q))$ , given  $(n(v, q), r(v, q))$ . This is called computing a *transition*. Let  $B_k$  be the  $B$ -field of the record at address  $n(v, q)$ . We assume for now the availability of a primitive operation tally  $(k, t)$  which takes the number  $B_k = 0 \dots 0.x_0 \dots x_{t-1}$  in binary and an index  $t$  ( $0 < t < \lambda$ ), and returns  $x_0 + x_1 + \dots + x_{t-1}$ . By definition, the first  $r(v, q)$  bits of  $B(v)$  are contained in the  $B$ -fields of  $W(v)$  preceding  $n(v, q)$ , and perhaps in a prefix of  $B_k$ . Using  $C_k$  and  $C'_k$  we can find the length of this prefix using  $+$ ,  $-$ . Next, we call the tally function to compute the number of 1's in the prefix, which gives us in turn the number of 0's among the first  $r(v, q)$  bits of  $B(v)$ , that is,  $r(v, q)$ . To compute  $n(w, q)$ , we just follow the pointer in  $E_k$ . Since each  $B$ -field has at most  $\lambda$  bits,  $n(w, q)$  will be either the address of the record where we land, or it will be the address of its predecessor in the list  $W(w)$ . To find out we can use  $q$  and  $D$ -fields, or if we prefer, we can use  $r(w, q)$  and  $C, C'$ -fields. We mention the latter method for the following reason: it is possible to compute a transition from  $(n(v, q), r(v, q))$  to  $(n(w, q), r(w, q))$  *without* prior knowledge of  $q$ . This will be used in the next section.

To implement  $\text{tally}(k, t)$ , we use the fields  $G_k$  and  $H_k$  as well as the primitive *shift* available on an APM. This, we should recall, takes  $i \leq \lfloor \log n \rfloor$  as input and returns  $10^i$  (1 followed by  $i$  0's). It is needed to simulate a RAM at the word level in constant time. To do so, let  $\alpha = \lfloor \log \lambda \rfloor + 1$ ,  $\beta = \lfloor \log \alpha \rfloor + 1$  and  $\gamma = \lfloor \log \beta \rfloor + 1$  (these quantities are computed by binary search in preprocessing). For any integer  $u$  ( $0 \leq u < n$ ), we define the function  $h(u, i, j) = \lfloor u/2^{\lambda-j} \rfloor - 2^{j-i} \lfloor u/2^{\lambda-i} \rfloor$  if  $0 \leq i < j \leq \lambda$ , and 0 otherwise: if  $u = u_0 \cdots u_{\lambda-1}$  in binary, then  $h(u, i, j) = u_i u_{i+1} \cdots u_{j-2} u_{j-1}$ . Next, we define a word  $\Theta = t_0 \cdots t_{\lambda-1}$  as follows: for  $i = 0, \dots, 2^\beta$ , let  $t_{i\gamma} \cdots t_{(i+1)\gamma-1}$  be the number of 1's in the binary representation of  $i$ . To set up the fields  $G_k = g_0 \cdots g_{\lambda-1}$  and  $H_k = h_0 \cdots h_{\lambda-1}$ , let  $p_0 \cdots p_{\lambda-1}$  be the  $\lambda$  meaningful bits of  $B_k$  (Fig. 1). Recall that the last  $B$ -field of  $W(v)$  might be short of  $\lambda$  meaningful bits; this special case can be treated easily, however, and will be omitted here.

- (1) For each  $i = 0, \dots, \lfloor \lambda/\alpha \rfloor - 1$ , we have  $g_{i\alpha} \cdots g_{(i+1)\alpha-1} = p_0 + \cdots + p_{(i+1)\alpha-1}$ .
- (2) For each  $i = 0, \dots, \lfloor \lambda/\alpha \rfloor - 1$  and each  $j = 0, \dots, \lfloor \alpha/\beta \rfloor - 1$ , we have  $h_{i\alpha+j\beta} \cdots h_{i\alpha+(j+1)\beta-1} = p_{i\alpha} + \cdots + p_{i\alpha+(j+1)\beta-1}$ . Since an integer  $x$  can be represented on  $\lceil \log(x+1) \rceil$  bits, one will easily check that the space provided for  $\Theta$  and the various blocks of  $G_k$  and  $H_k$  is sufficient, for  $n$  large enough.

We can now implement  $\text{tally}(k, t)$ . To do so, we evaluate in sequence

$$i = \alpha \left\lfloor \frac{t}{\alpha} \right\rfloor,$$

$$j = i - \alpha,$$

$$l = i + \beta \left\lfloor \frac{t-i}{\beta} \right\rfloor,$$

$$m = l - \beta,$$

$$p = h(B_k, l, t),$$

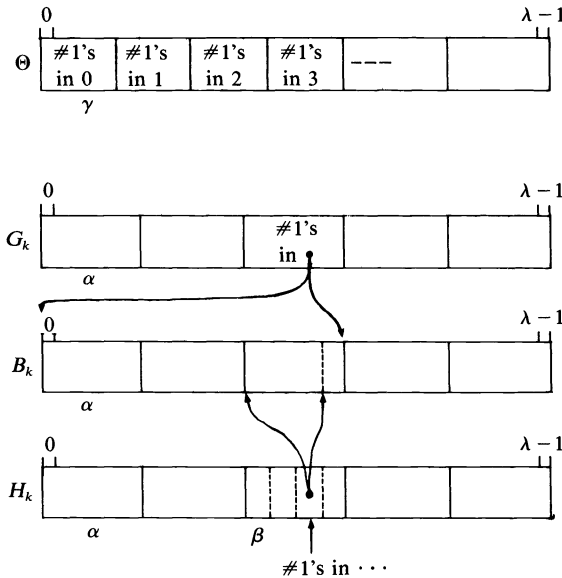


FIG. 1

then

$$\text{tally}(k, t) = h(G_k, j, i) + h(H_k, m, l) + h(\Theta, p\gamma, (p+1)\gamma).$$

The algorithm computes successive approximations of the answer. The vector  $p_0 \cdots p_{l-1}$  is broken up into three blocks  $p_0 \cdots p_{i-1}$ ,  $p_i \cdots p_{l-1}$ , and  $p_l \cdots p_{l-1}$ . The number of 1's in the first and second blocks are given directly by reading off one of the elementary pieces of  $G_k$  and  $H_k$ . For the last block, one turns to  $\Theta$  for the answer.

To summarize, range counting can be done in  $O(n)$  space and  $O(\log n)$  query time on an APM. What happens if we only have an EPM? In that case, we may ignore the fields  $H$  and  $G$  altogether. Given any prefix of a  $B$ -field, we can count its number of 0's or 1's by examining each of its bits. To do so, we set up a binary search tree on the keys  $\{0, \dots, 2^\lambda - 1\}$  so as to identify the turns of a search with the binary representation of the search key. At most a constant number of subtractions might be needed for each node of  $T$  visited. These can be simulated in  $O(\log n)$  time on an EPM. The query time will be  $O(\log^2 n)$ . This completes our discussion of the static case. The data structures described above for the various models form the skeleton of the remaining algorithms. We refer to them as  $M$ -structures ( $M$  for mergesort). They can all be built in  $O(n \log n)$  time.

*The dynamic case.* We begin with a number of remarks which apply to all our subsequent treatments of dynamic problems. As we said earlier on, we will always assume that in the dynamic case the underlying model of computation is an EPM. One difficulty in dynamizing the previous data structure is trying to assign a value to  $\lambda$ . Since  $n$  changes constantly in a dynamic environment, so will  $\lambda$  if we set its value to  $\lfloor \log n \rfloor$ , as in the static case. This may imply that the entire data structure will need to be reconfigured at all times. To overcome this problem we relax the assignment of  $\lambda$  and only require that its value should lie between  $\frac{1}{3} \log n$  and  $\log n$  (for  $n$  larger than some appropriate constant). This is called the *compaction invariant*. The idea is that as soon as the compaction invariant ceases to be satisfied the entire data structure is to be rebuilt from scratch for a value of  $\lambda$  equal to  $\lfloor \frac{2}{3} \log n \rfloor$ . Clearly, no such reconstruction will be needed until after  $\Omega(n)$  updates. Looking at a reconstruction as a sequence of insertions (at a polylogarithmic cost each), we will easily argue that the amortized cost of maintaining the compaction invariant is absorbed by the cost of individual updates. We now return to the specific problem of range counting in two dimensions.

A number of modifications to the  $M$ -structure are in order. To begin with, we replace fields  $C$  and  $C'$  by a dynamic search tree  $C(v)$ , e.g., a 2-3 tree (Aho et al. [AHU]), whose leaves are associated with the  $B$ -fields of  $W(v)$ . Each node (including the leaves) stores the number of meaningful 0's and the number of meaningful 1's in the  $B$ -fields of the leaves descending from it. This will essentially replace the use of  $C$ -fields while granting dynamic capabilities to the structure. Fields  $C$ ,  $C'$ ,  $D$ ,  $E$ ,  $F$ ,  $G$ ,  $H$  are no longer needed. The most important modification is the emulation of a *variable* word-size. In general,  $B$ -fields will not be filled entirely with meaningful bits. A  $B$ -field  $\beta$  will have only a suffix of meaningful bits, that is, a contiguous sequence in right-justified position. The length of this suffix varies between 0 and  $\lambda$ . It can be determined by adding the counts of 0's and 1's associated with the leaf  $\beta$  in  $C(v)$ .

**Invariant:** If  $\beta$  is not the only  $B$ -field at node  $v$ , then it must have between  $\lfloor \lambda/2 \rfloor$  and  $\lambda$  meaningful bits.

Without loss of generality, let  $w$  be the left child of  $v$ . We describe the computation of  $r(w, q)$ , given  $r(v, q)$ . Let  $\beta$  be the  $B$ -field that contains the bit of  $B(v)$  at position  $r(v, q)$ . We can find  $\beta$  by using the counts of 0's and 1's stored at each node of  $C(v)$

as partial ranks, and performing a simple binary search. This is similar to finding a key in a search tree, given its rank. Once  $\beta$  has been found, the position  $p$  of the desired bit in the  $B$ -field follows immediately. Note that this may require a subtraction, which we can afford since it takes only  $O(\log n)$  time on an EPM. As is often the case, however,  $p$  will only be used for comparisons, so it can be represented as a pair  $(r(v, q), p')$ , where  $p' = r(v, q) - p$  is the rank determined by the binary search. In this way, no subtraction is needed. While doing the search, we will collect partial counts of 0's: this is a special case of one-dimensional range search. If we visit a node  $z$  and branch to its right child, we update the partial count by adding to it the number of 0's provided with the left child of  $z$ . This is straightforward, so we omit the details. Finally, at the end of the search, we add up to this partial count the number of 0's among the first  $p$  bits of  $\beta$ . To enumerate these  $p$  bits we use binary search. Note that it is important to know the number of meaningful bits in  $\beta$ ; also the compaction invariant ensures that the tree needed for the enumeration remains of linear size. This gives us exactly  $r(w, q)$ . Clearly, a single transition takes  $O(\log n)$  steps. Answering a query can then be done in  $O(\log^2 n)$  time.

Next, we look at the problem of inserting or deleting a bit in a  $B$ -field  $\beta$ . We insert a new bit by locating its position in the  $B$ -field. To make room for the newcomer, we shift to the left the prefix of  $\beta$  ending at that position. Conversely, deleting a bit causes a right shift of a prefix of  $\beta$ . Every node of  $C(v)$  on the path from the current leaf to the root must have its partial counts updated, that is, incremented by  $-1$ ,  $0$ , or  $1$ . Although an EPM does not allow subtraction in constant time, we can get around this difficulty by maintaining a doubly-linked list of length  $n$  dynamically, each record storing its rank in the list. Instead of storing counts in  $C(v)$ , we store pointers to the record with the appropriate rank, which can always be done since no count can exceed  $n$ . This allows us to update counts in constant time while adding only a linear term to the size of the data structure.

What happens if the shift causes an overflow or an underflow? In the first case, we insert a new record in  $W(v)$  and use the  $B$ -field of that record to absorb the overflow. Since we then have  $\lambda + 1$  bits at our disposal, we clear the old  $B$ -field and allocate  $\lfloor \lambda/2 \rfloor$  of these bits to it, the remaining  $\lambda + 1 - \lfloor \lambda/2 \rfloor$  going into the new  $B$ -field. This causes the insertion of one node in  $C(v)$ . In case of underflow, we pick any adjacent  $B$ -field (one is always to be found since, by definition, no underflow can occur if  $W(v)$  has a single record) and evaluate their combined number of bits. If it exceeds  $\lambda$ , then we reassign the two  $B$ -fields with each at least  $\lfloor \lambda/2 \rfloor$  bits. Otherwise, we use a single  $B$ -field and discard the other one. If the tree  $C(v)$  falls out of balance, we apply the necessary rotations to reconfigure it into an acceptable shape. Rather than going into details, let us give a general argument, to be used again later on, showing why all counts can be updated in  $O(\log n)$  time. The main observation is that the Steiner minimal tree of all nodes whose counts need to be updated has size  $O(\log n)$ . We can then (conceptually) orient the edges of this subtree upwards and update every count by pebbling the resulting dag. Indeed, the counts at a node can be obtained from the counts of its children. See Knuth [K1] for variants.

Let us recap the sequence of operations for inserting a new point  $(x, y)$  into the structure. We start with a binary search in  $T$  with respect to  $x$ . This gives us the root-to-leaf path  $v_1, \dots, v_i$  at whose nodes an insertion must take place. At the root of  $T$  we perform a binary search in  $R(\text{root})$  to compute  $r(\text{root}, y)$ , and then descend in each of the relevant nodes by successive transitions. We are then ready to insert a new bit in each of the selected lists  $B(v_i)$ . Note that the value of this bit is determined by whether  $v_{i+1}$  is the right or left child of  $v_i$ . Deletions are handled in a similar fashion.

Of course, we must deal with the possibility of the tree  $T$  falling out of balance. For this purpose, we use a weight-balanced tree, also known as a  $\text{BB}[\alpha]$  tree. The idea, due to Lueker and Willard [L], [WL], [W2], is to rebuild the entire subtree rooted at the highest node going out of balance. This rebuilding is only expensive high up in the tree, but by chance weight-balanced trees go out of balance mostly at lower levels. With a proper choice of  $\alpha$ , it can be shown (Willard and Lueker [WL]) that a node  $v$  cannot go out of balance twice before a number of updates at least proportional to the size of the subtree rooted at  $v$  has taken place among its leaves.

A simple counting argument reveals the performance of this scheme. We create on the order of  $\log^2 n$  credits for every update, which we distribute evenly to each node  $v_i$  visited during the update. This should be enough to pay for the updates in each  $C(v_i)$  and still leave on the order of  $\log n$  credits to each node  $v_i$ . These remaining credits are stored in the *node-bank* of  $v_i$ . Suppose now that  $v_i$  falls out of balance. As we said earlier, after  $v_i$  last fell out of balance, at least  $\Omega(|H|)$  updates must have occurred at the leaves of the subtree  $H$  rooted at  $v_i$ . For this reason, the node-bank of  $v_i$  will have available at least on the order  $|H| \log n$  credits. Since constructing a data structure of size  $p$  takes  $O(p \log p)$  time, these credits can pay for the entire rebuilding of  $H$  and of all the structures in  $\{C(v) | v \in H\}$ . The node-bank at  $v_i$  is thus emptied, but ready to be refilled with future updates taking place below it. This proves that insertions and deletions can be performed in  $O(\log^2 n)$  amortized time. A more complex strategy (Willard and Lueker [WL]) can be invoked to spread the rebalancing over all the updates in an even manner and obtain an  $O(\log^2 n)$  worst-case update time.

We close this section with the problem of maintaining the compaction invariant. Our discussion will apply to range counting as well as to all the other dynamic problems considered later on. For this reason, we make the general assumption that any update among  $n$  active input elements requires  $O(\log^c n)$  amortized time ( $c$  is an arbitrary nonnegative constant). Between two consecutive reconstructions, assume that each update generates a number of credits equal to  $\log^c n$ , where  $n$  is the input size at the time of the update. We will show that these credits can cover the cost of the second reconstruction (up to within a constant factor). Let  $p$  (resp.,  $n$ ) be the size of the input during the first (resp., second) reconstruction. We have either  $\lfloor \frac{2}{3} \log p \rfloor < \frac{1}{3} \log n$  or  $\lfloor \frac{2}{3} \log p \rfloor > \log n$ . In the former case, we derive that  $n > p^2/8$ , therefore at least  $n - p = \Omega(n)$  insertions must have taken place between the two reconstructions. The credits generated thus amount to  $\Omega(n \log^c n)$ , which covers the cost of the second reconstruction. In the latter case,  $\lfloor \frac{2}{3} \log p \rfloor > \log n$ , we have  $p > n^{3/2}$ , which implies that at least  $p - n = \Omega(n^{3/2})$  deletions occurred. The credits released by these deletions trivially cover the  $O(n \log^c n)$  cost of reconstruction. Note that in both cases the credits released also cover the cost of maintaining auxiliary structures, such as the tree of size  $O(2^\lambda)$  used for bit enumeration.

**THEOREM 1.** *Range counting in two dimensions can be done in  $O(n)$  space and  $O(n \log n)$  preprocessing time. In the static case, the query time is  $O(\log n)$  on an APM. On a RAM the storage used is at most twice the number of words required to store the input. In the dynamic case, query and update times are  $O(\log^2 n)$  on an EPM.*

**4. On the process of identification.** As we will quickly find out, identifying the point associated with a given bit in  $B(v)$  is the computational bottleneck of most of the problems to be considered later on. For this reason, let us look at this process in some detail. Recall that computing a transition from  $(n(v, y), r(v, y))$  to  $(n(w, y), r(w, y))$  can be done without knowledge of  $y$ . This is convenient because it allows us to characterize any bit of  $B(v)$  by a pair  $(n_v, r_v)$ , knowing that the correspond-

ing bit in  $B(w)$ , denoted  $(n_w, r_w)$ , can be computed directly from  $(n_v, r_v)$ . This is the essential step for identification on a pointer machine.

LEMMA 1. *Let  $v$  be a node of  $T$  and let  $b$  denote an arbitrary bit of  $B(v)$ , specified by a pair of the form  $(n_v, r_v)$ . Computing a transition from  $b$  can be done in constant time on an APM. In the dynamic case, it can be done in  $O(\log n)$  time on an EPM.*

*Proof.* We review the basic steps of the algorithms, most of which we have already seen. To begin with, we determine the value of the bit  $b$  of  $B(v)$  corresponding to  $(n_v, r_v)$ . On an APM, we can find the position of the bit in the appropriate  $B$ -field in constant time, using  $C, C'$ -fields. Then we call on *shift* and  $\div$  to find  $b$ . Recall that the  $i$ th most significant bit of a  $\lambda$ -bit word  $x$  is  $h(x, i-1, i) = \lfloor x/2^{\lambda-i} \rfloor - 2 \lfloor x/2^{\lambda-i+1} \rfloor$ . Next, we set  $w$  to be the left child of  $v$  if  $b=0$ , or the right child if  $b=1$ . We are now ready to compute the transition from  $b$ , which is done exactly as described in the previous section. In the dynamic case, we use the tree  $C(v)$  to locate  $b$  in  $B(v)$  and then compute its location in  $B(w)$ . A single transition takes  $O(\log n)$  steps.  $\square$

The power or random access allows us to do much better on a RAM. The idea is to store a little more information at scattered nodes of  $T$  in order to compute repeated transitions in one fell swoop.

LEMMA 2. *Let  $v$  be a node of  $T$  and  $b$  denote an arbitrary bit of  $B(v)$  (at a known location), and let  $\varepsilon$  be any positive real. It is possible to modify an  $M$ -structure on a RAM so that identifying  $b$  can be accomplished in (1)  $O(n)$  space and  $O(\log^\varepsilon n)$  time; or (2)  $O(n \log \log n)$  space and  $O(\log \log n)$  time; or (3)  $O(n \log^\varepsilon n)$  space and constant time. In all cases, the transformation of the  $M$ -structure can be accomplished in  $O(n \log n)$  time.*

*Proof.* As usual in the static case we set  $\lambda = \lfloor \log n \rfloor$ . We define the *level* of a node of  $T$  as its number of ancestors. Let  $w$  be a descendent of  $v$ , and let  $b$  denote the bit of  $B(v)$  corresponding to  $(n_v, r_v)$ . How can we compute  $(n_w, r_w)$  from  $(n_v, r_v)$  in constant time? First note that  $w$  cannot be just any descendent of  $v$ , if  $B(w)$  is to have a bit in correspondence with  $b$ . At a given level in  $T$ , there is a unique node  $w$  that satisfies the criterion. Furthermore, recall that on a RAM the consideration of  $n_v$  and  $n_w$  is redundant, since  $r_v$  and  $r_w$  supply all the needed information in useful and efficient form. We can fully characterize the bit  $b$  by a pair of the form  $(v, r)$ . In practice, such a pair can be readily translated into an absolute address by adding  $r$  to the starting position of  $B(v)$ . Recall that this addressing at the bit level is virtual, but that it can be emulated in constant time on a RAM. By abuse of language, we will refer to  $b$  as “bit  $(v, r)$ .” The problem at hand is that of computing a batch of consecutive transitions, and more precisely, the function  $\theta(v, r, l) = (w, s)$ , where  $(w, s)$  is the bit at level  $l$  in correspondence with  $(v, r)$ . Whenever  $l$  is understood, we will say that bit  $\theta(v, r, l)$  is the *companion* of bit  $(v, r)$ .

We show how to compute  $\theta(v, r, l)$  in constant time, adding only a small amount of storage to the  $M$ -structure. We consider that  $l$  and  $v$  are both fixed and we assume that  $l$  is larger than the level of  $v$ . Let  $h$  be the difference between  $l$  and the level of  $v$ , and let the sequence  $w_0, \dots, w_{2^h-1}$  denote the descendents of  $v$  at level  $l$  (without loss of generality, we will assume that  $n$  is a power of two, so that  $v$  has exactly  $2^h$  descendents at level  $l$ ). As usual,  $B(v)$  is broken up into words  $\beta_0, \dots, \beta_{m-1}$ , each made of  $\lambda$  meaningful bits (except possibly for  $\beta_{m-1}$ , a special case that we will ignore in the remainder of this discussion). We cannot afford to attach with each bit of  $B(v)$  the address of its companion, but we might still want to do so for a sample of  $B(v)$ ; for example, to follow a known pattern, for the least significant bit of each  $\beta_i$ . There is a catch, however. Indeed, this could lead to a situation where, say, all the companions land in the same node at level  $l$ , thus providing useless information for most of the other bits of  $B(v)$ . To circumvent this difficulty, we do not link the bits to their

companions but to *neighboring bits* chosen in a round-robin fashion among the nodes at level  $l$ . We augment each  $\beta_i$  with a pointer  $\gamma_i$  and a bit vector  $\Gamma_i$ , defined as follows:

- (1) The pointers  $\gamma_i$  allow us to jump from selected places in  $B(v)$  to related places at level  $l$ . For any  $j$  such that  $0 \leq j < 2^h$ , we say that bit  $(w_j, s)$  is the  $j$ -*companion* of bit  $(v, r)$  if its identifier is the smallest value in  $R(w_j)$  that is at least as large as the identifier of  $(v, r)$ . Note that there is a unique  $j$  for which the  $j$ -companion of  $(v, r)$  is also its companion. For each  $i$  such that  $0 \leq i < m$ , we define  $\gamma_i$  as the address of the word in  $B(w_{i \bmod 2^h})$  that contains the  $(i \bmod 2^h)$ -companion of the least significant bit in  $\beta_i$ . If there is no such companion,  $\gamma_i$  need not be defined.
- (2) The bit vectors  $\Gamma_i$  allow us to correct the errors introduced by the  $\gamma_i$ 's. Let  $(v, r)$  be as usual an arbitrary bit of  $\beta_i$ . Let  $\theta(v, r, l) = (w_j, s)$ , and let  $k$  be the largest integer less than  $i$  such that  $\gamma_k$  points to a word of  $B(w_j)$ . Suppose for the time being that  $k$  is well defined. Then it cannot be too far away from  $i$  because of the round-robin scheme. We easily verify that  $k$  is always at least as large as  $i - 2^h$ . The address in  $\gamma_k$  allows us to jump from  $v$  to  $w_j$  within some reasonable accuracy. To complete the computation, we need information to (1) compute  $k$ , given  $(v, r)$ , and (2) go from  $\gamma_k$  to the actual companion of  $(v, r)$ . If  $(v, r)$  is the  $d$ th most significant bit of  $\beta_i$ , we define  $P_d$  as an  $(h+1)$ -long bit vector containing the binary representation of  $i-k$  in right-justified position. This allows us to complete the first task. For the second one, we define  $Q_d$  as a  $(\lfloor \log \lambda \rfloor + h + 3)$ -bit vector. Let  $d_0, d_1, d_2 \dots$  be the bits of  $B(w_j)$ , broken down into words,  $\delta_0, \delta_1, \delta_2$ , etc. Let  $\delta_a$  be the word pointed to by  $\gamma_k$ , let  $d_b$  be the most significant meaningful bit of  $\delta_a$ , and let  $d_c$  be the bit  $\theta(v, r, l)$ ; we define  $Q_d$  as the  $(\lfloor \log \lambda \rfloor + h + 3)$ -bit vector containing the binary representation of  $c-b$  in right-justified position (note that  $c \geq b$ ). We can verify that the number of bits provided by  $Q_d$  is sufficient for that purpose ( $\#$  bits needed  $= \lceil \log(c-b+1) \rceil < \lfloor \log \lambda \rfloor + h + 3$ ). If  $k$  is not defined then  $P_d$  and  $Q_d$  store the values of  $j$  and  $c$ , respectively, with a flag to indicate so; again  $\lfloor \log \lambda \rfloor + 2h + 4$  are more than sufficient for that purpose. Finally, we define  $\Gamma_i$  as the concatenated string  $P_1.Q_1.P_2.Q_2 \dots P_\lambda.Q_\lambda$ . Note that  $\Gamma_i$  is  $\lambda(\lfloor \log \lambda \rfloor + 2h + 4)$  bits long.

Computing  $\theta(v, r, l)$  is now straightforward. To begin with, we find the index  $i$  of the word  $\beta_i$  that contains bit  $(v, r)$ . This gives us the position  $d$  of  $(v, r)$  in  $\beta_i$ , which in turn leads to the two bit vectors  $P_d$  and  $Q_d$ . Note that both vectors can always be read in constant time. With  $P_d$  in hand, we can retrieve  $k$  as well as the address stored in  $\gamma_k$ . Using now  $Q_d$  as a relative address, we automatically gain access to  $\theta(v, r, l)$ . All these operations take constant time on a RAM. So, we have devised a technique for computing repeated transitions in constant time. But at what cost in storage? With the notation used above, the overhead amounts to one word  $\gamma_i$  and one bit vector  $\Gamma_i$  for each word  $\beta_i$  of  $B(v)$ . This gives an average of  $O((h + \log \lambda)/\lambda)$  extra words per bit of  $B(v)$ , hence a grand total of  $O(((h + \log \lambda)/\lambda)n + 2^{l-h})$  space, summing up over every node  $v$  at level  $l-h$ . Recall that there are exactly  $n$  bits in all the  $B(v)$ 's combined, at any given level. Of course,  $\Gamma_i$  will be organized as an array of words. Because each  $\Gamma_i$  is of the same length, these can be packed into one large array in order to simulate a two-dimensional matrix. To summarize, we have a scheme for computing  $h$  consecutive transitions in constant time from level  $j$ , using  $O(((h + \log \lambda)/\lambda)n + 2^j)$  added space.

What is the time needed to construct the set of  $\gamma_i$  and  $\Gamma_i$ ? With the random access features of a RAM, it is not difficult to carry out the preprocessing in  $O(n)$  time. We



briefly describe the procedure. Assuming that for each  $v$  at level  $l-h$  the array  $R(v)$  is available, we can compute the companion of each bit of  $B(v)$  in constant amortized time. To do so, we compute  $R(w_0), \dots, R(w_{2^h-1})$ , using bucket sort to place each element in its right set. This is possible because, if we work in rank space, the points of  $R(v)$  can be partitioned by vertical slabs of the same width, each corresponding to a distinct  $R(w_j)$ . Knowing the companions of the bits of  $B(v)$  serves two purposes: one is to provide the position of each companion in its list  $R(w_j)$ , and the other is to reveal the name of the node in question. With the first piece of information we compute  $Q_d$ , and with the latter we derive  $P_d$ . To do so, we compute the round-robin assignment of the bits of  $B(v)$  and form  $O(2^h)$  sorted lists of  $y$ -values, which we merge with  $R(w_0), \dots, R(w_{2^h-1})$ , in turn. This gives us all the  $\gamma_i$ 's in linear time. To compute the  $P_d$ 's, it suffices to scan  $B(v)$  and for each bit deduce  $P_d$  from the index  $j$  of the node  $w_j$  associated with its companion. Finally,  $Q_d$  can be found by simulating the computation of  $\theta(v, r, l)$  for each bit of  $B(v)$ . We can obtain  $Q_d$  as soon as it is needed in the computation, since we already know the value of  $\theta(v, r, l)$ . These explanations are more intuitive than formal, but we do not judge it necessary to dwell on this technical but simple aspect of the algorithm. For convenience, we denote the data structure which we have just described,  $\Theta(l-h, l)$ . In general,  $\Theta(a, b)$  is a structure that allows us to compute transitions from level  $a$  to level  $b$  in constant time, using  $O(((b-a + \log \lambda)/\lambda)n + 2^a)$  extra space. Incidentally, there is an obvious discrepancy with regard to this bound. If we set  $b-a=1$ , then we have just described a method for computing constant time transitions, while using an extra  $\log \log n$  bits for each bit of  $B(v)$ ! Of course, this result is useless since we already know how to compute constant time transitions with only constant overhead in space. We will therefore save the more complex method only as a way to batch many consecutive transitions together.

We are now ready to attack the main question: how to identify  $(v, r)$ ? Let  $m$  be the level of  $v$  ( $m < \lambda - 1$ ); the idea is to express  $m$  in some appropriate number system. We will build several data structures on the same model. Let  $D(x, y)$  denote a data structure for jumping from any level  $l_1$  to any level  $l_2$ , with  $\lambda - y \leq l_1 < l_2 \leq \lambda - x$ . Identification can be done by using  $D(0, \lambda)$ . We define  $D(x, y)$  in a recursive manner (for notational convenience we assume, without loss of generality, that  $x = 0$ ). Let  $\alpha(y)$  be an integer function of  $y$  such that  $0 \leq \alpha(y) \leq y/2$ . The data structure  $D(0, y)$  is made of

$$\Theta(\lambda - y, \lambda) \cup \left\{ D(0, \alpha(y) - 1), \dots, D(j\alpha(y), (j+1)\alpha(y) - 1), \dots, D\left(\left\lfloor \frac{y}{\alpha(y)} \right\rfloor \alpha(y), y\right) \right\}.$$

We stop the recursion at some threshold  $\nu$ , which is to say that  $D(a, b)$  is null for  $b - a \leq \nu$ . How do we physically store these structures? We want to emulate a situation where each  $\Theta(i, j)$  is accessible as defined earlier, that is, as though it were the only one attached to level  $i$ . Unfortunately, many  $\Theta(i, j)$ 's are bound to be assigned to the same level  $i$ . We solve this difficulty by storing the  $\Theta(i, j)$ 's as two-dimensional matrices (as before), and keeping an entry table  $E[0..\lambda + 1, 0..\lambda + 1]$  defined as follows:  $E[i, j]$  stores the address of the first entry in  $\Theta(i, j)$ , if  $\Theta(i, j)$  appears in the full expansion of  $D(0, \lambda)$ ; it stores 0 otherwise. The storage  $S(n, y)$  occupied by  $D(0, y)$  follows the recurrence

$$S(n, y) = \left\lfloor \frac{y}{\alpha(y)} \right\rfloor S(n, \alpha(y)) + S\left(n, y + 1 - \left\lfloor \frac{y}{\alpha(y)} \right\rfloor \alpha(y)\right) + O\left(\frac{y + \log \lambda}{\lambda} n\right),$$

and  $S(n, y) = 0$ , for  $y \leq \nu$  (threshold below which we stop the recursion). We have

omitted the term  $2^{\lambda-y}$  to be able to treat the recurrence uniformly. We will see later on the effect of this omission. Let  $\alpha^{(0)}(y) = y$  and, for any  $i > 0$ , let  $\alpha^{(i)}(y) = \alpha(\alpha^{(i-1)}(y))$ . Finally, let  $\alpha^*(y) = \max \{i | \alpha^{(i)}(y) \geq \nu\}$ . We derive the relation

$$(1) \quad S(n, \lambda) = O\left(\left(\alpha^*(\lambda) + 1 + r \frac{\log \lambda}{\nu}\right)n\right).$$

We can easily show that taking into account the terms of the form  $2^{\lambda-y}$  adds another term  $O((\alpha^*(\lambda) + 1)n)$ , which asymptotically is immaterial. We briefly describe the preprocessing. We define the first recursive layer  $L_1$  as the construction of  $\Theta(0, \lambda)$ . The second recursive layer  $L_2$  involves building in this order

$$\Theta\left(0, \lambda - \left\lfloor \frac{\lambda}{\alpha(\lambda)} \right\rfloor \alpha(\lambda)\right), \dots, \Theta(\lambda - (j+1)\alpha(\lambda) + 1, \lambda - j\alpha(\lambda)), \dots, \Theta(\lambda - \alpha(\lambda) + 1, \lambda),$$

and so forth, every time expanding each term with its recursive definition. If we look at the definition of  $D(0, \lambda)$  as inducing a tree of height roughly  $\alpha^*(\lambda)$ , then  $L_i$  represents the work to be accomplished to build each node of the tree at level  $i$ . Starting with  $R(\text{root})$ , we can construct each layer in time  $O(n|L_i|)$ , where  $|L_i|$  is the number of terms involved in the expression of  $L_i$ . This derives from our previous observation on the preprocessing of  $\Theta(a, b)$ . Note that when computing a  $\Theta$ -structure the set of lists  $R(v)$  needed for the computation can be obtained from the previous one in linear time. Since  $\alpha(y) \leq y/2$ , the sum  $\sum n|L_i|$  is a geometric series whose value is on the order of  $(\lambda/\nu)n$ . This gives a preprocessing time of  $O(n \log n)$ .

To identify  $(v, r)$ , we begin by checking to see if  $m$ , the level of  $v$ , is equal to 0. If this is the case, we can then use  $\Theta(0, \lambda)$  and conclude in constant time. If  $m > 0$ , we compute the starting interval of the form  $[j\alpha(\lambda), (j+1)\alpha(\lambda) - 1]$  (or  $[\lfloor \lambda/\alpha(\lambda) \rfloor \alpha(\lambda), \lambda]$ ) that contains  $\lambda - m$ . We leap from  $v$  to the companion of  $(v, r)$  at level  $\lambda - j\alpha(\lambda)$  (with  $j \leq \lfloor \lambda/\alpha(\lambda) \rfloor$ ) by calling the algorithm recursively, using  $D(j\alpha(\lambda), (j+1)\alpha(\lambda) - 1)$  (or  $D(\lfloor \lambda/\alpha(\lambda) \rfloor \alpha(\lambda), \lambda)$ ). When the recursion passes the cutoff point (i.e., is about to jump across fewer than  $\nu$  levels) we complete the computation by taking transitions one level at a time. Of course, we assume that we have the appropriate data structure for doing so. When this is done, we jump from level  $\lambda - j\alpha(\lambda)$  to level  $\lambda$  by taking intermediate steps. Let  $k$  be initially set to  $j$ .

- (1) Take a regular transition from  $\lambda - k\alpha(\lambda)$  to  $\lambda - (k\alpha(\lambda) - 1)$ ;
- (2) Jump from  $\lambda - (k\alpha(\lambda) - 1)$  to  $\lambda - (k-1)\alpha(\lambda)$ , using  $D((k-1)\alpha(\lambda), k\alpha(\lambda) - 1)$ ;
- (3) Decrement  $k$  by 1. If  $k > 0$  then go to step (1), else stop.

Let  $Q(n, m)$  be the time to identify  $(v, r)$ , that is, to jump across  $\lambda - m$  levels. We immediately find that

$$(2) \quad Q(n, m) = O\left(\nu + \sum_{\nu \leq \alpha^{(i)}(\lambda) \leq \lambda} \frac{\alpha^{(i)}(\lambda)}{\alpha^{(i+1)}(\lambda)}\right).$$

Setting  $\alpha(y) = \lfloor y/2 \rfloor$  and  $\nu = 1$ , from (1) and (2) we obtain  $S(n, \lambda) = O(n \log \log n)$  and  $Q(n, m) = O(\log \lambda) = O(\log \log n)$ . This completes the proof of the second part of Lemma 2. Let us now set  $\alpha(y) = \lfloor y/\lambda^\epsilon \rfloor$  and  $\nu = \log \lambda$ . We derive  $S(n, \lambda) = O(n)$  and  $Q(n, m) = O(\log^\epsilon n)$ , which gives us the first part of Lemma 2.

To prove the last part of Lemma 2, we must modify the overall design of the data structure:  $D(0, y)$  will now be made not only of

$$\left\{ D(0, \alpha(y) - 1), \dots, D(j\alpha(y), (j+1)\alpha(y) - 1), \dots, D\left(\left\lfloor \frac{y}{\alpha(y)} \right\rfloor \alpha(y), y\right) \right\},$$

but also of

$$\left\{ \Theta(\lambda - \alpha(y), \lambda), \dots, \Theta(\lambda - j\alpha(y), \lambda), \dots, \Theta\left(\lambda - \left\lfloor \frac{y}{\alpha(y)} \right\rfloor \alpha(y), \lambda\right), \Theta(\lambda - y, \lambda) \right\}.$$

As usual,  $D(a, b)$  is null if  $b - a \leq \nu$ . We assume that  $\alpha(y) = \lfloor y/\lambda^{\epsilon/2} \rfloor$  and  $\nu = 1$ . The storage needed follows the relation

$$S(n, y) = \left\lfloor \frac{y}{\alpha(y)} \right\rfloor S(n, \alpha(y)) + S\left(n, y + 1 - \left\lfloor \frac{y}{\alpha(y)} \right\rfloor \alpha(y)\right) + O\left(\frac{y(y + \log \lambda)}{\alpha(y)\lambda} n\right),$$

and  $S(n, y) = 0$ , for  $y \leq \nu$ . This gives  $S(n, \lambda) = O(n \log^{\epsilon/2} n \log \log n) = O(n \log^\epsilon n)$ . Once again, it is easily verified that the extra terms  $2^a$  omitted from the recurrence are absorbed asymptotically. The construction time is  $O(n \log n)$  since there are only a constant number of layers. To identify  $(v, r)$ , we proceed recursively within the starting interval. The key difference now is that at a given recursion level, only a single  $\Theta$ -structure need be used. The time thus becomes proportional to the number of recursive calls, which is constant. The proof of Lemma 2 is now complete.  $\square$

**5. Range reporting.** For this problem we use an  $M$ -structure with the various modifications described in the previous section. It is clear that to start out we might as well mimic the algorithm for range counting until all the bits in the appropriate  $B(v)$ 's have been located. These are the bits whose identifiers fall in the query range. They appear as  $O(\log n)$  sequences of consecutive bits, each sequence being associated with a distinct node of  $T$ . We call these sequences the *decomposition vectors* of the query. To complete the computation, it suffices to identify each bit of the decomposition vectors. This can be done by applying the techniques of the previous section to each bit individually.

How do we dynamize the data structure? On an EPM, there is no difficulty adapting to the problem at hand our solution to range counting. Identification proceeds by applying repeated transitions, using the methods of Lemma 1.

This technique leads to the results stated in the tables of § 1, except for one minor difference: we have just proven a slightly weaker set of results obtained by replacing each occurrence of  $n/k$  by  $n$  in the tables. Obviously, to obtain the results claimed it suffices to treat, say, the case  $k \geq n^{2/3}$  separately. Indeed, if  $k < n^{2/3}$  then  $\log n$  and  $\log(n/k)$  are within constant factors of each other. We will describe a dynamic algorithm on an EPM with a complexity of  $O(n)$  space,  $O(n \log n)$  preprocessing time,  $O(\log n)$  update time, and for  $k \geq n^{2/3}$ ,  $O(k)$  query time. Since  $k$  is not known beforehand the idea will be to dovetail between this method and the others (i.e., run the two competing algorithms in parallel). The first process to terminate will thus guarantee the upper bounds claimed in the tables. We define the alternative algorithm by combining standard data structuring techniques. Assuming for simplicity that all  $x$ -coordinates are distinct, we partition the set of  $n$  points by means of vertical slabs into groups of size between  $p(n)$  and  $4p(n)$ , where  $p(n) = \lfloor 2^{2\lceil \log n \rceil / 3} \rfloor$ . For each group of points we maintain a dynamic list containing the indices of the points sorted by  $y$ -coordinates. Given two query values  $y$  and  $y'$  we will use the list to report all the points in the group with  $y$ -coordinates between  $y$  and  $y'$ . Using, say, a 2-3 tree, it is easy to guarantee  $O(k + \log n)$  query time (where  $k$  is the number of points to be reported) and  $O(\log n)$  update time. The storage requirement is trivially linear. The utility of this data structure for the previous problem should be obvious. Given a query rectangle, our first task is to use the slabs to partition it into subrectangles, which is easily done in  $O(\log n)$  time. If the query rectangle falls entirely within a single slab,

then we naively check each point in the slab. If the partition is effective, however, then each subrectangle can be handled by using the appropriate list (checking each element in the list in the case of the leftmost and rightmost subrectangles, and using the tree structures for the other subrectangles, if any).

The query time is easily shown to be  $O(k + n^{1/3} \log n + n^{2/3})$ , which is  $O(k)$  if  $k \geq n^{2/3}$ . Insertions and deletions are performed in logarithmic time by updating the relevant lists. To handle overflows and underflows we will assume for the time being that although  $n$  varies,  $\lfloor \log n \rfloor$  and hence  $p(n)$  remain the same. Let  $m$  be the size of a group where an insertion has just taken place. If  $m > 4p(n)$  then break up the group into two subgroups of size  $\lfloor m/2 \rfloor$  and  $\lceil m/2 \rceil$ . If a deletion has taken place and  $m < p(n)$  then let  $m'$  be the size of one of its adjacent groups (guaranteed to exist if  $n$  is larger than some constant). If  $m + m' \leq 3p(n)$  then merge the two groups into one, otherwise form two groups of size  $\lfloor (m + m')/2 \rfloor$  and  $\lceil (m + m')/2 \rceil$ , respectively. In all cases the relevant lists are reconstructed from scratch. We easily check that each new group has its size  $\Omega(p(n))$  away from the two limits  $p(n)$  and  $4p(n)$ . A simple accounting argument can then be used to prove that an insertion/deletion has an  $O(\log n)$  amortized complexity. Note that once  $p(n)$  is known the only arithmetic needed is addition and subtraction by 1, which can be implemented in  $O(\log n)$  time on an EPM. (Or even in  $O(1)$ , if we maintain a linked list from 1 to  $n$  and pointers to the values of the various group sizes). If now  $\lfloor \log n \rfloor$  increases or decreases by 1, we simply rebuild the entire data structure from scratch, which is easily done in  $O(n \log n)$  time (or even  $O(n)$  using presorting). Note that to update the value of  $p(n)$  at each update can be easily done in constant amortized time (on an EPM). When it is decided that a reconstruction is needed, prior to it we replace  $p(n)$  by  $q(n) = \lfloor 2^{2 \lfloor \log(3n) \rfloor / 3} \rfloor$ , and always alternate back and forth between  $p(n)$  and  $q(n)$ . In this way, we are guaranteed to have  $\Omega(n)$  updates between two consecutive reconstructions. Obviously, everything we said earlier about  $p(n)$  applies to  $q(n)$  just as well. Since  $p(n)$  and  $q(n)$  can be computed in  $O(n)$  time, using only additions, reconstructions still preserve the logarithmic amortized complexity of an update.

**THEOREM 2.** *Data structures for range reporting in two dimensions can be constructed in  $O(n \log n)$  time in all cases. Let  $k - 1$  be the number of points to be reported, and let  $\varepsilon$  be any real  $> 0$ . On a RAM, a tradeoff is possible: in particular, we can achieve (1)  $O(k(\log(2n/k))^\varepsilon + \log n)$  query time and  $O(n)$  space; (2)  $O(k \log \log(4n/k) + \log n)$  query time and  $O(n \log \log n)$  space; (3)  $O(k + \log n)$  query time and  $O(n \log^\varepsilon n)$  space. On an APM, it is possible to achieve a query time of  $O(k \log(2n/k))$ , using  $O(n)$  space. In the dynamic case on an EPM, it is possible to achieve a query time of  $O(k(\log(2n/k))^2)$  and an update time of  $O(\log^2 n)$  time, using  $O(n)$  space.*

**6. Range searching for maximum.** As usual, we will make use of an  $M$ -structure, and we will start the algorithm by computing the decomposition vectors of the query. In general, these vectors will fail to fill whole sequences of  $B$ -fields. However, they can be broken up into three (or fewer) blocks, one of which corresponds to a sequence of  $B$ -fields, and the others to subparts. Discovering the identifier with maximum value in each block will readily lead to the final answer. Let  $v$  be an arbitrary node of  $T$ . To handle the first case, we add an  $M$ -field to each record of  $W(v)$  to indicate the maximum value attained by the identifiers of bits in the corresponding  $B$ -field. Then we set up a data structure  $M_1$  for doing one-dimensional range search for maximum with respect to the  $M$ -fields of  $W(v)$ . For the other blocks we need a data structure  $M_2$  for computing  $\maxval(\beta, k, l)$ , a function that takes as input a  $B$ -field  $\beta = x_0 \cdots x_{\lambda-1}$  and two bit positions  $0 \leq k \leq l < \lambda$ , and returns the position  $i$  of the bit  $x_i$

whose identifier has maximum value among  $\{x_k, x_{k+1}, \dots, x_l\}$ . We next show how to implement  $M_1$  and  $M_2$  in our various models of computation.

*Implementation of  $M_1$ .*

- (1) *On a RAM.* We use a technique of Gabow, Bentley, and Tarjan [GBT] to reduce one-dimensional range search for maximum to the computation of nearest common ancestors in some special tree. To begin with, we construct a *Cartesian tree* with respect to the  $M$ -fields of each  $W(v)$ . This tree, discovered by Vuillemin [V], is defined as follows: pick the maximum  $M$ -field and store it in the root of the tree. Remove this maximum from the sequence of  $M$ -fields, and define the left (resp., right) subtree recursively with respect to the left (resp., right) remaining subsequence. Two important facts about Cartesian trees state that: (a) they can be built in linear time; (b) one-dimensional range search for maximum is reducible to the computation of the nearest common ancestor (nca) of two nodes in a Cartesian tree. Harel and Tarjan [HT] have given an efficient method for computing nca's. Their method allows us to implement  $M_1$  in linear space and time, and obtain any answer in constant time.
- (2) *On an APM or an EPM.* More simply, we use a complete binary tree with each node  $v$  holding the maximum value in the range spanned by the subtree rooted at  $v$ . This heap structure is of standard use for one-dimensional range search, so we need not elaborate on it. Query time is  $O(\log n)$  in both models.

*Implementation of  $M_2$ .*

- (1) *On a RAM.* We still use Cartesian trees, but in an indirect manner. The key observation is that for our purposes Cartesian trees do not need to be labelled. Therefore they can be represented by a canonical index over a number of bits =  $O(\log \# \text{Cartesian trees of size } \lambda)$ . We can get a rough estimate on this number by representing a  $p$ -node Cartesian tree as a string of matching parentheses for internal nodes, 1 for leaves and 0 for missing leaves, as in  $((((10)((01)0))((01)1)))$ , for example. Missing leaves are the nonexistent brothers of single children (poor things!). There are  $p$  nodes and at most  $p - 1$  missing leaves, so  $6p$  bits are sufficient for the encoding. Since the number of nodes we are dealing with is quite small, we can precompute all possible Cartesian trees and preprocess them for efficient nca computation. It will then suffice to store a pointer from each record in  $W(v)$  to the appropriate Cartesian tree. Let  $\alpha p$  be the number of words required to store a  $p$ -node tree preprocessed for constant time nca computation (Harel and Tarjan [HT]). Storing all Cartesian trees of size  $\lambda$  will thus require  $4^{3\lambda} \alpha \lambda = \Theta(n^6 \log n)$  words! This is a bit too much, so we must resize the  $B$ -fields. We divide each  $B$ -field into subwords of at most  $\lfloor \lambda/7 \rfloor$  bits each, and for each piece we keep a pointer to its appropriate  $\lfloor \lambda/7 \rfloor$ -node Cartesian tree. This will multiply the query time by a constant factor, but it will also bring down the space requirement to a more acceptable  $O(n^{6/7} \log n) = O(n)$  bound. The time to compute  $\text{maxval}(\beta, k, l)$  is constant. But to identify the corresponding bit, we must resort to the techniques of Lemma 2. Since there are  $O(\log n)$  decomposition vectors, we can achieve a query time of  $O(\log^{1+\epsilon} n)$  with  $O(n)$  space; a query time of  $O(\log n \log \log n)$  with  $O(n \log \log n)$  space; or a query time of  $O(\log n)$  with  $O(n \log^\epsilon n)$  space.
- (2) *On an APM or an EPM.* We use the same technique, except for the preprocessing of Cartesian trees, now no longer needed. We compute an nca naively by examining the entire tree. This allows us to compute  $\text{maxval}(\beta, k, l)$  in

$O(\log n)$  time. The total cost of the computation will be  $O(\log^2 n)$  on an APM and  $O(\log^3 n)$  on an EPM.

In all cases (including on a RAM with constant-time nca computation), the preprocessing can be done in linear time per level of  $T$ , that is, in a total of  $O(n \log n)$  steps. This concludes our treatment of the static case.

*The dynamic case.* How can we dynamize these data structures on a pointer machine? The problem is quite similar to range counting, after all, so the proof of Theorem 1 can serve us as a guide. Certainly, the treatment of  $B$ -fields and the management of the structures  $M_1$  can follow the same lines used for range counting. Difficulties arise with  $M_2$ , however. Consider the insertion or deletion of a bit in some  $B$ -field  $\beta$ . What is wrong with the following scheme? Find which subpart of  $\beta$  houses the bit in question and follow its pointer to the appropriate Cartesian tree. Then update the tree naively by considering each of its nodes. There are two basic problems: one is to update a Cartesian tree without knowing the values of the keys. Another problem, though less serious, is that these trees are shared among many pointers, so pleasing one pointer with an update would most likely upset many others. Once again, a functional approach will take us out of this dilemma.

Instead of a Cartesian tree, we use a dynamic one-dimensional range tree (similar to  $M_1$ ); for example a 2-3 tree (Aho et al. [AHU]). Ideally, each node would store the maximum value associated with its leaves below (Fig. 2(a)). But this is as costly as having labelled trees, so we must seek a different solution. The key remark is that we need not store values in the nodes, but simply bits indicating where these values come from. These are called the *direction flags* of the tree. If the value at node  $z$  originates from a leaf of its leftmost subtree, the direction flag of  $z$  is 0; if it comes from the second subtree from the left, it is 1; else the direction flag is 2 (Fig. 2(b)).

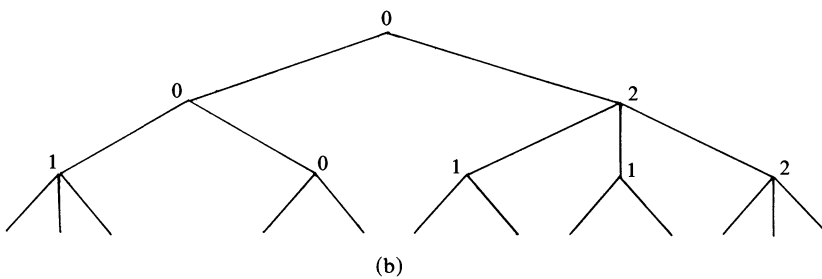
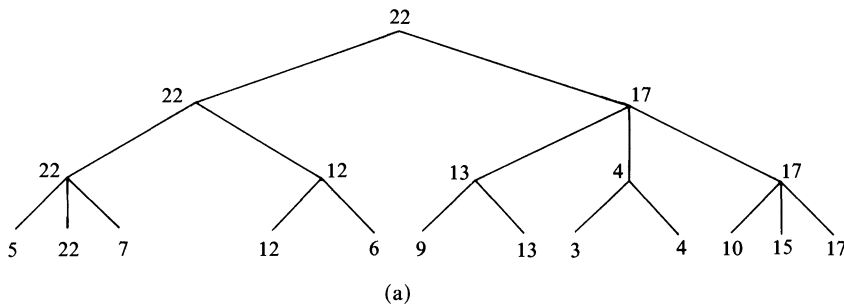


FIG. 2

We can represent such a tree by means of balanced parentheses. Each leaf is a 1 and each internal node is a pair  $(\cdot \cdot \cdot)$  immediately followed by its direction flag. For example, the tree of Fig. 2(b) is encoded as

$$(((111)1(11)0)0((11)1(11)1(111)2)2)0.$$

We represent each character on two bits: 0, 1, 2 as they are, “(” as 2, and “)” as 3, for example. Note that although 2 is used for different purposes there is no source of ambiguity, since 2 is always preceded by “)” and “)” can never occur. Encoding and decoding any such tree can easily be done in time linear in its size. A leaf takes up 1 character and an internal node 3, so the tree associated with a  $B$ -field of  $\lambda$  meaningful bits can be encoded over  $8\lambda - 6$  bits. Since every description starts with 10 there is no need for a boundary delimiter if the bits are right-justified. We will therefore assign 8 additional words for each  $B$ -field.

How can we use these trees to compute  $\maxval(\beta, k, l)$ ? To begin with, we decode the tree  $T_\beta$  associated with  $\beta$ . For (conceptual) simplicity, we might want to request  $O(\lambda)$  temporary storage to construct a “useful” copy of  $T_\beta$ , that is, using pointers, one-word fields for direction flags, etc. We can now compute  $\maxval(\beta, k, l)$  by performing one-dimensional range search in  $T_\beta$ . The canonical decomposition of  $[k, l]$  can be computed in  $O(\log \lambda)$  operations. This reduces the investigation to  $O(\log \lambda)$  nodes of  $T_\beta$ . For each such node, we determine the leaf from which its associated value originates, and we identify the corresponding bit. This will take a total of  $O(\log \lambda \log^2 n)$  time on an EPM.

Let us now look at insertions and deletions. We proceed as in the proof of Theorem 1. Suppose that no underflow or overflow occurs within the  $B$ -field  $\beta$ . In that case, we carry out the update in  $T_\beta$  alone. Intuitively, we can look at the tree in its entirety, but we should try to minimize the number of bits that must be identified since these are the costly operations. The advantage of using a balanced tree is to limit the number of necessary identifications to  $O(\log \lambda)$  (as opposed to  $O(\lambda)$ ). For each update, we apply the standard insert or delete procedure for 2-3 trees. This can only upset the direction flags of the nodes that are effectively manipulated by the update as well as the direction flags of their ancestors. We promptly repair the damage caused to the direction flags by identifying the bits associated with the children of each of these nodes. This will take time  $O(\log \lambda \log^2 n)$  on an EPM.

How do we deal with overflow or underflow? Following the proof of Theorem 1, the reconfiguration of the  $B$ -fields necessitates splitting or concatenating one or two trees. This can be done in  $O(\log \lambda)$  steps [AHU]. Again we must ask: how many direction flags must be recomputed? A careful examination of the split and concatenate procedures on a 2-3 tree shows that after each application of these operations the number of nodes that either have been manipulated or are ancestors of nodes that have been manipulated is at most logarithmic in the size of the tree. Since these are the only ones whose direction flags might need updating, we conclude that the time spent to handle underflow and overflow is asymptotically the same as the time to insert or delete a new element.

As soon as a tree has been updated, we re-encode it as a bit vector and re-attach the result to the relevant  $B$ -field, discarding the old tree. Once  $M_2$  has been computed, updating  $M_1$  can be done in  $O(\log n)$  operations on all models, assuming that it is implemented as a dynamic one-dimensional range search tree. The procedure is similar to the updating of  $C(v)$  in the case of range counting (§ 3.2). It is also possible to unify the treatment of  $M_1$  and  $M_2$  and consider a single master tree with two modes

of representation: with pointers for  $M_1$  (the top part) and with bit vectors for  $M_2$  (the lower levels).

To summarize, it takes  $O(\log^2 n \log \log n)$  time to carry an update with respect to a given node of  $T$ . Reconfiguring a subtree of  $T$  of size  $p$  can easily be done in  $O(p \log p)$  steps: play a knock-out tournament at each level of  $T$  being reconstructed. The usual counting argument shows that the amortized cost of rebuilding  $T$  when it goes out of balance is  $O(\log^2 n)$ . Unfortunately this is largely dominated by the  $O(\log^3 n \log \log n)$  steps incurred at the time of insertion or deletion.

**THEOREM 3.** *The data structures for range searching for maximum in two dimensions can be constructed in  $O(n \log n)$  time in all cases. On a RAM, a tradeoff is possible: in particular, we can achieve (1)  $O(\log^{1+\epsilon} n)$  query time and  $O(n)$  space; (2)  $O(\log n \log \log n)$  query time and  $O(n \log \log n)$  space; (3)  $O(\log n)$  query time and  $O(n \log^\epsilon n)$  space. On an APM (resp., EPM) it is possible to achieve a query time of  $O(\log^2 n)$  (resp.,  $O(\log^3 n)$ ), using  $O(n)$  space. In the dynamic case on an EPM, it is possible to achieve query and update times of  $O(\log^3 n \log \log n)$ , using  $O(n)$  space.*

**7. Semigroup range searching.** Of course, we assume that any value in the semigroup can be stored within one or, say, a constant number of computer words. We modify the  $M$ -structures by providing each node  $v$  with a complete binary tree whose leaves are associated with the  $B$ -fields of  $W(v)$ . Each leaf stores the semigroup sum of the values associated with the bits of its corresponding  $B$ -field. Similarly each internal node stores the sum of the values stored in its descending leaves. As in the case of range searching for maximum the algorithm computes the decomposition vectors of the query in a first stage. As usual, these are broken up into three (or fewer) blocks, one of which corresponds to sequences of  $B$ -fields, and the others to subparts. The semigroup sum associated with the identifiers of the first block can be readily obtained with two binary searches in the relevant auxiliary tree (this is a standard one-dimensional range search). For the other blocks, one must identify each of their bits to complete the computation, which will take a total of  $O(t(n) \log^2 n)$ , where  $t(n)$  is the time taken to identify one bit. We can then use Lemmas 1 and 2 to derive the performance of the algorithms.

Executing updates is similar to the dynamic treatment of range search for maximum, so no further elaboration is necessary. The  $M_2$ -structures can be eliminated altogether, therefore updating the data structure will require  $O(\log^2 n)$  identifications. Note that as usual the preprocessing requires  $O(n \log n)$  time.

**THEOREM 4.** *The data structures for semigroup range searching in two dimensions can be constructed in  $O(n \log n)$  time in all cases. On a RAM, a tradeoff is possible: in particular, we can achieve (1)  $O(\log^{2+\epsilon} n)$  query time and  $O(n)$  space; (2)  $O(\log^2 n \log \log n)$  query time and  $O(n \log \log n)$  space; (3)  $O(\log^2 n)$  query time and  $O(n \log^\epsilon n)$  space. On an APM it is possible to achieve a query time of  $O(\log^3 n)$ , using  $O(n)$  space. In the dynamic case on an EPM, it is possible to achieve query and update times of  $O(\log^4 n)$ , using  $O(n)$  space.*

**8. Rectangle searching: counting and reporting.** Using an equivalence result of Edelsbrunner and Overmars [EO], rectangle counting can be reduced to range counting. The basic idea is to subtract the number of rectangles which do *not* intersect the query from the total number of rectangles. Using inclusion-exclusion relations the problem reduces to a constant number of range counting problems. We immediately derive a result similar to Theorem 1. Note that the extra subtractions needed are inconsequential in the asymptotic complexity of the algorithms on an EPM.



**THEOREM 5.** *Rectangle counting in two dimensions can be done in  $O(n)$  space and  $O(n \log n)$  preprocessing time. In the static case, the query time is  $O(\log n)$  on an APM. In the dynamic case, query and update times are  $O(\log^2 n)$  on an EPM.*

As is well known (Edelsbrunner [Ed]), rectangle reporting in two dimensions can be reduced to three subproblems:

- (1) *Range reporting.* See above.
- (2) *Orthogonal segment intersection.* Given a set of horizontal segments in the plane and a (query) vertical segment  $q$ , report all intersections between  $q$  and the horizontal segments.
- (3) *Point enclosure.* Given a set of 2-ranges in the plane and a query point  $q$ , report all the 2-ranges that contain  $q$ .

Given a set  $V$  of 2-ranges and a query rectangle  $q$ , rectangle reporting can be performed by doing (1) range reporting with respect to the lower left corners of the rectangles in  $V$  and the query  $q$ , (2) orthogonal segment intersection with respect to the bottom (resp., left) sides of the rectangles and the left (resp., bottom) side of  $q$ ; (3) point enclosure with respect to the rectangles of  $V$  and the lower left corner of  $q$ .

Multiple reports caused by singular cases can be easily avoided using extra care. To complete our treatment of the static case, we simply observe that optimal data structures already exist for both the orthogonal segment intersection and the point enclosure problem (Chazelle [C1]). The size of these data structures is  $O(n)$  and their construction takes  $O(n \log n)$  time on an EPM. Queries can be answered in time  $O(k + \log n)$ , where  $k$  is the size of the output. Using our previous solutions to range reporting, we are then equipped to solve the problem at hand. The complexity is dominated by the cost of range reporting.

To deal with the dynamic version of rectangle reporting, we use a different approach. The previous sections involved a redesign of the *range tree* in linear space. We will now undertake a similar transformation with respect to the *segment tree*, a data structure due to Bentley [B1]. The idea again is to identify the functional components of the data structure and, on that basis, re-implement it completely differently.

*The dynamic case.* (1) *Orthogonal segment intersection.* For explanatory purposes, we make various simplifying assumptions. To remove them is tedious but does not affect the validity of our results. Our main assumption is that the coordinates of the segments, including the query, are all distinct. The problem to solve can be formulated as follows: let  $V$  be a set of  $n$  horizontal segments, each of the form  $(x_i, x'_i, y_i)$ , with  $x_i < x'_i$ . Given a vertical segment  $q$  compute all intersections between  $q$  and the horizontal segments.

We begin with the static version of the problem, and as a starter we introduce a little notation. The integer  $i$  is called the *index* of segment  $(x_i, x'_i, y_i)$ . For any  $j$  ( $1 \leq j \leq 2n$ ) let  $m_j$  denote the unique  $x$ -coordinate of rank  $j$  among the  $2n$  endpoints. We are now ready to define the segment tree of  $V$ . For convenience, we use a declarative definition, as opposed to a more standard procedural definition (Bentley and Wood [BW]). Let  $T$  be a  $(4n + 1)$ -node complete binary tree. For  $i = 2, \dots, 2n$ , we put the  $i$ th leaf from the left in correspondence with both the endpoint  $m_i$  and the interval  $(m_{i-1}, m_i]$ . The leftmost leaf is in correspondence with  $m_1$ . Each internal node is associated with the union of the intervals at the leaves descending from it. We label each node  $v$  of  $T$  as follows: if  $v$  is the root,  $b(v)$  is the null string; else  $b(v)$  is the string of 0's and 1's given by the left (0) and right (1) turns of the path from the root to  $v$ .

Each node  $v$  is associated with a *node-set*  $L(v)$ . The segment  $(x_i, x'_i, y_i)$  is represented in the tree  $T$  by including the index  $i$  into the node-sets of a collection of nodes

with label-set  $C_i$ . The set of indices associated with node  $v$  constitutes  $L(v)$ . Let  $l_i$  and  $r_i$  be, respectively, the leaf corresponding to  $x_i$  and the leaf immediately to the right of the one associated with  $x'_i$ . Let  $p$  be the longest common prefix of  $b(l_i)$  and  $b(r_i)$ . We can always write  $b(l_i)$  as a string  $p0a_1 \cdots a_l$  of 1's and 0's, and  $b(r_i)$  as a string  $p1b_1 \cdots b_r$ . We define

$$C_i = \{p0a_1 \cdots a_{j-1}1 | a_j = 0\} \cup \{p1b_1 \cdots b_{k-1}0 | b_k = 1\}.$$

Each interval  $(x_i, x'_i]$  is in this way *canonically* partitioned into intervals associated with nodes of  $T$ . Where our data structures for rectangle searching, which we call *S-structures*, differ from segment trees is in the encoding of node-sets. To begin with, we construct an *M-structure* (EPM version) with respect to the  $2n$  endpoints of segments in  $V$ : to break ties between the  $y$ -coordinates of the endpoints of a segment, we may agree for consistency that the left endpoint precedes its right counterpart along the  $y$ -axis. We use  $T$  as the underlying tree of the *M-structure*.

Let us examine the relationship between  $B(v)$  and  $L(v)$ . Recall that each bit of  $B(v)$  is associated with a unique endpoint  $x_j$  or  $x'_j$ . In either case we say that the bit has *index*  $j$ . Without loss of generality, assume that  $v$  is the right child of node  $z$ . Then, by construction, for each  $i$  in  $L(v)$  the endpoint  $x_i$  is associated with a leaf descending from  $z$  and therefore has a *trace* in  $B(z)$ . This allows us to encode  $L(v)$  as a bit vector  $A(v)$  of the same length as  $B(z)$ . The  $k$ th bit of  $A(v)$  is 1 if and only if the index of the  $k$ th bit of  $B(z)$  appears in  $L(v)$ . In this manner for every index in  $L(v)$  there is a unique 1 in  $A(v)$ . All the other bits of  $A(v)$  are set to 0.

Recall that  $B(z)$  appears in compact form in  $W(z)$  as a list of  $B$ -fields. We represent  $A(v)$  exactly like  $B(z)$  ( $A(v)$  and  $B(z)$  are isomorphic). Furthermore, we add pointers between the two lists to make it possible to jump directly between any  $B$ -field in  $B(z)$  into its associated  $A$ -field in  $A(v)$ . To answer a query  $(x, y, y')$  is now straightforward. We perform a binary search to identify the leaf of  $T$  whose associated interval contains  $x$  (if any). Next, we search for  $y$  and  $y'$  in the virtual list  $R(z)$  of each ancestor  $z$  of the leaf. Proceeding down from the root to the leaf, this can be readily done with the *M-structure*. After these preliminaries, we can locate both  $y$  and  $y'$  in each virtual list  $L(v)$  by jumping directly from its virtual counterpart  $R(z)$  ( $z$  is the parent of  $v$ ). Let  $\alpha_0, \alpha_1, \cdots$  be the  $A$ -fields of  $A(v)$ . The previous search leads to two bits, one in  $\alpha_i$  and the other in  $\alpha_j$  ( $i \leq j$ ). The problem is now to identify all the bits equal to 1 in each field of  $L = \{\alpha_{i+1}, \alpha_{i+2}, \cdots, \alpha_{j-1}\}$ , as well as in a certain suffix of  $\alpha_i$  and prefix  $\alpha_j$ . The latter part of the computation can be accomplished in  $O(\log^2 n)$  time per report with  $O(\log n)$  overhead. Assuming that  $L$  is not empty, retrieving the 1's among the fields of  $L$  is more difficult. We will say that  $\alpha_i$  is *vacuous* if each of its bits is set to 0. We cannot just examine the fields  $\alpha_{i+1}, \cdots, \alpha_{j-1}$ , in search of all the 1's, since we would run the risk of visiting many vacuous fields. Instead, we link together the nonvacuous  $A$ -fields. For each relevant bit found we can return to its companion in  $B(z)$  in  $O(\log n)$  time, and then proceed to identify it, which takes another  $O(\log^2 n)$  time.

We must now dynamize this static data structure. First, we describe the data structure and its invariants, and then we give an overview of the method used to insert a new horizontal segment (the case of a deletion is similar, so it will be omitted). The data structure is built from a dynamic *M-structure* (as in range reporting) augmented with the  $A$ -fields defined above. Recall that for each internal node  $z$  of  $T$ , we have a search tree  $C(z)$ , whose leaves are associated with the  $B$ -fields of  $W(z)$ . Let  $v$  be a child of  $z$ ; we define  $\mathcal{T}(v)$  to be a tree isomorphic to  $C(z)$ . The leaves of  $\mathcal{T}(v)$  are in

one-to-one correspondence with the  $A$ -fields of  $A(v)$  and, as before, these  $A$ -fields are isomorphic to the  $B$ -fields of  $B(z)$  and connected to them via pointers. Instead of linking together nonvacuous  $A$ -fields, it suffices to keep flags at each node of  $\mathcal{T}(v)$  which is the ancestor of at least one leaf corresponding to a nonvacuous  $A$ -field. With this scheme the position of each relevant bit in the nonvacuous fields at node  $v$  can be found in  $O(\log n)$  time. As before, the identification of these bits will dominate the query time, bringing it up to  $O(k \log^2 n)$ .

How do we insert a new segment? To begin with, we take the two endpoints of the segment and insert them in the underlying  $M$ -structure. This involves the addition of two new leaves in  $T$  and the updating of  $C(v)$  for each node  $v$  on the path from these leaves to the root. Note that for each such internal node and their left ( $v.l$ ) and right ( $v.r$ ) children we must update  $\mathcal{T}(v.l)$  and  $\mathcal{T}(v.r)$  to maintain the invariant of isomorphism with  $C(v)$ . Of course, the update of the flags and the  $A$ -fields will be different depending on whether  $v.l$  or  $v.r$  are nodes of the canonical decomposition of the new segment. It is a simple exercise to show that the updating can be performed in  $O(\log n)$  time per node  $v$ . Thus updating includes checking with neighboring nodes of the new leaves if previously inserted segments may have to undergo changes in their canonical decomposition because of the new insertion. Deletions are symmetric to insertions and therefore quite similar. For details on dynamic operations on segment trees, consult Edelsbrunner [Ed] and Mehlhorn [Me].

The last part of any dynamic operation is to check whether the tree  $T$  has fallen out of balance. If that is the case, we apply the method used for the  $M$ -structure. This implies finding the highest node  $v$  of  $T$  that falls out of balance and reconstructing the entire subtree below  $v$ . The key remark is that all the segments involved in the updating must have at least one endpoint identified by a leaf descending from the parent of  $v$ . Therefore the cost of rebuilding the structure is once again  $O(|H| \log n)$ , where  $H$  is the subtree of  $T$  rooted at  $v$ . The usual counting argument shows that  $O(\log^2 n)$  is an amortized upper bound on the update time.

As in the case of range reporting, we can cut down the query time to  $O(k(\log(2n/k))^2)$  by dovetailing between the method above and a slab-based technique (§ 5). Instead of a dynamic list one will now use a dynamic *interval tree* (Edelsbrunner [Ed])—see also Mehlhorn [Me]. This data structure stores  $n$  intervals on the real axis in linear storage, so that all  $k$  intervals containing an arbitrary query value can be reported in  $O(k + \log n)$  time. Edelsbrunner has shown, moreover, that insertions and deletions can be executed in  $O(\log n)$  amortized time. Using the technique of slabs developed in § 5 we immediately obtain a dynamic algorithm for orthogonal segment intersection with a complexity of  $O(n)$  space,  $O(n \log n)$  preprocessing time,  $O(\log n)$  update time and, for  $k \geq n^{2/3}$ ,  $O(k)$  query time.

*The dynamic case.* (2) *Point enclosure.* This section will not make use of the compaction technique described earlier. The basic approach will be mostly borrowed from McCreight [Mc] and Edelsbrunner [Ed]. For this reason, we will only sketch the data structures, and refer to the aforementioned sources the reader who wishes to reconstruct all the proofs in full detail.

McCreight [Mc] has given a dynamic algorithm for point enclosure with the following performance: the storage needed is  $O(n)$  and the query time is  $O(k + \log^3 n)$ , where  $k$  is the size of the output. We must extend his algorithm a little, however, because it handles only updates over a fixed universe. We assume that the reader is familiar with the concept of a *priority search tree* (McCreight [Mc]). For this reason we only briefly sketch the main features of this data structure. Given a set of  $n$  points in the plane and a fixed line  $L$ , a priority search tree allows us to report all the points

inside a query rectangle constrained to have one of its sides collinear with  $L$ . The complexity of the data structure is  $O(n)$  space,  $O(n \log n)$  preprocessing time and  $O(k + \log n)$  query time, where  $k$  is the number of points to be reported. Furthermore any point can be inserted or deleted in  $O(\log n)$  time.

Let  $V$  be a set of  $n$  2-ranges in  $\mathfrak{R}^2$ . Consider a vertical line  $L$  that separates the collection of  $2n$  vertical sides into two equal-size sets. We associate with the root of a binary tree the set of 2-ranges that intersect  $L$ , and to create the two subtrees below the root we iterate on this process with respect to the 2-ranges falling strictly to the left and strictly to the right of  $L$ . This defines a binary tree  $T$  of height  $O(\log n)$ . The reader familiar with Edelsbrunner [Ed] will recognize in  $T$  the interval tree of the projection of  $V$  on the horizontal axis. For each node  $v$  of  $T$ , clip the 2-ranges associated with  $v$ , using the dividing line at that node. This gives us a set of left ranges  $R_l(v)$  and a set of right ranges  $R_r(v)$ . We construct two similar structures with respect to these collections. We restrict our description to  $R_l(v)$ . Apply exactly the procedure just described to  $R_l(v)$  with respect to the horizontal (and not the vertical) direction. This leads to another interval tree-like structure,  $T_l(v)$ , whose nodes store lists of 2-ranges that are adjacent to both a horizontal and a vertical line. Given any point, finding the 2-ranges that contain it is equivalent to *domination searching* (Fig. 3). This problem involves preprocessing a set of points in the plane so that, for any query point  $q$ , finding which of the given points are dominated by  $q$  in both  $x$  and  $y$  coordinates can be done efficiently. But this is right up the alley of McCreight's priority search tree [Mc]. Putting all these simple observations together leads to a data structure of linear size for solving point enclosure reporting in time  $O(k + \log^3 n)$ , where  $k$  is the size of the output. Note that the cubic term comes from the fact that McCreight's structure is called upon on the order of  $\log n$  times for each level of  $T$ .

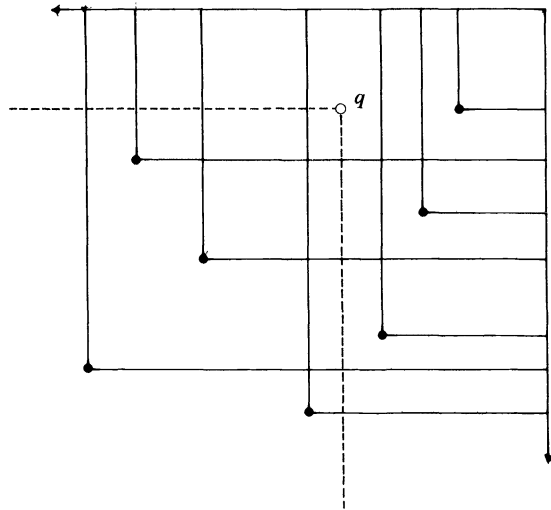


FIG. 3

Edelsbrunner [Ed] has shown that interval trees can be updated in  $O(\log n)$  amortized time. Using the fact that a priority search tree on presorted points can be computed in linear time, we can use Edelsbrunner's method almost verbatim and derive

an algorithm for updating each  $T_l(v)$  and  $T_r(v)$  in  $O(\log n)$  amortized time. A similar argument applied to  $T$  leads to an overall  $O(\log^2 n)$  update time. We omit the details. Putting all these results together we have a linear-size data structure for dynamic reporting on an EPM with  $O(\log^2 n)$  update time and  $O(k \log^2 n)$  query time. Dovetailing with the alternative algorithms will cut down this query time to the minimum of  $O(k \log^2 n + \log^3 n)$  and  $O(k + n^{2/3})$ , which gives  $O(k \log^2(2n/k) + \log^3 n)$ . Note that a similar observation can be made in the static case. Theorem 6 summarizes our results on rectangle reporting.

**THEOREM 6.** *Data structures for rectangle reporting in two dimensions can be constructed in  $O(n \log n)$  time in all cases. Let  $k - 1$  be the number of points to be reported, and let  $\epsilon$  be any real  $> 0$ . On a RAM, a tradeoff is possible: in particular, we can achieve (1)  $O(k(\log(2n/k))^\epsilon + \log n)$  query time and  $O(n)$  space; (2)  $O(k \log \log(4n/k) + \log n)$  query time and  $O(n \log \log n)$  space; (3)  $O(k + \log n)$  query time and  $O(n \log^\epsilon n)$  space. On an APM (resp., EPM), it is possible to achieve a query time of  $O(k \log(2n/k))$  (resp.,  $O(k(\log(2n/k))^2)$ ), using  $O(n)$  space. In the dynamic case on an EPM, it is possible to achieve a query time of  $O(k(\log(2n/k))^2 + \log^3 n)$  and an update time of  $O(\log^2 n)$  time, using  $O(n)$  space.*

**9. Going into higher dimensions.** Generalizing our result to any fixed dimension is straightforward. For range problems we use a classical technique of Bentley [B2]. Let  $(x_1, \dots, x_d)$  be a system of Cartesian coordinates in  $\mathbb{R}^d$ . We build a complete binary tree with respect to the  $x_d$ -coordinates of the points. Each query range is thus decomposed into  $O(\log n)$  canonical ranges, each of which can be handled by solving a range problem in  $\mathbb{R}^{d-1}$ . The technique generalizes readily for the dynamic case (see Lueker [L], Willard [W2], Willard and Lueker [WL], for example).

As regards rectangle searching, we also follow a standard technique (Edelsbrunner and Maurer [EM]). See also Mehlhorn [Me]. The *left* coordinate of a  $d$ -range refers to its smaller coordinate in dimension  $x_d$ ; its  $x_d$ -interval denotes the interval formed by projecting the  $d$ -range on the  $x_d$ -axis. Let  $R$  be a  $d$ -range and let  $x$  be its left coordinate. We define the *left* face of  $R$  to be the  $(d - 1)$ -range obtained by projecting  $R$  on the hyperplane  $x_d = x$ . A  $d$ -range intersects another if and only if one intersects the other's left face. Since the two conditions are mutually exclusive, barring equalities among coordinates, we can use this criterion for both reporting and counting purposes. The data structure consists of a range tree and segment tree built respectively with respect to the left coordinates and  $x_d$ -intervals of the input. Each node has a pointer to a  $(d - 1)$ -dimensional structure defined recursively. Once again, this generalization is sufficiently well known to avoid further elaboration.

Note that a problem in dimension  $d$  is in this way reduced to  $O(\log^{d-2} n)$  problems in two dimensions. Analyzing the complexity in the nonreporting cases is straightforward. In the reporting problems considered earlier we obtained query times of the form  $O(kf(n, k) + \log n)$ , where  $f(n, k)$  is decreasing in  $k$  and asymptotically equivalent to  $f(n, 1)$  for  $k = O(n^{3/4})$ . To be more accurate, the query times could be expressed as the minimum of  $O(kf(n, 1) + \log n)$  and  $O(k + n^{2/3})$ . This implies that in dimension  $d$  the query times  $Q(n, k)$  are of the form

$$O\left(\log^{d-2} n + \sum_{1 \leq i \leq m} \min(k_i f(n, 1) + \log n, k_i + n^{2/3})\right),$$

where  $\sum_{1 \leq i \leq m} k_i = k$  and  $m = O(\log^{d-2} n)$ . Let  $k = A + B$ , where  $A = \sum_{k_i < n^{2/3}} k_i$  and

$B = \sum_{k_i \geq n^{2/3}} k_i$ . We have

$$Q(n, k) = O(\log^{d-1} n + Af(n, 1) + B).$$

Note that  $A < mn^{2/3}$ . If  $B \geq n^{3/4}$ , then since  $m$  and  $f(n, 1)$  are at most polylogarithmic we have  $Q(n, k) = O(B) = O(k)$ . If now  $B < n^{3/4}$  we have  $k = O(n^{3/4})$ , and therefore  $Q(n, k) = O(\log^{d-1} n + kf(n, k))$ . In all cases, we have shown that  $Q(n, k) = O(kf(n, k) + \log^{d-1} n)$ .

**THEOREM 7.** *Each of the previous data structures (Theorems 1–6) can be extended to  $\mathfrak{R}^d$  ( $d > 2$ ). The complexity of the resulting data structures can be obtained by multiplying each expression in the complexity for two dimensions by a factor of  $\log^{d-2} n$ . (Note: the terms involving  $k$  remain unchanged, but an extra term  $\log^{d-1} n$  must be included in the query time).*

**10. Discussion.** We conclude with a few remarks and open problems.

The  $M$ -structures are essentially transcripts of sorting runs. We have used mergesort as our base sort but we could have turned to other methods as well. If we visualize the preprocessing played backwards and rotate the point set by 90 degrees, what we will see in action is no longer mergesort but quicksort (with median splitting). If we use radix-sort, we will see appear the possibility of tradeoffs between space and query time in higher dimensions, depending on the size of the radix. These simple observations show that the relationship between range search and sorting is rich, indeed, and deserves further investigation. An interesting open question is to determine whether there exist linear-size data structures with polylogarithmic response-time for solving range searching in *any* fixed dimension.

Our data structure for range counting in two dimensions is quite simple. It is very fast and surprisingly economical in its use of storage. Last but not least, it can be implemented with little effort. Unfortunately, this cannot be said of all the data structures in this paper. Some of our upper bounds are obtained at the price of a fairly heavy machinery. Can the algorithms be significantly simplified while keeping the same asymptotic complexity? Can their asymptotic performance be improved?

We believe that many of our bounds for dynamic multidimensional searching can be lowered if one has access to a more powerful machine than an EPM. For example, what improvements can the added power of a RAM buy us?

All the data structures in this paper rely crucially on the use of bit vectors. These have sometimes been used in previous data structures to encode sets—often as addresses in a RAM, like in (Gabow and Tarjan [GT]). Our use of bit vectors is quite different. The nodes of the  $M$ -structures, for example, store bit vectors that encode the history of a computation. This scheme is a departure from previous solutions and raises interesting questions concerning models of computation. It is evident from this work that even seemingly *minimal* models such as pointer machines with only comparison and addition can't avoid but allow exotic encodings of the sort. Indeed, it seems that only by placing very strong semantic limitations on the models of computation one might manage to invalidate these data structures. Let us observe that with a few notable exceptions (Tarjan [T], Chazelle [C2], for example) little is known on the computational power of pointer machines. Our results suggest that these might be, indeed, more powerful than expected.

**Acknowledgments.** I wish to thank the referee for several useful comments about this manuscript.

## REFERENCES

- [AHU] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [B1] J. L. BENTLEY, *Algorithms for Klee's rectangle problems*, Dept. of Comp. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, 1977, unpublished notes.
- [B2] ———, *Multidimensional divide and conquer*, Comm. ACM, 23 (1980), pp. 214-229.
- [BW] J. L. BENTLEY AND D. WOOD, *An optimal worst-case algorithm for reporting intersections of rectangles*, IEEE Trans. Comput., C-29 (1980), pp. 571-577.
- [C1] B. CHAZELLE, *Filtering search: a new approach to query-answering*, this Journal, 15 (1986), pp. 703-724.
- [C2] ———, *Lower bounds on the complexity of multidimensional searching*, Proc. 27th Annual IEEE Symposium on the Foundations of Computer Science, 1986, pp. 87-96.
- [Ea] J. EARLEY, *Toward an understanding of data structures*, Comm. ACM, 14 (1971), pp. 617-627.
- [Ed] H. EDELSBRUNNER, *Intersection problems in computational geometry*, Ph.D. thesis, Tech. Rep. F-93, Technical Univ. Graz, Graz, Austria 1982.
- [EM] H. EDELSBRUNNER AND H. A. MAURER, *On the intersection of orthogonal objects*, Inform. Process. Lett. 13 (1981), pp. 177-181.
- [EO] H. EDELSBRUNNER AND M. H. OVERMARS, *On the equivalence of some rectangle problems*, Inform. Process. Lett. 14 (1982), pp. 124-127.
- [EOS] H. EDELSBRUNNER, M. H. OVERMARS, AND R. SEIDEL, *Some methods of computational geometry applied to computer graphics*, Computer Vision, Graphics, and Image Processing, 28 (1984), pp. 92-108.
- [GBT] H. N. GABOW, J. L. BENTLEY, AND R. E. TARJAN, *Scaling and related techniques for geometry problems*, Proc. 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 135-143.
- [GT] H. N. GABOW AND R. E. TARJAN, *A linear-time algorithm for a special case of disjoint set union*, Proc. 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 246-251.
- [GoT] G. H. GONNET AND F. W. TOMPA, *A constructive approach to the design of algorithms and their data structures*, Comm. ACM, 26 (1983), pp. 912-920.
- [G] J. GUTTAG, *Abstract data types and the development of data structures*, Comm. ACM, 20 (1977), pp. 396-404.
- [HT] D. HAREL AND R. E. TARJAN, *Fast algorithms for finding nearest common ancestors*, this Journal, 13 (1984), pp. 338-355.
- [H] P. HENDERSON, *Functional Programming: Application and Implementation*, Prentice-Hall, Englewood Cliffs, NJ, 1980.
- [K1] D. E. KNUTH, *The Art of Computer Programming, Sorting and Searching*, Vol. 3, Addison-Wesley, Reading, MA, 1973.
- [K2] ———, *The Art of Computer Programming, Seminumerical Algorithms*, Vol. 2, Addison-Wesley, Reading, MA, 1981.
- [L] G. S. LUEKER, *A data structure for orthogonal range queries*, Proc. 19th Annual IEEE Symposium on the Foundations of Computer Science, 1978, pp. 28-34.
- [LP1] D. T. LEE AND F. P. PREPARATA, *Location of a point in a planar subdivision and its applications*, this Journal, (1977), pp. 594-606.
- [LP2] ———, *Computational geometry—a survey*, IEEE Trans. Comput., C-33 (1984), pp. 1072-1101.
- [Mc] E. M. MCCREIGHT, *Priority search trees*, this Journal, 14 (1985), pp. 257-276.
- [Me] K. MEHLHORN, *Data Structures and Algorithms: Vol. 3, Multidimensional Searching and Computational Geometry*, Springer-Verlag, Berlin, 1984.
- [O] M. OVERMARS, *The Design of Dynamic Data Structures*, Springer Lecture Notes in Computer Science 156, Springer-Verlag, Berlin, New York, 1983.
- [T] R. E. TARJAN, *A class of algorithms which require nonlinear time to maintain disjoint sets*, J. Comput. System Sci., 18 (1979), pp. 110-127.
- [V] J. VUILLEMIN, *A unifying look at data structures*, Comm. ACM, 23 (1980), pp. 229-239.
- [W1] D. E. WILLARD, *Reduced memory space for multi-dimensional search trees*, 2nd Annual STACS, LNCS, Springer-Verlag, Berlin, New York, 1985, pp. 363-374.
- [W2] ———, *New data structures for orthogonal range queries*, this Journal, 14 (1985), pp. 232-253.
- [WL] D. E. WILLARD AND G. S. LUEKER, *Adding range restriction capability to dynamic data structures*, J. Assoc. Comput. Mach., 32 (1985), pp. 597-617.

## PARALLEL TIME $O(\log n)$ ACCEPTANCE OF DETERMINISTIC CFLs ON AN EXCLUSIVE-WRITE P-RAM\*

PHILIP N. KLEIN† AND JOHN H. REIF‡

**Abstract.** We give an algorithm for accepting a deterministic context-free language on the P-RAM, an exclusive-write, concurrent-read model of parallel computation. Whereas on inputs of length  $n$ , a deterministic push-down automaton will use time linear in  $n$ , our algorithm runs in time  $O(\log n)$  on  $n^3$  processors. The algorithm is easily generalized to permit parallel simulation of any deterministic auxiliary pushdown automaton that uses space  $s(n) \cong \log n$  and time  $2^{O(s(n))}$ . The simulation runs in time  $O(s(n))$  on  $2^{O(s(n))}$  processors, and is nearly optimal, since we observe that any language accepted by a P-RAM in time  $T(n)$  is accepted by a deterministic auxiliary pushdown automaton in space  $T(n)$  and time  $2^{O(T(n)^2)}$ .

**Key words.** parallel algorithms, parallel computation, formal language theory, context-free language recognition

**AMS(MOS) subject classification.** 68Q20

**1. Introduction.** In this paper, we address the parallel complexity of parsing. The study of algorithms for parsing has a long history (see, e.g., [4]), but it is only fairly recently that parallel algorithms for parsing began to be investigated. Previous papers ([8], [10]) have contained parallel algorithms for recognition of general context-free languages. We give a faster parallel algorithm for recognition of deterministic context-free languages on an exclusive-write model of parallel computation. An algorithm for parsing such languages may be readily derived from our algorithm.

Parsing of deterministic context-free languages arises in program compilation. Deterministic context-free languages can be used to capture the syntax of many of the commonly used programming languages. The  $LR(k)$  grammars of [7], for example, generate exactly the deterministic context-free languages, so our algorithm may be applied to the parsing of programming languages generated by such grammars.

The deterministic context-free languages are exactly those accepted by deterministic pushdown automata, so our algorithm can effectively simulate any deterministic pushdown automaton. More generally, our algorithm can be used to simulate a deterministic auxiliary pushdown automaton, i.e., a deterministic pushdown automaton with a bounded worktape.

The model of parallel computation we use in this paper is the concurrent-read, exclusive-write P-RAM. The P-RAM is a parallel random access machine model defined by Fortune and Wyllie in [3]. It consists of a collection of synchronous deterministic unit-cost RAMs with shared memory locations indexed by the natural numbers. The P-RAM disallows concurrent writes. That is, no two processors can attempt to write into the same memory location on the same step. Concurrent reads, however, are permitted. During each step, any memory location may be read simultaneously by any number of processors.

---

\* Received by the editors March 16, 1984; accepted for publication (in revised form) February 11, 1987. A preliminary version of this paper appeared in the Proceedings of the 23rd Annual IEEE Symposium on the Foundations of Computer Science, 1982, pp. 290–296.

† Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The research of this author was supported in part by an Office of Naval Research Graduate Fellowship.

‡ Computer Science Department, Duke University, Durham, North Carolina 27706. The research of this author was supported by Office of Naval Research contract N00014-80-C-0647 and National Science Foundation grant DCR-85-03251.



Previously Fortune and Wyllie showed that any language accepted by a deterministic Turing machine with space bound  $s(n) \geq \log n$  is accepted in time  $O(s(n))$  by a P-RAM. Also, Ruzzo showed in [10] that any language accepted by a *nondeterministic* auxiliary pushdown machine with space  $s(n) \geq \log n$  and time  $2^{O(s(n))}$  can be accepted in time  $O(s(n)^2)$  by an apparently weaker parallel machine model, the uniform circuit family. In particular, he gave a parallel algorithm for recognition of general context-free languages that requires  $O(\log^2 n)$  time and  $n^6$  processors on a uniform circuit family. Although Ruzzo's algorithm requires only  $O(\log n)$  time on a more powerful parallel model in which concurrent writes are permitted, there seems no way to take advantage of the more limited power of the exclusive-write P-RAM to achieve  $O(\log n)$  time for recognition of an arbitrary context-free language.

Our contribution is to show that, if a context-free language is in fact deterministic, then  $O(\log n)$  time recognition is indeed possible on a P-RAM. We give a very simple algorithm to solve this problem—so simple, it might be practical to implement. Its proof, however, is rather complicated. This is perhaps understandable in view of the fact that our result implies Cook's result [2] that a deterministic context-free language can be recognized in  $O(\log^2 n)$  space and polynomial time on a Turing machine. We also show as a corollary to our main result that a P-RAM can simulate a *deterministic*  $s(n)$  space-bounded,  $2^{O(s(n))}$  time-bounded auxiliary pushdown automaton in parallel time  $O(s(n))$  with  $2^{O(s(n))}$  processors.

One further difference between Ruzzo's algorithm and ours may be of greater practical significance: the processor bounds. Ruzzo's algorithm for context-free language recognition seems to require  $n^6$  processors to achieve  $O(\log^2 n)$  parallel time on a uniform circuit family. However, if the language is *deterministic* context-free, our algorithm can recognize it in  $O(\log n)$  time on only  $n^3$  processors of a P-RAM. Moreover, our algorithm can be modified [6] to run on only  $n^2 \log n$  processors (but in  $O(\log^2 n)$  time).

The form of the paper is as follows: in § 2, we describe a well-known parallel algorithm for simulating a deterministic space-bounded Turing machine; the algorithm and its proof serve to prepare the ground for our algorithm for simulating a deterministic push-down automaton (DPDA). In § 3, we describe some assumptions we make of the simulated DPDA in order to simplify the presentation of the algorithm. In § 4, we introduce the notion of a surface configuration, and define some notation that will be useful in the algorithm's proof of correctness. In § 5, we present the parallel algorithm itself, and in § 6, we explain the algorithm and prove its correctness. In § 7, we remark on improving the processor bound of our algorithm. In § 8, we observe that the P-RAM algorithm may be generalized to simulate any deterministic auxiliary pushdown automaton. Finally, in § 9, we observe that our simulation of § 8 is almost optimal, since there is a comparable simulation of P-RAMs by deterministic auxiliary pushdown automata.

**2. Simulating a deterministic space-bounded Turing machine.** To motivate the algorithm for simulating a deterministic pushdown automaton, we first discuss the algorithm for a simpler related case, in which the simulated automaton does not possess a stack. A careful reading of the proof of correctness for this latter algorithm will aid the reader in understanding the proof of correctness for the former algorithm.

Let  $M$  be an  $S(n)$  space-bounded deterministic Turing machine, where  $S(n) = \Omega(\log n)$ . We may assume that  $M$  is  $T(n)$  time-bounded, where  $T(n) = 2^{O(S(n))}$ . We give an algorithm for simulating  $M$  that takes  $O(\log T(n))$  time on  $2^{O(S(n))}$  processors of a P-RAM. This is a well-known simulation due to Fortune and Wyllie [3].

We will call an encoding of the Turing machine's finite state, worktape contents, and tape head positions a *configuration*. Let  $\mathcal{C}_\omega$  denote the set of configurations of the Turing machine on input  $\omega$ . It is easy to see that the number of configurations  $|\mathcal{C}_\omega|$  is  $2^{O(S(n))}$ . Let  $c_0, c_{\text{accept}} \in \mathcal{C}_\omega$  be the initial and accepting configurations. Let  $\vdash \subset \mathcal{C}_\omega \times \mathcal{C}_\omega$  be the next move relation, and let  $\vdash^+$  and  $\vdash^*$  be its transitive and reflexive-transitive closures, respectively. For any nonnegative integer  $i$ , let  $\vdash^i$  be the  $i$ -fold composition of  $\vdash$  (reachability in exactly  $i$  steps). We assume that there is no next move from the unique accepting configuration  $c_{\text{accept}}$ .

We shall simulate the Turing machine  $M$  in  $O(\log T(n))$  steps, using one processor for each configuration. The algorithm consists of  $\lceil \log_2 T(n) \rceil + 1$  stages. At each stage, every processor inspects the values of a constant number of memory locations (each containing a configuration) and stores a configuration into another memory location. The memory is represented by a collection of arrays  $FAR_k$  ( $k = 0, 1, \dots, \lceil \log_2 T(n) \rceil$ ) of configurations, indexed by configurations. The  $k$ th stage of the algorithm ( $k = 0, 1, \dots, \lceil \log_2 T(n) \rceil$ ) computes, for each configuration  $c$ , the configuration  $FAR_k[c]$  such that  $c \vdash^{2^k} FAR_k[c]$ . (If there is no configuration  $2^k$  steps from  $c$ , then  $FAR_k[c]$  is the configuration that is as many steps from  $c$  as possible.) After the last stage, we can examine the configuration  $c' = FAR_{\lceil \log T(n) \rceil}[c_0]$  to determine if  $c' = c_{\text{accept}}$ . If in fact  $c'$  is the accepting configuration, we may conclude that the automaton  $M$  accepted its input. Conversely, if  $M$  eventually enters the accepting configuration  $c_{\text{accept}}$  when started in the initial configuration  $c_0$ , then it does so in at most  $T(n)$  steps. Hence  $FAR_k[c_0] = c_{\text{accept}}$ .

The algorithm for computing  $FAR_k[c]$  for each configuration  $c$  and each  $k = 0, 1, \dots, \lceil \log_2 T(n) \rceil$  is as follows:

ALGORITHM: *TM-SIM*( $M$ ).

Input: a string  $\omega \in \Sigma^n$

Output: *YES* if  $M$  accepts  $\omega$ , *NO* otherwise.

- 1 To initialize (stage 0),
- 2 for all configurations  $c$  in parallel,
- 3 set  $FAR_0[c] := \begin{cases} c' & \text{if } c \vdash c', \\ c & \text{else.} \end{cases}$
- 4 For each subsequent stage  $k+1$  ( $k = 0, 1, \dots, \lceil \log_2 T(n) \rceil$ ),
- 5 for all configurations  $c$  in parallel,
- 6 set  $FAR_{k+1}[c] := FAR_k[FAR_k[c]]$ .
- 7 Finally, output *YES* if  $FAR_k[c_0] = c_{\text{accept}}$ ,
- 8 *NO* otherwise.

Intuitively, the algorithm works by composing two computations of length  $2^k$  to get a single computation of length  $2^{k+1}$ . Figure 1 illustrates this process.

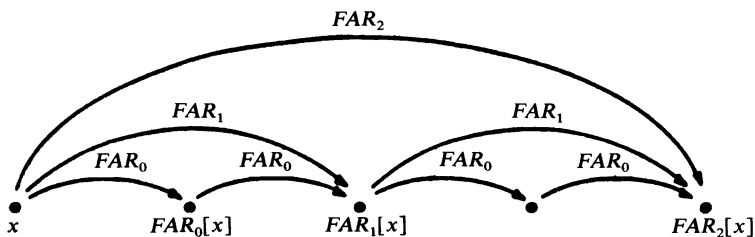


FIG. 1. Three stages of the Turing machine simulation.

The proof of correctness is a simple induction; we give it in detail in order to prepare the ground for similar proofs that are to come.

DEFINITION. Let  $c$  be a configuration, and suppose another configuration  $c'$  reachable from  $c$  is either

- at least  $2^k$  steps from  $c$ , or
- the last configuration reachable from  $c$ .

Then we say that  $c'$  is  $k$ -far from  $c$ .

THEOREM 1. For each configuration  $c$ , for each  $k = 0, 1, \dots, \lceil \log_2 T(n) \rceil$ ,  $FAR_k[c]$  is  $k$ -far from  $c$  (correctness of  $FAR_k$ ).

*Proof of Theorem 1 by induction on  $k$ .* The basis  $k = 0$  is easy to check. Suppose the theorem holds for  $k$ , and let  $c$  be any configuration. In line 6, the algorithm sets  $FAR_{k+1}[c] = FAR_k[FAR_k[c]]$ . Let  $c' = FAR_k[c]$ , so  $FAR_{k+1}[c] = FAR_k[c']$ . By the induction hypothesis,  $c'$  is  $k$ -far from  $c$  and  $FAR_k[c']$  is  $k$ -far from  $c'$ . We have immediately that  $c \vdash^* c'$  and  $c' \vdash^* FAR_k[c']$ , hence  $c \vdash^* FAR_k[c']$ .

If  $c'$  is the last configuration reachable from  $c$ , then  $FAR_k[c'] = c'$ , so  $FAR_k[c']$  is also the last configuration reachable from  $c$ . In this case, certainly  $FAR_k[c']$  is  $(k+1)$ -far from  $c$ , and we are done. Therefore, assume that  $FAR_k[c']$  is not the last configuration reachable from  $c$ . It follows that  $FAR_k[c']$  is also not the last configuration reachable from  $c'$ .

Because  $c'$  is  $k$ -far from  $c$  (but not the last configuration reachable from  $c$ ), we must conclude that  $c'$  is at least  $2^k$  steps from  $c$ . Similarly,  $FAR_k[c']$  is at least  $2^k$  steps from  $c'$ , so  $FAR_k[c']$  is at least  $2^k + 2^k = 2^{k+1}$  steps from  $c$ , and we are done.  $\square$

**3. A normal form for deterministic pushdown automata.** This section defines a normal form for deterministic pushdown automata, and proves that every deterministic context-free language is accepted by some pushdown automaton in normal form. In § 5, we will give an algorithm for simulating any pushdown automaton in normal form. Throughout this paper, we will denote the length of a string  $s$  by  $|s|$ , and the empty string by  $\epsilon$ . We denote the concatenation of strings  $s_1$  and  $s_2$  by  $s_1 \circ s_2$ . A *prefix* of a string  $s$  is an initial substring of  $s$ .

We say a deterministic pushdown automaton is in *normal form* if it satisfies the following assumptions:

*Assumptions.*

(1) All pushes and advances of the input head occur together; a step of  $M$  involves a push if and only if the head advances on that step.

(2)  $M$  accepts by a unique final state, and  $M$  has no move from a configuration with a final state.

(3) The only stack operations are to push one symbol or to erase the top element (pop).

(4) Whether  $M$ 's move from a configuration involves a push or pop does not depend on the current stack contents; it depends only on the current state and the input symbol currently being scanned.

(5) If  $M$ 's move from a configuration does not involve a pop, then the new *state* also does not depend on the stack contents.

(6)  $M$ 's initial configuration has an empty stack.

There is an apparent paradox. Let  $c$  be any configuration of  $M$  with a nonempty stack, and let  $c'$  be the corresponding configuration with an empty stack (i.e., same state, same input head position). By our assumption (4), if  $M$ 's move from  $c$  involves a pop, then  $M$ 's move from  $c'$  involves a pop—but  $c'$  has an empty stack, so a pop is impossible! We resolve this conflict by requiring that if  $M$ 's move from  $c$  involves a pop, then  $M$  has *no* move from  $c'$ , and we say that  $M$  is *stuck* at  $c'$ . Intuitively,  $M$

“tried” to pop at  $c'$ , but discovered that its stack was empty, forcing  $M$  to freeze. This situation will arise in our algorithm to simulate  $M$ .

Note that assumption 1 ensures that the stack height never grows by more than  $n$ , the size of the input.

The following lemma ensures that we can make the above assumptions of  $M$  without loss of generality.

**LEMMA 1.** *For any deterministic context-free language  $L$ , there is a deterministic pushdown automaton  $M$  satisfying assumptions (1), (2), (3), (4), (5), and (6) that accepts the language  $L\$$  ( $L\$ = \{w\$ \mid w \in L\}$ , where  $\$$  is a special symbol).*

*Proof.* For any DCFL  $L$ , there is an  $LR(0)$  grammar  $G$  generating  $L\$$ . Construct a DPDA  $M_1$  that accepts  $L\$$  as described in Theorem 10.10 of [5]; the automaton  $M_1$  is a shift-reduce parser for  $L\$$ , and can be constructed so as to push a single symbol onto the stack for each input symbol it consumes—and never push otherwise. Thus  $M_1$  satisfies assumption (1). Moreover,  $M_1$  is constructed so as to pop at most one symbol from the stack in a single move. Also,  $M_1$  can be made to accept by a unique final state, and we can easily ensure that  $M_1$  has no transitions from its final state. Thus  $M_1$  satisfies assumption (2). Next, construct a DPDA  $M_2$  that simulates  $M_1$  but keeps the top symbol of  $M_1$ 's stack in its finite control. The DPDA  $M_2$  need never look at its stack except to update the stored stack symbol in the event that it pops. Thus  $M_2$  satisfies assumptions (3), (4), and (5). Because the initial configuration of  $M_1$  had a stack of one symbol, and  $M_2$  stores one stack symbol in its finite control, the initial configuration of  $M_2$  has an empty stack. Thus  $M_2$  satisfies assumption (6).  $\square$

**4. Surface configurations.** The technique of § 2 could be directly applied to the simulation of deterministic pushdown automata. A configuration of a pushdown automaton includes the contents of its stack, however, and the stack may be  $n$  symbols high. This means that the number of configurations of a PDA could be exponential. Recall that in the algorithm of § 2, there is a processor for each configuration. In order to obtain even a subexponential processor bound for simulating a PDA, let alone a polynomial bound, we must therefore take a somewhat different approach. The question therefore arises: can we do without stack information? Surprisingly, the answer is yes. Through a more sophisticated approach to computing the  $FAR_k$  values, we can in fact make do with impoverished configurations that contain only stack height information, rather than the entire stack contents. The reason for this is the limited way in which a pushdown automaton accesses its stack. Suppose that our pushdown automaton enters the configuration  $c$  at some point during its computation, and consider the subcomputation starting from  $c$  and continuing until the stack height first goes below its height at  $c$ . We can simulate this subcomputation without knowing the stack at  $c$ , and we need only keep track of those stack symbols in excess of the stack at  $c$  to carry out this simulation. (We call  $c$  the *base* of the subcomputation.) Note, moreover, that we can apply the same technique to the subcomputation. By repeatedly applying this technique to simulate subcomputations, we can simulate the entire computation in such a way that we never need to associate with a given configuration more than one symbol of its stack. Our algorithm is in fact a dynamic-programming approach to carrying out such a simulation.

Our algorithm manipulates *surface configurations* instead of full configurations. A surface configuration is like a full configuration, but lacks information about stack contents.

Let  $M$  be a deterministic pushdown automaton in normal form with input alphabet  $\Sigma$ , state set  $Q$ , and stack alphabet  $\Gamma$ . Fix an input  $\omega_1 \cdots \omega_n \in \Sigma^n$ . For our purposes, a surface configuration  $x$  will include  $M$ 's current state  $q(x) \in Q$ , the current position

of the input head  $p(x) \in \{1, \dots, n+1\}$ , and a stack height parameter  $h(x)$ , a value between 0 and  $n$ . (By assumptions (1) and (6), the stack of  $M$  never exceeds  $n$  in height.) Let  $X_n = Q \times \{1, \dots, n+1\} \times \{0, \dots, n\}$  be the set of surface configurations associated with an input of length  $n$ . The number of such surface configurations is  $|Q|(n+1)^2$ , which is  $O(n^2)$  for a fixed pushdown automaton. A full configuration is obtained from a surface configuration by appending the stack contents. For  $s \in \Gamma^*$ , the pair  $(x, s)$  represents the configuration of  $M$  when it is in state  $q(x)$ , with input head position  $p(x)$ , and stack  $s$  (where the last symbol of  $s$  is the top of the stack).

We let  $\vdash$  denote the next-move relation:  $(x, s) \vdash (x', s')$  holds if  $M$  moves from state  $q(x)$  with input head position  $p(x)$  and stack  $s$  to state  $q(x')$  with input head position  $p(x')$  and stack  $s'$ , and the following technical condition is satisfied:

$$(1) \qquad h(x') - h(x) = |s'| - |s|.$$

Condition (1) guarantees that changes in the stack height parameter  $h(\cdot)$  reflect changes in stack height. Note that we do not require of a configuration  $(x, s)$  that  $h(x)$  be the actual height of the stack  $s$ . Indeed, in simulating the subcomputations out of which we shall construct the full computation, we find it useful to associate with a single surface configuration  $x$  a (possibly) different stack for each subcomputation in which  $x$  occurs. For example, suppose  $(x, s)$  is the base of a subcomputation in our simulation. It suffices to carry out the simulation of the subcomputation from the configuration  $(x, \varepsilon)$ . Condition 1 is a compromise that allows us this flexibility, while still enabling us to track the stack height over the course of a single subcomputation.

By a simple induction, we obtain:

LEMMA 2 (Stack Height Lemma). *For any  $x, x' \in X$  and  $s, s' \in \Gamma^*$ , if  $(x, s) \vdash^*(x', s')$  then  $|s'| - |s| = h(x') - h(x)$ .  $\square$*

Because the input head is one-way, we have

$$(2) \qquad \text{if } (x, s) \vdash^*(y, s') \text{ then } p(y) \cong p(x).$$

Because we assumed that each push is accompanied by an advance of the input head (assumption (1)), we have

$$(3) \qquad \text{if } (x, s) \vdash^*(y, s') \text{ and } |s'| > |s|, \text{ then } p(y) > p(x).$$

One might well ask: given that surface configurations lack stack contents information, what use are they? We would like to define a next-move relation, analogous to  $\vdash$ , between surface configurations, but without the addition of stack contents, such a relation would not be well defined. However, there is a way to finesse this difficulty. We will define a separate next-move relation  $\vdash_u$  for each surface configuration  $u$ . Before defining this notation, we must prove a lemma which ensures its well definedness.

LEMMA 3 (Uniqueness Lemma). *If  $(u, s) \vdash^*(x, s_1)$  and  $(u, s) \vdash^*(x, s_2)$  then  $s_1 = s_2$  (i.e.,  $s_1$  is uniquely determined by  $u, s$ , and  $x$ ).*

*Proof.* Either  $(x, s_1) \vdash^*(x, s_2)$  or  $(x, s_2) \vdash^*(x, s_1)$ . Suppose without loss of generality that  $(x, s_1) \vdash^*(x, s_2)$ . By the Stack Height Lemma,  $|s_1| = |s_2|$ . If any intermediate configuration  $(y, s_3)$  had  $|s_3| > |s_1|$  then, by (2) and (3), we would have  $p(x) \cong p(y) > p(x)$ , a contradiction. Similarly  $|s_3| < |s_1|$  leads to a contradiction. Hence  $|s_3| = |s_1|$ , i.e., every intermediate configuration has stack height equal to  $|s_1|$ . But since  $M$  never changes the stack without pushing or popping, we may conclude  $s_1 = s_2$ .  $\square$

DEFINITION. For any surface configuration  $u$ , we define the relation  $\vdash_u$  on surface configurations as follows:

$$x \vdash_u y \text{ if for some } s, s' \in \Gamma^*, (u, \varepsilon) \vdash^*(x, s) \vdash (y, s').$$

By the Uniqueness Lemma,  $u$  and  $x$  together determine  $s$ ; similarly,  $u$  and  $y$  determine  $s'$ . Thus when we write  $x \vdash_u y$ , the stack contents associated with  $x$  and  $y$  are implicit. We call  $u$  the *base* of the computation  $x \vdash_u y$ , and say that, *relative* to the base  $u$ ,  $M$  moves from  $x$  to  $y$ . The relations  $\vdash_u^*$ ,  $\vdash_u^+$ , and  $\vdash_u^i$  for  $i \geq 0$  are defined analogously.

Intuitively,  $x \vdash_u^* y$  is an assertion that there is a computation passing through  $u$ ,  $x$ , and  $y$ , in that order, in which the stack height never goes below the stack height at  $u$ .

We illustrate a possible computation  $x \vdash_u^* y$  in Fig. 2. This diagram charts the changes in stack height of the automaton  $M$  as it proceeds through the computation from  $(u, \varepsilon)$  to  $(x, s)$  to  $(y, s')$ . We use such diagrams throughout this paper. They may be viewed as plots of stack height versus number of steps taken, with the base appearing as the origin in the bottom left corner of the diagram. Figures 2 and 3 also show the stack contents associated with the labelled surface configurations.

For any surface configuration  $u$ , we define the predicate  $LAST_u(\cdot)$  on surface configurations as follows:

$LAST_u(x)$  holds iff  $u \vdash_u^* x$  but there is no surface configuration,  $y$  satisfying  $x \vdash_u y$ .

That is,  $LAST_u(x)$  holds if  $x$  is the last surface configuration reachable from  $(u, \varepsilon)$ .

Remarks.

(1)  $x \vdash_x^* y$  means simply that  $(x, \varepsilon) \vdash^*(y, s)$  for some  $s$  (determined, of course, by  $x$  and  $y$ ).

(2)  $x \vdash_u y$  implies that  $h(x) \geq h(u)$  (by the Stack Height Lemma, since  $(u, \varepsilon) \vdash^*(x, s)$  for some  $s$ ). Similarly,  $h(y) \geq h(u)$ .

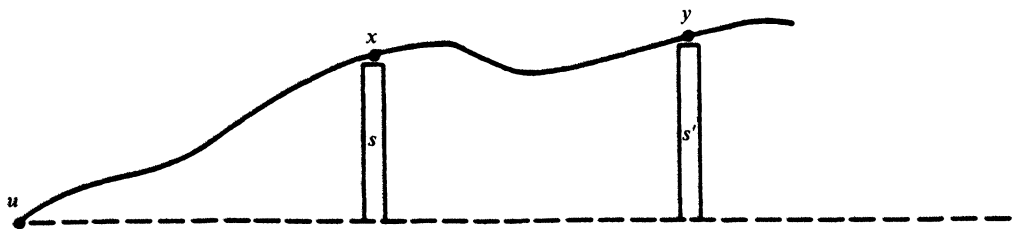


FIG. 2.  $x \vdash_u^* y$ .

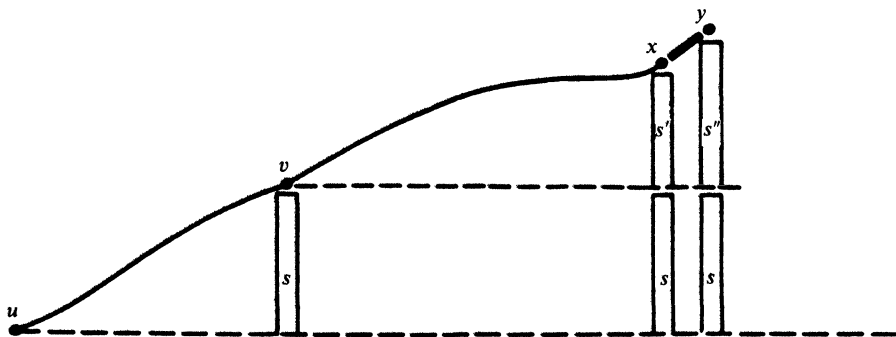


FIG. 3.  $x \vdash_v y$  implies  $x \vdash_u y$ .

(3) Suppose that  $(u, \varepsilon) \vdash^*(v, s)$ . Then, as illustrated in Fig. 3, we have that  $x \vdash_v y$  implies  $x \vdash_u y$  as follows (note the change in base):  $x \vdash_v y$  means that  $(v, \varepsilon) \vdash^*(x, s') \vdash (y, s'')$  for some  $s', s'' \in \Gamma^*$ . But then, by the definition of a pushdown automaton,  $(v, s) \vdash^*(x, s \circ s') \vdash (y, s \circ s'')$ . By combining the computations  $(u, \varepsilon) \vdash^*(v, s)$  and  $(v, s) \vdash^*(x, s \circ s') \vdash (y, s \circ s'')$ , we obtain  $(u, \varepsilon) \vdash^*(x, s \circ s') \vdash (y, s \circ s'')$ , which is to say  $x \vdash_u y$ .

We strengthen and generalize these remarks in the following lemma and its corollary.

**LEMMA 4 (Stack Lemma).** *Suppose  $u \vdash_u^* v$  and  $v \vdash_v^* x$ . Then for any surface configuration  $y$  and nonnegative integer  $a$ ,  $x \vdash_v^a y$  if and only if*

- $x \vdash_u^a y$ , and
- $h(w) \geq h(v)$  for each  $w$  such that  $x \vdash_u^* w \vdash_u^* y$ .

*Proof.* The *only-if* direction is a straightforward application of the definitions and the Stack Height Lemma. The argument follows the reasoning of remarks (2) and (3) above. The *if* direction is proved by induction on  $a$ . The case  $a = 0$  is trivial. Assume that  $x \vdash_u^a y \vdash_u z$ , so  $x \vdash_u^{a+1} z$ , and that  $h(w) \geq h(v)$  for each  $w$  such that  $x \vdash_u^* w \vdash_u^* y$ .

By the induction hypothesis, we have  $x \vdash_v^a y$ . We need only show that  $y \vdash_v z$  follows from  $y \vdash_u z$ . We know that for some stack  $s_y$ ,  $(v, \varepsilon) \vdash^*(y, s_y)$ , and that for some stack  $s$ ,  $(u, \varepsilon) \vdash^*(v, s)$ . Hence  $(u, \varepsilon) \vdash^*(v, s) \vdash^*(y, s \circ s_y)$  by definition of a pushdown automaton.

Now,  $y \vdash_u z$  means that  $(y, s \circ s_y) \vdash (z, s')$ . Recall that our pushdown automaton  $M$  never looks at its stack unless it is about to pop a symbol. Moreover, even if  $M$  is about to pop, it only looks at the top symbol of its stack. Thus the configurations  $(y, s \circ s_y)$  and  $(y, s_y)$  look the same to  $M$  unless  $M$  is about to pop and  $s_y = \varepsilon$ .

It follows that either  $(y, s_y) \vdash (z, s'')$  for some stack  $s''$ , or  $M$ 's move from  $(y, s \circ s_y)$  is a pop and  $s_y = \varepsilon$ . In the first case,  $y \vdash_v z$ , and we are done. In the second case,  $h(z) < h(y)$  (because the move from  $y$  to  $z$  involves a pop), and  $h(y) = h(v)$  (because  $(v, \varepsilon) \vdash^*(y, \varepsilon)$ ). We must conclude that  $h(z) < h(v)$ , which is a contradiction.  $\square$

**COROLLARY.** *Suppose  $u \vdash_u^* v \vdash_u^* x$  and  $x \vdash_v^* y$ . Then the surface configurations making up the computation  $x \vdash_u^* y$  are exactly the surface configurations making up the computation  $x \vdash_v^* y$ . Formally, any  $w$  satisfying  $x \vdash_u^* w \vdash_u^* y$  also satisfies  $x \vdash_v^* w \vdash_v^* y$ .*

*Proof.* Suppose  $w$  satisfies  $x \vdash_u^* w \vdash_u^* y$ . By the Stack Lemma, since  $x \vdash_v^* y$ , we have  $h(w') \geq h(v)$  for every  $w'$  such that  $x \vdash_v^* w' \vdash_v^* y$ . This includes all  $w'$  such that  $x \vdash_u^* w' \vdash_u^* w$ , so we have  $x \vdash_v^* w$  by the *if* direction of the Stack Lemma. We can similarly obtain  $w \vdash_v^* y$ , so  $x \vdash_v^* w \vdash_v^* y$ , and we are done.  $\square$

**5. An algorithm for simulating a deterministic PDA.** In this section, we give the algorithm for simulating a deterministic pushdown automaton  $M$  that is in normal form.

Let  $M$  be a deterministic pushdown automaton in normal form with input alphabet  $\Sigma$ , state set  $Q$ , start state  $q_0$ , accepting state  $q_{\text{accept}}$ , and stack alphabet  $\Gamma$ . Let  $x_0$  be the surface configuration with  $q(x_0) = q_0$ ,  $p(x_0) = 1$ , and  $h(x_0) = 0$ . Thus  $(x_0, \varepsilon)$  is the starting configuration. Let  $X_n = Q \times \{1, \dots, n+1\} \times \{0, \dots, n\}$  be the set of surface configurations for an input of length  $n$ .

Let  $T(n) = 2|Q|(n+1)$ . We claim that if  $M$  accepts an input of length  $n$ , it does so in at most  $T(n)$  steps. The automaton  $M$  cannot go more than  $|Q|$  steps without pushing, popping, or looping. Moreover,  $M$  cannot push more than  $n$  times, because each push is accompanied by an advance of the input head. Finally,  $M$  cannot pop more times than it pushes, so the claim follows.

Our algorithm will simulate  $T(n)$  steps of  $M$  in  $\lceil \log T(n) \rceil + 1 = O(\log n)$  stages. It makes use of the arrays  $FAR_k$ ,  $HOP_k$ , and  $LOW_k$  for  $k = 0, 1, \dots, \lceil \log_2 T(n) \rceil$ .

In direct analogy with the algorithm of § 2 for simulating an arbitrary deterministic automaton, the algorithm for simulating a pushdown automaton consists of stages  $k = 0, 1, \dots, \lceil \log_2 T(n) \rceil$ . At stage  $k$ , for each surface configuration  $x$ , the array element  $FAR_k[x]$  is a surface configuration such that  $x \vdash_x^* FAR_k[x]$ . After stage  $\lceil \log T(n) \rceil$ , in lines 21–22 the algorithm inspects  $FAR_{\lceil \log T(n) \rceil}[x_0]$  and determines whether it is an accepting configuration. If so, the algorithm concludes that the pushdown automaton  $M$  accepts its input, and outputs YES. Otherwise, it outputs NO.

ALGORITHM: *DPDA-SIM*( $M$ ).

Input: a string  $w \in \Sigma^n$

Output: YES if  $M$  accepts  $w$ , NO otherwise.

- 1 To initialize (stage 0),
- 2 for all surface configurations  $x \in X_n$  in parallel,
- 3 set  $FAR_0[x] := \begin{cases} y & \text{if } (x, \varepsilon) \vdash (y, \sigma), \\ x & \text{if } LAST_x(x), \end{cases}$
- 4 set  $HOP_0[x] := \begin{cases} FAR_0[x] & \text{if } h(FAR_0[x]) = h(x), \\ x & \text{else.} \end{cases}$

Note that  $h(x) \leq h(FAR_0[x]) \leq h(x) + 1$ .

- 5 for all  $x, y \in X_n$  such that  $h(x) \leq h(y) \leq h(FAR_0[x])$  in parallel,
- 6 if  $M$  is stuck at  $(x, \varepsilon)$  then set  $LOW_0[x, y] := x$
- 7 else let  $\sigma$  satisfy  $(x, \varepsilon) \vdash (FAR_0[x], \sigma)$ ,
- 8 and let  $s = \begin{cases} \sigma & \text{if } h(y) = h(FAR_0[x]), \\ \varepsilon & \text{if } h(y) < h(FAR_0[x]) \end{cases}$
- 9 If  $M$  is stuck at  $(y, s)$  then set  $LOW_0[x, y] := y$
- 10 else let  $z, s'$  satisfy  $(y, s) \vdash (z, s')$
- 11 and set  $LOW_0[x, y] := \begin{cases} z & \text{if } h(z) \leq h(y), \\ y & \text{else.} \end{cases}$

(See Figs. 4 and 5 for examples.)

- 11 For each subsequent stage  $k + 1$  ( $k = 0, 1, \dots, \lceil \log_2 T(n) \rceil$ ),
- 12 for all configurations  $x \in X_n$  in parallel,
- 13 let  $\hat{x}$  denote  $LOW_k[x, FAR_k[x]]$
- 14 set  $FAR_{k+1}[x] := FAR_k[\hat{x}]$ .
- 15 set  $HOP_{k+1}[x] := \begin{cases} \hat{x} & \text{if } h(\hat{x}) = h(x), \\ HOP_k[x] & \text{else} \end{cases}$
- 16 For all  $x, y \in X_n$ ,

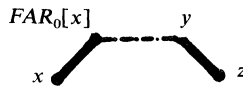


FIG. 4.  $LOW_0[x, y] = z$  in case  $h(z) \leq h(y)$ .

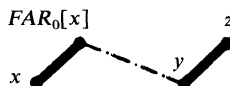


FIG. 5.  $LOW_0[x, y] = y$  in case  $h(z) > h(y)$ .



17 let  $LHL_{k+1}[x, y]$  denote  $LOW_k[x, HOP_{k+1}[LOW_k[x, y]]]$   
 18 For all  $x, y \in X_n$  such that  $h(x) \leq h(y) \leq h(FAR_{k+1}[x])$  in parallel,  
 19 set  $LOW_{k+1}[x, y] := \begin{cases} LHL_{k+1}[x, y] & \text{if } h(y) \leq h(\hat{x}), \\ LHL_{k+1}[x, z] & \text{if } h(y) > h(\hat{x}) \text{ and } h(z) = h(\hat{x}), \\ z & \text{else} \end{cases}$   
 20 where  $z = LHL_{k+1}[\hat{x}, y]$ .  
 21 Finally, output YES if  $q(FAR_{\lceil \log T(n) \rceil}[x_0]) = q_{\text{accept}}$   
 22 NO otherwise.

Note that in lines 5–10 and lines 18–20, not all locations in the arrays  $LOW_k$  are assigned values. Specifically,  $LOW_k[x, y]$  is only assigned a value if  $h(x) \leq h(y) \leq h(FAR_k[x])$ . The other locations in these arrays need not be assigned values, as they have no effect on the eventual output.

**6. Proof of correctness of the algorithm.** We now proceed to explain and prove the algorithm.

There are two fundamental ideas that make the algorithm work. One idea we have already seen in action: that of composing two computations to obtain a computation of twice the length. This was the fundamental idea in the algorithm of § 2 for simulating a deterministic automaton, and it recurs several times in the algorithm for simulating a pushdown automaton. It is by no means a new idea; it has played a role in many proofs, e.g., that of Savitch's theorem. The second idea, introduced in § 4, is that of considering computations between surface configurations relative to a base surface configuration.

To show that the algorithm is correct, we prove that the arrays  $FAR_k$ ,  $HOP_k$ , and  $LOW_k$  satisfy certain *correctness conditions* for  $k = 0, 1, \dots, \lceil \log_2 T(n) \rceil$ . Then the correctness of the algorithm will follow from the fact that the array  $FAR_{\lceil \log T(n) \rceil}$  satisfies its correctness condition. The proof is by induction on  $k$ : assuming that  $FAR_k$ ,  $HOP_k$ , and  $LOW_k$  satisfy their correctness conditions, we show that  $FAR_{k+1}$ ,  $HOP_{k+1}$ , and  $LOW_{k+1}$  satisfy their correctness conditions. The algorithm and proof also make use at each stage  $k$  of a temporary array,  $LHL_{k+1}$ . The complicated dependencies in the algorithm are reflected in a complicated proof. To aid the reader in following the proof, we provide in Fig. 6 a diagram illustrating the dependencies between the correctness conditions for the various arrays.

**DEFINITION.** Let  $x$  be a surface configuration. We say that another surface configuration  $y$  reachable from  $(x, \varepsilon)$  is *k-far* from  $x$  if

- either  $y$  is at least  $2^k$  steps from  $(x, \varepsilon)$  (i.e.,  $x \vdash_x^i y$  for some  $i \geq 2^k$ ),
- or  $y$  is the last surface configuration reachable from  $(x, \varepsilon)$  (i.e.,  $LAST_x(y)$  holds).

The *correctness condition* for  $FAR_k$  is:

for all  $x \in X_n$ ,  $FAR_k[x]$  is  $k$ -far from  $x$ .

The essential idea in computing  $FAR_{k+1}$  from  $FAR_k$  is combining two  $k$ -far computations to obtain a  $(k+1)$ -far computation. There is a subtlety involved, however. Suppose that, as illustrated in Fig. 7,  $x \vdash_x^{2^k} y$  and  $LAST_y(z)$ , but  $z$  is fewer than  $2^k$  steps from  $y$ . It happens in this case that  $y$  is  $k$ -far from  $x$  and  $z$  is  $k$ -far from  $y$ , but  $z$  is not  $k$ -far from  $x$ . The reason that  $LAST_y(z)$  holds is that the computation from  $(y, \varepsilon)$  reached the configuration  $(z, \varepsilon)$ , from which the next move involved popping a symbol from the stack. The stack is empty, however—there is nothing to pop! As pointed out in § 3, the pushdown automaton  $M$  is stuck at such a configuration. We

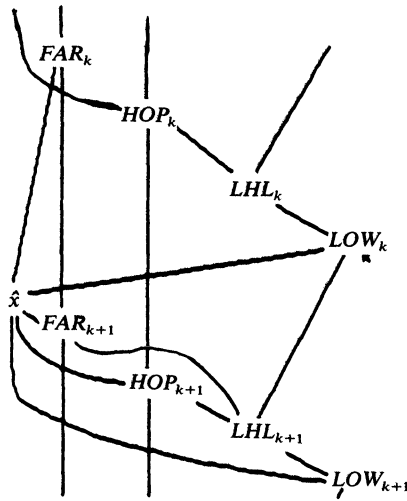


FIG. 6. Dependencies between correctness conditions for the arrays.

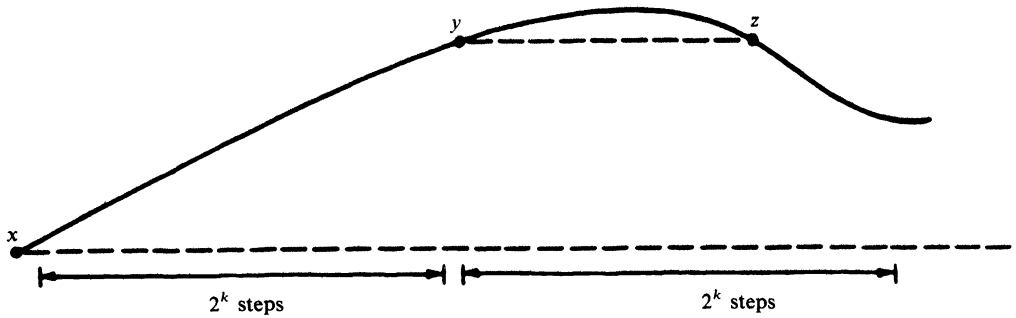


FIG. 7.  $y$  is  $k$ -far from  $x$ , and  $z$  is  $k$ -far from  $y$ , but  $z$  is not  $(k+1)$ -far from  $x$ .

see that from the perspective of the base  $y$ , the surface configuration  $z$  is an impassable barrier. Viewed from the perspective of the base  $x$ , however,  $z$  is not a barrier. When the automaton  $M$  starts at  $(x, \epsilon)$  it arrives at  $z$  with a nonempty stack. It can then pop a symbol without any fuss, and continue with the computation. Thus  $z$  is a discriminating barrier. It obstructs the computation from  $(y, \epsilon)$  but permits the computation from  $(x, \epsilon)$  to continue unimpeded.

Such discriminating barriers can only arise when the pushdown automaton  $M$  has a disposition to pop a symbol but is prevented from doing so by an empty stack. This is the idea underlying the following lemma.

LEMMA 5 (Barrier Lemma). *Suppose  $x \vdash_x^* z$  and  $y \vdash_y^* z$ . If  $LAST_y(z)$  but not  $LAST_x(z)$  then  $h(z) = h(y)$ .*

*Proof.* Since  $x \vdash_x^* z$  and not  $LAST_x(z)$ , there is a surface configuration  $v$  such that  $z \vdash_x v$ . Since  $y \vdash_y^* z$  (so  $h(z) \geq h(y)$ ) but not  $z \vdash_y v$ , by the Stack Lemma we obtain  $h(v) < h(y)$ . But stack height can change by at most one in a single move, so  $h(v) \geq h(z) - 1$ . It follows that  $h(z) \leq h(v) + 1 \leq h(y)$ . We must conclude that  $h(z) = h(y)$ .  $\square$

The Barrier Lemma enables us to render barriers innocuous, by a careful choice of the surface configuration  $y$ .

DEFINITION. Let  $x, x', y \in X_n$ . We say that  $y$  is  $k$ -low from  $x'$  with respect to the base  $x$  if

1.  $x' \vdash_x^* y$ , and
2. for any  $w$  satisfying  $y \vdash_x^+ w$ , if  $x' \vdash_x^i w$  for some  $i \leq 2^k$ , then  $h(w) > h(y)$ .

We may at times leave the base unspecified if it is clear from the context which base is intended.

Figure 8 depicts a computation in which  $y$  is  $k$ -low from  $x'$  with respect to  $x$ . The intuition is that the stack height of  $y$  is as low as is possible within  $2^k$  steps of  $x'$ . Note, however, that  $y$  may in fact be  $2^k$  or more steps from  $x'$ , in which case (2) is trivially satisfied. Note, moreover, that (2) is again trivially satisfied if  $y$  is the last surface configuration reachable from  $x$ . For example, in Fig. 9, the automaton  $M$  is stuck at  $(y, \epsilon)$ , so  $y$  is  $k$ -low from  $x'$  with respect to  $x$ .

The following lemma will show how we can achieve our aim of combining two  $k$ -far computations to obtain a computation that is  $(k+1)$ -far.

LEMMA 6 (Correctness of  $FAR_{k+1}$ ). Suppose that for the surface configuration  $x$ ,

- (1)  $FAR_k[x]$  is  $k$ -far from  $x$ ;
- (2)  $\hat{x}$  is  $k$ -low from  $FAR_k[x]$  with respect to  $x$ ;
- (3)  $FAR_k[\hat{x}]$  is  $k$ -far from  $\hat{x}$ .

As in line 14 of the algorithm, let  $FAR_{k+1}[x] = FAR_k[\hat{x}]$ . Then  $FAR_{k+1}[x]$  is  $(k+1)$ -far from  $x$ . (See Fig. 10.)

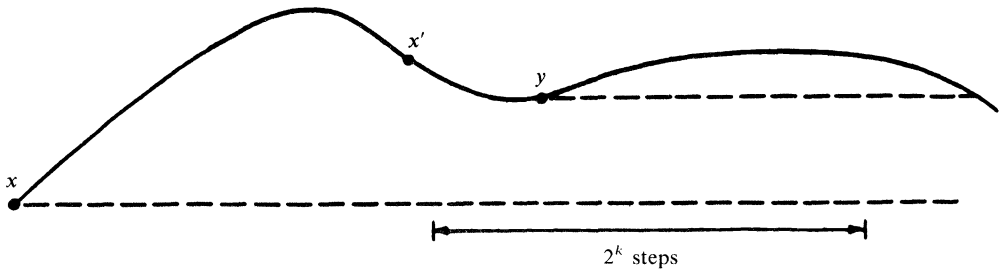


FIG. 8.  $y$  is  $k$ -low from  $x'$  with respect to  $x$ .

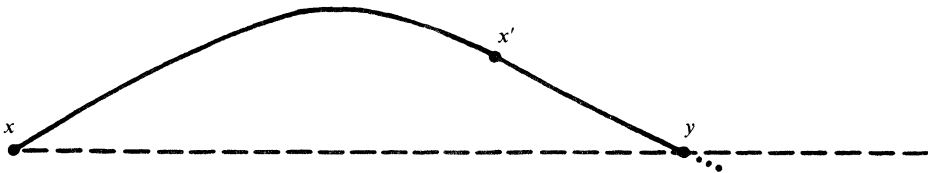


FIG. 9.  $y$  is  $k$ -low from  $x'$  with respect to  $x$  because  $M$  is stuck at  $(y, \epsilon)$ .

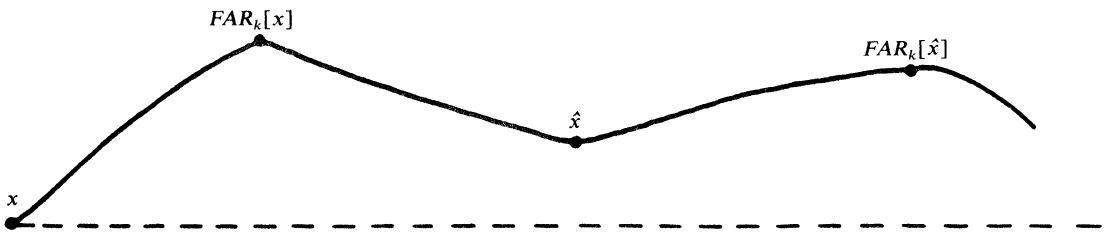


FIG. 10.  $x \vdash_x^* FAR_k[x] \vdash_x^* \hat{x} \vdash_x^* FAR_k[\hat{x}] = FAR_{k+1}[x]$ .

*Proof.* It follows from (2) by definition that  $x \vdash_x^* \hat{x}$ . It follows from (3) that  $\hat{x} \vdash_x^* FAR_k[\hat{x}]$ , so  $\hat{x} \vdash_x^* FAR_k[\hat{x}] = FAR_{k+1}[x]$  by the Stack Lemma. Combining these two computations yields  $x \vdash_x^* FAR_{k+1}[x]$ .

Now if both the computations  $x \vdash_x^* FAR_k[x]$  and  $\hat{x} \vdash_x^* FAR_k[\hat{x}]$  are at least  $2^k$  steps in length, then certainly the computation  $x \vdash_x^* FAR_k[\hat{x}]$  is at least  $2^{k+1}$  steps, and hence  $FAR_k[\hat{x}]$  is  $(k+1)$ -far from  $x$ . Therefore, we can assume that one of these computations is fewer than  $2^k$  steps in length.

If the computation  $x \vdash_x^* FAR_k[x]$  is fewer than  $2^k$  steps in length, then by (1) it must be that  $LAST_x(FAR_k[x])$  holds. That is,  $FAR_k[x]$  is the last surface configuration reachable from  $(x, \epsilon)$ . But  $\hat{x}$  and  $FAR_k[\hat{x}]$  are also reachable from  $(x, \epsilon)$ , so it must be that  $\hat{x} = FAR_k[\hat{x}] = FAR_k[x]$ . Then  $LAST_x(FAR_k[\hat{x}])$ , so  $FAR_k[\hat{x}]$  is  $(k+1)$ -far from  $x$ .

Now assume that the computation  $x \vdash_x^* FAR_k[x]$  is at least  $2^k$  steps in length, and the computation  $\hat{x} \vdash_x^* FAR_k[\hat{x}]$  is fewer than  $2^k$  steps. By (3), it must be that  $LAST_{\hat{x}}(FAR_k[\hat{x}])$  holds. If in addition  $LAST_x(FAR_k[\hat{x}])$  holds, then  $FAR_k[\hat{x}]$  is  $(k+1)$ -far from  $x$ , and we are done. If  $LAST_x(FAR_k[\hat{x}])$  does not hold, then  $FAR_k[\hat{x}]$  must be a barrier. By the Barrier Lemma,  $h(FAR_k[\hat{x}]) = h(\hat{x})$ . But then, by (2),  $FAR_k[\hat{x}]$  must be more than  $2^k$  steps from  $FAR_k[x]$ , hence more than  $2^{k+1}$  steps from  $x$ , and we are done.  $\square$

Lemma 6 shows that if we can compute a  $k$ -low surface configuration  $\hat{x}$  for each surface configuration  $x$  at each stage  $k$ , we can then compute the  $FAR_{k+1}$  values and thereby carry out the DPDA simulation. The remaining problem, then, is to compute  $\hat{x}$ . We find it necessary to solve a more general problem in order to make possible the computation of a new value of  $\hat{x}$  at each stage. We first introduce yet another new term.

DEFINITION. We say that  $y$  is  $k$ -valid with respect to  $x$  if

- $FAR_k[x] \vdash_x^* y$ , and
- For each  $w$  satisfying  $FAR_k[x] \vdash_x^* w \vdash_x^* y$ , we have  $h(w) \geq h(y)$ .

That is, as Fig. 11 illustrates, there is a computation from  $FAR_k[x]$  to  $y$ , and no configuration in this computation has stack height below that at  $y$ .

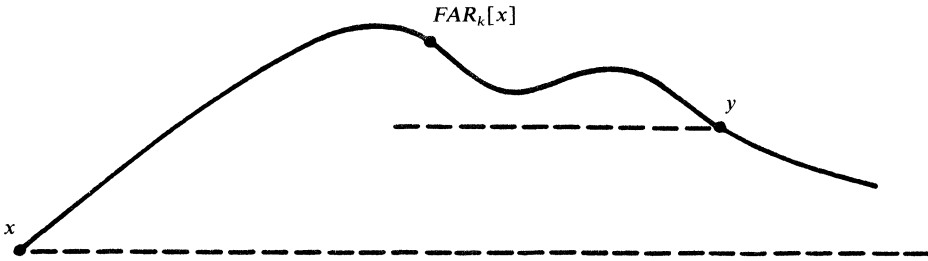


FIG. 11.  $y$  is  $k$ -valid with respect to  $x$

The point of this definition is that if  $y$  is  $k$ -valid with respect to  $x$ , then the stack at  $y$  is a prefix<sup>1</sup> of the stack at each previous configuration, all the way back to  $FAR_k[x]$ . In the course of the computation from  $FAR_k[x]$  to  $y$ , the stack height never went below its height at  $y$ , and so the pushdown automaton  $M$  never had an opportunity to modify these lowest symbols of the stack. This is a consequence of our assumption that the automaton  $M$  does not change the top element of the stack.

<sup>1</sup> Recall that we represent a stack as a string with the top element of the stack being the last symbol of the string.

The choice of the term “ $k$ -valid” is motivated by the following: for any surface configuration  $x$ , the  $y$ 's that are “valid” second indices in  $LOW_k[x, \cdot]$  are those  $y$ 's that are  $k$ -valid; if  $y$  is not  $k$ -valid, then  $LOW_k[x, y]$  is meaningless.

The following lemma shows that  $k$ -validity is preserved under computations starting and ending with empty stacks.

LEMMA 7. *If  $y$  is  $k$ -valid with respect to  $x$ , and  $(y, \varepsilon) \vdash^*(y', \varepsilon)$ , then  $y'$  is also  $k$ -valid with respect to  $x$ .*

This is most easily seen by inspecting Fig. 12 and applying the definitions. A rather detailed proof follows:

*Proof.* By definition of  $k$ -validity,  $FAR_k[x] \vdash_x^* y$  and for any  $w$  such that  $FAR_k[x] \vdash_x^* w \vdash_x^* y$ ,  $w$  satisfies  $h(w) \geq h(y)$ . We have that  $(y, \varepsilon) \vdash^*(y', \varepsilon)$ , so by the Stack Lemma:

- (a)  $y \vdash_x^* y'$ , and
- (b)  $h(w) \geq h(y)$  for each  $w$  such that  $y \vdash_x^* w \vdash_x^* y'$ .

By combining the computations  $FAR_k[x] \vdash_x^* y$  and  $y \vdash_x^* y'$ , we obtain  $FAR_k[x] \vdash_x^* y'$ . Now suppose  $w$  satisfies  $FAR_k[x] \vdash_x^* w \vdash_x^* y'$ . There are two possibilities: either  $FAR_k[x] \vdash_x^* w \vdash_x^* y$  or  $y \vdash_x^* w \vdash_x^* y'$ . In the first case,  $h(w) \geq h(y)$  by the  $k$ -validity of  $y$ . In the second case,  $h(w) \geq h(y)$  by (b). In either case, since  $h(y') = h(y)$  by the Stack Height Lemma, we have  $h(w) \geq h(y')$ , so we may conclude that  $y'$  is  $k$ -valid with respect to  $x$ .  $\square$

To make it possible to compute  $\hat{x}$  for each  $x \in X_n$ , we compute a surface configuration  $LOW_k[x, y]$  for pairs  $x, y \in X_n$ . We will inductively prove the following *correctness condition for  $LOW_k$* :

- For all  $x, y \in X_n$ ,
- if  $(\dagger)$   $y$  is  $k$ -valid with respect to  $x$ ,
- then  $LOW_k[x, y]$  is
  - (A)  $k$ -valid with respect to  $x$ , and
  - (B)  $k$ -low from  $y$  with respect to  $x$ .

This means that if  $(\dagger)$  holds, then, as Fig. 13 illustrates,  $LOW_k[x, y]$  has stack height so low that (A) there is no configuration between  $FAR_k[x]$  and  $y$  having stack height

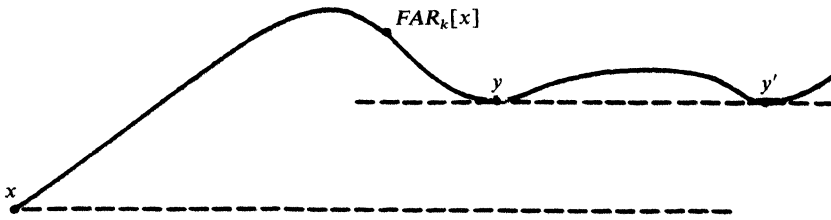


FIG. 12. *If  $y$  is  $k$ -valid with respect to  $x$  and  $(y, \varepsilon) \vdash^*(y', \varepsilon)$ , then  $y'$  is also  $k$ -valid with respect to  $x$ .*

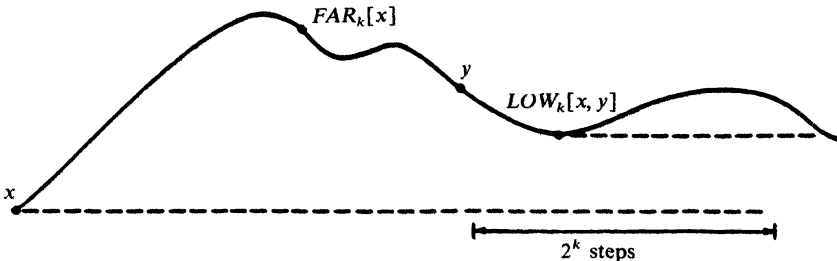


FIG. 13.  *$LOW_k[x, y]$  is  $k$ -valid and  $k$ -low from  $y$  with respect to  $x$ .*

lower, and (B) there is no configuration after  $LOW_k[x, y]$  and within  $2^k$  steps of  $y$  having stack height even as low as that at  $LOW_k[x, y]$ .

In particular, consider  $LOW_k[x, FAR_k[x]]$ . Noting that  $FAR_k[x]$  itself is certainly  $k$ -valid with respect to  $x$ , we see that  $LOW_k[x, FAR_k[x]]$  is  $k$ -low from  $FAR_k[x]$  with respect to  $x$ . This condition is exactly what Lemma 6 requires of  $\hat{x}$ , suggesting that we let  $\hat{x}$  be  $LOW_k[x, FAR_k[x]]$ . This is, in fact, how  $\hat{x}$  is obtained, as the reader can check by inspecting line 13 of the algorithm. As a consequence, it turns out that  $\hat{x}$  is  $k$ -valid in addition to being  $k$ -low from  $FAR_k[x]$ . In what follows, as a notational convenience, we assume for all  $x \in X_n$  that  $\hat{x}$  denotes a surface configuration such that

- (I)  $x \vdash_x^* FAR_k[x] \vdash_x^* \hat{x}$ ;
- (II)  $\hat{x}$  is  $k$ -valid and  $k$ -low from  $FAR_k[x]$  with respect to  $x$ ;
- (III)  $FAR_{k+1}[x] = FAR_k[\hat{x}]$ .

Note that a consequence of (I) is that  $\hat{x} \vdash_{\hat{x}}^* FAR_k[\hat{x}]$ .

LEMMA 8. Any  $w \in X_n$  satisfying  $FAR_k[x] \vdash_x^* w \vdash_x^* FAR_{k+1}[x]$  also satisfies  $h(w) \cong h(\hat{x})$ .

*Proof.* There are two possibilities for  $w$ . Either (1)  $FAR_k[x] \vdash_x^* w \vdash_x^* \hat{x}$  or (2)  $\hat{x} \vdash_x^* w \vdash_x^* FAR_{k+1}[x]$ . In case (1),  $h(\hat{x}) \cong h(w)$  follows from (II). In case (2), using (III), (I), and the corollary to the Stack Lemma, we see that  $\hat{x} \vdash_{\hat{x}}^* w \vdash_{\hat{x}}^* FAR_{k+1}[x]$  (note the change in base). But then  $h(w) \cong h(\hat{x})$  by the Stack Height Lemma.  $\square$

The computation of the  $LOW_{k+1}[x, y]$  values makes use of an array  $HOP_k$ . For each surface configuration  $x$ ,  $HOP_k[x]$  is a surface configuration such that  $(x, \varepsilon) \vdash^*(HOP_k[x], \varepsilon)$ . The reason for the name ‘‘HOP’’ should be evident from Fig. 14.

The array  $HOP_k$  satisfies the following *correctness condition*:

For all  $x \in X_n$ ,

Either  $HOP_k[x]$  is some surface configuration  $w$  such that

$$x \vdash_x^+ w \vdash_x^* FAR_k[x] \text{ and } h(w) = h(x),$$

or, if there is no such  $w$ , then  $HOP_k[x] = x$ .

In either case, we have that  $(x, \varepsilon) \vdash^*(HOP_k[x], \varepsilon)$ . Lemma 7 shows that consequently  $HOP_k[x]$  preserves the  $k'$ -validity of  $x$ , for any  $k'$ . That is, if  $x$  is  $k'$ -valid with respect to some surface configuration  $u$ , then  $HOP_k[x]$  is also  $k'$ -valid with respect to  $u$ .



FIG. 14.  $HOP_k[x]$ .

The next lemma shows that the algorithm inductively ensures the correctness of  $HOP_k$ .

LEMMA 9 (Correctness of  $HOP_k$ ). Suppose that  $HOP_k$  satisfies its correctness condition. As in line 15 of the algorithm, let

$$HOP_{k+1}[x] := \begin{cases} \hat{x} & \text{if } h(\hat{x}) = h(x), \\ HOP_k[x] & \text{else.} \end{cases}$$

Then  $HOP_{k+1}$  satisfies its correctness condition.

Figures 15 and 16 illustrate the two cases in the definition of  $HOP_{k+1}[x]$ .

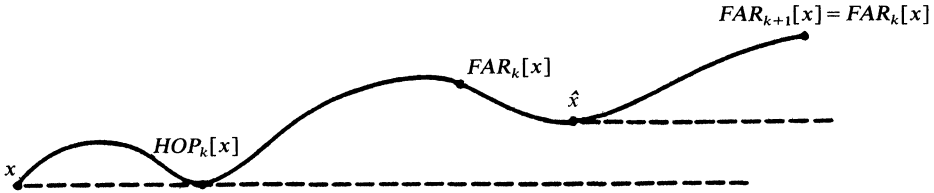


FIG. 15.  $HOP_{k+1}[x] = HOP_k[x]$  in case  $h(\hat{x}) > h(x)$ .

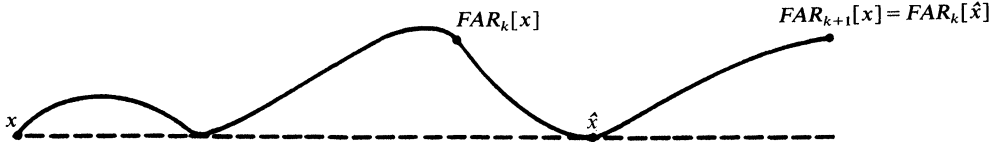


FIG. 16.  $HOP_{k+1}[x] = \hat{x}$  in case  $h(\hat{x}) = h(x)$ .

*Proof.* We must show that either  $HOP_{k+1}[x]$  is some surface configuration  $w$  such that  $x \vdash_x^+ w \vdash_x^* FAR_{k+1}[x]$  and  $h(w) = h(x)$ , or, if there is no such  $w$ ,  $HOP_{k+1}[x] = x$ . If  $h(\hat{x}) = h(x)$ , then by (II),  $\hat{x}$  is such a  $w$ , so setting  $HOP_{k+1}[x] = \hat{x}$  satisfies the correctness condition for  $HOP_{k+1}[x]$ . Suppose therefore, that  $h(\hat{x}) > h(x)$ . In this case  $HOP_{k+1}[x]$  is defined to be  $HOP_k[x]$ . Now, by our assumption of the correctness of  $HOP_k$ , either  $HOP_k[x]$  is some surface configuration  $w$  such that  $x \vdash_x^+ w \vdash_x^* FAR_k[x]$  and  $h(w) = h(x)$ , or, if there is no such  $w$ ,  $HOP_k[x] = x$ . In the former case,  $w$  certainly satisfies  $x \vdash_x^+ w \vdash_x^* FAR_{k+1}[x]$ , because  $FAR_k[x] \vdash_x^* FAR_{k+1}[x]$ . Consider therefore the latter case:  $HOP_k[x] = x$  because there is no  $w$  satisfying both  $x \vdash_x^+ w \vdash_x^* FAR_k[x]$  and  $h(w) = h(x)$ . Note that by Lemma 8 there is also no  $w$  satisfying both  $FAR_k[x] \vdash_x^* w \vdash_x^* FAR_{k+1}[x]$  and  $h(w) = h(x)$ , so setting  $HOP_{k+1}[x] = x$  satisfies the correctness condition for  $HOP_{k+1}[x]$ .  $\square$

We now consider how the algorithm computes the  $LOW_{k+1}$  values from the  $LOW_k$  values. The computation of the  $LOW_{k+1}$  values makes frequent use of the expression  $LOW_k[x, HOP_{k+1}[LOW_k[x, y]]]$ . For the sake of brevity, therefore, we denote this expression by  $LHL_{k+1}[x, y]$ . Here the name “LHL” is intended to signify that  $LHL_{k+1}[x, y]$  represents a composition of  $LOW_k$ ,  $HOP_{k+1}$ , and again  $LOW_k$ . The reason that this expression is so useful in deriving the  $LOW_{k+1}$  values from the  $LOW_k$  values is that it satisfies a condition “halfway” between the condition on  $LOW_k[x, y]$  and the condition on  $LOW_{k+1}[x, y]$ . Recall the correctness condition for  $LOW_k$ :

- For all  $x, y \in X_n$ ,  
 if  $y$  is  $k$ -valid with respect to  $x$ ,  
 then  $LOW_k[x, y]$  is  
 (A)  $k$ -valid with respect to  $x$ , and  
 (B)  $k$ -low from  $y$  with respect to  $x$ .

The correctness condition for  $LHL_{k+1}$  is:

- For all  $x, y \in X_n$ ,  
 if  $y$  is  $k$ -valid with respect to  $x$ ,  
 then  $LHL_{k+1}[x, y]$  is  
 (A)  $k$ -valid with respect to  $x$ , and  
 (B)  $(k + 1)$ -low from  $y$  with respect to  $x$ .

Notice that the only difference between the condition on  $LOW_k[x, y]$  and that on  $LHL_{k+1}[x, y]$  is that the latter promises  $(k + 1)$ -lowness, while the former can promise only  $k$ -lowness.

LEMMA 10 (Correctness of  $LHL_{k+1}$ ). *As in line 17 of the algorithm, let  $LHL_{k+1}[x, y] = LOW_k[x, HOP_{k+1}[LOW_k[x, y]]]$ . Then the correctness conditions for  $LOW_k$ ,  $FAR_{k+1}$ , and  $HOP_{k+1}$  imply the correctness condition for  $LHL_{k+1}$ .*

*Proof.* Let  $y' = LOW_k[x, y]$  and  $z = LOW_k[x, HOP_{k+1}[y']]$ , so  $LHL_{k+1}[x, y] = z$ . Assume that  $y$  is  $k$ -valid with respect to  $x$ . By the correctness of  $LOW_k[x, y]$ , it follows that  $y'$  is  $k$ -low from  $y$  and  $k$ -valid with respect to  $x$ . By Lemma 7 and the correctness of  $HOP_{k+1}$ , it follows from the latter that  $HOP_{k+1}[y']$  is also  $k$ -valid. Then, by the correctness of  $LOW_k$ ,  $z = LOW_k[x, HOP_{k+1}[y']]$  is also  $k$ -valid. It remains only to show that  $z$  is  $(k + 1)$ -low from  $y$ .

We distinguish two cases, corresponding to the two cases in the correctness condition for  $HOP_{k+1}[y']$ .

*Case 1.*  $HOP_{k+1}[y']$  is some surface configuration  $w$  such that  $h(w) = h(y')$  and  $y' \vdash_{y'}^+ w$ . (This case is illustrated in Fig. 17.) We know that  $y'$  is  $k$ -low from  $y$ , so  $h(HOP_{k+1}[y']) = h(y')$  implies that  $HOP_{k+1}[y']$  must be more than  $2^k$  steps from  $y$ . Now, consider  $z = LOW_k[x, HOP_{k+1}[y']]$ . Since  $z$  is  $k$ -low from  $HOP_{k+1}[y']$ , and  $HOP_{k+1}[y']$  is at least  $2^k$  steps after  $y$ , it follows that  $z$  is  $(k + 1)$ -low from  $y$ , and we are done.

*Case 2.* There is no  $w$  satisfying both  $h(w) = h(y')$  and  $y' \vdash_{y'}^+ w \vdash_{y'}^* FAR_{k+1}[y']$ . (In this case,  $HOP_{k+1}[y'] = y'$ , and so  $z = LOW_k[x, y']$ . See Fig. 18.) There are two possibilities: either  $FAR_{k+1}[y'] = y'$  (i.e.,  $LAST_{y'}(y)$  holds), or  $h(FAR_{k+1}[y']) > h(y')$ . We consider these two cases separately.

*Subcase A.*  $LAST_{y'}(y')$  holds. If in addition  $LAST_x(y')$  holds, then  $z = LOW_k[x, y'] = y'$ , and  $z$  is trivially  $k$ -low, since there is no  $w$  such that  $z \vdash_x^+ w$ . Assume therefore that  $LAST_x(y')$  does not hold, and let  $y''$  be the surface configuration such

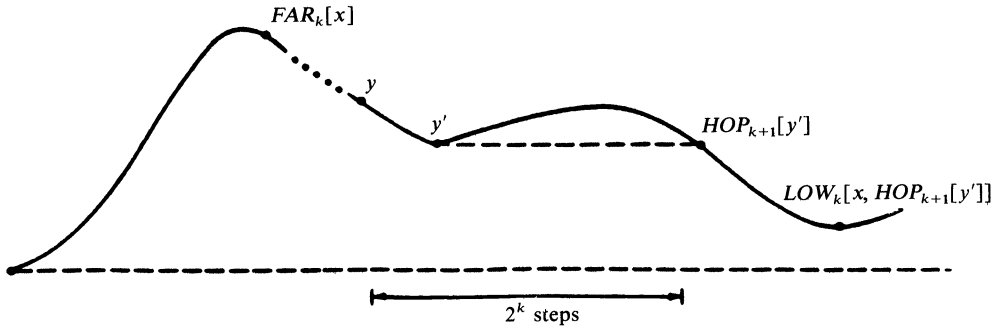


FIG. 17.  $LHL_{k+1}[x, y] = LOW_k[x, HOP_{k+1}[y']]$ , in the case where  $y' \vdash_{y'}^+ HOP_{k+1}[y']$ .

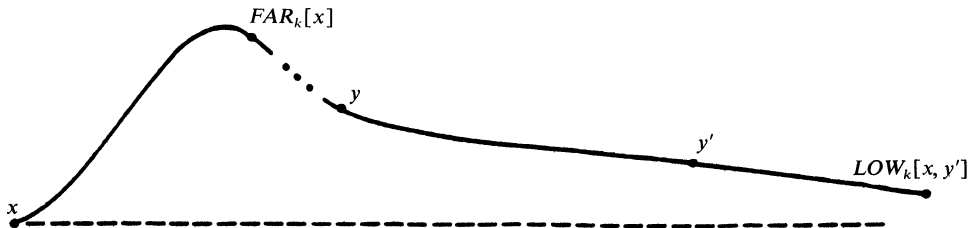


FIG. 18.  $LHL_{k+1}[x, y] = LOW_k[x, y']$ , in the case where  $HOP_{k+1}[y'] = y'$ .



that  $y' \vdash_x y''$ . By the Stack Lemma,  $h(y'') < h(y')$ . But then by the  $k$ -lowness of  $y'$ ,  $y''$  is more than  $2^k$  steps after  $y$ . Hence  $y'$  is at least  $2^k$  steps after  $y$ . As in Case 1, since  $z$  is  $k$ -low from  $y'$  and  $y'$  is at least  $2^k$  steps after  $y$ , it follows that  $z$  is  $(k+1)$ -low from  $y$ .

*Subcase B.*  $h(FAR_{k+1}[y']) > h(y')$ . By the Barrier Lemma, it follows that  $LAST_{y'}(FAR_{k+1}[y'])$  implies  $LAST_x(FAR_{k+1}[y'])$ . (Note change in base.) But then by correctness of  $FAR_{k+1}[y']$  (Lemma 6), either  $FAR_{k+1}[y']$  is at least  $2^{k+1}$  steps from  $y'$ , or  $LAST_x(FAR_{k+1}[y'])$ . In either case it follows that every surface configuration  $w$  within  $2^{k+1}$  steps of  $y'$  satisfies  $y' \vdash_{y'}^+ w \vdash_{y'}^* FAR_{k+1}[y']$  and hence  $h(w) > h(y')$ . Since  $z = LOW_k[x, y']$  is  $k$ -low from  $y'$ , certainly  $h(z) \leq h(y')$ . It follows that  $z$  is  $(k+1)$ -low from  $y'$ , so  $z$  is certainly  $(k+1)$ -low from  $y$ , and the proof is complete.  $\square$

We finally show how to compute the  $LOW_{k+1}$  values. The correctness condition for  $LOW_{k+1}$  is:

- For all  $x, y \in X_n$ ,
- if  $y$  is  $(k+1)$ -valid with respect to  $x$
- then  $LOW_{k+1}[x, y]$  is
- (A)  $(k+1)$ -valid with respect to  $x$ , and
- (B)  $(k+1)$ -low from  $y$  with respect to  $x$ .

Note that the only difference between the condition on  $LOW_{k+1}[x, y]$  and that on  $LHL_{k+1}[x, y]$  is that  $LOW_{k+1}[x, y]$  assumes (and delivers)  $(k+1)$ -validity, whereas  $LHL_{k+1}[x, y]$  assumes (and delivers)  $k$ -validity.

We can make use of  $k$ -validity by noting special circumstances under which  $(k+1)$ -validity implies  $k$ -validity, and vice versa.

LEMMA 11. *Suppose that  $y$  is  $(k+1)$ -valid with respect to  $x$ , and  $h(y) \leq h(\hat{x})$ . Then  $y$  is  $k$ -valid with respect to  $x$ .*

*Proof.* See Fig. 19. Since  $FAR_k[x] \vdash_x^* FAR_{k+1}[x]$ , and  $FAR_{k+1}[x] \vdash_x^* y$  by the  $(k+1)$ -validity of  $y$ , it follows that  $FAR_k[x] \vdash_x^* y$ . It remains to be shown that for any  $w \in X_n$  such that  $FAR_k[x] \vdash_x^* w \vdash_x^* y$ ,  $w$  satisfies  $h(w) \geq h(y)$ .

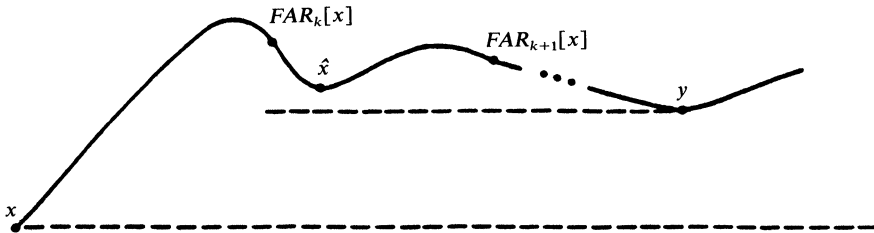


FIG. 19.  $y$  is  $k$ -valid with respect to  $x$ .

There are two cases: either  $FAR_k[x] \vdash_x^* w \vdash_x^* FAR_{k+1}[x]$  or  $FAR_{k+1}[x] \vdash_x^* w \vdash_x^* y$ . In the former case,  $h(w) \geq h(\hat{x})$  by Lemma 8, so  $h(w) \geq h(y)$  by our assumption that  $h(y) \leq h(\hat{x})$ . In the latter case,  $h(w) \geq h(y)$  follows directly from the  $(k+1)$ -validity of  $y$ .  $\square$

LEMMA 12. *Suppose that  $y$  is  $(k+1)$ -valid with respect to  $x$  and  $h(y) \geq h(\hat{x})$ . Then  $y$  is  $k$ -valid with respect to  $\hat{x}$ .*

*Proof.* See Fig. 20. Consider the computation  $FAR_{k+1}[x] \vdash_x^* y$ . By the  $(k+1)$ -validity of  $y$ , this computation consists only of surface configurations  $w$  such that  $h(w) \geq h(y) \geq h(\hat{x})$ . Moreover, recall that  $\hat{x} \vdash_{\hat{x}}^* FAR_k[\hat{x}] = FAR_{k+1}[x]$  (note that the base is  $\hat{x}$ ). It follows by the Stack Lemma that  $FAR_k[\hat{x}] \vdash_{\hat{x}}^* y$ . It remains to show that any surface configuration  $w$  satisfying  $FAR_k[\hat{x}] \vdash_{\hat{x}}^* w \vdash_{\hat{x}}^* y$  has  $h(w) \geq h(y)$ . But any

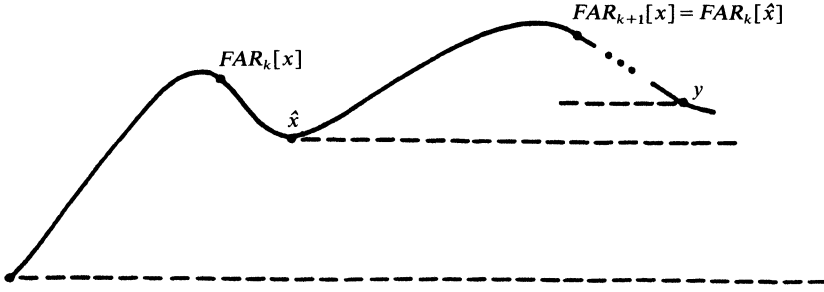


FIG. 20.  $y$  is  $k$ -valid with respect to  $\hat{x}$ .

such  $w$  must also satisfy  $FAR_k[\hat{x}] \vdash_x^* w \vdash_x^* y$  by the corollary to the Stack Lemma, and hence  $h(w) \geq h(y)$  by the  $(k+1)$ -validity of  $y$ .  $\square$

LEMMA 13. Suppose that  $y$  is  $k$ -valid with respect to  $x$ , and  $FAR_{k+1}[x] \vdash_x^* y$ . Then  $y$  is  $(k+1)$ -valid with respect to  $x$ .

Proof. All we need to show is that for any surface configuration  $w$  such that  $FAR_{k+1}[x] \vdash_x^* w \vdash_x^* y$ , we have  $h(w) \geq h(y)$ . But this follows immediately from the  $k$ -validity of  $y$ , because any such  $w$  also satisfies  $FAR_k[x] \vdash_x^* w \vdash_x^* y$ .  $\square$

LEMMA 14. Suppose  $y$  is  $k$ -valid with respect to  $\hat{x}$ . Then  $y$  is  $(k+1)$ -valid with respect to  $x$ .

Proof. By definition of  $k$ -validity,  $FAR_k[\hat{x}] \vdash_x^* y$ , so  $FAR_k[\hat{x}] \vdash_x^* y$  by the Stack Lemma. Recalling that  $FAR_{k+1}[x] = FAR_k[\hat{x}]$ , we have  $FAR_{k+1}[x] \vdash_x^* y$ . Suppose the surface configuration  $w$  satisfies  $FAR_{k+1}[x] \vdash_x^* w \vdash_x^* y$ . We must show that  $h(w) \geq h(y)$ . But by the corollary to the Stack Lemma,  $w$  satisfies  $FAR_k[\hat{x}] \vdash_x^* w \vdash_x^* y$ , so  $h(w) \geq h(y)$  by definition of  $k$ -validity.  $\square$

LEMMA 15 (Correctness of  $LOW_{k+1}$ ). Assume that  $LHL_{k+1}$  satisfies its correctness condition. As in lines 18–19 of the algorithm, for each  $x, y \in X_n$  such that  $h(x) \leq h(y) \leq h(FAR_{k+1}[x])$ , let

$$LOW_{k+1}[x, y] = \begin{cases} LHL_{k+1}[x, y] & \text{if } h(y) \leq h(\hat{x}), \\ LHL_{k+1}[x, z] & \text{if } h(y) > h(\hat{x}) \text{ and } h(z) = h(\hat{x}), \\ z & \text{else,} \end{cases}$$

where  $z = LHL_{k+1}[\hat{x}, y]$ . Then  $LOW_{k+1}$  satisfies its correctness condition.

Proof. Assume that  $y$  is  $(k+1)$ -valid with respect to  $x$ . We consider the following two cases:

Case 1.  $h(y) \leq h(\hat{x})$ . In this case,  $LOW_{k+1}[x, y]$  is defined to be  $LHL_{k+1}[x, y]$ . See Fig. 21. By Lemma 11,  $y$  is  $k$ -valid with respect to  $x$ . Hence by the correctness of

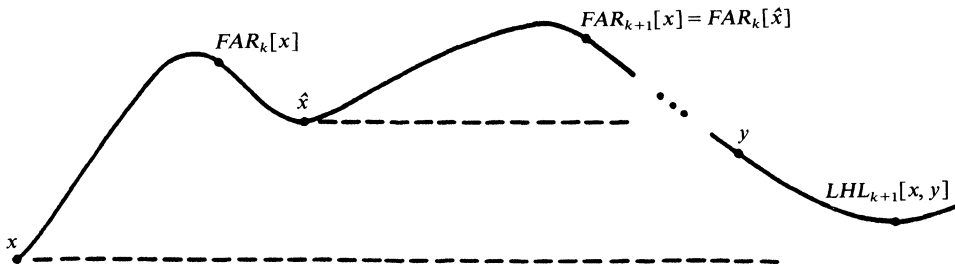


FIG. 21.  $LOW_{k+1}[x, y] = LHL_{k+1}[x, y]$  in case  $h(y) \leq h(\hat{x})$ .

$LHL_{k+1}$ ,  $LHL_{k+1}[x, y]$  is  $k$ -valid with respect to  $x$  and  $(k+1)$ -low from  $y$  with respect to  $x$ . By Lemma 13,  $LHL_{k+1}[x, y]$  is also  $(k+1)$ -valid with respect to  $x$ , so we are done.

Case 2.  $h(y) > h(\hat{x})$ . We assumed that  $y$  is  $(k+1)$ -valid with respect to  $x$ , so by Lemma 12,  $y$  is  $k$ -valid with respect to  $\hat{x}$ . Now consider  $z = LHL_{k+1}[\hat{x}, y]$ . By the correctness of  $LHL_{k+1}$ ,  $z$  is  $k$ -valid with respect to  $\hat{x}$  and  $(k+1)$ -low from  $y$  with respect to  $\hat{x}$ . By Lemma 14,  $z$  is  $(k+1)$ -valid with respect to  $x$ . We must again distinguish two cases:

Case a.  $h(z) = h(\hat{x})$ . In this case,  $LOW_{k+1}[x, y]$  is defined to be  $LHL_{k+1}[x, z]$ . See Fig. 22. As noted above,  $z$  is  $(k+1)$ -valid with respect to  $x$ , and we assume that  $h(z) = h(\hat{x})$ , so by Lemma 11,  $z$  is  $k$ -valid with respect to  $x$ . Then by the correctness of  $LHL_{k+1}$ ,  $LHL_{k+1}[x, z]$  is  $k$ -valid with respect to  $x$ , and  $(k+1)$ -low from  $z$  with respect to  $x$ . Since  $LHL_{k+1}[x, z]$  is  $k$ -valid with respect to  $x$  and occurs after  $FAR_{k+1}[x]$ , it follows by Lemma 13 that  $LHL_{k+1}[x, z]$  is  $(k+1)$ -valid with respect to  $x$ . Since  $LHL_{k+1}[x, z]$  is  $(k+1)$ -low from  $z$ , and  $z$  occurs after  $y$ , it follows that  $LHL_{k+1}[x, z]$  is certainly  $(k+1)$ -low from  $y$ , so we are done.

Case b.  $h(z) > h(\hat{x})$ . In this case,  $LOW_{k+1}[x, y]$  is defined to be  $z$ . See Fig. 23. We already know that  $z$  is  $(k+1)$ -valid with respect to  $x$ . It remains to show that  $z$  is  $(k+1)$ -low from  $y$  with respect to  $x$ . We prove this by contradiction. We will start by assuming the existence of a counterexample  $w$ , and show that  $w$  must also be a counterexample to  $z$ 's being  $(k+1)$ -low from  $y$  with respect to  $\hat{x}$ . We know, however, that  $z$  is  $(k+1)$ -low from  $y$  with respect to  $\hat{x}$ , so there can be no such counterexample.

Let  $w$  be the first surface configuration such that  $z \vdash_x^+ w$  and  $h(w) \leq h(z)$ , and assume  $w$  is within  $2^{k+1}$  steps of  $y$ . (If there is no such surface configuration  $w$ , or if the first such  $w$  is more than  $2^{k+1}$  steps from  $y$ , then there is no counterexample to  $z$ 's being  $(k+1)$ -low from  $y$  with respect to  $x$ .) We first show that  $z \vdash_x^+ w$  (note that the base is  $\hat{x}$ ).

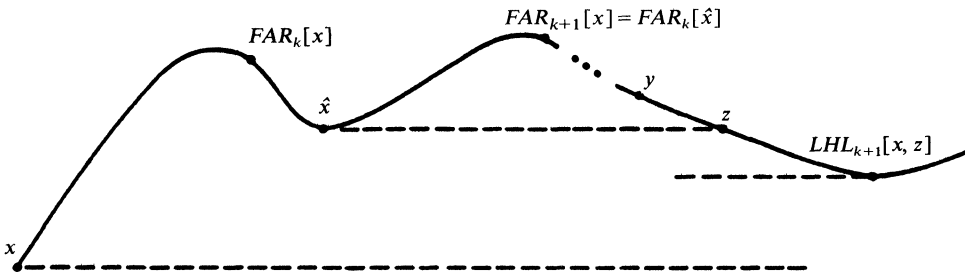


FIG. 22.  $LOW_{k+1}[x, y] = LHL_{k+1}[x, z]$  in case  $h(y) > h(\hat{x})$  and  $h(z) = h(\hat{x})$ .

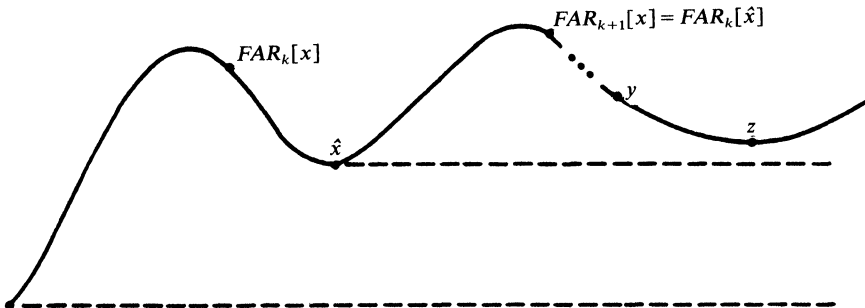


FIG. 23.  $LOW_{k+1}[x, y] = z$  in case  $h(y) > h(\hat{x})$  and  $h(z) > h(\hat{x})$ .

Consider the surface configuration  $u$  immediately following  $z$ . That is,  $z \vdash_x u$ . Since  $h(z) > h(\hat{x})$  and the stack height can change by at most one in a single move, we have  $h(u) \geq h(\hat{x})$  and hence  $z \vdash_{\hat{x}} u$  by the Stack Lemma. Now if  $h(u) \leq h(z)$ , then  $u$  is our counterexample  $w$ , and we are done. If, on the other hand,  $h(u) > h(z)$ , we consider the surface configuration  $u'$  immediately following  $u$ , and repeat the argument. Continuing in this way, we discover that  $z \vdash_{\hat{x}} u \vdash_{\hat{x}} \cdots \vdash_{\hat{x}} w$ . Since  $h(w) \leq h(z)$  and  $w$  was assumed to be within  $2^{k+1}$  steps of  $y$ , we see that  $w$  is a counterexample to  $z$ 's being  $(k+1)$ -low from  $y$  with respect to  $\hat{x}$ . But no such counterexample can exist, for in fact  $z$  is  $(k+1)$ -low from  $y$  with respect to  $\hat{x}$  by the correctness of  $LHL_{k+1}[x, y]$ . Hence we must conclude that  $z$  is also  $(k+1)$ -low from  $y$  with respect to  $x$ .  $\square$

The following theorem asserts the correctness of our algorithm for simulating a deterministic pushdown automaton.

**THEOREM 2.** *The arrays  $FAR_k$ ,  $HOP_k$ , and  $LOW_k$  satisfy their correctness conditions for all  $k = 0, 1, \dots, \lceil \log_2 T(n) \rceil$ .*

For convenience, we reproduce the correctness conditions for  $FAR_k$ ,  $HOP_k$ , and  $LOW_k$ .

**$FAR_k$ :**

For all  $x \in X_n$ ,  $FAR_k[x]$  is  $k$ -far from  $x$ .

**$HOP_k$ :**

For all  $x \in X_n$ ,

Either  $HOP_k[x]$  is some surface configuration  $w$  such that

$x \vdash_x^+ w \vdash_x^* FAR_k[x]$  and  $h(w) = h(x)$ ,

or, if there is no such  $w$ , then  $HOP_k[x] = x$ .

**$LOW_k$ :**

For all  $x, y \in X_n$ ,

if  $y$  is  $k$ -valid with respect to  $x$ ,

then  $LOW_k[x, y]$  is

(A)  $k$ -valid with respect to  $x$ , and

(B)  $k$ -low from  $y$  with respect to  $x$ .

*Proof by induction on  $k$ .* The reader can check that the correctness conditions are satisfied for  $k = 0$  by the initialization stage, lines 1-10 of the algorithm. The induction step from  $k$  to  $k+1$  follows from Lemmas 6-15.  $\square$

We now state the main theorem of this paper.

**THEOREM 3.** *Let  $L$  be a deterministic context-free language. Then  $L$  is accepted by a P-RAM with  $O(n^4)$  processors in time  $O(\log n)$ .*

In § 3, we saw that there is a deterministic pushdown automaton  $M$  in normal form that accepts  $L$ . Use the algorithm of § 5 to simulate  $M$ . Each of the  $\lceil \log T(n) \rceil$  stages of the algorithm can be implemented in constant time on a P-RAM with one processor for each pair of surface configurations  $x, y \in X_n$ . Thus the algorithm runs in time  $O(\log T(n)) = O(\log n)$ . Since the number of surface configurations  $|X_n|$  is  $O(n^2)$ , we obtain a processor bound of  $O(n^4)$ .  $\square$

Rytter shows in [11] how the algorithm of Theorem 3 may be interpreted as a recursive procedure to recognize a deterministic context-free language in  $O(\log^2 n)$  space and polynomial time, thus giving an alternative proof that DCFL recognition is in SC, a result due to Cook [2].

**7. Improvement of the processor bounds.** The following observation leads to an easy improvement of the processor bound from  $O(n^4)$  to  $O(n^3)$ . It is sufficient for the algorithm to compute  $LOW_k[x, y]$  for the  $O(n^3)$  pairs  $x, y \in X_n$  such that  $h(x) = 0$ .

This is because, as noted in § 4, the stack height parameter reflects only relative stack height. All that matters is the difference between  $h(y)$  and  $h(x)$ . The value of  $LOW_k[x, y]$  for an arbitrary pair  $x, y \in X_n$  can be computed on demand from the value of  $LOW_k[x', y']$  for a corresponding pair  $x', y' \in X_n$  with  $h(x') = 0$  by adding  $h(x)$  to the stack height parameter of  $LOW_k[x', y']$ .<sup>2</sup> This does not increase the asymptotic running time of the algorithm. We therefore have:

**THEOREM 4.** *Let  $L$  be a deterministic context-free language. Then  $L$  is accepted by a P-RAM with  $O(n^3)$  processors in time  $O(\log n)$ .*

A more drastic reduction of the processor bound to  $O(n^2 \log n)$  can be obtained at the expense of an increase in running time to  $O(\log^2 n)$ . The essential idea is similar to the previous reduction. The algorithm computes  $LOW_k[x, y]$  only for those pairs  $x, y \in X_n$  such that  $h(x)$  is 0 and  $h(y)$  is a power of two. It is shown in [6] how the algorithm may be modified so that the value of  $LOW_k[x, y]$  for an arbitrary pair  $x, y \in X_n$  can be computed in  $O(\log n)$  time by a single processor from the values for these special pairs.

**8. Simulation of a deterministic auxiliary pushdown automaton.** We now consider the simulation by a P-RAM of a space- and time-bounded deterministic auxiliary pushdown automaton. An auxiliary pushdown automaton is a Turing machine with a work tape and a stack. Let  $dAPDA(S(n))$  denote the class of languages accepted by  $S(n)$  space-bounded,  $2^{O(S(n))}$  time-bounded deterministic auxiliary pushdown automata. Sudborough has exhibited [12] a deterministic context-free language  $L$  that is log-space complete for the class  $dAPDA(\log n)$ . That is, for any  $L' \in dAPDA(\log n)$ , there is a function  $f$  from  $L'$  to  $L$ , computable in  $O(\log n)$  space, such that  $x \in L'$  if and only if  $f(x) \in L$ , for all strings  $x$  over the alphabet of  $L'$ . This suggests a two-stage algorithm for recognizing  $L'$ : for any string  $x$  over the alphabet of  $L'$ , first compute  $f(x)$  and then determine if  $f(x) \in L$ . A log  $n$  space-bounded Turing machine can only run for  $2^{O(\log n)} = n^{O(1)}$  steps, so the length of  $f(x)$  is polynomial in the length of  $x$ . It is shown in [3] that a P-RAM can simulate any deterministic  $S(n)$  space-bounded Turing machine in parallel time  $O(S(n))$  on  $2^{O(S(n))}$  processors. Thus the first stage of our proposed two-stage algorithm can be carried out in  $O(\log n)$  time on  $n^{O(1)}$  processors. By our Theorem 3, the second stage can be carried out in time  $O(\log |f(x)|) = O(\log n)$  time on  $|f(x)|^3 = n^{O(1)}$  processors. We conclude that any language in  $dAPDA(\log n)$  can be recognized by a P-RAM in  $O(\log n)$  time on a polynomial number of processors. By a standard "padding" argument, Theorem 5 follows.

**THEOREM 5.** *For any fully space-constructible  $S(n) = \Omega(\log n)$ , a language in  $dAPDA(S(n))$  can be recognized by a P-RAM in  $O(S(n))$  time on  $2^{O(S(n))}$  processors.*

Sudborough conjectures in [12] that  $DSPACE(\log n)$  is strictly contained in  $dAPDA(\log n)$ , i.e., that adding a stack to a log  $n$  space-bounded, poly-time-bounded Turing machine gives it additional power. By our Theorem 5, Sudborough's conjecture implies that  $S(n)$  time-bounded P-RAM's are strictly more powerful than  $S(n)$  space-bounded Turing machines.

**9. Simulation of P-RAMs by deterministic auxiliary pushdown automata.** We observe in this section that our P-RAM algorithm for simulating any deterministic auxiliary pushdown automaton is nearly optimal, since there is a complementary simulation of P-RAMs by deterministic auxiliary pushdown automata.

<sup>2</sup>That is,  $q(x') = q(x)$  and  $q(y') = q(y)$ ,  $p(x') = p(x)$  and  $p(y') = p(y)$ , but  $h(x') = 0$  and  $h(y') = h(y) - h(x)$ .

**THEOREM 6.** *Let  $L$  be accepted by a P-RAM in time  $T(n)$ . Then  $L$  is accepted by a deterministic auxiliary pushdown automaton with space  $T(n)$  and time  $2^{O(T(n)^2)}$ .*

This theorem is a fairly easy consequence of some known relations between various computational models. Here we give a more direct proof:

*Proof.* Fortune and Wyllie prove in Lemma 1b of [3] that  $L$  is accepted by a deterministic Turing machine with space  $T(n)^2$  and time  $2^{O(T(n)^2)}$ . We observe that their algorithm may be implemented on a deterministic auxiliary pushdown automaton. Their algorithm is recursive and requires a pushdown stack of size at most  $T(n)$ , where each element on the stack can be represented by a bit sequence of length  $O(T(n))$ . Thus only  $T(n)$  space is required by our simulating deterministic auxiliary pushdown automaton.

**Acknowledgments.** Thanks to Charles Leiserson and David Shmoys for their helpful comments. Many thanks also to the referee for extremely conscientious and careful reading of drafts of this article.

#### REFERENCES

- [1] B. VON BRAUNMÜHL, S. COOK, K. MELHORN, AND R. VERBEEK, *The recognition of deterministic CFLs in small time and space*, Inform. and Control, 56 (1983), pp. 34–51.
- [2] S. COOK, *Deterministic CFLs are accepted simultaneously in polynomial time and log squared space*, in Proc. 11th Annual ACM Symposium on Theory of Computing, 1979, pp. 338–345.
- [3] S. FORTUNE AND J. WYLLIE, *Parallelism in random access machines*, in Proc. 10th ACM Symposium on Theory of Computation, 1978, pp. 114–118.
- [4] M. HARRISON, *Introduction to Formal Language Theory*, Addison-Wesley, Reading, MA, 1978.
- [5] J. HOPCROFT AND J. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.
- [6] P. KLEIN *Parallel recognition of context-free languages*, B.A. thesis, Harvard University, Cambridge, MA, 1984.
- [7] D. KNUTH, *On the translation of languages from left to right*, Inform. and Control, 8 (1965), pp. 607–639.
- [8] R. KOSARAJU, *Speed of recognition of context-free languages by array automata*, this Journal, 4 (1975), pp. 331–340.
- [9] J. REIF, *Parallel time  $O(\log n)$  time acceptance of deterministic CFLs*, in Proc. 23rd Annual IEEE Symposium on the Foundations of Computer Science, 1982, pp. 290–296.
- [10] W. RUZZO, *Tree-size bounded alternation*, J. Comput. System Sci., 21 (1980), pp. 218–235.
- [11] W. RYTTER, *On the recognition of context-free languages*, in Proc. Fifth Symposium on Computation Theory, Lecture Notes in Computer Science 208, Springer-Verlag, Berlin, Heidelberg, New York, 1985, pp. 318–325.
- [12] I. SUDBOROUGH, *On the tape complexity of deterministic context-free languages*, J. Assoc. Comput. Mach., 25 (1978), pp. 405–414.

## A NEARLY OPTIMAL PARALLEL ALGORITHM FOR CONSTRUCTING DEPTH FIRST SPANNING TREES IN PLANAR GRAPHS\*

XIN HE† AND YAACOV YESHA‡

**Abstract.** This paper presents a parallel algorithm for constructing depth first spanning trees in planar graphs. The algorithm takes  $O(\log^2 n)$  time with  $O(n)$  processors on a concurrent read concurrent write parallel random access machine (PRAM). The best previously known algorithm for the problem takes  $O(\log^3 n)$  time with  $O(n^4)$  processors on a PRAM. Our algorithm is within an  $O(\log^2 n)$  factor of optimality.

**Key words.** parallel algorithms, planar graphs, depth first spanning trees, graph algorithms

**AMS(MOS) subject classifications.** 05C05, 05C38, 68Q25, 68R10

**1. Introduction.** The *depth first spanning trees* of graphs have been used as a basis for many graph algorithms (Tarjan [11]). It has been conjectured, however, that the construction of a depth first spanning tree of a graph is inherently sequential (Reif [8] and Smith [9]). In spite of many attempts, the depth first spanning tree problem for general graphs is still not known to be in the NC class. (The NC class is the class of problems solvable in  $(\log n)^{O(1)}$  time by using  $n^{O(1)}$  processors. The NC class is widely accepted as the class of problems which can be solved fast in parallel with reasonably many processors [2].)

Recently, Smith discovered a nice parallel algorithm which finds a depth first spanning tree in a *planar* graph in  $O(\log^3 n)$  time with  $O(n^4)$  processors [9]. This work is important since it is the first which shows that the depth first spanning tree problem for planar graphs is in the NC class. However, due to the importance of the depth first spanning trees in designing graph algorithms, it is very desirable to have a more efficient parallel algorithm.

In this paper we develop such an algorithm. Our algorithm takes only  $O(\log^2 n)$  time with  $O(n)$  processors. It is within an  $O(\log^2 n)$  factor of optimal and is a substantial improvement over Smith's algorithm.

The computation model used in this paper is a *concurrent read concurrent write parallel random access machine* (CRCW PRAM). The model consists of a number of identical processors and a common memory. In each time unit, a processor can read from a memory cell, perform an arithmetic or a logical computation, and write into a memory cell. Both concurrent read from and write into the same memory cell by different processors are permitted. If a write conflict occurs, an arbitrary processor succeeds.

The model we use is slightly stronger than the *concurrent read exclusive write* (CREW) PRAM model used in [9] where the concurrent writes are disallowed. One step of our model can be simulated by the weaker CREW PRAM model in  $O(\log n)$  steps. (Vishkin [13] showed that this simulation can be performed in  $O(\log^2 n)$  steps by using parallel sorting. Later, Cole [1] showed that the sorting can be done in

---

\* Received by the editors December 8, 1986; accepted for publication (in revised form) May 29, 1987.

† Department of Computer and Information Science, Ohio State University, Columbus, Ohio 43210. The work of this author was supported in part by an Ohio State University Presidential Fellowship, and by the Office of Research and Graduate Studies of Ohio State University. Present address, Department of Computer Science, State University of New York, Buffalo, New York 14260.

‡ Department of Computer and Information Science, Ohio State University, Columbus, Ohio 43210. The work of this author was supported in part by the National Science Foundation under grant DCR-8606366.

$O(\log n)$  time with  $O(n)$  processors on a CREW PRAM. Hence by combining the algorithms in [13] and [1], this simulation can be performed in  $O(\log n)$  time with  $O(n)$  processors.) Therefore our algorithm can be implemented on the CREW PRAM model in  $O(\log^3 n)$  time with  $O(n)$  processors, still much better than Smith's algorithm.

In § 2, we review the basic strategy of Smith's algorithm. The implementation and analysis of our algorithm is presented in § 3.

**2. Preliminary.** Throughout this paper,  $G = (V, E)$  denotes a connected planar graph with vertex set  $V$  and edge set  $E$ .  $n = |V|$  denotes the number of vertices of  $G$ .

A *depth first spanning tree*  $T$  of  $G$  rooted at a specified vertex  $r \in V$  is constructed as follows: Starting from  $r$ , successively add edges into  $T$  until a vertex is reached all of whose exit edges (i.e., incident edges other than the one used to arrive at the vertex) are incident upon  $T$ . At this point, backtrack a minimum distance until a vertex is reached that has exit edges not incident upon  $T$  and continue. Repeat this process until all vertices of  $G$  are exhausted [11]. The problem considered in this paper is how to construct a depth first spanning tree for a given planar graph  $G$  with a specified root  $r$ .

A *plane embedding* of a planar graph  $G$  is a drawing of  $G$  in the plane such that no two edges of  $G$  intersect with each other in the drawing.

Suppose  $G$  is embedded in the plane. Let  $C$  be a (simple) cycle of  $G$ . The vertices of  $G$  are partitioned into three parts: the vertices on  $C$ , the vertices in the interior of  $C$  and the vertices in the exterior of  $C$ . If both the interior and the exterior of  $C$  contain no more than  $2n/3$  vertices,  $C$  is called a *separating cycle* of  $G$ .

Let  $T$  be any spanning tree of  $G$  rooted at a vertex  $t \in V$ .  $T$  induces a partial order on  $V$ :  $v_1 < v_2$  if and only if  $v_2$  is an ancestor of  $v_1$  in  $T$ . In general, if  $v_1$  and  $v_2$  are two arbitrary vertices they may be noncomparable in the partial order induced by  $T$ . A depth first spanning tree  $T$  can be characterized by the following well-known fact:

LEMMA 1 [9]. *A spanning tree  $T$  of  $G$  is a depth first spanning tree if and only if for any edge  $e \in E$  the end vertices of  $e$  are comparable in the partial order induced by  $T$ . □*

Let  $P_r$  be a (simple) path in  $G$  with  $r$  as one of its end vertices. Suppose  $G - P_r$  is the union of connected components  $\{G_i\}_{1 \leq i \leq k}$  for some  $k$ . Let  $e$  be an edge of  $G$ . Define  $e$  to be [9]:

(a) A *touching edge* of  $G_i$  if one end vertex of  $e$  is in  $G_i$  and another end vertex of  $e$  is in  $P_r$ —call this vertex the point where  $e$  touches  $P_r$ .

(b) An *inessential touching edge* of  $G_i$  if it is a touching edge of  $G_i$  and there exists another touching edge  $e'$  of  $G_i$  which touches  $P_r$  at a point further (in  $P_r$ ) from  $r$  than the point at which  $e$  touches  $P_r$ .

(c) An *essential touching edge* of  $G_i$  if it is a touching edge of  $G_i$  and the conditions in statement (b) are not satisfied.

Let  $e_i$  be an essential touching edge of  $G_i$  ( $1 \leq i \leq k$ ) and let  $x_i$  be the end vertex in  $e_i$  in  $G_i$ . Suppose  $T_i$  is a depth first spanning tree of  $G_i$  rooted at  $x_i$  ( $1 \leq i \leq k$ ). From Lemma 1, it is easy to show the following.

LEMMA 2 [9]. *Let  $T$  be the union of  $P_r$ ,  $\{e_i\}$  ( $1 \leq i \leq k$ ) and  $\{T_i\}$  ( $1 \leq i \leq k$ ). Then  $T$  is a depth first spanning tree of  $G$  rooted at  $r$ . □*

Figure 1 shows an example of Lemma 2. The planar graph shown in Fig. 1 is divided by the path  $P_r$  into two connected components  $G_1$  and  $G_2$ . The edges  $e_1$  and  $e_2$  are two touching edges of  $G_1$ , where  $e_2$  is the essential touching edge of  $G_1$ . The edges  $e_3$  and  $e_4$  are two touching edges of  $G_2$ , where  $e_4$  is the essential touching edge of  $G_2$ .  $T_1$  and  $T_2$  are the depth first spanning trees of  $G_1$  and  $G_2$ , respectively. The union of  $P_r$ ,  $e_2$ ,  $e_4$ ,  $T_1$ , and  $T_2$  is a depth first spanning tree of  $G$  rooted at  $r$ .



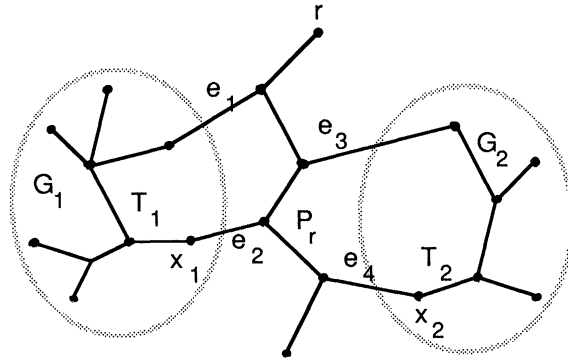


FIG. 1. An example of Lemma 2.

The basic idea of Smith's algorithm is as follows; Find a path  $P_r$  in  $G$ , with  $r$  as one end vertex such that  $G - P_r$  is a union of connected components  $\{G_i\} (1 \leq i \leq k)$  and  $|G_i| \leq 2n/3$  for each  $1 \leq i \leq k$ . Then find an essential touching edge  $e_i$  for each  $G_i (1 \leq i \leq k)$  and let  $x_i$  be the end vertex of  $e_i$  in  $G_i$ . Next recursively call the algorithm on each  $G_i (1 \leq i \leq k)$  in parallel to construct a depth first spanning tree  $T_i$  of  $G_i$  rooted at  $x_i$ . Finally construct a depth first spanning tree  $T$  for  $G$  rooted at  $r$  as in Lemma 2.

In order to find the *separating path*  $P_r$ , described above, Smith uses the following method: First find a separating cycle  $C$  in  $G$ . Then find a path  $P_1$  from the root  $r$  to an arbitrary vertex  $x$  on  $C$  such that no other vertex of  $C$  is on  $P_1$ . Let  $e$  be an edge on  $C$  incident to  $x$ . Define  $P_r = P_1 \cup C - \{e\}$ . Each connected component  $G_i$  of  $G - P_r$  is contained either in the interior or in the exterior of  $C$ . Thus each  $G_i$  contains at most  $2n/3$  vertices. Therefore  $P_r$  is the required separating path.

**3. Implementation and analysis.** We discuss the implementation and analyze the resource bounds of the parallel depth first spanning tree algorithm in this section.

Our algorithm works for any planar graph  $G = (V, E)$  given by the *edge list* form. (Namely, the edge set of  $G$  is given as a list in arbitrary order.) In order to make fast parallel computation possible, we will preprocess  $G$  to construct a more suitable representation, the *combinatorial representation*, defined as follows.

Suppose  $G = (V, E)$  is embedded in the plane.  $G$  partitions the plane into a number of connected regions. Each connected region is called a *face*. The combinatorial representation of the plane embedded graph  $G$  is defined as follows: Each edge  $e = (x, y) \in E$  is represented by two directed *darts*  $(x, y)$  and  $(y, x)$ . The combinatorial representation of  $G$  consists of two lists of these darts. The first list  $L_1$  is arranged in the order such that for each vertex  $v \in V$  the darts "leaving  $v$ " are consecutive in  $L_1$  and are in the order they appear in the embedding in the counterclockwise direction. The second list  $L_2$  is arranged in the order such that for each face  $f$  of  $G$  the darts on the boundary of  $f$  are consecutive in  $L_2$  and are in the order they appear in the embedding in the clockwise direction. Given a planar graph  $G$ , the combinatorial representation of a plane embedding of  $G$  can be found in  $O(\log^2 n)$  time with  $O(n)$  processors (Klein and Reif [3]). We will use this representation in our algorithm.

The algorithm is as follows. (It is different from Smith's algorithm in that the resources are allocated more carefully and each step is implemented more efficiently in order to achieve the  $O(\log^2 n)$  time and  $O(n)$  processor bounds.)

## DFST ALGORITHM.

Input: A connected planar graph  $G = (V, E)$  with a specified root  $r \in V$  given by edge list form;

Initialization: Find a plane embedding and a combinatorial representation for  $G$ ;

- (1) Find the *biconnected components*  $\{G_j\} (1 \leq j \leq q \text{ for some } q)$  of  $G$ . Let  $x_j (1 \leq j \leq q)$  be the vertex in  $G_j$  which is closest to the root  $r$ . Perform the following steps on each  $G_j$  simultaneously to construct a depth first spanning tree  $T_j$  rooted at  $x_j$  for each  $G_j (1 \leq j \leq q)$ . The union of  $T_j (1 \leq j \leq q)$  will be a depth first spanning tree of  $G$  rooted at  $r$  (Smith [9]). (In order to avoid double subscripts, we still use symbols  $G$  and  $r$  for  $G_j$  and  $x_j$  in the following statements.)
- (2) Find a separating cycle  $C$  in  $G$ . (Since  $G$  is a biconnected planar graph, such a separating cycle exists; see Smith [9] and Miller [6].)
- (3) Find a path  $P_1$  from the root  $r$  to any vertex  $x$  on  $C$  such that no other vertex of  $C$  is on  $P_1$ . Let  $e$  be an edge of  $C$  incident to  $x$ . Define  $P_r = P_1 \cup C - \{e\}$ .
- (4) Find the connected components  $\{G_i\} (1 \leq i \leq k \text{ for some } k)$  of  $G - P_r$ .
- (5) For each  $G_i (1 \leq i \leq k)$  find an essential touching edge  $e_i$ . Let  $x_i$  be the end vertex of  $e_i$  in  $G_i$ .
- (6) Construct the combinatorial representation of the subgraphs  $G_i (1 \leq i \leq k)$  from the combinatorial representation of  $G$ .
- (7) Recursively call the DFST algorithm on each  $G_i$  in parallel to construct a depth first spanning tree  $T_i$  for  $G_i$  rooted at  $x_i (1 \leq i \leq k)$ .
- (8) Let  $T = \cup_i (T_i \cup e_i) \cup P_r$ .  $T$  is a depth first spanning tree of  $G$  rooted at  $r$ .

End DFST.

We next discuss the implementation and analyze the resource bounds of the DFST algorithm.

The initialization step can be performed in  $O(\log^2 n)$  time with  $O(n)$  processors on a CRCW PRAM [3]. This step is performed only once. When the DFST algorithm is recursively called on a subgraph  $G_i$  of  $G$  at step (7), the combinatorial representation of  $G_i$  is obtained from the combinatorial representation of  $G$  in a more efficient way (details are discussed later).

Step (1) can be implemented by using the algorithm of Tarjan and Vishkin [12] in  $O(\log n)$  time with  $O(n)$  processors on a CRCW PRAM.

Step (2) is the most resource-consuming part of the algorithm. In Smith's algorithm, this step takes  $O(\log^2 n)$  time with  $O(n^4)$  processors. Miller [6] developed a parallel algorithm which finds a separating cycle in a biconnected planar graph with size at most  $O(n^{1/2})$ . If the input is given by the combinatorial representation form (Miller uses a slightly different variation of this representation in [6]), this algorithm takes  $O(\log n)$  time with  $O(n^3)$  processors. However, since we do not have to restrict the size of the separating cycle in the DFST algorithm, this step can be implemented much more efficiently. Actually, one subroutine in Miller's algorithm finds a separating cycle  $C$  of  $G$  in  $O(\log n)$  time with  $O(n)$  processors on a CRCW PRAM, (although the size of  $C$  might be large). Thus step (2) can be implemented by using this subroutine in  $O(\log n)$  time with  $O(n)$  processors.

Step (3) can be performed as follows: by using the algorithm of Shiloach and Vishkin [10], we first construct an arbitrary spanning tree  $\bar{T}$  of  $G$  rooted at  $r$ . This can be done in  $O(\log n)$  time with  $O(n)$  processors. Next pick up an arbitrary vertex  $x_0$  on  $C$  and identify the path  $P_0$  in  $\bar{T}$  from  $x_0$  to  $r$ . This can be done by using standard pointer jump technique in  $O(\log n)$  time with  $O(n)$  processors. Let  $x$  be the vertex on  $P_0$  which is on  $C$  and is closest to  $r$ . Let  $P_1$  be the portion of  $P_0$  from  $x$  to  $r$ . Then

$x$  and  $P_1$  is what we want to find in step (3). Thus step (3) totally takes  $O(\log n)$  time with  $O(n)$  processors.

Step (4) can be performed by using the algorithm of Shiloach and Vishkin [10] in  $O(\log n)$  time with  $O(n)$  processors on a CRCW PRAM.

In order to implement step (5), we first compute the index number of the vertices on the path  $P_r$  (i.e.,  $r$  has index number 1, the vertex on  $P_r$  adjacent to  $r$  has index number 2 and so on). The set  $E_i(1 \leq i \leq k)$  of touching edges for each connected component  $G_i$  can be identified in constant time. Next we sort each  $E_i(1 \leq i \leq k)$  according to the index number of the end vertex of each  $e \in E_i$  that is on  $P_r$ . This sorting can be done in  $O(\log n)$  time with  $O(n)$  processors [1]. The essential touching edge of  $G_i$  is the last edge in the sorted list of  $E_i$ . Thus step (5) takes  $O(\log n)$  time with  $O(n)$  processors.

Step (6) can be implemented as follows: Each subgraph  $G_i(1 \leq i \leq k)$  of  $G$  has a natural plane embedding inherited from the embedding of  $G$ . The combinatorial representation for  $G_i$  is computed from the lists  $(L_1, L_2)$  of the combinatorial representation of  $G$  as follows; In the list  $L_1$ , delete all darts that have at least one end vertex in  $P_r$ . Then perform a parallel sort on the remaining darts of  $L_1$  such that the darts belonging to the same connected component  $G_i$  are consecutive in the resulting list. Perform a similar operation for the list  $L_2$ . This gives the combinatorial representation for each  $G_i$ . Since the sorting can be done in  $O(\log n)$  time with  $O(n)$  processors (Cole [1]), step (6) can be performed within the same time and processor bounds.

Step (7) is the recursive call. Step (8) clearly takes constant time.

Since one application of the DFST algorithm on  $G$  reduces the problem to the subgraphs of size at most  $2n/3$ , the depth of recursive calls is at most  $\log_{3/2} n$ . Since each step takes at most  $O(\log n)$  time with  $O(n)$  processors, the entire algorithm takes  $O(\log^2 n)$  time with  $O(n)$  processors as claimed. In summary we have Theorem 3.

**THEOREM 3.** *Let  $G$  be a planar graph given by edge list form and let  $r$  be a specified vertex of  $G$ . A depth first spanning tree of  $G$  rooted at  $r$  can be constructed in  $O(\log^2 n)$  time with  $O(n)$  processors on a CRCW PRAM.  $\square$*

**Acknowledgments.** The authors are grateful to an anonymous referee who pointed out Miller's paper [6] to us. This helped to substantially simplify and shorten the presentation of the our paper.

#### REFERENCES

- [1] R. COLE, *Parallel merge sort*, Proc. 27th Annual IEEE Symposium on the Foundations of Computer Science, 1986, pp. 511-516.
- [2] S. A. COOK, *The classification of problems which have fast parallel algorithms*, Proc. 1983 International Foundation of Computation Theory Conference, Lecture Notes in Computer Science 158, Springer-Verlag, pp. 78-93.
- [3] P. N. KLEIN AND J. H. REIF, *An efficient parallel algorithm for planarity*, Proc. 27th Annual IEEE Symposium on the Foundations of Computer Science, 1986, pp. 465-477.
- [4] R. J. LIPTON AND R. E. TARJAN, *A separator theorem for planar graphs*, SIAM J. Appl. Math. 36 (1979), pp. 177-189.
- [5] ———, *Applications of the planar separator theorem*, SIAM J. Comput., 9 (1980), pp. 615-627.
- [6] G. L. MILLER, *Finding small simple cycle separator for 2-connected planar graphs*, J. Comput. System. Sci., 32 (1986), pp. 265-279.
- [7] G. L. MILLER AND J. H. REIF, *Parallel tree contraction and its application*, Proc. 26th Annual IEEE Symposium on the Foundations of Computer Science, 1985, pp. 478-489.
- [8] J. REIF, *Depth first search is inherently sequential*, Aiken Computation Lab Technical Report TR-27-38, Harvard University, Cambridge, MA, November, 1983.
- [9] J. SMITH, *Parallel algorithms for depth-first searches I: Planar graphs*, SIAM J. Comput., 15 (1986), pp. 814-830.

- [10] Y. SHILOACH AND U. VISHKIN, *An  $O(\log n)$  parallel connectivity algorithm*, J. Algorithms, 3 (1982), pp. 57–67.
- [11] R. E. TARJAN, *Depth first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.
- [12] R. E. TARJAN AND U. VISHKIN, *An efficient parallel biconnectivity algorithm*, SIAM J. Comput., 14 (1985), pp. 862–874.
- [13] U. VISHKIN, *Implementation of simultaneous memory address access in models that forbid it*, J. Algorithms, 4 (1983), pp. 45–50.

## ON SEARCH TIMES FOR EARLY-INSERTION COALESCED HASHING\*

BORIS PITTEL† AND JENN-HWA YU†

**Abstract.** The distributions of the search times for an early-insertion form of coalesced hashing (first proposed by Vitter [11], [12]) are studied. It is demonstrated, in particular, that the largest search time is very close, in probability, to the one for the late-insertion coalesced hashing [8]. In addition, a formula for the expected successful search time obtained by Chen and Vitter [1] and, independently, by Knott [7] is shown to follow directly from the presented analysis.

**Key words.** search algorithm, hashing, largest search time, probabilistic analysis

**AMS(MOS) subject classifications.** primary 68P10, 68Q25, 68R05; secondary 60C05, 05C99

**1. Introduction. Results.** Coalesced hashing is a hashing algorithm in which collisions are resolved by forming chains of occupied locations (cells) of the table; ordering in each chain is conveniently specified by the consecutive chain references associated with every cell of the chain. To have a complete description of coalescing, one has to decide how a chain grows in cases when the next key is hashed to one of its cells. In the most well-known version [4], [6] a cell which accommodates such a key becomes the *last* cell of the corresponding chain. For the sake of references below, let us call it the late-insertion coalesced hashing method, or lich. In contrast, an early-insertion coalesced hashing method (eich) suggested by Vitter [11] and independently by Knott [7] is specified by the following agreement. Suppose that a key is hashed to an already occupied cell, say  $i$ . Then a cell which admits this key, that is its eventual location, is inserted into the chain passing through the cell  $i$  right between the cell  $i$  and its immediate successor. (Thus, only when cell  $i$  is the last in its chain, will the new cell be the last one of the enlarged chain.) As for the actual location of the key in case of collision, in the simplest case, it is found—in both lich and eich—via the sequential search of the table from, say, left to right (see, however, Knott [7]). It should be clear therefore that, for a given set of keys, the two algorithms lead to the same partition of the set of occupied cells into chains. It is the ordering in the chains where the algorithms differ from each other.

Let  $m$  be the size of the table, and let the  $n$  keys be  $x_1, x_2, \dots, x_n$ , indexed according to the order of insertion. Naturally,  $n \leq m$ . As usual, we assume that the hashing function  $h$  which maps the universe of keys into  $S = \{1, \dots, m\}$  is such that  $h(x_1), h(x_2), \dots, h(x_n)$  are independent random variables uniformly distributed over  $S$ . The central characteristics of either of the algorithms are the random search times  $T_1(i, n), T_2(i, n), 1 \leq i \leq n$ . Formally, they are defined as follows:  $T_1(i, n)$  is the number of cells of the subchain which originates at the cell  $h(x_i)$  and ends at the actual location of the key  $x_i$  after  $n$  keys have been inserted;  $T_2(i, n)$  is the number of extra cells searched sequentially when  $x_i$  is inserted to find a vacant cell for the key  $x_i$  when the cell  $h(x_i)$  is already occupied, and 0 when  $h(x_i)$  is not already occupied. The random variables  $T_1(i, n)$  and  $T_2(i, n)$  depend formally on the random keys  $x_1, x_2, \dots, x_n$ . Clearly, the sequences  $\{T_2(i, n) : 1 \leq i \leq n\}$  are the same for both algorithms. Besides,  $T_2(i, n)$  is completely determined by the values of  $\{h(x_j) : 1 \leq j \leq i\}$ , thus does not depend on those  $n - i$  keys inserted after the  $i$ th key. The same is true for  $T_1(i, n)$  in

\*Received by the editors May 27, 1986; accepted for publication (in revised form) June 11, 1987.

† Department of Mathematics, Ohio State University, Columbus, Ohio 43210.

the case of the lich algorithm, since then every chain grows only at its end. For the eich algorithm, however,  $T_1(i, n)$  does depend on those  $n - i$  keys inserted after the  $i$ th key since the keys arriving after  $x_i$  may push a cell with the key  $x_i$  in it (along the corresponding chain) farther away from the cell  $h(x_i)$ .

Exact and asymptotic moment-type formulae for  $T_j(i, n)$ ,  $j = 1, 2$ , were obtained by Knuth [6] (lich), by Chen and Vitter [1], and Knott [7] (eich) (see [2] for the mean values analysis of other versions of coalesced hashing). A central observation is that the average search times remain bounded by a constant even when the load factor  $n/m = 1$ .

The distributions of  $T_1(i, n)$ , and  $T_2(i, n)$ ,  $1 \leq i \leq n$  for lich were studied by the first author in [8]. It was shown, in particular, that when  $n, m \rightarrow \infty$ , such that  $n/m = a \in (0, 1]$  and is fixed,

$$(1.1) \quad \begin{aligned} U_1(n) &= \max \{T_1(i, n): 1 \leq i \leq n\} = \log_b n - 2 \log_b \log_b n + O_p(1), \\ U_2(n) &= \max \{T_2(i, n): 1 \leq i \leq n\} = \log_b n - \log_b \log_b n + O_p(1), \end{aligned}$$

where  $b = (1 - e^{-a})^{-1}$ , and  $O_p(1)$  stands for random terms bounded in probability as  $n, m \rightarrow \infty$ . Formally, a random variable  $A_{nm}$  is bounded in probability if  $P(|A_{nm}| \geq \omega_{nm}) \rightarrow 0$  for  $\omega_{nm} \rightarrow \infty$  however slowly as  $n, m \rightarrow \infty$ . ( $U_1(n)$  represents the largest successful search time.  $U_2(n)$  represents the largest number of the left-to-right probes needed to find a vacant cell in case of an unsuccessful search.)

Our goal in this paper is to study the sequence  $\{T_1(i, n): 1 \leq i \leq n\}$  for the eich algorithm. It turns out to be possible to derive the exact formula for the distribution of  $T_1(i, n)$ . Namely, we prove the following theorem.

THEOREM 1.

$$\begin{aligned} P(T_1(i, n) \geq l) &= (i-1)/m \sum_{j=0}^{l-2} (-1)^j \binom{l-2}{j} (1-j/m)^{n-i}, \quad 2 \leq l \leq n-i+2, \\ &= 1, \quad l = 1. \end{aligned}$$

*Comment.* The factors in the formula above have a transparent probabilistic interpretation. Namely,  $(i-1)/m$  is the probability that the key  $x_i$  is hashed to a nonempty location; the sum equals the probability that in the classical allocation model (with  $(n-i)$  balls and  $m$  cells) some specified  $(l-2)$  cells are nonempty. In a striking contrast, our derivation of the formula is quite analytic, involving contour integration and residues.

COROLLARY.

$$P(T_1(i, n) \geq l) \sim (i-1)/m [1 - \exp(-a + i/m)]^{l-2},$$

if  $n-i \rightarrow \infty$  and  $l^2 = o(n-i)$ . Here and below, the relation  $f(n) \sim g(n)$  means that  $f(n)/g(n) \rightarrow 1$  as  $n \rightarrow \infty$ .

This corollary contains a very simple approximate formula for the distribution of  $T_1(i, n)$  in cases when  $i$  the index of a key is considerably smaller than  $n$ . Furthermore, let  $\mathcal{C}_n$  be the successful search time of a key chosen at random among all  $n$  keys. The above results will enable us to prove the following theorem.

THEOREM 2. As  $n, m \rightarrow \infty$ , with  $n/m = a$ ,  $\mathcal{C}_n$  converges in distribution, and in terms of all moments, to a random variable  $\mathcal{C}$  such that

$$(1.2) \quad \begin{aligned} P(\mathcal{C} = l) &= 1 - a/2 \quad \text{if } l = 1, \\ &= a^{-1}(l-1)^{-1} \int_0^a [1 - \exp(-a+y)]^{l-1} dy \quad \text{if } l \geq 2. \end{aligned}$$

In particular,

$$(1.3) \quad \lim E(\mathcal{C}_n) = E(\mathcal{C}) = (e^a - a)/a,$$

$$(1.4) \quad \lim \text{var}(\mathcal{C}_n) = \text{var}(\mathcal{C}) = (e^{2a}/2 + e^a - a - 3/2)/a - (e^a - 1)^2/a^2.$$

The formula (1.3) was derived differently in [1], [7].

We will also study the asymptotic behavior of  $U_1(n) = \max \{T_1(i, n) : 1 \leq i \leq n\}$ , which represents the largest successful search time for each.

THEOREM 3. Let  $\omega(n) \rightarrow \infty$  however slowly. Then

$$P(|U_1(n) - \log_b n + 2 \log_b \log_b n| \leq \omega(n)) \rightarrow 1,$$

or in short

$$(1.5) \quad U_1(n) = \log_b n - 2 \log_b \log_b n + O_p(1),$$

where  $O_p(1)$  represents a random term bounded in probability.

Comments. Comparing (1.1) and (1.5) shows that the asymptotic behavior of  $U_1(n)$  is very similar for both versions of coalesced hashing. This is quite surprising since there are no close relations between these two random variables. To see it, and to better understand the difference between algorithms, let us look at a couple of examples.

Example 1.  $n = 7, m = 10, \{h(x_i) : 1 \leq i \leq 7\} = \{10, 8, 10, 7, 8, 1, 10\}$ .

(a) lich. Since a chain grows at its end, it can be easily seen that the chains of occupied cells are  $\{10, 1, 3, 4\}, \{8, 2\}$  and  $\{7\}$ . Also  $\{T_1(i, 7) : 1 \leq i \leq 7\} = \{1, 1, 2, 1, 2, 2, 4\}$  and  $U_1(7) = 4$  (Fig. 1).

(b) eich. Up until the key  $x_7$ , the algorithms do not differ. The key  $x_7$  is hashed to the cell 10, which is already occupied by the key  $x_1$ . So,  $x_7$  goes into the cell 4 which is the left-most empty cell at this moment. Now, the cell 4 is to be absorbed by the (sub)chain  $\{10, 1, 3\}$ ; unlike the lich version, it is inserted between the cells 10 and 1 which results in a larger chain  $\{10, 4, 1, 3\}$ . (Two other chains are  $\{8, 2\}$  and  $\{7\}$ .) Compared with the lich scheme,  $T_1(3, 7)$  is increased by 1, and  $T_1(7, 7)$  is reduced by 2. Hence  $\{T_1(i, 7) : 1 \leq i \leq 7\} = \{1, 1, 3, 1, 2, 2, 2\}$  and  $U_1(7) = 3$  as opposed to 4 in the lich case Fig. 2).

Example 2.  $n = 6, m = 9, \{h(x_i) : 1 \leq i \leq 6\} = \{1, 1, 1, 3, 4, 5\}$ . Here, in both schemes, there is only one chain comprised by the cells 1, 2,  $\dots$ , 6. However, in lich case,  $\{T_1(i, 6) : 1 \leq i \leq 6\} = \{1, 2, 3, 2, 2, 2\}$  and  $U_1(6) = 3$ , while, for eich,  $\{T_1(i, 6) : 1 \leq i \leq 6\} = \{1, 6, 2, 2, 2, 2\}$  and  $U_1(6) = 6$ . (See Figs. 3 and 4.)

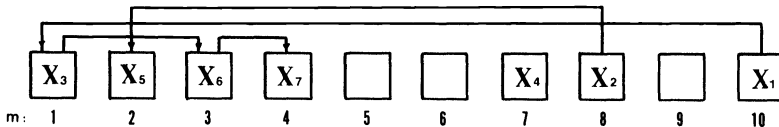


FIG. 1. lich,  $m = 10, n = 7$ .

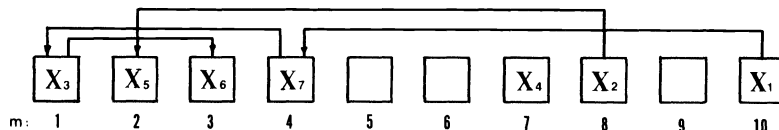


FIG. 2. eich,  $m = 10, n = 7$ .

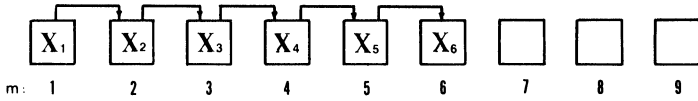


FIG. 3. *lich*,  $m = 9$ ,  $n = 6$ .

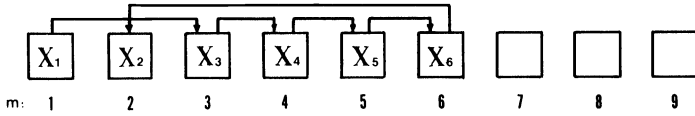


FIG. 4. *eich*,  $m = 9$ ,  $n = 6$ .

The reader interested in the results regarding the expected largest search times for other hashing techniques is referred to [3], [5], [9].

**2. Distribution of  $T_1(i, n)$ .** Recall that  $T_1(i, n)$  is the length of the subchain from the cell  $h(x_i)$  to the cell which actually contains the key  $x_i$  after all  $n$  keys are inserted,  $1 \leq i \leq n$ . Denote the index of this actual location by  $\tilde{h}(x_i)$ . Clearly,

$$T_1(i, n) = 1 \quad \text{and} \quad 1 \leq T_1(i, n) \leq n - i + 2 \quad \text{for } 2 \leq i \leq n.$$

Suppose that  $2 \leq l \leq n - i + 2$ . By the definition of the *eich* algorithm,  $T_1(i, n) \geq l$  if and only if in the subchain which connects the cells  $h(x_i)$  and  $\tilde{h}(x_i)$  there are  $j \geq l - 2$  cells occupied by some  $j$  keys inserted after  $x_i$ . Let  $A = \{x_{i_1} < x_{i_2} < \dots < x_{i_{l-2}}\}$  be a set of  $l - 2$  keys inserted after  $x_i$ . Denote by  $E_A$  the event that, when the subchain from  $h(x_i)$  to  $\tilde{h}(x_i)$  is of length  $l$  for the first time, it is the keys from the set  $A$  which occupy  $l - 2$  intermediate cells. (In other words,  $E_A$  means that insertion of the key  $x_{i_{l-2}}$  results in a subchain  $\{h(x_i), \tilde{h}(y_1), \dots, \tilde{h}(y_{l-2}), \tilde{h}(x_i)\}$ , where  $\{y_1, \dots, y_{l-2}\}$  is a permutation of  $A$ .) Then obviously

$$(2.1) \quad P(T_1(i, n) \geq l) = \sum_A P(E_A),$$

where summation is taken over all  $(l - 2)$ -element subsets  $A$  of  $\{x_{i+1}, \dots, x_n\}$ . Let us compute  $P(E_A)$ . According to the definition of the *eich* algorithm, the probability of the event  $E_A$  is the product of the probability of the following  $l - 1$  independent events:  $B_1 = \{\text{the key } x_i \text{ is hashed to a nonempty cell}\}$ , with

$$P(B_1) = (i - 1)/m;$$

$B_2 = \{\text{for every key } y \text{ from } x_{i+1} \text{ to } x_{i_{l-1}}, h(y) \neq h(x_i) \text{ and } h(x_{i_l}) = h(x_i)\}$ , with

$$P(B_2) = (1 - 1/m)^{i_1 - i - 1} (1/m); \dots;$$

$B_{l-1} = \{\text{for every key } y \text{ from } x_{i_{l-3}+1} \text{ to } x_{i_{l-2}-1}, h(y) \notin \{h(x_i), \tilde{h}(x_{i_l}), \dots, \tilde{h}(x_{i_{l-3}})\} \text{ and } h(x_{i_{l-2}}) \in \{h(x_i), \tilde{h}(x_{i_l}), \dots, \tilde{h}(x_{i_{l-3}})\}\}$ , with

$$P(B_{l-1}) = (1 - (l - 2)/m)^{i_{l-2} - i_{l-3} - 1} (l - 2)/m.$$

Hence

$$(2.2) \quad \begin{aligned} P(E_A) &= (i - 1)/m (1 - 1/m)^{i_1 - i - 1} (1/m) \dots \\ &\cdot [1 - (l - 2)/m]^{i_{l-2} - i_{l-3} - 1} (l - 2)/m \\ &= [(i - 1)/m] [(l - 2)!/m^{l-2}] \prod_{j=1}^{l-2} (1 - j/m)^{s_j} \cdot 1^{s_{l-1}}, \end{aligned}$$



where

$$s_j = i_j - i_{j-1} - 1 \quad (j \geq 2), \quad s_1 = i_1 - i - 1, \quad s_{l-1} = n - i_{l-2}.$$

Then, by (2.1) and (2.2),

$$P(T_1(i, n) \geq l) = [(i-1)/m][(l-2)!/m^{l-2}] \sum_s \prod_{j=1}^{l-2} (1-j/m)^{s_j} \cdot 1^{s_{l-1}},$$

where the summation is over all tuples  $s = (s_1, \dots, s_{l-1})$  such that  $s_j \geq 0, 1 \leq j \leq l-1$  and  $\sum_j s_j = n - i - l + 2$ .

Let us simplify the last formula (as in [8]). To this end, notice first that the sequence

$$\{P[T_1(i, u + (i+l-2)) \geq l] \cdot [(i-1)m^{-1}(l-2)!m^{-(l-2)}]^{-1}; u \geq 0\}$$

is an  $(l-1)$ -fold convolution of the  $(l-2)$  sequences  $\{(1-j/m)^u; u \geq 0\}, 1 \leq j \leq l-2$ , and the sequence  $\{1^u; u \geq 0\}$ . Since the generating function of a convolution is the product of individual generating functions, we obtain

$$P(T_1(i, n) \geq l) = (i-1)m^{-1}(l-2)!m^{-(l-2)} \text{coeff}_{z^{n-i-l+2}} F(z, l),$$

where

$$F(z, l) = \prod_{j=0}^{l-2} [1 - z(1-j/m)]^{-1}, \quad |z| < 1.$$

Applying the Cauchy integral formula so as to select  $\text{coeff}_{z^{n-i-l+2}} F(z, l)$  from  $F(z, l)$ , we have

$$P(T_1(i, n) \geq l) = [(i-1)/m][(l-2)!/m^{l-2}](2\pi i)^{-1} \int_C G(z) dz,$$

where

$$G(z) = F(z, l)/z^{n-i-l+3} \quad \text{and} \quad C = \{re^{i\theta}; -\pi < \theta \leq \pi\}, \quad r < 1.$$

Outside the contour  $C$ , the function  $G(z)$  has simple poles at  $z_j = (1-j/m)^{-1}, 0 \leq j \leq l-2$ . A direct computation shows that

$$\text{res}_{z=z_j} G(z) = (-1)^{j+1} m^{l-2} (1-j/m)^{n-i} / [j!(l-2-j)!], \quad 0 \leq j \leq l-2.$$

Also, the residue at  $z = \infty$  equals 0 since

$$d \cdot \max \{|G(z)|; |z| = d\} \rightarrow 0 \quad \text{when } d \rightarrow \infty.$$

According to the residue theorem, we then have

$$\begin{aligned} P(T_1(i, n) \geq l) &= -[(i-1)/m][(l-2)!/m^{l-2}] \sum_{j=0}^{l-2} \text{res}_{z=z_j} G(z) \\ (2.3) \quad &= [(i-1)/m] \sum_{j=0}^{l-2} (-1)^j \binom{l-2}{j} (1-j/m)^{n-i}, \quad 2 \leq l \leq n-i+2. \end{aligned}$$

Therefore, Theorem 1 is proven.

The last formula is most convenient when  $l$  is not large. In fact, if  $l \geq 2$  is fixed and  $n-i \rightarrow \infty$ , we get from (2.3):

$$\begin{aligned} P(T_1(i, n) \geq l) &\sim [(i-1)/m] \sum_{j=0}^{l-2} (-1)^j \binom{l-2}{j} \exp[-j(n-i)/m] \\ (2.4) \quad &= [(i-1)/m][1 - \exp(-a + i/m)]^{l-2}, \quad n, m \rightarrow \infty. \end{aligned}$$

It is remarkable that this simple formula holds true even when  $l \rightarrow \infty$  together with  $n$  and  $m$ , slowly enough such that

$$(2.5) \quad l^2/(n-i) = o(1).$$

This result follows from the proof of Lemma 2 in [8]. For the sake of completeness, let us give a sketch of the necessary argument. Denote the sum appearing in (2.3) by  $p(n-i, l-2)$ . Because of the alternating signs in the sum, a direct asymptotic analysis appears impossible. Fortunately, we can get around this obstacle. Suppose that  $(n-i)$  distinguishable balls are allocated independently and uniformly among  $m$  cells. According to the inclusion-exclusion formula,  $p(n-i, l-2)$  happens to be the probability that the first  $(l-2)$  cells are all nonempty. Subsequently,

$$p(n-i, l-2) = \left( \sum_s (n-i)!/[s_1! \cdots s_m!] \right) m^{-(n-i)},$$

where the summation is taken over all nonnegative integer tuples  $s = (s_1, \dots, s_m)$  such that

$$\sum_{i=1}^m s_i = n-i, \quad s_i \geq 1, \quad 1 \leq i \leq l-2.$$

A little reflection shows then that

$$(2.6) \quad p(n-i, l-2) = (n-i)! \operatorname{coeff}_z z^{n-i} [(e^{z/m} - 1)^{l-2} (e^{z/m})^{m-l+2}].$$

Applying the Cauchy integral formula and the saddle-point method, we obtain (via more or less standard arguments) that under the condition (2.5):

$$p(n-i, l-2) \sim [1 - \exp(-a + i/m)]^{l-2}.$$

This, of course, leads again to (2.4).

Finally in this section, consider  $\mathcal{C}_n$  the successful search time of a key chosen uniformly at random among  $n$  keys. Clearly,

$$(2.7) \quad P(\mathcal{C}_n \geq l) = n^{-1} \sum_{i=1}^n P(T_1(i, n) \geq l).$$

Let  $l$  be fixed. Set  $n_1 = [n - \log n]$ . Then according to (2.4), we have

$$\begin{aligned} P(\mathcal{C}_n \geq l) &= n^{-1} \sum_{i=1}^{n_1} P(T_1(i, n) \geq l) + O(\log n/n) \\ &= (1 + o(1)) a^{-1} m^{-1} \sum_{i=1}^{n_1} (i/m) [1 - \exp(-a + i/m)]^{l-2} + o(1) \\ (2.8) \quad &\rightarrow a^{-1} \int_0^a u [1 - \exp(-a + u)]^{l-2} du, \quad n, m \rightarrow \infty. \end{aligned}$$

Since  $P(\mathcal{C}_n \geq 1) = 1$ , the relations show that  $\mathcal{C}_n \rightarrow \mathcal{C}$  where

$$(2.9) \quad \begin{aligned} P(\mathcal{C} \geq l) &= 1 \quad \text{for } l = 1, \\ P(\mathcal{C} \geq l) &= a^{-1} \int_0^a u [1 - \exp(-a + u)]^{l-2} du \quad \text{for } l \geq 2. \end{aligned}$$

Let us demonstrate that, in fact, for every integer  $k \geq 1$ ,

$$(2.10) \quad E(\mathcal{C}_n^k) \rightarrow E(\mathcal{C}^k), \quad n, m \rightarrow \infty.$$

It is easy to see that (2.10) takes place provided that

$$(2.11) \quad \lim_{n,m \rightarrow \infty} S_n(k) = S(k), \quad k = 1, 2, \dots,$$

where

$$S_n(k) = \sum_{l=1}^n l^k P(\mathcal{C}_n \geq l), \quad S(k) = \sum_{l=1}^{\infty} l^k P(\mathcal{C} \geq l).$$

To prove (2.11), set  $\nu = [(k+2)^3 \log_b^3 n]$ , ( $b = (1 - e^{-a})^{-1}$ ) and break the sum as follows:

$$S_n(k) = 1 + S_{n1}(k) + S_{n2}(k) + S_{n3}(k),$$

where

$$S_{n1}(k) = n^{-1} \sum_{i=2}^{n-\nu} \sum_{l=2}^{t(i)} l^k P(T_1(i, n) \geq l), \quad t(i) = [(n-i)^{1/3}],$$

$$S_{n2}(k) = n^{-1} \sum_{i=2}^{n-\nu} \sum_{l=t(i)+1}^{n-i+2} l^k P(T_1(i, n) \geq l),$$

$$S_{n3}(k) = n^{-1} \sum_{i=n-\nu+1}^n \sum_{l=2}^{n-i+2} l^k P(T_1(i, n) \geq l).$$

Consider  $S_{n1}(k)$ . By the definition of  $\nu$  and  $t(i)$ , the range of  $i$  and  $l$  in  $S_{n1}(k)$  is such that

$$l^2/(n-i) \leq t^2(i)/(n-i) \leq (n-i)^{-1/3} \leq \nu^{-1/3} \rightarrow 0, \quad m, n \rightarrow \infty.$$

Hence, by (2.4) and (2.5),

$$(2.12) \quad \begin{aligned} 1 + S_{n1}(k) &= 1 + [1 + o(1)] a^{-1} m^{-1} \sum_{i=2}^{n-\nu} (i/m) \sum_{l=2}^{t(i)} l^k [1 - \exp(-a + i/m)]^{l-2} \\ &\sim 1 + a^{-1} \int_0^a u \left\{ \sum_{l=2}^{\infty} l^k [1 - \exp(-a + u)]^{l-2} \right\} du \\ &= S(k), \end{aligned}$$

where  $S(k) = \sum_{l=1}^{\infty} l^k P(\mathcal{C} \geq l)$  (see (2.9) for  $P(\mathcal{C} \geq l)$ ). Further, the range of  $i$  in  $S_{n2}(k)$  is the same as in  $S_{n1}(k)$ . Since  $P(T_1(i, n) \geq l)$  is a decreasing function of  $l$  and  $t(i) \geq \nu^{1/3} - 1$  for  $i \leq n - \nu$ , we obtain

$$(2.13) \quad \begin{aligned} S_{n2}(k) &\leq n^{-1} \sum_{i=2}^{n-\nu} (n-i) n^k P(T_1(i, n) \geq t(i)) \\ &\leq cn^{k+1} (1 - e^{-a})^{\nu^{1/3}} = cn^{k+1} b^{-(k+2) \log_b n} \\ &= O(n^{-1}), \quad m, n \rightarrow \infty. \end{aligned}$$

Also

$$(2.14) \quad S_{n3}(k) \leq n^{-1} \sum_{i=n-\nu+1}^n (n-i+2)^{k+1} = O(\nu^{k+2}/n) = o(1), \quad m, n \rightarrow \infty.$$

The relations (2.12)–(2.14) imply (2.11). Theorem 2 is proved.

**3. The asymptotic behavior of the largest successful search time.** Recall that  $U_1(n) = \max \{T_1(i, n) : 1 \leq i \leq n\}$ . Our goal here is to prove that the largest successful search time for  $n$  stored random items is

$$U_1(n) = \log_b n - 2 \log_b \log_b n + O_p(1),$$

where  $b = (1 - e^{-a})^{-1}$ . It is plausible to conjecture that

$$E(U_1(n)) = \log_b n - 2 \log_b \log_b n + O(1),$$

but we do not propose to prove it here.

(a) *Upper estimate of  $U_1(n)$ .* Introduce an integer

$$(3.1) \quad l = \log_b n - 2 \log_b \log_b n + \omega(n),$$

where  $\omega(n) = o(\log n)$  as  $\omega(n) \rightarrow \infty$ . Let us show that

$$(3.2) \quad \lim_{n \rightarrow \infty} P(U_1(n) > l) = 0.$$

To this end, write first an obvious inequality

$$P(U_1(n) > l) \leq \sum_{i=1}^n P(T_1(i, n) > l) = \sum_{i=1}^{n-l+1} P(T_1(i, n) > l) = \sum_1 + \sum_2;$$

where  $\sum_1, \sum_2$  correspond, respectively, to sums for  $1 \leq i \leq s, s < i \leq t$ , with

$$s = [n - \log^3 n] \quad \text{and} \quad t = n - l + 1.$$

We will prove that  $\sum_1, \sum_2 \rightarrow 0$ . Begin with  $\sum_1$ , which actually dominates  $\sum_2$ . Since  $l^2/(n-i) \rightarrow 0$  uniformly for  $1 \leq i \leq s$ , by (2.4) we have

$$(3.3) \quad \sum_1 \leq c \sum_{i=1}^s (i/m) [1 - \exp(-a + i/m)]^{l-1} = c \sum_{i=1}^s F(i/m);$$

where  $F(y) = y[1 - \exp(-a + y)]^{l-1}$ , and  $c$  is a constant. It turns out (as shown in the Appendix) that, on the interval  $[0, a]$ ,  $F(y)$  achieves its maximum at

$$(3.4) \quad y_0 \sim (e^a - 1)/l,$$

and

$$(3.5) \quad F(y_0) \leq c_1/(lb^l), \quad \text{with } b = (1 - e^{-a})^{-1}, \text{ and } a = n/m.$$

Now, let  $\sum_{11} = \sum_{i=1}^r F(i/m), \sum_{12} = \sum_{i=r+1}^s F(i/m)$ , with  $r = [2y_0 m]$  so that  $y_0 m < r < s$ . Then

$$\sum_{i=1}^s F(i/m) = \sum_{11} + \sum_{12}.$$

Clearly (see (3.4)),

$$(3.6) \quad \sum_{11} \leq rF(y_0) = O(F(y_0)m/l).$$

On the other hand,

$$F((i+1)/m)/F(i/m) \leq \exp(-c_2 l/m), \quad c_2 > 0,$$

for  $r < i \leq n$  (see the Appendix). Hence,

$$(3.7) \quad \begin{aligned} \sum_{12} &\leq F(y_0) \sum_{j \geq 0} [\exp(-c_2 l/m)]^j = F(y_0) [1 - \exp(-c_2 l/m)]^{-1} \\ &= O(F(y_0)m/l). \end{aligned}$$

Combining (3.3) and (3.5)–(3.7) yields

$$(3.8) \quad \sum_1 = O(F(y_0)m/l) = O(mb^{-l}l^{-2}) = O(b^{-\omega(n)}) = o(1), \quad n \rightarrow \infty.$$

Turn now to  $\sum_2$ . Observe that, by (2.3) and (2.6),

$$\begin{aligned} P(T_1(i, n) > l) &< (n-i)! \text{coeff}_{z^{n-i}} [(e^{z/m} - 1)^{l-1} (e^{z/m})^{m-l+1}] \\ &< (n-i)! \text{coeff}_{z^{n-i}} [(e^{z/m} - 1)^{l-1} e^z] \\ &< (n-i)! [(e^{z/m} - 1)^{l-1} e^z] / z^{n-i}, \quad z > 0. \end{aligned}$$

Select  $z = n - i$ . Since  $n - i = O(\log^3 n)$  for  $i > s = \lceil n - \log^3 n \rceil$ ,

$$(e^{z/m} - 1)^{l-1} = (z/m)^{l-1} [1 + O(\log^4 n/m)] \leq 2(\log^3 n/m)^{l-1}.$$

So, by Stirling’s formula applied to  $(n - i)!$ ,

$$\begin{aligned} P(T_1(i, n) > l) &\leq 2(n-i)! (e/n - i)^{n-i} (\log^3 n/m)^{l-1} \\ &\leq c_2 \log^{3/2} n (\log^3 n/m)^{l-1}, \end{aligned}$$

and

$$(3.9) \quad \sum_2 \leq c_3 \log^{9/2} n (\log^3 n/m)^{l-1} \rightarrow 0, \quad m, n \rightarrow \infty.$$

The relations (3.8), (3.9) imply (3.2).

(b) *Lower estimate of  $U_1(n)$ .* Introduce an integer

$$l = \log_b n - 2 \log_b \log_b n - \omega(n),$$

where  $\omega(n) = o(\log n)$  as  $\omega(n) \rightarrow \infty$ . Let us show that

$$(3.10) \quad \lim P(U_1(n) > l) = 1.$$

(1) Introduce an auxiliary random variable  $X(n, l)$ , which is the total number of chains induced by insertion of all  $n$  keys, whose lengths exceed  $l$ . Since both versions of coalesced hashing yield the same—though differently ordered—chains, we can use the results obtained in [8] for the lich algorithm. First of all,

$$X(n, l) = \sum_B \delta(B),$$

where the sum is taken over all  $(l + 1)$ -element subsets  $B = \{x_{i_1} < \dots < x_{i_{l+1}}\}$  of the set of  $n$  keys  $\{x_1, \dots, x_n\}$ , and  $\delta(B) \in \{0, 1\}$  is defined as follows:  $\delta(B) = 1$  if and only if, after the key  $x_{i_{l+1}}$  is inserted, the cells occupied by the keys  $x_{i_1}, \dots, x_{i_{l+1}}$  constitute a subchain which originates at the cell occupied by the key  $x_{i_1}$  (and then, of course,  $h(x_{i_i}) = \tilde{h}(x_{i_i})$  for either of the algorithms). Hence

$$(3.11) \quad E(X(n, l)) = \sum_B E(\delta(B)) = \sum_B P(\delta(B) = 1),$$

where, similarly to (2.2),

$$\begin{aligned} P(\delta(B) = 1) &= [1 - (i_1 - 1)/m] [(1 - 1/m)^{i_2 - i_1 - 1} / m] \dots [(1 - l/m)^{i_{l+1} - i_l - 1} / m] \\ (3.12) \quad &= (l! / m^l) (1 - s_0/m) \prod_{j=1}^l (1 - j/m)^{s_j} \quad \text{with } s_j = i_{j+1} - i_j - 1 \quad (i_0 = 0). \end{aligned}$$

Also, by Lemma 2 in [8],

$$(3.13) \quad E(X(n, l)) = (1 + o(1))cm / (lb^l), \quad b = (1 - e^{-a})^{-1},$$

where  $c$  is a positive constant. The relations (3.11)–(3.13) will be needed shortly.

(2) To proceed we need a notion of a special chain. Consider a chain of length  $\geq l + 1$ , and let  $x_{i_1} < \dots < x_{i_{l+1}}$  be the first  $(l + 1)$  keys whose locations are absorbed by this chain. According to the rule, it means that  $h(x_{i_1}) = \tilde{h}(x_{i_1}) = h(x_{i_2}) \neq \tilde{h}(x_{i_2})$ ,  $h(x_j) \neq h(x_{i_1})$  for  $i_1 < j < i_2$ , or that insertion of  $x_{i_1}$  starts a new chain, while  $x_{i_2}$  is the first key whose insertion results in collision with  $x_{i_1}$  and forms a chain of length 2. We call such a chain special if insertions of all the subsequent keys  $x_{i_3}, \dots, x_{i_{l+1}}$  have resulted in pushing the cell  $\tilde{h}(x_{i_2})$  along the chain farther and farther from the cell  $h(x_{i_1})$ , so that once the key  $x_{i_{l+1}}$  has been inserted the length of the subchain connecting the cells  $h(x_{i_1}) (= h(x_{i_2}))$  and  $\tilde{h}(x_{i_2})$  becomes, for the first time, equal to  $(l + 1)$ .

Denote the total number of special chains by  $Y(n, l)$ . It should be clear from the definition above that

$$\{U_1(n) > l\} \supset \{Y(n, l) > 0\}.$$

Thus, the relation (3.10) will be proved if we can show that

$$\lim_{n \rightarrow \infty} P(Y(n, l) > 0) = 1.$$

But, by a corollary of Chebyshev's inequality,

$$P(Y(n, l) > 0) \geq E^2(Y(n, l)) / E(Y^2(n, l));$$

so, it remains to prove that

$$(3.14) \quad E(Y^2(n, l)) \leq [1 + o(1)]E^2(Y(n, l)), \quad n, m \rightarrow \infty.$$

To evaluate the first two moments of  $Y(n, l)$ , write first

$$Y(n, l) = \sum_B \varepsilon(B);$$

here the summation is taken over all  $(l + 1)$ -element subsets  $B = \{x_{i_1} < \dots < x_{i_{l+1}}\}$  of  $\{x_1, \dots, x_n\}$ ,  $\varepsilon(B) \in \{0, 1\}$  and  $\varepsilon(B) = 1$  if and only if  $x_{i_1}, \dots, x_{i_{l+1}}$  are the first  $(l + 1)$  keys whose locations are absorbed by a special chain. Similarly to (2.2) and (3.12), we have

$$\begin{aligned} P(\varepsilon(B) = 1) &= [1 - (i_1 - 1)/m][(1 - 1/m)^{i_2 - i_1 - 1} / m] \\ &\quad \cdot [(1 - 2/m)^{i_3 - i_2 - 1} / m] \cdot \dots \cdot [(1 - l/m)^{i_{l+1} - i_l - 1} (l - 1)/m] \\ &= P(\delta(B) = 1) / l = E(\delta(B)) / l. \end{aligned}$$

By (3.11) and (3.13), it follows then that

$$\begin{aligned} E(Y(n, l)) &= \sum_B E(\varepsilon(B)) = l^{-1} \sum_B E(\delta(B)) \\ &= l^{-1} E(X(n, l)) = [1 + o(1)]cm / (l^2 b^l) \\ &= [1 + o(1)]ca^{-1} b^{\omega(n)}. \end{aligned}$$

Turn now to the second-order moment. First observe that

$$\varepsilon(B_1) \cdot \varepsilon(B_2) = 0 \quad \text{if } B_1 \neq B_2 \quad \text{and} \quad B_1 \cap B_2 \neq \emptyset.$$

Subsequently,

$$(3.15) \quad E(Y^2(n, l)) = E(Y(n, l)) + \sum_{B_1, B_2} P(\varepsilon(B_1) = 1, \varepsilon(B_2) = 1),$$

where the summation is taken over all ordered pairs of disjoint subsets  $B_1, B_2$  of  $\{x_1, \dots, x_n\}$  with  $|B_1| = |B_2| = l + 1$ .

To estimate the sum in (3.15) we need the following lemma.

LEMMA. For every pair of disjoint  $(l + 1)$ -element subsets  $B_1, B_2$ ,

$$P(\varepsilon(B_1) = 1, \varepsilon(B_2) = 1) \leq c(l, m)P(\varepsilon(B_1) = 1)P(\varepsilon(B_2) = 1),$$

where

$$c(l, m) = [1 - 2(l + 1)/m]^{-2(l+1)}.$$

(This inequality is similar to Lemma 3, for the lich algorithm, proven in [8] and we omit its proof.)

By (3.15) and the lemma,

$$\begin{aligned} E(Y^2(n, l)) &\leq E(Y(n, l)) + c(l, m) \sum_{B_1, B_2} P(\varepsilon(B_1) = 1) \cdot P(\varepsilon(B_2) = 1) \\ &\leq E(Y(n, l)) + c(l, m)E^2(Y(n, l)) \\ &= E^2(Y(n, l))[O(b^{-\omega(n)}) + 1 + O(l^2/m)] \\ &= (1 + o(1))E^2(Y(n, l)), \end{aligned}$$

as  $\omega(n) \rightarrow \infty$ . Thus, the relation (3.14) is proven, and it implies the relation (3.10).

Combination of (3.2) and (3.10) enables us to conclude that, for every  $\omega(n)$ :

$$P(|U_1(n) - \log_b n + 2 \log_b \log_b n| \leq \omega(n)) \rightarrow 1 \quad \text{as } \omega(n) \rightarrow \infty$$

which proves Theorem 3.

**Appendix.** Consider a function  $F(y) = y[1 - \exp(-a + y)]^{l-1}$ ,  $y \in (0, a)$ . Denote  $\log F(y)$  by  $h(y)$ .

(i) Since

$$h'(y) = y^{-1} - (l - 1)[\exp(a - y) - 1]^{-1} \rightarrow \begin{cases} \infty & \text{if } y \rightarrow 0, \\ -\infty & \text{if } y \rightarrow a, \end{cases}$$

there must exist  $y_0 \in (0, a)$ , where  $F(y)$  achieves its maximum and

$$y_0^{-1} - (l - 1)[\exp(a - y_0) - 1]^{-1} = 0.$$

(This point is unique since  $h''(y) < 0$ ,  $y \in (0, a)$ .) Let  $l \rightarrow \infty$ ; it follows then that  $y_0 \rightarrow 0$  so

$$y_0 = [1 + o(1)](e^a - 1)/l, \quad m, n \rightarrow \infty.$$

Furthermore,

$$\begin{aligned} h(y_0) &= \log y_0 + (l - 1) \log [1 - \exp(-a + y_0)] \\ &= -\log l + \log(e^a - 1) + o(1) + (l - 1) \log(1 - e^{-a}) - 1 + o(1), \end{aligned}$$

so

$$F(y_0) = (1 + o(1))c/(lb^l), \quad (b = (1 - e^{-a})^{-1}, c = e^{a-1}).$$

(ii) Note that  $h'(y)$  is a decreasing function, since  $h''(y) < 0$ . So, for  $[2y_0m] + 1 \leq i \leq n$ ,

$$\begin{aligned} h[(i + 1)/m] - h(i/m) &\leq h'(2y_0)/m \\ &= \{(2y_0)^{-1} - (l - 1)[\exp(a - 2y_0) - 1]^{-1}\}/m \\ &= [(2y_0)^{-1} - y_0^{-1}(1 + o(1))]/m \\ &\leq -(3y_0m)^{-1} \leq -c_1l/m, \quad c_1 > 0, \end{aligned}$$

whence

$$F[(i+1)/m]/F(i/m) \leq \exp(-c_1 i/m).$$

**Acknowledgments.** We wish to express our gratitude to the referees for many insightful comments which helped us to improve the presentation of the results. One of the referees took time to read through two consecutive versions of the paper, and we are much indebted to him for his meticulous care about essence and details of the paper.

#### REFERENCES

- [1] W.-C. CHEN AND J. S. VITTER, *Analysis of early insertion standard coalesced hashing*, this Journal, 12 (1983), pp. 667-676.
- [2] ———, *Analysis of new variants of coalesced hashing*, ACM Trans. Database Systems, 9 (1984), pp. 616-645.
- [3] L. DEVROYE, *The expected length of the longest probe sequence for bucket searching when the distribution is not uniform*, J. Algorithms, 6 (1985), pp. 1-9.
- [4] D. M. GREENE AND D. E. KNUTH, *Mathematics for the Analysis of Algorithms*, 2nd edition, Birkhäuser, Boston, 1982.
- [5] G. H. GONNET, *Expected length of the longest probe sequence in hash code searching*, J. Assoc. Comput. Mach., 25 (1981), pp. 289-304.
- [6] D. E. KNUTH, *The Art of Computer Programming*, 3: *Sorting and Searching*, Section 6.4, Addison-Wesley, Reading, MA, 1973.
- [7] G. D. KNOTT, *Direct-chaining with coalescing lists*, J. Algorithms, 5 (1984), pp. 7-21.
- [8] B. PITTEL, *On probabilistic analysis of a coalesced hashing algorithm*, Ann. Probab., 15 (1987), pp. 1180-1202.
- [9] ———, *Linear probing: the probable largest search time grows logarithmically with the number of records*, J. Algorithms, 8 (1987), pp. 236-249.
- [10] J. S. VITTER, *Analysis of the search performance of coalesced hashing*, J. Assoc. Comput. Mach., 30 (1983), pp. 231-258.
- [11] ———, *Analysis of coalesced hashing*, Ph.D. dissertation, Dept. of Computer Science, Technical Rep. STAN-CS-80-817, Stanford University, Stanford, CA, 1980.
- [12] ———, *Implementations for coalesced hashing*, Commun. ACM, 25 (1982), pp. 911-926.



## LOWNESS PROPERTIES OF SETS IN THE EXPONENTIAL-TIME HIERARCHY\*

R. BOOK†, P. ORPONEN‡, D. RUSSO§, AND O. WATANABE¶

**Abstract.** The notion of “lowness” was introduced in computational complexity theory by Schöning [J. Comput. Systems Sci., 27 (1983), pp. 14–28] who studied sets in the class NP. This notion may be interpreted as setting an upper bound on the amount of information that can be encoded by a set. Here ideas from previous studies are incorporated in order to capture the notion of a set being exponentially low.

The main result asserts the existence of a sparse set  $E$  such that  $\text{DEXT}(E) = \text{DEXT}$ , i.e.,  $E$  is “exponentially low,” but  $E$  is not in the class  $P$ . In contrast, any set with small generalized Kolmogorov complexity that is exponentially low must be in the class  $P$ . In addition, we show that for each  $k \geq 2$ , any sparse set  $S$  that is low with respect to the class  $\Sigma_k^E$  of the exponential-time hierarchy (i.e.,  $\Sigma_k^E(S) = \Sigma_k^E$ ) must be in the class  $\Sigma_k^P$  of the polynomial-time hierarchy. Similarly, for each  $k \geq 4$ , any set with polynomial-size circuits that is low with respect to the class  $\Sigma_k^E$  must be in the class  $\Sigma_k^P$ .

**Key words.** exponential lowness, polynomial lowness, small generalized Kolmogorov complexity, sparse sets

**AMS(MOS) subject classifications.** 68Q15, 68Q30, 03D15

**1. Introduction.** How does one measure the complexity of a set of strings? The most common approach is, of course, to study the inherent computational complexity of the characteristic function of the set; for example, a set that can be recognized in exponential time but not in polynomial time is considered to be more complex than a set that can be recognized in polynomial time. A different idea is to consider a function that bounds the Kolmogorov complexity of the strings in the set. Then a set with large Kolmogorov complexity is considered to be more complex than a set with small Kolmogorov complexity. A third method is to consider how much the set, when used as an oracle, “helps” different types of computations; for example, if  $\text{NP}(A) = \Sigma_2^P$  but  $\text{NP}(B) = \Sigma_3^P$ , then  $B$  is considered to be more complex than  $A$ .

The purpose of this paper is to consider a specific case of the third method and to compare that with the first. More specifically, we consider the following question: if a set  $A$  has the property that  $\text{DEXT}(A) = \text{DEXT}$ , does this mean that  $A$  is in  $P$ ? Here,  $\text{DEXT}(A)$  denotes the class of sets reducible to  $A$  by means of Turing reducibilities that are computed deterministically in exponential time. Any set  $A$  with the property that  $\text{DEXT}(A) = \text{DEXT}$  is called *DEXT-low* (informally, “exponentially low”). Not every set in  $\text{DEXT}$  is exponentially low since  $\text{DEXT}(\text{DEXT}) \neq \text{DEXT}$ . Being exponentially low means that the set in question cannot encode a great deal of information since applying the  $\text{DEXT}(\ )$ -operator to that set does not produce sets not already in  $\text{DEXT}$ . Every set in  $P$  is already exponentially low, so the question of whether every exponentially low set is in  $P$  is equivalent to asking whether the property of being exponentially low characterizes the sets in  $P$ . We will show that the answer

---

\* Received by the editors July 31, 1985; accepted for publication (in revised form) July 28, 1987. This research was supported in part by the National Science Foundation under grants DCR-8312472 and CCR-8611980. The results presented here were announced at the 13th International Colloquium on Automata, Languages and Programming, Rennes, France, July 1986.

† Department of Mathematics, University of California, Santa Barbara, California 93106.

‡ Department of Computer Science, University of Helsinki, Tukholmankatu 2, SF-00250 Helsinki 25, Finland. The work of this author was supported in part by the Academy of Finland.

§ Digital Sound, Inc., Santa Barbara, California 93103.

¶ Department of Computer Science, Tokyo Institute of Technology, Ookayama, Meguro-ku, Tokyo 152, Japan.

to the question is “no,” that is, there exist sets  $A$  such that  $A$  is exponentially low but  $A \in \text{DEXT}-P$ : we will exhibit a sparse set  $A$  with this property. To accomplish this we consider sets whose generalized Kolmogorov complexity is bounded by certain functions, so that the second method of characterizing the information encoded by a set of strings is also considered.

The notion of “lowness” was introduced in computational complexity theory by Schöning [14] who called a set  $A$  in NP “low” if for some  $i$ ,  $\Sigma_i^P(A) = \Sigma_i^P$ . Ko and Schöning [12] showed that if  $A \in \text{NP}$  is sparse, then  $\Sigma_2^P(A) = \Sigma_2^P$ , and if  $A \in \text{NP}$  has polynomial-size circuits, then  $\Sigma_3^P(A) = \Sigma_3^P$ . Balcázar, Book, and Schöning [2] introduced a more general type of lowness in the context of sets in the union of the polynomial-time hierarchy. In general, the notion of lowness with respect to the classes  $\Sigma_i^P$ ,  $i > 0$ , may be interpreted as setting an upper bound on the amount of information that can be encoded in the set, that is, the power of a low set in the polynomial-time hierarchy is subsumed by a bounded number of alternating quantifiers or, equivalently, a bounded number of applications of the NP ( )-operator. More formally, if  $A \in \Sigma_n^P - \Sigma_{n-1}^P$  and for every  $i$ ,  $\Sigma_i^P(A) \subseteq \Sigma_{i+j}^P$  where  $j < i + n$ , then  $A$  can encode information equivalent to at most  $j$  alternating quantifiers. Here we incorporate ideas from the previous studies in order to capture the notion of a set being exponentially low.

Other types of lowness can be studied in the context of classes in the exponential hierarchy. In this direction we have several results about sets with “compact descriptions.” (The notion of a set having a “compact description” is similar to notions of “language compression by machine” which extend the classical data compression problem (see [6]).) We show that if a set with a compact description is low with respect to certain classes in the exponential-time hierarchy, then it is also low with respect to the corresponding classes in the polynomial-time hierarchy.

We establish the following results.

**THEOREM 3.3.** *A set with small generalized Kolmogorov complexity (this includes the tally sets) is DEXT-low if and only if it is in  $P$ .*

**THEOREM 4.5.** *There is a sparse set  $E \in \text{DEXT}-P$  that is DEXT-low.*

**THEOREM 5.5.** *For  $k \geq 2$ , a sparse set is  $\Sigma_k^E$ -low if and only if it is  $\Sigma_k^P$ -low.*

**THEOREM 5.6.** *For  $k \geq 4$ , a set with polynomial-size circuits is  $\Sigma_k^E$ -low if and only if it is  $\Sigma_k^P$ -low.*

Recall that there is a fundamental difference between the relativizations of polynomial-time classes and the relativizations of exponential-time classes (at least if the standard model with unrestricted access to the oracle is used). For example, while  $P(P) = P$  and  $\text{NP}(P) = \text{NP}$ , it is the case that  $\text{DEXT}(\text{DEXT}) \neq \text{DEXT}$  and  $\text{NEXT}(\text{DEXT}) \neq \text{NEXT}$ ; also, while  $\text{NP}(\text{NP}) = \Sigma_2^P$ ,  $\text{NEXT}(\text{NEXT}) \neq \Sigma_2^E$ . Because of this difference, strange anomalies arise: exponential time Turing reducibility is not a transitive relation; it is possible for the exponential-time hierarchy to collapse at some level (i.e.,  $\Sigma_i^E = \text{co-}\Sigma_i^E$ ) but be proper at a higher level (i.e.,  $\Sigma_j^E \neq \text{co-}\Sigma_j^E$  for some  $j > i$ ) [17], [9]; and it is possible to construct an oracle set such that relative to that set the exponential-time hierarchy and the alternating space classes are different [13], in contrast to the unrelativized case [4].

Basically, all of these difficulties arise from the fact that the class of single exponential functions is not closed under composition. Hence, when an oracle machine makes a query of exponential length to an exponentially complex oracle set, the amount of information it obtains is in a sense double exponential. However, this explanation is not helpful in increasing our understanding of exponential relativizations. We would like to know how this basic arithmetical fact is reflected in the structure of the sets causing the anomalous behavior, that is, in the sets in DEXT which are *not* DEXT-low.

This paper is organized in the following way. Section 2 contains notation and some facts about sets with small generalized Kolmogorov complexity. In § 3, exponentially low oracle sets are discussed. The main result is contained in § 4. A set  $E$  is defined and shown to be in  $\text{DEXT-}P$ ; in Theorem 4.4 it is shown that the set  $E$  is  $\text{DEXT-low}$ . In § 5, we consider lowness with respect to levels of the polynomial-time hierarchy and of the exponential-time hierarchy. Section 6 is devoted to remarks and open questions.

**2. Preliminaries and generalized Kolmogorov complexity.** We use  $\Sigma = \{0, 1\}$  as our alphabet. For each integer  $n \geq 0$ , let  $\text{bin}(n)$  be a standard unique binary representation of  $n$  over  $\Sigma$ . Let  $\mathbf{N}$  and  $\mathbf{bin}(\mathbf{N})$  denote the set of all nonnegative integers and  $\{\text{bin}(n) \mid n \in \mathbf{N}\}$ , respectively. Denote the length of  $x \in \Sigma^*$  by  $|x|$  and denote the cardinality of set  $S$  by  $\|S\|$ .

A set  $T$  is a *tally* set if  $T \subseteq \{0\}^*$ . For each tally set  $T$ ,  $\text{bin}(T)$  denotes  $\{\text{bin}(n) \mid O^n \in T\}$ . For each set  $B \subseteq \Sigma^*$ ,  $\text{TALLY}(B)$  denotes  $\{O^n \mid \text{bin}(n) \in B\}$ .

A set  $S$  is *sparse* if there exists a polynomial  $q$  such that for all  $n$ ,  $\|\{s \in S \mid |s| \leq n\}\| \leq q(|n|)$ .

We use standard deterministic and nondeterministic time-bounded oracle machines that act as acceptors. If  $M$  is an acceptor, then  $L(M, A)$  denotes the set of strings recognized relative to oracle set  $A$ , and  $L(M)$  is written for  $L(M, \emptyset)$ . We assume a standard enumeration of such machines, say  $M_1, M_2, \dots$ , indicating by context whether we are discussing the class of deterministic machines or the class of nondeterministic machines. A function is *time constructible* if it is the running time of some deterministic Turing machine.

We also consider standard deterministic time-bounded Turing machines that act as transducers. We assume a standard enumeration of such transducers, say  $N_1, N_2, \dots$ . For each  $i$ , let  $T_i$  be the running time of the transducer  $N_i$ , let  $f_i$  be the function computed by the transducer  $N_i$ , and let  $R_i$  denote the range of  $f_i$ . We assume that the enumeration has the property that there exists a *universal Turing transducer*  $N_u$  and a *description function*  $d$  with the following properties: for every  $i > 0$  there is a constant  $c_i$  such that for all  $x \in \Sigma^*$ ,

- (a)  $d(i)$  is not a prefix of  $d(j)$  if  $i \neq j$ ,
- (b)  $f_u(d(i)x) = f_i(x)$ , and
- (c)  $T_u(d(i)x) \leq c_i \cdot T_i(x) \cdot \log T_i(x) + c_i$ .

Hartmanis [8] introduced a “generalized Kolmogorov complexity measure.” Informally, it measures how far and how fast a string can be compressed. More formally, for each pair  $g, t$  of functions, we consider the following sets:

$$K_i[g(n), t(n)] = \{y \mid (\exists x)[|x| \leq g(n), f_i(x) = y, \text{ and } T_i(x) \leq t(n)]\}, \text{ where } n = |y|,$$

$$K[g(n), t(n)] = K_u[g(n), t(n)], \text{ where } u \text{ denotes the index of the universal Turing transducer considered above.}$$

We have the following fact.

LEMMA 2.1 [8]. *Let  $i$  be any index and let  $g(n)$  and  $t(n)$  be time-constructible functions. Then there exists  $c > 0$  and  $d > 0$  such that  $K_i[g(n), t(n)] \leq K[g(n) + d, c \cdot t(n) \cdot \log t(n) + c]$ .*

This lemma shows that the generalized Kolmogorov complexity defined by the universal Turing machine  $N_u$  is within small factors of any other complexity defined by a different Turing machine. Thus we will use the generalized Kolmogorov complexity defined by  $N_u$  to state our results.

We will say that a set  $A$  has *small generalized Kolmogorov complexity* if the elements of  $A$  have uniformly bounded generalized Kolmogorov complexity that is small in the sense that there exist constants  $c > 0$  and  $d > 0$  such that  $A \subseteq K[d \cdot \log n, n^c]$ . Properties of sets with small generalized Kolmogorov complexity are investigated in [1], [8], [19].

We assume that the reader is familiar with the classes  $P$ ,  $NP$ , and  $\Sigma_i^P$  for  $i \geq 0$ , and their relativizations to an arbitrary oracle set  $A$ ,  $P(A)$ ,  $NP(A)$ , and  $\Sigma_i^P(A)$ , respectively. We will consider the classes  $DEXT = \{L(M) \mid M \text{ is a deterministic acceptor that runs in time } 2^{cn} \text{ for some } c > 0\}$  and  $NEXT = \{L(M) \mid M \text{ is a nondeterministic acceptor that runs in time } 2^{cn} \text{ for some } c > 0\}$ , and will consider their relativizations to an arbitrary oracle set  $A$ ,  $DEXT(A)$ , and  $NEXT(A)$ .

**3. Exponentially low oracle sets.** Let  $\mathcal{C}$  be an operator that maps  $2^{\Sigma^*}$  into  $2^{2^{\Sigma^*}}$  with the property that for every set  $A \subseteq \Sigma^*$ ,  $\mathcal{C}(\emptyset) \subseteq \mathcal{C}(A)$ . For example,  $P(A) = \{B \mid B \leq_T^P A\}$ . We write  $\mathcal{C}$  for  $\mathcal{C}(\emptyset)$ . A set  $A \subseteq \Sigma^*$  is  $\mathcal{C}$ -low if  $\mathcal{C}(A) = \mathcal{C}$ .

We are particularly interested in the class of  $DEXT$ -low sets. Observe that only sets in  $DEXT$  can be  $DEXT$ -low. But there are sets in  $DEXT$  which are known *not* to be  $DEXT$ -low since  $DEXT(DEXT) = DDEXT$  (recall that  $DDEXT = \bigcup \{DTIME(2^{2^{cn}}) \mid c > 0\}$ ) which is a proper superset of  $DEXT$ . We have the following fact.

**PROPOSITION 3.1.** *Every set in  $P$  is  $DEXT$ -low.*

*Proof.* Let  $A \in P$  and let  $M_1$  be a machine recognizing  $A$  that runs in time  $n^c$ . Let  $B \in DEXT(A)$  and let  $M_2$  be an oracle machine such that  $L(M_2, A) = B$  and  $M_2$  runs in time  $2^{dn}$ . Membership in  $B$  can be decided by a machine  $M_3$  that simulates  $M_2$  but instead of querying the oracle simply simulates  $M_1$ . Clearly, such an  $M_3$  need have running time only  $O(2^{dn} \cdot (2^{dn})^c) = O(2^{d(1+c)n})$  (since on input of length  $n$ ,  $M_2$  can make at most  $2^{dn}$  oracle queries, each of length at most  $2^{dn}$ ). Thus,  $M_3$  witnesses  $B \in DEXT$ .  $\square$

We are interested in the structure of sets that are  $DEXT$ -low. Our next result (and the results in § 5) show that sets that are  $DEXT$ -low but not in  $P$  must be structurally complex.

**PROPOSITION 3.2.** *A set  $A$  is  $DEXT$ -low if and only if there are no tally sets in  $P(A) - P$ .*

*Proof.* Let  $A$  be  $DEXT$ -low so that  $DEXT(A) = DEXT$ , and suppose that  $T$  is a tally set in  $P(A)$ . Then the set  $\text{bin}(T)$  can be seen to be in  $DEXT(A)$ : an input string  $x = \text{bin}(n)$  can be expanded to be in the form  $1^n$  in time  $O(2^{c|x|})$  and whether  $1^n$  is in  $T$  can be decided relative to  $A$  in time  $O(n^d) = O(2^{d|x|})$ . But since  $DEXT(A) = DEXT$ , there is an unrelativized decision procedure for  $\text{bin}(T)$  that runs in time  $O(2^{e|x|})$  and this procedure can be used to decide membership in  $T$  in time  $O(2^{e|\text{bin}(n)|}) = O(n^e)$ . Hence,  $T$  is in  $P$ .

Conversely, let  $A$  be a set that is not  $DEXT$ -low and let  $B \in DEXT(A) - DEXT$ . We claim that the set  $\text{TALLY}(B)$  is in  $P(A) - P$ . A decision procedure for membership in  $B$  running in time  $O(2^{e|x|})$  relative to  $A$  can be used to decide membership in  $\text{TALLY}(B)$  in time  $O(2^{e|\text{bin}(n)|}) = O(n^e)$  relative to  $A$ ; hence,  $\text{TALLY}(B) \in P(A)$ . On the other hand, if  $\text{TALLY}(B)$  had an unrelativized decision procedure running in time  $O(n^d)$ , we could obtain from this a decision procedure for  $B$  running in time  $O(2^{d|x|})$  so that  $B \in DEXT$ , contrary to our hypothesis. Hence,  $\text{TALLY}(B)$  is not in  $P$ .  $\square$

The proof of Proposition 3.2 is essentially that given by Book [3]. It is a simple example of a more general technique that will be used in § 5.

Recall that every tally set has small generalized Kolmogorov complexity. This fact suggests the following result.

**THEOREM 3.3.** *Let  $A$  be a set with small generalized Kolmogorov complexity. Then  $A$  is DEXT-low if and only if  $A$  is in  $P$ .*

*Proof.* The “if” portion follows from Proposition 3.1. To prove the “only if” portion, assume that  $A$  has small generalized Kolmogorov complexity and  $A$  is not in  $P$ . There exist  $c, d > 0$  such that  $A \subseteq K[d \cdot \log n, n^c]$ . We will show that  $\text{DEXT}(A) \neq \text{DEXT}$  so that  $A$  is not DEXT-low.

Let  $N_0$  be an exponential time-bounded transducer such that the function  $f_0$  computed by  $N_0$  is defined by

$$f_0(x) = \begin{cases} f_U(x) & \text{if } N_U \text{ halts on } x \text{ within } 2^{(c/d)|x|} \text{ steps,} \\ 0 & \text{otherwise,} \end{cases}$$

where  $f_U$  and  $N_U$  are the universal function and machine, respectively (as in § 2).

Let  $L_A = \{x \mid f_0(x) \in A\}$ . Then  $L_A \in \text{DEXT}(A)$ . Notice that  $f_0$  enumerates all strings of Kolmogorov complexity less than or equal to that of the members of  $A$ . The language  $L_A$  consists of a set of witnesses for the small generalized Kolmogorov complexity of the elements of  $A$ . Assuming that  $L_A$  is in DEXT will yield the contradiction.

Let  $y$  be an element of  $A$ ; then,  $y \in K[d \cdot \log k, k^c]$  where  $k = |y|$ . Hence, there exists some  $x$  such that  $|x| \leq d \cdot \log k$  and on input  $x$ ,  $N_U$  outputs  $y$  and halts within  $k^c$  (i.e.,  $2^{(c/d)|x|}$ ) steps; hence,  $f_0(x) = y$ . This means that for all  $y, y \in A$  if and only if  $(\exists x)[|x| \leq d \cdot \log |y| \text{ and } f_0(x) = y \text{ and } x \in L_A]$ . Using this fact we can construct a polynomial time-bounded deterministic acceptor for  $A$  that for  $y$  behaves as follows:

```
begin
  input y;
  for all x such that  $|x| \leq d \cdot \log |y|$  do
    if  $x \in L_A$  and  $f_0(x) = y$  then halt and accept
end.
```

Therefore,  $A \in P$ , contradicting the hypothesis.  $\square$

**COROLLARY 3.4.** *For every tally set  $T$ ,  $T$  is DEXT-low if and only if  $T$  is in  $P$ .*

The reader should note that Corollary 3.4 demonstrates that Theorem 3.3 generalizes Proposition 3.2.

Notice that there is a tally set in DEXT that is *not* DEXT-low: if  $L$  is a set that is  $\leq_m^P$ -complete for DDEXT, the set  $T = \text{TALLY}(L)$  is not in  $P$  since  $\text{DEXT} \neq \text{DDEXT}$  so by Corollary 3.4,  $T$  is not DEXT-low. Theorem 3.3 can be deduced from Corollary 3.4 by means of the following fact.

A *polynomial semi-isomorphism* from set  $A$  to set  $B$  is a function  $f$  computable in polynomial time such that  $f$  witnesses the fact that  $A \leq_m^P B$  and the restriction of  $f$  to  $A$  is a polynomially invertible bijection from  $A$  to  $B$ . Balcázar and Book [1] showed that a set has small generalized Kolmogorov complexity if and only if it is polynomially semi-isomorphic to a tally set. Later, Allender and Rubinfeld [19] showed that “semi-isomorphic” could be replaced by “isomorphic.”

**4. There are DEXT-low sets that are not in  $P$ .** Here we establish the main result, showing the existence of a set that is DEXT-low but is not in  $P$ . We saw in § 3 that any set that is both DEXT-low and not in  $P$  cannot have small generalized Kolmogorov complexity and must contain infinitely many complex (noncompressible) strings. Recall the result of Allender and Rubinfeld [19] that a set has small generalized Kolmogorov complexity if and only if it is polynomially isomorphic to a tally set. A set that is polynomially isomorphic to a tally set must be sparse. The main result shows that there is a sparse set that is DEXT-low but not in  $P$ , so this sparse set cannot be polynomially

isomorphic to any tally set. (The reader may wish to review § 2 for notation on Kolmogorov complexity.)

Let  $\varepsilon > 0$  be arbitrarily fixed, and consider the set  $K[n/2, 2^{(1+\varepsilon)n}]$ . For brevity, we let  $K$  denote  $K[n/2, 2^{(1+\varepsilon)n}]$  and  $\bar{K}$  denote the complement of  $K$ . A simple counting argument is enough to show that for every  $n > 0$ , there exists at least one string in  $\bar{K}$  of length  $n$ : there are not enough strings in  $\Sigma^*$  of length  $n/2$  or less to map onto all of the strings of length  $n$ .

Intuitively,  $\bar{K}$  is a good candidate for a DEXT-low set in DEXT- $P$ . What follows is an intuitive explanation of our idea. Note that  $\bar{K} \in \text{DEXT}$ . Thus, if the lengths of queries made by a deterministic exponential time-bounded oracle machine  $M$  are small, e.g., linear bounded with respect to the length of the input, then it can be simulated deterministically by an exponential time-bounded machine that makes no oracle queries. On the other hand, if  $M$  makes a long query with respect to the length of the input, then this query may be computable from a short description in exponential time and may be in  $K$  (i.e., not in  $\bar{K}$ ). Thus, the answer to this query may be “no” (i.e., we do not need the oracle here). The next lemma establishes this type of property for the set  $\bar{K}$ .

LEMMA 4.1. *Let  $N_i$  be any exponential time-bounded transducer. There exists a  $d_i$  such that for all but finitely many  $x$ ,  $|f_i(x)| > d_i \cdot |x|$  implies that  $f_i(x) \in K$ .*

*Proof.* Let  $c$  be a constant such that  $N_i$  runs in time  $2^{cn}$ . Let  $d > 1$  be arbitrary. For any  $x$  such that  $|f_i(x)| > d|x|$ , we have  $f_i(x) \in K_i[n/d, 2^{c(n/d)}]$ . From Lemma 2.1 we see that  $K_i[n/d, 2^{c(n/d)}] \subseteq K[n/d + d', c'(n/d) \cdot 2^{c(n/d)}]$ , where  $d'$  and  $c'$  are constants determined by  $N_i$ .

Choose a constant  $d_i$  such that  $d' + n/d_i \leq n/2$  and  $c'(n/d_i) \cdot 2^{c(n/d_i)} < 2^n$  for all but finitely many  $n$ . Then we have  $K[n/d_i + d', c'(n/d_i) \cdot 2^{c(n/d_i)}] \subseteq K[n/2, 2^{(1+\varepsilon)n}]$  for all but finitely many  $n$ . Therefore, for all but finitely many  $n$  such that  $|f_i(x)| > d_i|x|$ , we have  $f_i(x) \in K[n/2, 2^{(1+\varepsilon)n}] = K$ .  $\square$

However, we cannot prove the DEXT-lowness of  $\bar{K}$  in this way. The machine  $M$  may use some information obtained by answers to the previous queries. So the query may not result from a short description in exponential time even if it is long. But if the oracle is very sparse, we can compress this information.

Let  $E$  be any infinite subset of  $\bar{K}$  in DEXT with the property that for each  $n > 0$ ,  $E$  contains at most one element of length between  $n$  and  $2^n$ . For example, we can define  $E$  to be the set  $\{x \mid x \text{ is the smallest element (in lexicographic order) of } \bar{K} \text{ of length } 2^{2^{\cdot^{\cdot^2}}m}, \text{ for some } m > 0\}$ ; then it is easy to show that this choice for  $E$  is in DEXT. Moreover the following lemmas show that  $E$  is not in  $P$ .

LEMMA 4.2. *Let  $t$  be any time-constructible function. Let  $A$  be any infinite set accepted by a deterministic Turing acceptor that runs in time  $t$ . Then there exists a deterministic transducer  $N_i$  such that  $A \cap K_i[\lceil \log n \rceil, 2^n \cdot t(n)]$  is infinite.*

*Proof.* Let  $M$  be any deterministic acceptor that runs in time  $t$  and recognizes  $A$ . From  $M$  construct a transducer  $N_i$  which computes the function  $f_i$  defined as

$$f_i(\text{bin}(n)) = \begin{cases} \text{the smallest (in lexicographic order) } y \in A \\ \text{such that } |y| = n \text{ if such a } y \text{ exists,} \\ 0 \text{ if no such } y \text{ exists.} \end{cases}$$

It is easy to implement  $N_i$  so that it runs in time  $2^n \cdot t(n)$ . Thus, we have  $R_i = K_i[\lceil \log n \rceil, 2^n \cdot t(n)] \subseteq A$ . Since  $A$  is infinite, it is clear that  $R_i$  is infinite.  $\square$

LEMMA 4.3. *If  $X$  is any infinite subset of  $\bar{K}$ , then  $X$  is not in  $P$ . Hence,  $E$  is not in  $P$ .*

*Proof.* Assume to the contrary that there is an infinite  $X \in P$  so that  $X \subseteq \bar{K}$ . By Lemma 4.2, for some polynomial  $p$  there exists a deterministic transducer  $N_i$  such that

$X \cap K_i[[\log n], 2^n \cdot p(n)]$  is infinite. But from Lemma 2.1 we see that  $K_i[[\log n], 2^n \cdot p(n)] \subseteq K[(\log n) + d, (cn2^n \cdot p(n) \cdot \log p(n)) + c] \subset K$  for all but finitely many  $n$ . Therefore,  $\bar{K} \cap K \neq \emptyset$ , a contradiction.  $\square$

Now we can show that  $E$  is DEXT-low.

**THEOREM 4.4.** *If  $L$  is in DEXT( $E$ ), then  $L$  is in DEXT. Hence, DEXT( $E$ ) = DEXT.*

*Proof.* Let  $M_i$  be a deterministic exponential time-bounded oracle machine that witnesses  $L \in \text{DEXT}(E)$ , that is,  $L = L(M_i, E)$ . We must prove that  $L$  is in DEXT.

**CLAIM.** There exists a constant  $d > 0$  such that for all but finitely many  $x$ , if  $y$  is a query made by  $M_i$  during its computation on  $x$ , then  $|y| > d|x|$  implies  $y$  is not in  $E$ .

*Proof of the Claim.* The proof is based on the notion that one can convert  $M_i$  to the transducer  $N_0$  that will produce each query string  $y$  from  $x$  with some additional information bits.

Let the running time of  $M_i$  be  $2^{cn}$ . Suppose that in  $M_i$ 's computation on  $x$  relative to  $E$ , the oracle is queried about string  $y$ . Let  $n = |x|$  and let  $q_1, \dots, q_m$  ( $m \leq 2^{cn}$ ) be the strings previously queried during this computation (i.e.,  $y$  is the  $(m+1)$ st query). We first show that with input  $x$  and some additional information, all of which requires at most  $(3n+3cn)$  bits to encode, this computation can be simulated up to this point without making any oracle calls.

Let  $\{\alpha_1, \dots, \alpha_k\}$  be the set of strings in  $E$  of length at most  $n$ . Notice that by the definition of  $E$ ,  $|\alpha_1| + \dots + |\alpha_k| \leq 2n$ . Hence, by using  $2n$  bits of additional information, we can simulate an oracle call to  $E$  if the length of the string is at most  $n$ .

Note that  $|q_i| \leq 2^{cn}$  for all  $i$ ,  $1 \leq i \leq m$ . Furthermore, there is at most one element in  $E$  with length greater than  $n$  but no greater than  $2^n$ . Since  $2^{cn} < 2^{2^n}$  for almost all  $n$ , there exist at most two different  $q_i$ 's that are in  $E$  and have lengths that are greater than  $n$ . Thus, if some  $q_j$  is in  $E$  but  $|q_j| > n$ , then we can indicate that fact by knowing its index  $j$  which costs at most  $cn$  bits of additional information for each  $q_j$ . This means that we can simulate the oracle calls to  $E$  even if the length of the string is greater than  $n$ , just so long as we have  $2cn$  bits of additional information.

Note that the string  $y$  is indicated by its index (i.e.,  $m+1$ ). Hence, we can conclude that it is possible to simulate  $M_i$ 's computation on  $x$  relative to  $E$  up to the query  $y$  by making no oracle calls if we have enough additional information: at most  $(2n+2cn+cn)$  bits of such information are required. We encode this information using exactly  $(2n+3cn)$  bits and add them to the input string  $x$ ; hence,  $(3n+3cn)$  bits are needed. Therefore, one can construct a deterministic exponential time-bounded transducer  $N_0$  which produces each query  $y$  from some  $z$  where  $z$  has less than  $(3+3c)n$  bits. Now it follows from Lemma 4.1 that there exists  $d_0 > 0$  such that for all but finitely many  $y$ ,  $|y| > d_0 \cdot |z|$  implies  $y \in K$  so that  $y$  is not in  $E$ . Letting  $d$  be  $d_0(3+3c)$ , we have that  $|y| > d \cdot |x| = d_0(3+3c) \cdot |x| \geq d_0|z|$  implies  $y$  is not in  $E$ , for all but finitely many  $y$ .

This concludes the proof of the claim.

To continue with the proof of the theorem, let  $M_E$  be a deterministic exponential time-bounded acceptor which recognizes  $E$ . Then there is a deterministic exponential time-bounded acceptor  $M_L$  which recognizes  $L$  without making any oracle queries that operates as follows:

```

begin
  input  $x$ ;
  simulate  $M_i$  on  $x$  where
    if  $M_i$  queries the oracle about a string  $y$  and  $|y| > d|x|$ 
      then answer "NO" to the query;

```

if  $M_i$  queries the oracle about a string  $y$  and  $|y| \leq d|x|$   
 then  
   begin  
     simulate  $M_E$  on  $y$  to determine  
     whether  $y \in E$   
     if  $y \in E$  then answer “YES” else “NO”  
   end  
 end.

It is clear that  $M_L$  recognizes  $L$  and operates in exponential time. Thus,  $L$  is in DEXT.  $\square$

From Lemma 4.2 and Theorem 4.4, we have the desired result.

**THEOREM 4.5.** *There is a sparse set  $E \in \text{DEXT} - P$  that is DEXT-low.*

The technique used in the proof of the claim in Theorem 4.4 is similar to one used by Goldberg and Sipser [6].

**5. Polynomial lowness and exponential lowness.** Here we will be concerned with lowness with respect to levels of the polynomial hierarchy or levels of the exponential hierarchy. To begin we review the definitions of the two hierarchies.

Let  $A$  be a set of strings. Define  $\Sigma_0^P(A) = \Pi_0^P(A) = \Delta_0^P(A) = \Delta_1^P(A) = P(A)$ , and for each integer  $i \geq 0$ , define  $\Delta_{i+1}^P(A) = P(\Sigma_{i+1}^P(A))$ ,  $\Sigma_{i+1}^P(A) = NP(\Sigma_i^P(A))$ , and  $\Pi_{i+1}^P(A) = \text{co-}\Sigma_{i+1}^P(A)$ . The structure  $\{(\Sigma_i^P(A), \Pi_i^P(A), \Delta_i^P(A))\}_{i \geq 0}$  is the *polynomial-time hierarchy relative to  $A$* . Define  $\text{PH}(A) = \bigcup_{i \geq 0} \Sigma_i^P(A)$ . If  $A = \emptyset$ , then we have the *polynomial-time hierarchy*, in which case we write  $\Sigma_i^P$  instead of  $\Sigma_i^P(\emptyset)$ , etc. Let  $\text{PH} = \text{PH}(\emptyset)$ .

It is useful to characterize the classes in the polynomial-time hierarchy in terms of quantifiers. To do this we use the notation  $(\exists y)_n$  and  $(\forall y)_n$  to abbreviate  $(\exists y, |y| \leq n)$  and  $(\forall y, |y| \leq n)$ , respectively.

**LEMMA 5.1** [18]. *For any  $A$  and  $k > 0$ , a set  $B$  is in  $\Sigma_k^P(A)$  if and only if there exist a set  $B_0 \in P(A)$  and a polynomial  $q$  such that for all  $x, x \in B$  if and only if  $(\exists y_1)_{q(|x|)} (\forall y_2)_{q(|x|)} \cdots (Q y_k)_{q(|x|)} [\langle x, y_1, \dots, y_k \rangle \in B_0]$ , where  $Q_k$  is  $\exists$  if  $k$  is odd and  $\forall$  if  $k$  is even.*

For properties of the polynomial-time hierarchy, see [16], [18].

Now we turn to the exponential-time hierarchy.

Let  $A$  be a set of strings. Define  $\Sigma_0^E(A) = \Pi_0^E(A) = \Delta_0^E(A) = \Delta_1^E(A) = \text{DEXT}(A)$ , and for each integer  $i \geq 0$ , define  $\Delta_{i+1}^E(A) = \text{DEXT}(\Sigma_{i+1}^E(A))$ ,  $\Sigma_{i+1}^E(A) = \text{NEXT}(\Sigma_i^E(A))$ , and  $\Pi_{i+1}^E(A) = \text{co-}\Sigma_{i+1}^E(A)$ . The structure  $\{(\Sigma_i^E(A), \Pi_i^E(A), \Delta_i^E(A))\}_{i \geq 0}$  is the *exponential-time hierarchy relative to  $A$* . If  $A = \emptyset$ , then we have the *exponential-time hierarchy*, in which case we write  $\Sigma_i^E$  instead of  $\Sigma_i^E(\emptyset)$ , etc.

Also, the classes in the exponential-time hierarchy can be characterized in terms of quantifiers.

**LEMMA 5.2** [17]. *For any  $A$  and  $k > 0$ , a set  $B$  is in  $\Sigma_k^E(A)$  if and only if there exist a set  $B_0 \in P(A)$  and a function  $e(n) = 2^{cn}$  such that for all  $x, x \in B$  if and only if*

$$(\exists y_1)_{e(|x|)} (\forall y_2)_{e(|x|)} \cdots (Q y_k)_{e(|x|)} [\langle x, y_1, \dots, y_k \rangle \in B_0].$$

Aspects of the exponential-time hierarchy have been studied by Dekhtyar [5], Heller [10], Orponen [13], Simon [15], and Wilson [17].

Now we can turn to lowness. Schöning [14] first studied the notion of lowness with respect to classes in the polynomial-time hierarchy. He showed that a set is  $P$ -low if and only if it is in  $P$ , and a set is  $NP$ -low if and only if it is in  $NP \cap \text{co-}NP$ . No similar characterizations are known for the higher levels, but the  $\Sigma_2^P$ - or  $\Sigma_3^P$ -lowness



of several types of sets was established by Ko and Schöning [12]. Our interest is in the notion of lowness with respect to the classes in the exponential-time hierarchy. We note that it is not known whether every set that is  $\Sigma_k^E$ -low is also  $\Sigma_{k+1}^E$ -low.

Our first result extends Proposition 3.1.

**THEOREM 5.3.** *For every  $k \geq 0$ , if a set is  $\Sigma_k^P$ -low, then it is  $\Sigma_k^E$ -low.*

*Proof.* The case  $k = 0$  is Proposition 3.1. Let  $k > 0$  and let  $A$  be such that  $\Sigma_k^P(A) = \Sigma_k^P$ . It suffices to show that  $\Sigma_k^E(A) \subseteq \Sigma_k^E$ , so let  $B \in \Sigma_k^E(A)$  be chosen arbitrarily. By Lemma 5.2 there exist  $B_0 \in P(A)$  and constant  $c > 0$  such that  $x \in B$  if and only if  $(\exists y_1)_{e(|x|)} (\forall y_2)_{e(|x|)} \cdots (Qy_k)_{e(|x|)} [\langle x, y_1, \dots, y_k \rangle \in B_0]$ , where  $e(n) = 2^{cn}$ . Let  $B_1 = \{\langle x, 0^{e|x|} \rangle \mid x \in B\}$ . Then there exists a set  $B_2$  in  $P(A)$  that is defined in the obvious way from  $B_0$  with the property that  $x' = \langle x, 0^{e|x|} \rangle \in B_1$  if and only if  $x \in B$  if and only if  $(\exists y_1)_{q(|x'|)} (\forall y_2)_{q(|x'|)} \cdots (Qy_k)_{q(|x'|)} [\langle x', y_1, \dots, y_k \rangle \in B_2]$ , where  $q(n) = n^c$ . Clearly,  $B_2$  is in  $P(A)$ , and so by Lemma 5.1 we have  $B_1 \in \Sigma_k^P(A) = \Sigma_k^P$ . By removing “padding,” it can be seen that  $B_1 \in \Sigma_k^P$  implies  $B \in \Sigma_k^E$ . Hence,  $\Sigma_k^E(A) = \Sigma_k^E$  as claimed.  $\square$

The results in § 4 show that for  $k = 0$ , the converse of this result does not hold. Analogously to Theorem 3.3, however, we can show that for certain types of compressible sets the notions of polynomial and exponential lowness coincide for sufficiently large values of  $k$ . To make this precise, we first prove a technical lemma.

For a set  $C$ , denote by  $P_{\log}(C)$  the class of sets recognized relative to  $C$  by deterministic polynomial time-bounded oracle machines that on input of length  $n$  make only queries of length  $O(\log n)$ . If  $A \in P_{\log}(C)$ , then we say that  $C$  is a *compact description* of  $A$ .

The reader should notice that every set  $A$  having small generalized Kolmogorov complexity has a compact description, e.g.,  $N_U^{-1}(A)$ . This is consistent with the fact that such a set is polynomially isomorphic to a tally set. It is clear that a set with a compact description need not have small generalized Kolmogorov complexity: there exist sets with polynomial size circuits that do not have small generalized Kolmogorov complexity, but all sets with polynomial size circuits have compact descriptions (see the proofs of Theorems 5.5 and 5.6).

**LEMMA 5.4.** *Let  $A$  be a set with a compact description in the class  $\Delta_k^E(A)$  for some  $k \geq 0$ . If  $A$  is  $\Sigma_k^E$ -low, then  $A$  is  $\Sigma_k^P$ -low.*

*Proof.* The case for  $k = 0$  is argued just like the proof of Proposition 3.2 and is left to the reader.

Let  $k > 0$ . Let  $A$  be  $\Sigma_k^E$ -low and let  $C \in \Delta_k^E(A)$  be such that  $A \in P_{\log}(C)$ . We must show that  $\Sigma_k^P(A) = \Sigma_k^P$  so that  $A$  is  $\Sigma_k^P$ -low. To do this we consider an arbitrary set  $B$  in  $\Sigma_k^P(A)$  and show that  $B$  is in  $\Sigma_k^P$ .

First, recall from Lemma 5.1 that there exist a set  $B_0 \in P(A)$  and a constant  $c$  such that  $x \in B$  if and only if  $(\exists y_1)_{q(|x|)} (\forall y_2)_{q(|x|)} \cdots (Qy_k)_{q(|x|)} [\langle x, y_1, \dots, y_k \rangle \in B_0]$ , where  $q(n) = n^c$ .

Second, observe that  $P(A) \subseteq P_{\log}(C)$  since  $A \in P_{\log}(C)$  implies that polynomial length queries about membership in  $A$  can be replaced by polynomial time computations with log-length queries about membership in  $C$ . Hence,  $B_0 \in P_{\log}(C)$ ; let  $M_0$  be a  $P_{\log}$ -oracle machine such that  $L(M_0, C) = B_0$  and let  $d > 0$  be such that on any input of length  $m$ ,  $M_0$  makes queries about strings of length at most  $d \cdot \log m$  (notice that there are at most  $m^{2d}$  such strings).

We will consider a machine that simulates computations of  $M_0$  but replaces oracle queries by table lookups. Let  $M_1$  be the deterministic polynomial time-bounded acceptor which accepts inputs of the form  $\langle x, T_{\text{YES}}, T_{\text{NO}}, y_1, \dots, y_k \rangle$ , where  $T_{\text{YES}}$  and  $T_{\text{NO}}$  are two disjoint finite sequences of strings, if  $M_0$  accepts input  $\langle x, y_1, \dots, y_k \rangle$  when all oracle queries are in  $T_{\text{YES}} \cup T_{\text{NO}}$  and queries in  $T_{\text{YES}}$  are answered “yes”

while queries in  $T_{\text{NO}}$  are answered “no.” Let  $B_1$  denote the set of strings accepted by  $M_1$ . Let  $B_2$  denote the  $\Sigma_k^P$  set obtained from  $B_1$  by defining  $\langle x, T_{\text{YES}}, T_{\text{NO}} \rangle \in B_2$  if and only if

$$(\exists y_1)_{q(|x|)} (\forall y_2)_{q(|x|)} \cdots (Qy_k)_{q(|x|)} [\langle x, T_{\text{YES}}, T_{\text{NO}}, y_1, \dots, y_k \rangle \in B_1].$$

Thus, for some polynomial  $p$ , we have

$$(*) \quad x \in B \text{ if and only if } (\exists T_{\text{YES}}, T_{\text{NO}})_{p(|x|)} [(y \in T_{\text{YES}} \cup T_{\text{NO}})$$

implies  $|y| \leq \log|x|^d$  and  $T_{\text{YES}}$  is included in  $C$  and

$T_{\text{NO}}$  is included in  $\bar{C}$  (the complement of  $C$ ) and  $\langle x, T_{\text{YES}}, T_{\text{NO}} \rangle \in B_2]$ .

By hypothesis  $C \in \Delta_k^E(A)$  so that  $\bar{C} \in \Delta_k^E(A)$ , and  $\Delta_k^E(A) = \text{DEX}(\Sigma_{k-1}^P(A)) \subseteq \text{NEXT}(\Sigma_{k-1}^P(A)) = \Sigma_k^E(A) = \Sigma_k^E$  since  $A$  is  $\Sigma_k^E$ -low. Thus,  $C$  and  $\bar{C}$  are in  $\Sigma_k^E$ . By Lemma 5.2 there exist a set  $C_0 \in P$  and a constant  $t$  such that  $z \in C$  if and only if  $(\exists y_1)_{f(|z|)} (\forall y_2)_{f(|z|)} \cdots (Qy_k)_{f(|z|)} [\langle z, y_1, \dots, y_k \rangle \in C_0]$ , where  $f(n) = 2^{tn}$ . A similar formula can be obtained to characterize membership in  $\bar{C}$ .

Let  $D_1$  be the set defined by  $\langle x, \langle z_1, \dots, z_m \rangle \rangle \in D_1$  if and only if  $|z_1|, \dots, |z_m| \leq \log|x|^d$  and  $\{z_1, \dots, z_m\} \subseteq C$ . Then, by Lemma 5.1 the following representation shows that  $D_1$  is in  $\Sigma_k^P$ :

$$\langle x, \langle z_1, \dots, z_m \rangle \rangle \in D_1 \quad \text{if and only if}$$

$$(\exists y_{1,1}, \dots, y_{1,m})_{h(|x|)} (\forall y_{2,1}, \dots, y_{2,m})_{h(|x|)} \cdots$$

$$(Qy_{k,1}, \dots, y_{k,m})_{h(|x|)} [\langle z_1, y_{1,1}, \dots, y_{m,1} \rangle \in C_0 \text{ and } \cdots \text{ and}$$

$$\langle z_m, y_{1,m}, \dots, y_{k,m} \rangle \in C_0] \quad \text{where } h \text{ is an appropriate polynomial.}$$

Similarly, the set  $D_2 = \{\langle x, \langle z_1, \dots, z_m \rangle \rangle \mid |z_1|, \dots, |z_m| \leq \log|x|^d \text{ and } \{z_1, \dots, z_m\} \subseteq \bar{C}\}$  is in  $\Sigma_k^P$ .

The representation (\*) for membership in  $B$  can now be rewritten as  $x \in B$  if and only if

$$(**) \quad (\exists T_{\text{YES}}, T_{\text{NO}})_{p(|x|)} [\langle x, T_{\text{YES}} \rangle \in D_1 \text{ and } \langle x, T_{\text{NO}} \rangle \in D_2 \text{ and } \langle x, T_{\text{YES}}, T_{\text{NO}} \rangle \in B_2].$$

Now  $D_1, D_2$  and  $B_2$  are all in  $\Sigma_k^P$ , and  $\Sigma_k^P$  is closed under polynomial-bounded existential quantification, so that (\*\*) shows that  $B$  is in  $\Sigma_k^P$  as desired.  $\square$

While the proof of Lemma 5.4 is quite technical, the result is very useful as is seen by the next two theorems.

**THEOREM 5.5.** *Let  $S$  be a sparse set and let  $k \geq 2$ . If  $S$  is  $\Sigma_k^E$ -low, then  $S$  is also  $\Sigma_k^P$ -low.*

*Proof.* Since  $S$  is sparse, we can apply the coding technique developed in [7], [9]. Let  $C_S = \{\langle n, c, i, j, b \rangle \mid S \text{ contains at least } c \text{ strings } x_1 < x_2 < \cdots < x_c \text{ of length } n, \text{ and the } j\text{th symbol of } x_i \text{ is } b\}$ . (For brevity, we have identified integers with their representations. To be precise, we should have written  $\langle \text{bin}(n), \text{bin}(c), \text{bin}(i), \text{bin}(j), b \rangle$  for  $\langle n, c, i, j, b \rangle$ .) The symbol  $<$  denotes the lexicographic ordering of strings of  $\Sigma^*$ .

The following  $P_{\log}$ -type algorithm recognizes  $S$  relative to  $C_S$ , which shows that  $C_S$  is a compact description for  $S$ .

```

input  $x$ ;
 $n := |x|$ ;           // Note that  $|n| \approx \log|x|$ . //
 $c := 0$ ;
while  $\langle n, c+1, 1, 1, 0 \rangle \in C_S$  or  $\langle n, c+1, 1, 1, 1 \rangle \in C_S$  do
   $c := c+1$ ;       // The maximum value of  $c$  is
                    // polynomially bounded in  $|x|$ . //
for  $i = 1$  to  $c$  do

```

```

begin
  y := λ;
  for j = 1 to n do
    if ⟨n, c, i, j, 0⟩ ∈ CS then y := y0 else y := y1
    if x = y then halt and accept
  end;
halt and reject.

```

The set  $C_S$  can be accepted nondeterministically in exponential time relative to  $S$ : on input  $\langle n, c, i, j, b \rangle$ , a nondeterministic machine can guess an appropriate sequence  $x_1, \dots, x_c$  of strings, query the oracle to determine whether each string is in  $S$ , and verify that the  $j$ th symbol of  $x_i$  is  $b$ . Hence,  $C_S \in \text{NEXT}(S) \subseteq \Delta_k^E(S)$  for every  $k \geq 2$ . Thus,  $S$  has a compact description in  $\Delta_k^E(S)$  so that  $S$  being  $\Sigma_k^E$ -low implies that  $S$  is  $\Sigma_k^P$ -low by Lemma 5.4.  $\square$

From the results in § 4, we see that the result of Theorem 5.5 does not hold for  $k = 0$ . Whether the result holds for  $k = 1$  is an open question.

Recall that a set  $A$  has polynomial-size circuits if there exists a sparse set  $S$  such that  $A \in P(S)$ . Another characterization [11], [12] is as follows:  $A$  has polynomial-size circuits if there exists a set  $A_0 \in P$  and a polynomial  $p(n)$  with the property that for every  $n$  there exists an “advice” string  $w$ ,  $|w| \leq p(n)$ , such that for all  $x$  with  $|x| \leq n$ ,  $x \in A$  if and only if  $\langle w, x \rangle \in A_0$ .

**THEOREM 5.6.** *Let  $A$  be a set that has polynomial-size circuits and let  $k \geq 4$ . If  $A$  is  $\Sigma_k^E$ -low, then  $A$  is also  $\Sigma_k^P$ -low.*

*Proof.* Let  $A_0 \in P$  be a set that witnesses the fact that  $A$  has polynomial-size circuits. A description  $C_A$  for  $A$  can be obtained as follows:  $C_A = \{ \langle n, m, i, b \rangle \mid (\exists w)_m [ (\forall x)_n (x \in A \Leftrightarrow \langle w, x \rangle \in A_0) \text{ and } (\forall w' < w) (\exists x)_n (x \in A \Leftrightarrow \langle w', x \rangle \in A_0) \text{ and the } i\text{th symbol of } w \text{ is } b] \}$ . It can be verified that  $C_A \in \Sigma_3^E(A) \subseteq \Delta_4^E(A)$ . Further, the following  $P_{\log}$ -type algorithm decides membership in  $A$  relative to  $C_A$ .

```

input x;
n := |x|;
m := 0;
while neither ⟨n, m, 1, 0⟩ nor ⟨n, m, 1, 1⟩ is in CA do
  m := m + 1;
w := λ;
for i = 1 to m do
  if ⟨n, m, i, 0⟩ ∈ CA then w := w0 else w := w1;
  if ⟨w, x⟩ ∈ A0 then halt and accept else halt and reject.

```

Since  $C_A$  is a compact description for  $A$ ,  $C_A \in \Delta_4^E(A)$ , and  $A$  is  $\Sigma_k^E$ -low for some  $k \geq 4$ , it follows from Lemma 5.4 that  $A$  is  $\Sigma_k^P$ -low.  $\square$

Balcázar, Book and Schöning [2] studied a more general type of lowness in the polynomial-time hierarchy and the notions that they introduced are of interest here.

For integers  $i, j \geq 0$ , a set  $A$  is in  $L^P(i, j)$  if  $\Sigma_i^P(A) \subseteq \Sigma_j^P$ .

It is clear that a set in some class  $L^P(i, j)$  is itself in PH. Suppose that  $A \in \Sigma_n^P$ . Then for every  $i$ ,  $\Sigma_i^P(A) \subseteq \Sigma_{i+n}^P$  so that the notion of  $A \in L^P(i, j)$  can have significance only if  $j < i + n$ . Informally, we interpret  $A \in L^P(i, j)$  with  $k = j - i$  to mean that using the set  $A$  as an oracle has the power of at most  $k$  alternating quantifiers or of at most the  $k$ -fold composition of the NP ( ) operator.

We “lift” the definition to sets in the exponential hierarchy in the obvious way.

For integers  $i, j \geq 0$ , a set  $A$  is in  $L^E(i, j)$  if  $\Sigma_i^E(A) \subseteq \Sigma_j^E$ .

Clearly, for every  $i, j$ ,  $L^E(i, j) \subseteq \Sigma_j^E$ . It is easy to see that for every choice of  $i, j$  with  $i \leq j$ ,  $L^P(i, j) \subseteq L^P(i+1, j+1)$ . However, we do not know whether  $i \leq j$  implies  $L^E(i, j) \subseteq L^E(i+1, j+1)$ .

The following result is the obvious generalization of Theorem 5.3.

PROPOSITION 5.7. *For every  $i, j$  with  $i \leq j$ ,  $L^P(i, j) \subseteq L^E(i, j)$ .*

To generalize the results in § 3 on tally sets and Theorems 5.5 and 5.6, we have the following result.

PROPOSITION 5.8. (a) *Let  $T$  be a set with small Kolmogorov complexity. For every  $i, j$  with  $i \leq j$ ,  $T \in L^E(i, j)$  if and only if  $T \in L^P(i, j)$ .*

(b) *Let  $S$  be a sparse set. For every  $i, j$  with  $2 \leq i \leq j$ ,  $S \in L^E(i, j)$ , if and only if  $S \in L^P(i, j)$ .*

(c) *Let  $A$  be a set with polynomial-size circuits. For every  $i, j$  with  $4 \leq i \leq j$ ,  $A \in L^E(i, j)$  if and only if  $A \in L^P(i, j)$ .*

Recall that a set is  $P$ -immune if it has no infinite subset in  $P$ . Lemma 4.2 shows that there is a sparse set that is both  $P$ -immune and exponentially low.

THEOREM 5.9. *If there exists an infinite set that is both  $P$ -immune and  $\Sigma_k^E$ -low, where  $k \geq 3$ , then there is an infinite  $P$ -immune set in  $\Sigma_k^P$ ; hence  $P \neq \Sigma_k^P$  so that  $P \neq \text{NP}$ .*

*Proof.* Let  $A$  be any infinite  $P$ -immune  $\Sigma_k^E$ -low set. Consider the infinite sparse set  $S = \{x \mid x \text{ is the "smallest" string of length } |x| \text{ in } A\}$ . Since  $S \subseteq A$ ,  $S$  is  $P$ -immune. It is clear that  $S$  is in  $\text{co-NP}(A)$ .

Notice that

$$\begin{aligned} \Sigma_{k-1}^E(S) &\subseteq \Sigma_{k-1}^E(\text{co-NP}(A)) \\ &= \Sigma_{k-1}^E(\text{NP}(A)) = \Sigma_k^E(A) = \Sigma_k^E \end{aligned}$$

since  $A$  is  $\Sigma_k^E$ -low. Thus,  $S \in L^E(k-1, k)$  so by Proposition 5.8(b),  $S \in L^P(k-1, k)$ . Hence,  $S \in \Sigma_k^P$ .  $\square$

Another way to generalize the notions studied here is to extend the notion of compact descriptions. Instead of requiring that  $A \in P_{\log}(C)$ , we could require that for some  $k$ ,  $A \in \Delta_{k, \log}^P(C)$ . Then we could establish the obvious generalizations of Lemma 5.4 and Theorems 5.5 and 5.6. The proof is clear but tedious and the details are left to the reader.

**6. Concluding remarks.** We have studied the existence and properties of sets relative to which exponential complexity classes do not exhibit the pathology of not being closed under relativization. As a main result, we have established the existence of a set  $E$  in  $\text{DEXT-}P$  such that  $\text{DEXT}(E) = \text{DEXT}$ . Moreover, we have shown that this set can be taken to be sparse. On the other hand, we have shown that for  $k \geq 2$ , if a sparse set  $S$  satisfies  $\Sigma_k^E(S) = \Sigma_k^E$ , then it is also true that  $\Sigma_k^P(S) = \Sigma_k^P$ . (The main result shows that this is not the case for  $k=0$ .) In the case  $k=1$ , it remains unresolved whether  $\text{NEXT}(S) = \text{NEXT}$  implies  $\text{NP}(S) = \text{NP}$  (i.e.,  $S \in \text{NP} \cap \text{co-NP}$ ) for sparse  $S$ .

For tally sets and, more generally, sets with small Kolmogorov complexity, we have shown that  $\Sigma_k^E(T) = \Sigma_k^E$  implies  $\Sigma_k^P(T) = \Sigma_k^P$  for  $k \geq 0$ . For sets with polynomial-size circuits, this implication holds when  $k \geq 4$ .

In general, our results indicate that sets that are "exponentially low" but not "polynomially low"—whenever they exist—must have complex strings. This is witnessed by the fact that under various "simplicity" assumptions about a set  $A$ , we know that for sufficiently large  $k$ ,  $\Sigma_k^E(A) = \Sigma_k^E$  if and only if  $\Sigma_k^P(A) = \Sigma_k^P$ . Also, the proof of the main result relies on choosing a set of strings having reasonably high generalized Kolmogorov complexity.

The principal open questions may be summarized as follows: for each  $k \geq 0$ , characterize the class of sets that are  $\Sigma_k^E$ -low. The case of  $k = 0$  is of particular interest due to the main result of the present paper. We have shown that for every value of  $k$ , if a set is  $\Sigma_k^P$ -low, then it is  $\Sigma_k^E$ -low, but we have only partial results regarding the converse. Since the notion of exponential lowness is less restrictive than its polynomial counterpart, it appears that it might be easier to establish the lowness characteristics of various structural properties of sets in the exponential than in the polynomial sense (cf. [12]). What must not be overlooked is that the results of Schöning [14], Ko and Schöning [12], and Balcázar, Book and Schöning [2] do not characterize sets that are  $\Sigma_k^P$ -low.

## REFERENCES

- [1] J. BALCÁZAR AND R. BOOK, *Sets with small generalized Kolmogorov complexity*, Acta Inform., 23 (1986), pp. 679–688.
- [2] J. BALCÁZAR, R. BOOK, AND U. SCHÖNING, *Sparse oracles, lowness, and highness*, this Journal, 15 (1986), pp. 739–747.
- [3] R. BOOK, *Tally languages and complexity classes*, Inform. and Control, 26 (1974), pp. 186–193.
- [4] A. CHANDRA, D. KOZEN, AND L. STOCKMEYER, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114–133.
- [5] M. DEKHTYAR, *On the relation of deterministic and nondeterministic complexity classes*, in Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 45, Springer-Verlag, New York, Berlin, 1977, pp. 282–287.
- [6] A. GOLDBERG AND M. SIPSER, *Compression and ranking*, Proc. 17th Annual ACM Symposium on Theory of Computing, 1985, pp. 440–448.
- [7] J. HARTMANIS, *On sparse sets in NP-P*, Inform. Process. Lett., 16 (1983), pp. 55–60.
- [8] ———, *Generalized Kolmogorov complexity and the structure of feasible computations*, Proc. 24th Annual IEEE Symposium on the Foundations of Computer Science, 1983, pp. 439–445.
- [9] J. HARTMANIS, V. SEWELSON, AND N. IMMERMANN, *Sparse sets in NP-P: EXPTIME versus NEXPTIME*, Proc. 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 382–391.
- [10] H. HELLER, *On relativized polynomial and exponential computations*, this Journal, 13 (1984), pp. 717–725.
- [11] R. KARP AND R. LIPTON, *Some connections between nonuniform and uniform complexity classes*, Proc. 12th Annual ACM Symposium on Theory of Computing, 1980, pp. 302–309.
- [12] K. KO AND U. SCHÖNING, *On circuit-size complexity and the low hierarchy in NP*, this Journal, 14 (1985), pp. 41–51.
- [13] P. ORPONEN, *Complexity class of alternating machines with oracles*, in Automata, Languages, and Programming, Lecture Notes in Computer Science 154, Springer-Verlag, New York, Berlin, 1983, pp. 573–584.
- [14] U. SCHÖNING, *A low and a high hierarchy in NP*, J. Comput. System Sci., 27 (1983), pp. 14–28.
- [15] J. SIMON, *On some central problems in computational complexity*, Ph.D. dissertation, Cornell University, Ithaca, NY, 1975.
- [16] L. STOCKMEYER, *The polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1976), pp. 1–22.
- [17] C. WILSON, *Relativization, reducibilities, and the exponential hierarchy*, M.Sc. thesis, Univ. of Toronto, Toronto, Ontario, Canada, 1980.
- [18] C. WRATHALL, *Complete sets and the polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1976), pp. 23–33.
- [19] E. ALLENDER AND R. RUBINSTEIN, *P-printable sets*, submitted for publication.

## MONOTONE BIPARTITE GRAPH PROPERTIES ARE EVASIVE\*

ANDREW CHI-CHIH YAO†

**Abstract.** A Boolean function  $P$  from  $\{0, 1\}^t$  into  $\{0, 1\}$  is said to be *evasive*, if every decision-tree algorithm for evaluating  $P$  must examine all  $t$  arguments in the worst case. It was known that any nontrivial monotone bipartite graph property on vertex set  $V \times W$  must be evasive, when  $|V| \cdot |W|$  is a power of a prime number. In this paper, we prove that every nontrivial monotone bipartite graph property is evasive.

**Key words.** bipartite graph property, complexity, decision tree, evasive, monotone property

**AMS(MOS) subject classifications.** 68Q05, 68R05, 05C25

**1. Introduction.** In [RV2], Rivest and Vuillemin proved the Aanderra-Rosenberg Conjecture [R] which states that, to evaluate any nontrivial monotone graph property on  $n$  vertices, every decision-tree algorithm must examine  $\Omega(n^2)$  entries of the adjacency matrix in the worst case. A stronger conjecture, suggested by Karp (see [R]), that all such graph properties are *evasive*, i.e., *all* entries must be examined in the worst case, was left unresolved. Recently, Khan, Saks, and Sturtevant [KSS] gave a partial solution by showing that, when  $n$  is a power of a prime, all such graph properties are evasive; their proof employed an ingenious topological approach to this complexity problem.

The method used in Rivest and Vuillemin [RV1], [RV2] (also discovered in Best et al. [BBL]) yields immediately that any nontrivial monotone bipartite graph property on vertex set  $V \times W$  must be evasive, when  $|V| \cdot |W|$  is a power of a prime number. The purpose of this paper is to show that, in fact, every nontrivial monotone bipartite graph property is evasive. We will adopt the topological view for this problem as espoused in [KSS].

**2. Main theorem.** Let  $V = \{1, 2, \dots, m\}$ ,  $W = \{1, 2, \dots, n\}$  and  $\mathcal{G}_{m,n}$  be the set of all bipartite graphs  $G = (V \times W, E)$ , where  $E \subseteq V \times W$ . For any two  $G = (V \times W, E)$ ,  $G' = (V \times W, E')$ , we write  $G \leq G'$  if  $E \subseteq E'$ ; we say that  $G$  and  $G'$  are *isomorphic* if there exist permutations  $p_1, p_2$  of  $V, W$  such that  $(i, j) \in E$  if and only if  $(p_1(i), p_2(j)) \in E'$ . A bipartite graph property on  $V \times W$  is a function  $P: \mathcal{G}_{m,n} \rightarrow \{0, 1\}$  satisfying the constraint that  $P(G) = P(G')$  if  $G$  and  $G'$  are isomorphic. A bipartite graph property  $P$  on  $V \times W$  is *monotone* if  $P(G) \leq P(G')$  for all  $G \leq G'$ ;  $P$  is *nontrivial* if it is not a constant function.

Let  $P$  be any bipartite graph property on  $V \times W$ . We are interested in evaluating  $P(G)$ , where the input graph  $G = (V \times W, E)$  is given as an  $m \times n$  adjacency matrix  $(a_{ij})$  with  $a_{ij} = 1$  for  $(i, j) \in E$  and 0 otherwise. A *decision-tree algorithm*  $T$  proceeds by asking a sequence of queries:  $a_{i_1 j_1} = ?$ ,  $a_{i_2 j_2} = ?$ ,  $\dots$ , until the value of  $P(G)$  can be determined; the choice of the  $(k+1)$ st query can depend on the results of all the preceding  $k$  values  $a_{i_1 j_1}, \dots, a_{i_k j_k}$ . The *cost* of  $T$ ,  $\text{cost}(T)$ , is the maximum number of queries asked for any input  $G \in \mathcal{G}_{m,n}$ . The *complexity* of property  $P$  is defined as  $\min \{\text{cost}(T) \mid T \in \mathcal{T}(P)\}$ , when  $\mathcal{T}(P)$  is the set of all decision-tree algorithms for property  $P$ . We say that  $P$  is *evasive* if  $C(P) = |V| \cdot |W|$ . Our main result is the following theorem; the remainder of this paper is devoted to its proof.

\* Received by the editors April 20, 1987; accepted for publication May 28, 1987. This research was supported in part by the National Science Foundation under grant DCR-8308109.

† Department of Computer Science, Princeton University, Princeton, New Jersey 08544.

**THEOREM 1.** *Every nontrivial monotone bipartite graph property is evasive.*

**3. Preliminaries.** We review some needed terminology and facts from standard topology and from [KSS].

**3.1. Abstract complex.** An *abstract complex* on a finite set  $X = \{x_1, x_2, \dots, x_t\}$  is a collection  $\Delta$  of subsets of  $X$  with the property that  $A \subseteq B \in \Delta$  implies  $A \in \Delta$ . Each  $A \in \Delta$  is a *face*; the *dimension* of  $A$  is  $|A| - 1$ . We call  $x_i$  the *vertices*. The *Euler characteristic* of  $\Delta$  is  $\chi(\Delta) = \sum_{i \geq 0} (-1)^i f_i$ , where  $f_i$  is the number of  $i$ -dimensional faces of  $\Delta$ . We say that  $\Delta$  is *rationally acyclic* if the homology groups of  $\Delta$  are  $H_0(\Delta) = \mathbb{Z}$  and  $H_i(\Delta) = 0$  for all  $i > 0$ .

Let  $\Gamma$  be any permutation group of  $X$ . Assume that  $\Delta$  is *invariant under  $\Gamma$* , i.e., for all  $\sigma \in \Gamma$ ,  $(x_{i_1}, x_{i_2}, \dots, x_{i_k}) \in \Delta$  implies  $(x_{\sigma(i_1)}, x_{\sigma(i_2)}, \dots, x_{\sigma(i_k)}) \in \Delta$ . A face  $F = \{x_{i_1}, x_{i_2}, \dots, x_{i_k}\}$  is said to be *minimally invariant under  $\Gamma$*  if, for all  $\sigma \in \Gamma$ ,  $\{\sigma(i_1), \sigma(i_2), \dots, \sigma(i_k)\} = \{i_1, i_2, \dots, i_k\}$ , and if in addition, no proper nonempty subset of  $F$  has this property. Let  $\mathcal{A}(\Delta, \Gamma)$  be the set of all nonempty faces of  $\Delta$  that are minimally invariant under  $\Gamma$ .

**DEFINITION 1.** Suppose  $\Delta$  is invariant under  $\Gamma$ . If  $\mathcal{A}(\Delta, \Gamma) = \emptyset$ , let  $\Delta_\Gamma = \emptyset$ . If  $\mathcal{A}(\Delta, \Gamma) = \{A_1, A_2, \dots, A_s\} \neq \emptyset$ , let  $\Delta_\Gamma$  be the abstract complex on  $\mathcal{A}(\Delta, \Gamma)$  defined by  $\Delta_\Gamma = \{\{A_i \mid i \in D\} \mid D \subseteq \{1, 2, \dots, s\}, \cup_{i \in D} A_i \in \Delta\}$ .

**3.2. Geometric complex.** Let  $\{v_1, v_2, \dots, v_k\}$  be a set of  $k$  independent points in  $R^q$  where  $q > 0$  is an integer. Denote by  $\langle v_1, v_2, \dots, v_k \rangle$  their *convex hull*, i.e., the set

$$\left\{ \sum_{1 \leq i \leq k} \lambda_i v_i \mid \lambda_i \geq 0 \text{ for all } i, \text{ and } \sum_{1 \leq i \leq k} \lambda_i = 1 \right\}.$$

A set  $M \subseteq R^q$  is called a *geometric realization* of an abstract complex  $\Delta$  on  $X = \{x_1, x_2, \dots, x_t\}$  if there exists a set of independent points  $\{v_1, v_2, \dots, v_t\}$ , called the *base*, such that  $M = \cup_{A \in \Delta} Y_A$ , where  $Y_A = \langle v_{i_1}, v_{i_2}, \dots, v_{i_k} \rangle$  for  $A = \{x_{i_1}, x_{i_2}, \dots, x_{i_k}\}$ . Clearly, any abstract complex  $\Delta$  on  $X = \{x_1, x_2, \dots, x_t\}$  has a geometric realization in  $R^q$  if  $q \geq t - 1$ .

We will call  $M \subseteq R^q$  a *geometric complex* if  $M$  is a geometric realization of some abstract complex  $\Delta$ . It is a well-known fact in topology (see, e.g. [Mu]) that if  $M$  is a geometric realization of two abstract complexes  $\Delta$  and  $\Delta'$ , then  $\chi(\Delta) = \chi(\Delta')$ . Thus, we can define  $\chi(M)$  as  $\chi(\Delta)$  unambiguously.

**3.3. Fixed points.** Let  $\Delta$  be an abstract complex on  $X = \{x_1, x_2, \dots, x_t\}$ , invariant under a permutation group  $\Gamma$  of  $\{1, 2, \dots, t\}$ . Let  $M$  be a geometric realization of  $\Delta$  with base  $\{v_1, v_2, \dots, v_t\}$ . Then  $\Gamma$  induces a natural automorphism group on  $M$ . Precisely, for each  $\sigma \in \Gamma$ , let  $f_\sigma$  be the automorphism on  $M$  defined by

$$f_\sigma \left( \sum_{1 \leq i \leq k} \lambda_i v_i \right) = \sum_{1 \leq i \leq k} \lambda_i v_{\sigma(i)}$$

for  $\lambda_i \geq 0, \sum_{1 \leq i \leq k} \lambda_i = 1$ . Let  $M^\Gamma$  denote the set of *fixed points* of this automorphism group, i.e.,  $M^\Gamma \equiv \{v \in M, f_\sigma(v) = v \text{ for all } \sigma \in \Gamma\}$ .

**THEOREM 2 [KSS].**  $M^\Gamma$  is a geometric realization of  $\Delta_\Gamma$ .

For any two groups  $F$  and  $L$ , we say that  $L$  is a *homomorphic image* of  $F$  if there is a 1-1 homomorphism from  $F$  onto  $L$ . Let  $\mathcal{L}_l$  be the cyclic group of order  $l$ .

<sup>1</sup> In the paper all  $i_j$ 's are distinct whenever they appear in the notation  $\{x_{i_1}, x_{i_2}, \dots, x_{i_k}\}$ .

**THEOREM 3 [O].** *If  $\Delta$  is rationally acyclic and  $\Gamma$  is a homomorphic image of  $\mathcal{L}_1$ , then  $\chi(M^\Gamma) = 1$ .*

**3.4. General string properties and topology.** In the study of the complexity of evaluating graph properties, it has been found useful ([BBL], [RV2]) to consider the complexity of evaluating a more general class of functions, the *string properties*. A *string property*  $P$  is a function from  $\{0, 1\}^t$  into  $\{0, 1\}$ . As done for graph properties in § 1, we consider decision tree algorithms  $T$  for evaluating  $P(a_1, a_2, \dots, a_t)$  by asking an adaptive sequence of queries  $a_{i_1} = ?, a_{i_2} = ?, \dots$ ; we define  $\text{cost}(T)$  and  $C(P)$  in the same way. The property  $P$  is said to be *evasive* if  $C(P) = t$ . We say that  $P$  is *nontrivial* if  $P$  is not a constant;  $P$  is *monotone* if  $P(a_1, a_2, \dots, a_t) \leq P(a'_1, a'_2, \dots, a'_t)$  when  $a_i \leq a'_i$  for all  $i$ .

In [KSS], the approach to study a string property  $P$  is to associate with  $P$  the abstract complex  $\Delta$  on  $X = \{x_1, x_2, \dots, x_t\}$  defined as follows:  $\{x_{i_1}, x_{i_2}, \dots, x_{i_k}\} \in \Delta$  if  $P(a_1, a_2, \dots, a_t) = 0$  where  $a_{i_1} = a_{i_2} = \dots = a_{i_k} = 1$  and  $a_j = 0$  for  $j \neq i_l$ . The following fundamental observation was made.

**THEOREM 4 [KSS].** *If  $P$  is not evasive, then the associated  $\Delta$  is rationally acyclic.*

We need one more concept. Let  $\Gamma$  be a permutation group of  $\{1, 2, \dots, t\}$ . We say that  $P$  is *invariant under  $\Gamma$*  if  $P(a_1, a_2, \dots, a_t) = P(a_{\sigma(1)}, a_{\sigma(2)}, \dots, a_{\sigma(t)})$  for all  $\sigma \in \Gamma$ . It is clear that if  $P$  is invariant under  $\Gamma$ , so is the associated abstract complex  $\Delta$ .

**4. Proof of Theorem 1.** First we rephrase the problem in the terminology of string property (§ 3.4). Let  $\Sigma_{m,n} = S_m \oplus S_n$ , where  $S_m, S_n$  are the symmetric groups on  $V = \{1, 2, \dots, m\}, W = \{1, 2, \dots, n\}$ . Each  $\sigma \in \Sigma_{m,n}$  is a permutation of  $\{(i, j) | 1 \leq i \leq m, 1 \leq j \leq n\}$ , that is, if  $\sigma = (p_1, p_2)$ , where  $p_1 \in S_m, p_2 \in S_n$ , then  $\sigma(i, j) = (p_1(i), p_2(j))$  for all  $i, j$ . Let us regard a bipartite graph property  $P$  on  $V \times W$  as a string property in the following way: Any input graph  $G \in \mathcal{G}_{m,n}$  is identified with  $a \equiv (a_{11}, a_{12}, \dots, a_{ij}, \dots, a_{mn}) \in \{0, 1\}^{mn}$ , where  $a$  is obtained from the adjacency matrix  $(a_{ij})$  of  $G$  by concatenating the entries row by row; this naturally induces a string property  $P': \{0, 1\}^{mn} \rightarrow \{0, 1\}$ . It is easy to see that if  $P$  is nontrivial and monotone, so is  $P'$  as a string property; also  $P$  is evasive if and only if  $P'$  is. In addition,  $P'$  is invariant under  $\Sigma_{m,n}$ .

To prove Theorem 1, let  $P$  be a nontrivial monotone bipartite graph property on  $V \times W$ . Denote by  $P'$  the corresponding string property on  $\{0, 1\}^{mn}$ . Assume that  $P$  is not evasive, implying that  $P'$  is not evasive; we will derive a contradiction.

Let  $D \subseteq \{1, 2, \dots, m\}$ . Denote by  $b_D$  the vector  $(a_{11}, a_{12}, \dots, a_{ij}, \dots, a_{mn})$ , where  $a_{il} = 1$  for  $i \in D, 1 \leq l \leq n$ , and 0 otherwise.

**LEMMA 1.** *There exists an integer  $0 \leq r(P') < m$  such that  $P'(b_D) = 0$  if  $|D| \leq r(P')$  and 1 otherwise.*

*Proof.* As  $P'$  is invariant under  $\Sigma_{m,n}$ ,  $P'(b_D) = P'(b_{D'})$  if  $|D| = |D'|$ . It then follows from the monotonicity of  $P'$  that there exists an integer  $-1 \leq r(P') \leq m$  such that  $P'(b_D) = 0$  if and only if  $|D| \leq r(P')$ . Finally,  $r(P') \neq -1, m$ , since  $P'$  is nontrivial.  $\square$

Let  $\Delta$  be the abstract complex associated with  $P'$ . Then  $\Delta$  is rationally acyclic by Theorem 4. Let  $\Gamma$  be the subgroup  $1 \oplus Z_n$  of  $\Sigma_{m,n}$ , i.e.,  $\Gamma = \{\sigma_0, \sigma_1, \dots, \sigma_{n-1}\}$  with  $\sigma_l(i, j) = (i, (j+l) \bmod n)$ . Then  $\Delta$  is invariant under  $\Gamma$  since  $P'$  is invariant under  $\Gamma$ .

Now let  $M$  be a geometric realization of  $\Delta$ . As  $\Gamma$  is clearly a homomorphic image of  $\mathcal{L}_n$ , we have by Theorem 3 that  $\chi(M^\Gamma) = 1$ . Thus,  $\chi(\Delta_\Gamma) = \chi(M^\Gamma) = 1$ , since by Theorem 2  $M^\Gamma$  is a geometric realization of  $\Delta_\Gamma$ .

On the other hand, we have from the definition of  $\Delta_\Gamma$  that  $\Delta_\Gamma = \{\{A_i | i \in D\} | D \subseteq \{1, 2, \dots, m\}, P'(b_D) = 0\}$  whenever  $\Delta_\Gamma \neq \emptyset$ . Thus, either  $\Delta_\Gamma = \emptyset$  in



which case  $\chi(\Delta_\Gamma) = 0 \neq 1$ , or we have by Lemma 1 that

$$\begin{aligned}\chi(\Delta_\Gamma) &= \sum_{0 \leq j < r(P')} (-1)^j \binom{m}{j+1} \\ &= \sum_{0 \leq j < r(P')} (-1)^j \left[ \binom{m-1}{j+1} + \binom{m-1}{j} \right] \\ &= 1 + (-1)^{r(P')-1} \binom{m-1}{r(P')} \\ &\neq 1.\end{aligned}$$

This contradicts the conclusion of the last paragraph.

We have proved Theorem 1.

**5. Remarks.** The most tantalizing open question in this subject is whether all nontrivial monotone graph properties are evasive. As mentioned in [KSS], their topological approach cannot resolve this question when only the transitive nature of the underlying group for graph properties is exploited. The proof of our result on bipartite graphs, as well as the proof of evasiveness for graph properties on six vertices in [KSS], suggests that further progress might be possible if one examines in detail the structures of the geometric complexes associated with graph properties.

Another interesting direction for further work is to prove evasiveness for other classes of string properties. For example, any nontrivial monotone string properties that are transitively invariant under cyclic group  $C_m$  must be evasive (as can be seen from Theorem 2 in [KSS], or from Theorems 3 and 4 in this paper). Is the analogous result true for string properties invariant under  $C_m \oplus C_n$ ?

#### REFERENCES

- [BBL] M. BEST, P. VAN ENDE BOAS, AND H. W. LENSTRA, JR., *A sharpened version of the Aanderra-Rosenberg Conjecture*, Tech. Report ZW 30/74, Stichting Mathematisch Centrum, Amsterdam, 1974.
- [KSS] J. KAHN, M. SAKS, AND D. STURTEVANT, *A topological approach to evasiveness*, *Combinatorica*, 4 (1984), pp. 297-306.
- [Mu] J. MUNKRES, *Elements of Algebraic Topology*, Addison-Wesley, Reading, MA, 1984.
- [O] R. OLIVER, *Fixed-point sets of group actions on finite acyclic complexes*, *Comment. Math. Helv.* 50 (1975), pp. 155-177.
- [RV1] R. L. RIVEST AND J. VUILLEMIN, *On the number of argument evaluations required to compute boolean functions*, Electronics Research Laboratory Memorandum ERL-M472, Univ. of California, Berkeley, CA, October 1974.
- [RV2] —, *On recognizing graph properties from adjacency matrices*, *Theoret. Comput. Sci.*, 3 (1976), pp. 371-384.
- [R] A. L. ROSENBERG, *On the time required to recognize properties of graphs: A problem*, *SIGACT News*, 5 (1973), pp. 15-16.

## DISTANCE-HEREDITARY GRAPHS, STEINER TREES, AND CONNECTED DOMINATION\*

ALESSANDRO D'ATRI<sup>†</sup> AND MARINA MOSCARINI<sup>‡</sup>

**Abstract.** Distance-hereditary graphs have been introduced by Howorka and studied in the literature with respect to their metric properties. In this paper several equivalent characterizations of these graphs are given: in terms of existence of particular kinds of vertices (isolated, leaves, twins) and in terms of properties of connections, separators, and hangings. Distance-hereditary graphs are then studied from the algorithmic viewpoint: simple recognition algorithms are given and it is shown that the problems of finding cardinality Steiner trees and connected dominating sets are polynomially solvable in a distance-hereditary graph.

**Key words.** distance-hereditary graph, crossing chords, perfect graphs, connected dominating set, Steiner tree

**AMS(MOS) subject classifications.** 05C38, 05C75, 68C25, 68E10

**1. Introduction.** A graph is *distance-hereditary* if every two vertices have the same distance in every connected induced subgraph containing both (where the *distance* between two vertices is the length of a shortest path connecting them). These graphs have been introduced and studied by Howorka [18] and further characterized by Bandelt and Mulder [2] in terms of the distance function and forbidden isometric subgraphs.

In particular, it has been proved [18] that distance-hereditary graphs are perfect and can be characterized in terms of the existence of chords in cycles in the following way: a graph is distance-hereditary if and only if each cycle on five or more vertices has at least two crossing chords (where two chords  $(u, v)$  and  $(w, z)$  *cross* if the vertices  $u, w, v, z$  are distinct and in this order on the cycle). The proofs of the results in this paper are mainly based on this characterization; for this reason in a preliminary version [10] we called these graphs *cross* graphs. Similarly, several classes of perfect graphs [16] have also been characterized in the literature by means of adjacency properties of vertices in cycles (e.g., chordal and strongly chordal graphs [14], Meyniel's graphs [21], cographs [7], parity graphs [5]). Furthermore, starting from the studies on acyclic hypergraphs in database theory [9], [12], [13], it has been shown that acyclicity properties in hypergraphs correspond to chordality properties in bipartite graphs and vice versa [1], [11].

In this paper new characterizations of distance-hereditary graphs are given; furthermore, these graphs are related to the above classes of perfect graphs, and are studied with respect to classical problems that are open or computationally hard in several classes of perfect graphs.

In § 3, the new characterizations are proved. In particular, it is shown:

(1) That the class of distance-hereditary graphs is closed under multiplication of vertices [16] and appending of leaves and isolated vertices (distance-hereditary graphs are exactly those graphs generated from a single vertex by those operations); we note that this result has been obtained independently by Bandelt and Mulder [2] in the case of connected distance-hereditary graphs;

---

\* Received by the editors August 11, 1986; accepted for publication (in revised form) May 13, 1987. This work was partially supported by MPI within the National projects on "Design and Analysis of Algorithms" and on "Formal Aspects of Databases."

<sup>†</sup> Dipartimento di Ingegneria Elettrica, Università dell'Aquila, Monteluco, Roio, I-67040 L'Aquila, Italy.

<sup>‡</sup> Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, I-00185 Rome, Italy.

(2) That these graphs correspond exactly to graphs in which all minimal connected subgraphs joining a given set of vertices have the same number of vertices (in other words, the distance-hereditary property can be extended to an arbitrary set of vertices);

(3) That distance-hereditary graphs can be characterized also in terms of properties of their minimal separators and hangings.

In § 4, distance-hereditary graphs are related to various classes of perfect graphs: cographs, parity graphs, strongly chordal graphs,  $(6, 2)$ -chordal bipartite graphs [1], comparability and permutation graphs [16].

In § 5, distance-hereditary graphs are studied from the algorithmic viewpoint. Polynomial algorithms are given:

(1) To recognize distance-hereditary graphs (based on some of the above characterizations);

(2) To solve the cardinality Steiner tree and the connected dominating set problem in this class of graphs.

We note that the Steiner and the connected domination problems play a significant role in discrete optimization and, although there is no obvious relationship between them, they are of the same complexity for a variety of classes of graphs. In particular, the cardinality Steiner tree problem is known to be polynomially solvable for the following classes of perfect graphs: strongly chordal graphs [22], permutation graphs [6], cographs [20], and  $(6, 2)$ -chordal bipartite graphs [1]; on the other hand, this problem is NP-complete [15] for chordal and split graphs [22], bipartite and comparability graphs [20]. Similar results hold for the connected dominating set problem [20].

Since the class of distance-hereditary graphs properly contains some of the largest classes of graphs in which the problems are polynomially solvable, the results given in this paper may be considered as a contribution to a better definition of the bound between tractability and intractability for these problems.

**2. Terminology and notation.** We shall consider finite, simple loopless, undirected graphs  $G = (V, E)$ , where  $V$  is the vertex set and  $E$  is the edge set, and we shall use more-or-less standard terminology from graph theory [3], [17], some of which we review here.

Let  $S$  and  $T$  be two subsets of  $V$ . We denote with  $\langle S \rangle$  the subgraph of  $G$  induced by  $S$ , with  $N_T(S)$  the *neighborhood of  $S$  in  $T$* , that is the set of vertices in  $T$  that are adjacent to any vertex in  $S$  and with  $N_T[S]$  the *closed neighborhood of  $S$  in  $T$* , that is  $N_T(S) \cup S$  (for simplicity we shall omit  $T$  if it coincides with  $V$  and we shall use  $v$  instead of  $\{v\}$  whenever no ambiguity arises). A vertex  $v$  is *isolated* if  $N(v) = \emptyset$ , is a *leaf* if  $|N(v)| = 1$ . Two vertices  $v$  and  $v'$  are *twins* if they have the same neighborhood ( $N(v) = N(v')$ ) or the same closed neighborhood ( $N[v] = N[v']$ ).

The set of vertices  $S$  is a *separator* of  $G$  if either  $|S| = |V| - 1$ , or the removal of  $S$  from  $G$  increases the number of its connected components. A separator of  $G$  is *minimal* if none of its proper subsets is a separator. A *block* of  $G$  is a maximal connected subgraph of  $G$  that contains no cutpoint.

The set of vertices  $S$  is a *domination of  $T$*  if  $T \subseteq N[S]$  and for no  $S' \subseteq V$  such that  $T \subseteq N[S']$  we have  $|S'| < |S|$ . The set  $S$  is a *connected domination of  $T$*  if  $T \subseteq N[S]$ , the induced subgraph  $\langle S \rangle$  is connected and for no  $S' \subseteq V$ , such that  $T \subseteq N[S']$  and  $\langle S' \rangle$  is connected, we have  $|S'| < |S|$ . A (*connected*) *dominating set* of  $G$  is a (connected) domination of  $V$ .

The induced subgraph  $\langle S \rangle$  of  $G$  is a *connection of  $T$*  if  $\langle S \rangle$  is connected and  $T \subseteq S$ . A connection  $\langle S \rangle$  of  $T$  is *nonredundant* if, for no  $v \in S$ , we have that  $\langle S - \{v\} \rangle$  is a connection of  $T$ ; it is a *minimum connection of  $T$*  if no connection of  $T$  has less

than  $|S|$  vertices. A *Steiner tree* of  $T$  is a spanning tree of a minimum connection of  $T$ . An *induced path* is a nonredundant connection of a pair of vertices; an induced path is a *shortest path* if it is a minimum connection. The *distance* between  $u$  and  $v$  in  $G$ , i.e., the length of a shortest path between  $u$  and  $v$  in  $G$ , will be denoted by  $d_G(u, v)$ .

If  $G$  is connected and  $v$  is one of its vertices, the *hanging* of  $G$  by  $v$  is the function  $h_v$  that associates to every  $u \in V$  the value  $d_G(u, v)$  ( $h_v(u)$  is the *level* of  $u$  in the hanging  $h_v$ ).  $L^i(h_v)$  (or simply  $L^i$ , when no ambiguity arises) denotes the set of vertices with level  $i$  in  $h_v$ . Given a hanging  $h_v$  of  $G$ , the *horizontal part* of  $G$  with respect to  $h_v$ , denoted  $H(G, v)$ , is the subgraph  $(V, E')$  of  $G$  in which  $E'$  is the set of edges in  $E$  between pairs of vertices having the same level in  $h_v$ , and the *vertical part* of  $G$  with respect to  $h_v$ , denoted  $V(G, v)$  is the complement of  $H(G, v)$  in  $G$  (see Fig. 1). The concepts of hanging, level, horizontal part and vertical part can be extended to disconnected graphs by considering one hanging vertex for every connected component.

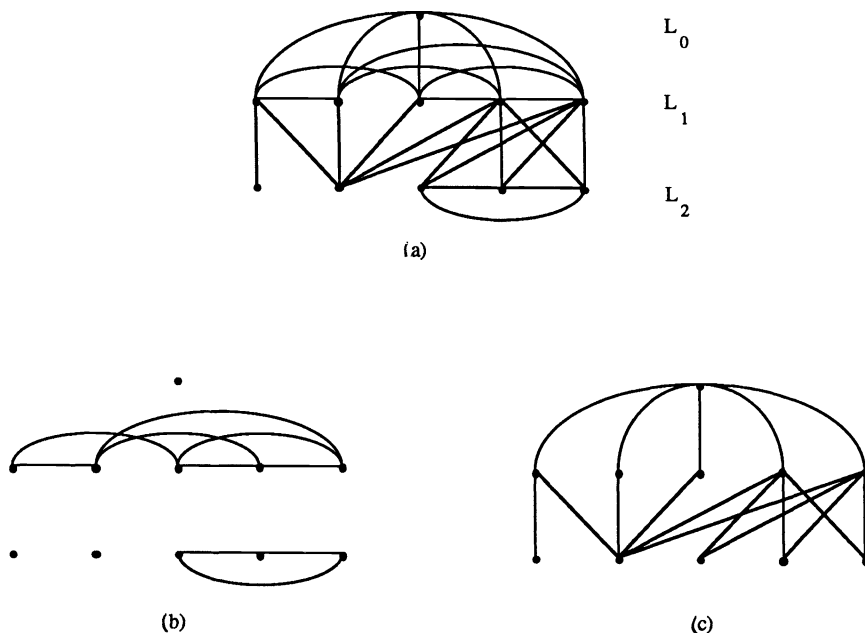


FIG. 1. A hanging of a distance hereditary graph (a) and its horizontal (b) and vertical (c) parts.

A *cograph* [5], [7] is a graph that does not contain any induced path of length three; a *chordal graph* is a graph having a chord in every cycle on four or more vertices; a *strongly chordal graph* [14] is a chordal graph in which every cycle on six or more vertices contains a strong chord (i.e., a chord joining two vertices with an odd distance in the cycle); a *ptolemaic graph* is a connected graph such that for every four vertices  $v_1, v_2, v_3$ , and  $v_4$  the following inequality is satisfied:  $d_G(v_1, v_2) * d_G(v_3, v_4) \leq d_G(v_1, v_3) * d_G(v_2, v_4) + d_G(v_1, v_4) * d_G(v_2, v_3)$ ; a *parity graph* [5] is a graph having two crossing chords in every odd cycle on five or more vertices; a *(6, 2)-chordal bipartite graph* [1] is a bipartite graph having two chords in every cycle on six or more vertices; a *comparability graph* [16] is a graph that is transitively orientable (i.e., there exists an assignment of directions to edges such that if  $(u, v)$  and  $(v, w)$  are arcs then so is  $(u, w)$ ); a *permutation graph* [16] is a comparability graph whose complement is also a comparability graph.

**3. Characterizations.** The following theorem provides several equivalent definitions of distance-hereditary graphs that characterize them in terms of existence of particular kinds of vertices (isolated, leaves, twins) and in terms of properties of connections, separators, and hangings. The results from (1) to (3) are showed by Howorka in [18]; in this section the proof of the remaining equivalences is given. Bandelt and Mulder [2] provided several interesting characterizations of connected distance-hereditary graphs, based on metric properties and forbidden configurations, and also independently proved the equivalence between (1) and (7), in the connected case, by using a different proof technique.

**THEOREM 1.** *Given a graph  $G = (V, E)$  the following statements are equivalent:*

- (1)  $G$  is a distance-hereditary graph.
- (2) Every cycle in  $G$  with five or more vertices has two crossing chords.
- (3) Every induced path in  $G$  is a shortest path.
- (4) Every nonredundant connection in  $G$  is a minimum connection.
- (5) For every  $T \subseteq V$  and every vertex  $v$  in a minimal separator  $S$  of  $\langle T \rangle$  we have  $N_{T-S}(v) = N_{T-S}(S)$ .
- (6) For every cycle  $c$  in  $G$  if  $\{u, v\}$  is a separator of the subgraph  $G'$  induced by  $c$  then  $u$  and  $v$  are twins in  $G'$ .
- (7) Every induced subgraph of  $G$  contains an isolated vertex, or a leaf or a pair of twins.
- (8) For every hanging  $h$  of  $G$  and every pair of vertices  $u, v \in L^i$ ,  $1 \leq i \leq k$ , that are connected in  $\langle V - L^{i-1} \rangle$ , we have  $L^{i-1} \cap N(u) = L^{i-1} \cap N(v)$ .

*Proof.* Throughout the proof we assume that  $|V| \geq 5$  and that  $G$  is connected. The other cases follow immediately. Since the equivalences of statements from (1) to (3) are proved in [18], we will prove the following implications:

$$(4) \Leftrightarrow (3) \Rightarrow (5) \Rightarrow (6) \Rightarrow (2) \Leftrightarrow (7) \quad \text{and} \quad (2) \Leftrightarrow (8).$$

$(3) \Rightarrow (5)$ . Statement (5) trivially holds if  $|S| = 1$  or  $|S| = |V| - 1$ . Therefore, let us suppose that  $1 < |S| < |V| - 1$ . Let  $u$  and  $v$  be two distinct vertices in  $S$  and  $C'$  and  $C''$  be two distinct connected components of  $\langle V - S \rangle$  (if  $|S| < |V| - 1$ , then  $\langle V - S \rangle$  has at least two connected components). Since  $S$  is minimal, we have that  $u$  is adjacent to at least one vertex in  $C'$  and to at least one vertex in  $C''$ ; the same must also be true for  $v$ . Let  $u'$  and  $u''$  be two vertices, in  $C'$  and  $C''$ , respectively, adjacent to  $u$ ; from these considerations it follows that there exist an induced path  $(u', v'_1, \dots, v'_n, v)$  from  $u'$  to  $v$  whose internal vertices are in  $C'$  and an induced path  $(v, v''_1, \dots, v''_m, u'')$  from  $v$  to  $u''$  whose internal vertices are in  $C''$ . Consider the path  $p = (u', v'_1, \dots, v'_n, v, v''_1, \dots, v''_m, u'')$ ; since no vertex in  $C'$  is adjacent to a vertex in  $C''$ ,  $p$  is induced and, due to (3), its length must be two. Hence, both  $u'$  and  $u''$  must be adjacent to  $v$ ; this implies that for every vertex  $v$  in a minimal separator  $S$  of  $G$  we have  $N_{V-S}(v) = N_{V-S}(S)$ . Since every induced subgraph of a distance-hereditary graph is distance-hereditary, (5) is proved.

$(5) \Rightarrow (6)$ . The proof is trivial; in fact,  $\{u, v\}$  is also a minimal separator in  $G'$ .

$(6) \Rightarrow (2)$ . Let us suppose, by contradiction, that there exists in  $G$  a cycle  $c = (v_1, v_2, \dots, v_q, v_1)$ ,  $q \geq 5$ , with no pair of crossing chords. Consider the subgraph  $G'$  of  $G$  induced by the vertices in  $c$ . If  $c$  has no chord then every pair of nonconsecutive vertices in  $c$  is a separator of  $G'$  whose vertices are not twins in  $G'$ . Otherwise, let  $(v_i, v_j)$ ,  $1 < i < j < q$ , be a chord in  $c$ . Since  $c$  has no pair of crossing chords, no vertex in  $\{v_{i+1}, \dots, v_{j-1}\}$  can be adjacent to a vertex in  $\{v_{j+1}, \dots, v_q, v_1, \dots, v_{i-1}\}$ . Therefore, the set  $S = \{v_i, v_j\}$  is a minimal separator for  $G'$ . Furthermore, since  $c$  has no pair of crossing chords,  $(v_{i-1}, v_j)$  and  $(v_i, v_{j+1})$  cannot be both in  $G'$  and a contradiction arises.

(3) $\Rightarrow$ (4). Let  $T$ , with  $|T| \geq 3$  (otherwise the implication is trivial), be the set of vertices in  $G$  to be connected and let  $G' = (V', E')$  and  $G'' = (V'', E'')$  be two distinct nonredundant connections of  $T$ ; we will show that if (3) holds, then there exists a 1-1 correspondence between  $V'$  and  $V''$ . In fact, we associate with every vertex in  $V' - V''$  a vertex in  $V'' - V'$  and we prove that such an association is a 1-1 mapping. First, we prove that:

(a) Given a vertex  $v' \in V' - V''$ , for every pair  $C_1 = \langle V_1, E_1 \rangle$  and  $C_2 = \langle V_2, E_2 \rangle$  of connected components of  $\langle V' - \{v'\} \rangle$  there exists a vertex  $v'' \in V'' - V'$  such that  $N_{V_1 \cup V_2}(v'') = N_{V_1 \cup V_2}(v')$ .

Let  $v_1$  and  $v_k$  be two vertices in  $T \cap V_1$  and  $T \cap V_2$ , respectively, such that  $p' = (v_1, \dots, v_{i-1}, v_i, v_{i+1}, \dots, v_k)$ ,  $k \geq 3$ , is an induced path in  $G'$  from  $v_1$  to  $v_k$  in which  $v_i = v'$  (such a pair of vertices always exists because of the definition of nonredundant connection). Since (3) holds, there exists in  $G''$  an induced path  $p''$  connecting  $v_1$  and  $v_k$  with the same length as  $p'$ . In order to prove (a), we will show that:

(b) There exists in  $p''$  a vertex  $v''$  that is adjacent to both  $v_{i-1}$  and  $v_{i+1}$ .

If  $k = 3$ , the existence of  $v''$  trivially follows from (3). If  $k \geq 4$ , let us suppose, by contradiction, that no vertex in  $p''$  is adjacent to both  $v_{i-1}$  and  $v_{i+1}$ . Let  $v_j, j < i$ , be a vertex in  $p'$  adjacent to at least one vertex in  $p''$  and such that no vertex in  $p'$  between  $v_j$  and  $v_i$  is adjacent to any vertex in  $p''$ ; analogously, let  $v_h, h > i$ , be a vertex in  $p'$  adjacent to at least one vertex in  $p''$  and such that no vertex in  $p'$  between  $v_i$  and  $v_h$  is adjacent to any vertex in  $p''$ ; furthermore, let  $v^*$  and  $v^+$  be two vertices in  $p''$  such that  $v^*$  is adjacent to  $v_j$ ,  $v^+$  is adjacent to  $v_h$  and no vertex in  $p''$ , between  $v^*$  and  $v^+$ , is adjacent to either  $v_j$  or  $v_h$ . Observe that if  $v_j$  and  $v_h$  coincide with  $v_{i-1}$  and  $v_{i+1}$ , respectively, then  $v^*$  and  $v^+$  have to be distinct, otherwise there would be a vertex in  $p''$  adjacent to both  $v_{i-1}$  and  $v_{i+1}$ . Therefore, due to the fact that  $p'$  and  $p''$  are induced paths, there exists an induced path in  $G$  of length greater or equal to three from  $v_{i-1}$  to  $v_{i+1}$ . This implies a contradiction and proves (b).

Now let  $w$  be a vertex in  $V_2$  distinct from  $v_{i+1}$  and adjacent to  $v'$ ; we will show that  $w$  is adjacent to  $v''$ . Let us suppose, by contradiction, that  $w$  is not adjacent to  $v''$ . Since (3) holds,  $w$  and  $v_{i+1}$  cannot be connected in  $C_2$  via an induced path of length greater than two. If  $w$  and  $v_{i+1}$  are adjacent, the path  $(v_{i-1}, v'', v_{i+1}, w)$  is an induced path longer than  $(v_{i-1}, v', w)$  (contradiction). If  $w$  and  $v_{i+1}$  are not adjacent, let  $z$  be a vertex in  $V_2$  adjacent to both  $w$  and  $v_{i+1}$ . In this case, the only admissible chord in the path  $p^* = (v_{i-1}, v'', v_{i+1}, z, w)$  is  $(v'', z)$ . Therefore, there exists an induced subpath of  $p^*$  between  $v_{i-1}$  and  $w$  longer than  $(v_{i-1}, v', w)$  (contradiction).

Since analogous considerations can be done if  $w$  is a vertex in  $V_1$  distinct from  $v_{i-1}$  and adjacent to  $v'$ , we can conclude that  $v''$  is adjacent to every vertex in  $N_{V_1 \cup V_2}(v')$ . Analogously, it is possible to prove that  $v'$  is adjacent to every vertex in  $N_{V_1 \cup V_2}(v'')$ . Furthermore,  $v''$  cannot be in  $V'$ , otherwise  $G'$  would not be nonredundant, and (a) is proved.

We now show that:

(c) There exists exactly one vertex  $v''$  belonging to  $V'' - V'$  such that  $N_{V'}(v') = N_{V' - \{v'\}}(v'')$ ; furthermore, for every  $v^* \in V'$  distinct from  $v'$  we have  $N_{V'}(v'') \neq N_{V'}(v^*)$ .

If  $|N_{V'}(v')| = 2$ , then the existence of  $v''$  trivially follows from (a). If  $|N_{V'}(v')| \geq 3$ , let  $C_1$  and  $C_2$  be as in (a); then there exists a vertex  $v''_1 \in V''$  such that  $N_{V_1 \cup V_2}(v''_1) = N_{V_1 \cup V_2}(v')$ . Therefore, if there exists a vertex  $v_3 \in N_{V'}(v')$  that is not adjacent to  $v''_1$ , such a vertex must be in a connected component  $C_3 = \langle V_3, E_3 \rangle$  of  $\langle V' - \{v'\} \rangle$  distinct from both  $C_1$  and  $C_2$ . Because of (a) there exists a vertex  $v''_2 \in V''$  (distinct from  $v''_1$ ) such that  $N_{V_2 \cup V_3}(v''_2) = N_{V_2 \cup V_3}(v')$ . We will show that every vertex in  $C_1$  adjacent to  $v'$  is adjacent to  $v''_2$ . In fact, let us suppose, by contradiction, that there exists a vertex

$v_1$  in  $C_1$  adjacent to  $v'$  and not adjacent to  $v_2''$  and let  $v_2$  be any vertex in  $C_2$  adjacent to  $v'$ . In this case, the only admissible chord in the path  $p^* = (v_1, v_1'', v_2, v_2'', v_3)$  is  $(v_1'', v_2'')$ . Therefore, there exists an induced subpath of  $p^*$  between  $v_1$  and  $v_3$  longer than  $(v_1, v', v_3)$  (contradiction).

By repeatedly applying the above argument, because of the finiteness of  $V'$ , it is possible to see that there exists a vertex  $v'' \in V''$  such that  $N_{V'}(v') \subseteq N_{V'-\{v'\}}(v'')$ . In order to prove that  $N_{V'}(v') = N_{V'-\{v'\}}(v'')$ , let  $C_1$  and  $C_2$  be as above, and let us suppose, by contradiction, that there exists a vertex  $u$  in  $C_1$  adjacent to  $v''$  and not adjacent to  $v'$ ; furthermore, let  $v_1$  and  $v_2$  be two vertices in  $C_1$  and  $C_2$ , respectively, adjacent to  $v'$ , and let  $(u, u_1, \dots, u_n, v_1, v')$ ,  $n \geq 0$ , be an induced path from  $u$  to  $v'$  whose internal vertices are in  $C_1$ ; we have that the path  $(u, u_1, \dots, u_n, v_1, v', v_2)$  is an induced path from  $u$  to  $v_2$  longer than  $(u, v'', v_2)$ ; hence, a contradiction arises.

Furthermore, due to the fact that  $G''$  is nonredundant, no vertex in  $V''$  distinct from  $v''$  can enjoy the same property. Finally, due to the fact that  $G'$  is nonredundant,  $v'' \notin V'$  and for no vertex  $v^*$  in  $G'$  distinct from  $v'$  we may have  $N_{V'}(v'') = N_{V'}(v^*)$ .

(4) $\Rightarrow$ (3). It trivially follows from the definitions of nonredundant connection and induced path.

(2) $\Rightarrow$ (7). Since an induced subgraph of a distance-hereditary graph is distance-hereditary, it is sufficient to prove that if every cycle in  $G$  of length five or more has two crossing chords then  $G$  contains an isolated vertex or a leaf or a pair of twins.

Since every cycle in a graph is contained in one block, at least one block  $G' = (V', E')$  exists in  $G$  sharing at most one vertex with the remaining blocks. If  $|V'| = 1$  then  $G$  has an isolated vertex. If  $|V'| = 2$  then the vertex belonging only to  $G'$  is a leaf in  $G$ . Therefore, it is sufficient to prove that:

(a) If  $|V'| > 2$  there exist in  $G'$  two pairs of twins.

In fact, if  $v$  is the vertex that  $G'$  shares with the remaining blocks and (a) holds then there exists at least one pair of vertices of  $G$  (both distinct from  $v$ ) that are twins in  $G'$ ; since they cannot be in a block distinct from  $G'$ , they are also twins in  $G$  and (2) $\Rightarrow$ (7) is proved. In order to prove (a) we will prove a number of facts.

(b) If every cycle in  $G$  of length five or more has two crossing chords then for every minimal separator  $S$  of  $G$  we have that  $\langle S \rangle$  is a cograph.

Since (2) $\Rightarrow$ (5), if  $v$  is a vertex in  $N_{V-S}(S)$  (because of the definition of minimal separator  $|N_{V-S}(S)| \geq 1$ ) all vertices in  $S$  have the same level in  $h_v$ . Then, by (2) $\Rightarrow$ (3), every induced path in  $\langle S \rangle$  cannot be longer than two and (b) is proved.

(c) If every cycle in  $G$  of length five or more has two crossing chords every minimal separator  $S$  of  $G$  either is a cutpoint or contains at least one pair of twins.

Because of (b),  $\langle S \rangle$  is a cograph. It has been proved in [7] that in every nontrivial cograph there are two vertices that are twins; therefore if  $S$  is not a cutpoint it contains two vertices  $u$  and  $v$  such that  $N_S(u) = N_S(v)$  or  $N_S[u] = N_S[v]$ . Since (2) $\Rightarrow$ (5) it follows that  $N_V(u) = N_V(v)$  or  $N_V[u] = N_V[v]$  and (c) is proved.

(d) In every nontrivial connected graph there exist at least two minimal separators.

Let  $G$  be a nontrivial connected graph with at least two vertices.  $G$  must contain at least one nonempty minimal separator  $S$ . Let  $v$  be a vertex in  $S$ ; we have that  $V - \{v\}$  is a separator of  $G$ , then  $G$  must have a minimal separator distinct from  $S$  and (d) is proved.

We are now able to prove (a). In fact, since  $G'$  is a block, every minimal separator  $S$  in  $G'$  contains at least two vertices. Because of (d) this implies that  $G$  has at least two minimal separators whose cardinality is greater than or equal to two. Finally, due to (c), (a) is proved.

(7) $\Rightarrow$ (2). Let us suppose, by contradiction, that there exists in  $G$  a cycle  $c = (v_1, v_2, \dots, v_q, v_1)$ ,  $q \geq 5$ , with no pair of crossing chords. Consider the subgraph  $G'$

of  $G$  induced by the vertices in  $c$ ; no vertex in  $c$  can be a leaf or an isolated vertex in  $G'$ . Furthermore, let  $v_i$  and  $v_j$  be any pair of distinct vertices in  $c$ ; since  $q \geq 5$  we may assume without loss of generality that  $1 < i < j < q$ . Since  $c$  has no pair of crossing chords,  $(v_{i-1}, v_j)$  and  $(v_i, v_{j+1})$  cannot be both in  $G'$ ; hence  $v_i$  and  $v_j$  cannot be twins in  $G'$  (contradiction).

(2) $\Rightarrow$ (8). Let  $h_v$  be a hanging of  $G$ , and  $v'$  and  $v''$  be a pair of vertices having the same level  $i$  in  $h_v$  and connected by an induced path  $p$  whose vertices have level in  $h_v$  greater or equal to  $i$ . Since the implication trivially holds for  $i = 1$ , let  $i > 1$  and let us suppose, by contradiction, that  $L^{i-1} \cap N(v') \neq L^{i-1} \cap N(v'')$ . Without loss of generality, we assume that there exists a vertex  $v^* \in L^{i-1}(h_v) \cap (N(v') - N(v''))$ . Consider two induced paths  $(v'_i, v'_{i-1}, \dots, v'_1, v'_0)$  and  $(v''_i, v''_{i-1}, \dots, v''_1, v''_0)$ , of length greater than one, such that  $v'_i = v'$ ,  $v''_i = v''$ ,  $v'_{i-1} = v^*$  and for every  $k$ ,  $0 \leq k \leq i-1$ ,  $v'_k$  and  $v''_k$  have level  $k$  in  $h_v$  (therefore  $v'_0 = v''_0 = v$ ); let  $j$  be the maximum  $k$ ,  $k < i$ , such that  $v'_k$  is adjacent to  $v''_{k-1}$  or  $v''_{k-1}$  is adjacent to  $v'_k$ . Suppose that  $v'_j$  is adjacent to  $v''_{j-1}$  and let us consider the induced paths  $p' = (v'_{i-1}, \dots, v'_j, v''_{j-1})$  and  $p'' = (v''_i, v''_{i-1}, \dots, v''_{j-1})$ . Let  $v^+$  be the nearest vertex to  $v''$  among those of  $p$  that are adjacent to  $v'_{i-1}$  and let us consider the cycle generated by  $p'$ ,  $p''$ ,  $(v'_{i-1}, v^+)$  and the subpath of  $p$  between  $v^+$  and  $v''$ . The length of such a cycle is greater than four; furthermore, the only admissible chords in this cycle are the chords  $(v'_k, v''_k)$ ,  $i-1 \leq k < j-1$  (that is chords connecting vertices with the same level), and chords connecting  $v''_{i-1}$  and a vertex in  $p$ ; therefore, the cycle contains no pair of crossing chords (contradiction). Analogous considerations can be done if  $v''_j$  is adjacent to  $v'_{j-1}$ .

(8) $\Rightarrow$ (2). Let us suppose, by contradiction, that there exists in  $G$  a cycle  $c = (v_1, v_2, \dots, v_q, v_1)$ ,  $q \geq 5$ , with no pair of crossing chords. First we prove that:

(a) There exists at least one vertex in  $c$  that is not an endpoint of any chord in  $c$ .

Let us suppose, by contradiction, that every vertex in  $c$  be an endpoint of a chord in  $c$ . Let  $(v_i, v_j)$ ,  $i < j$ , be a chord in  $c$ ; if there exists a vertex in  $c$  that is consecutive with both  $v_i$  and  $v_j$ , then every chord starting from such a vertex crosses  $(v_i, v_j)$  (contradiction).

Otherwise, if no chord crosses  $(v_i, v_j)$ , we can consider a chord  $(v_h, v_k)$ , such that  $i < h < k < j$ , and repeatedly apply the same argument until a contradiction arises and (a) is proved.

Now let  $v_n$  be a vertex in  $c$  that is not an endpoint of any chord in  $c$ . Without loss of generality, we can assume that  $2 < n < q-1$ . Consider the hanging  $h_{v_n}$  of  $G$ . Since no chord starts from  $v_n$ , we have that the only vertices in  $c$  having level one in  $h_{v_n}$  are  $v_{n-1}$  and  $v_{n+1}$ ; therefore  $v_{n-2}$  and  $v_{n+2}$  have level two in  $h_{v_n}$  and are connected by a path (the path in  $c$  that does not have  $v_{n-1}$ ,  $v_n$ , and  $v_{n+1}$  as internal vertices) whose vertices have level greater or equal to two. Since  $c$  has no pair of crossing chords, the edges  $(v_{n-2}, v_{n+1})$  and  $(v_{n-1}, v_{n+2})$  cannot be both in  $G$  and a contradiction arises.  $\square$

Characterization (8) provides a simple property to be satisfied by every hanging in order that a graph be distance-hereditary. Observe that Bandelt and Mulder, starting from an arbitrary hanging, provided a more complex characterization based on five conditions (see [2, Thm. 3]); two of them (conditions (1) and (3)) are implied by our characterization (8).

**4. Relationships between distance-hereditary graphs and other classes of perfect graphs.** We now relate distance-hereditary graphs with other kinds of graphs defined in § 2.

From the definitions of a distance-hereditary graph and a parity graph it trivially follows that a distance-hereditary graph is a parity graph. Furthermore, Howorka [19]



proved that Ptolemaic graphs are precisely chordal distance hereditary graphs. The following proposition states the relationships between the class of distance-hereditary graphs and the classes of cographs, (6, 2)-chordal bipartite graphs, comparability graphs and permutation graphs (see Fig. 2).

PROPOSITION 1. (i) *Cographs are distance-hereditary graphs; there are distance-hereditary graphs that are not cographs.*

(ii) *(6, 2)-chordal bipartite graphs are precisely bipartite distance-hereditary graphs.*

(iii) *There are distance-hereditary graphs that are not comparability graphs.*

(iv) *There are permutation graphs that are not distance-hereditary graphs.*

*Proof of (i).* It has been proved in [7] that in every nontrivial cograph there are two vertices that are twins, and that every induced subgraph of a cograph is a cograph; due to (7) in Theorem 1, this implies that a cograph is a distance-hereditary graph. The graph in Fig. 3 is a distance-hereditary graph that is not a cograph.

*Proof of (ii).* The proof immediately follows from the following result provided by Howorka [18]: a graph is distance-hereditary if and only if every cycle with five or more vertices has two chords and every cycle with five vertices has two crossing chords.

*Proof of (iii) and (iv).* See the graphs in Figs. 4 and 5.  $\square$

Finally, the following theorem gives a property of distance-hereditary graphs which is similar to a decomposition property proved in [4] and [5] for Meyniel and parity graphs; in particular it shows that a distance-hereditary graph can be easily decomposed into a cograph and a (6, 2)-chordal bipartite graph (without common edges) by hanging it with respect to one of its vertices. This property has been

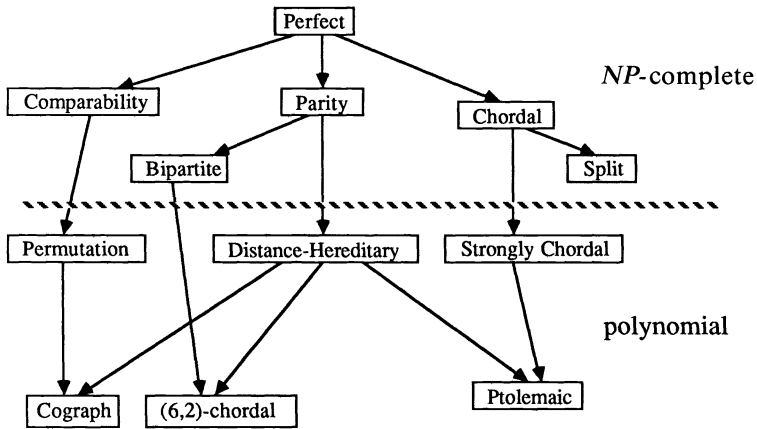


FIG. 2. Containment relations and bound between tractability and intractability of Steiner tree and connected domination problems for classes discussed in this paper.

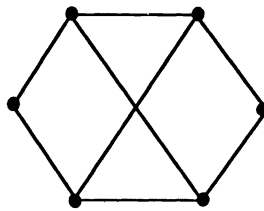


FIG. 3. A distance-hereditary graph that is not a cograph.

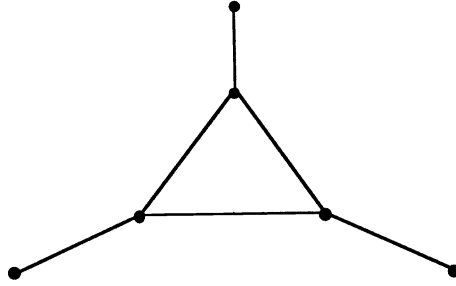


FIG. 4. A distance-hereditary graph that is not a comparability graph.

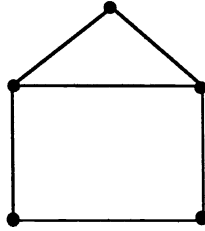


FIG. 5. A permutation graph that is not a distance-hereditary graph.

independently proved by Bandelt and Mulder [2, Property (2) in Theorem 3 and Corollary 5].

**THEOREM 2.** *Let  $G$  be a distance-hereditary graph. For every hanging  $h$  of  $G$ , the horizontal and the vertical part of  $G$  with respect to  $h$  are a cograph and a  $(6, 2)$ -chordal bipartite graph, respectively.*

*Proof.* It has been shown [5] that, if  $G$  is a parity graph,  $H(G, h)$  is a cograph. Due to the containment of the class of distance-hereditary graphs in the class of parity graphs, the first part of the statement is proved.

In order to prove the second part, let us assume, without loss of generality, that  $G$  is connected and let  $h_v$  be a hanging of  $G$  and  $k$  be the maximum level of vertices of  $G$  in  $h_v$ . The statement is trivial if  $k \leq 1$ . Otherwise, let  $V_1$  and  $V_2$  be the set of vertices in  $V$  having level even and odd, respectively; it is easy to see that  $V(G, h_v)$  is bipartite with respect to  $V_1$  and  $V_2$ . Furthermore, let us suppose, by contradiction, that there exists a cycle  $c = (v_1, v_2, \dots, v_q, v_1)$  in  $V(G, h_v)$ , of length  $q \geq 6$ , with at most one chord. Let  $j$  be the maximum level in  $h_v$  of vertices in  $c$  and let  $v_i$  be a vertex in  $c$  having level  $j$ ; without loss of generality, we may assume that  $3 \leq i \leq q - 2$ . The following cases may arise.

*Case (1).* Both  $v_{i-2}$  and  $v_{i+2}$  have level  $j - 2$ . In this case  $(v_{i-1}, v_i, v_{i+1})$  is a path between  $v_{i-1}$  and  $v_{i+1}$  whose vertices have level greater than or equal to  $j$ . Since either  $(v_{i-2}, v_{i+1})$  or  $(v_{i-1}, v_{i+2})$  is not in  $E$ , due to (8) in Theorem 1, a contradiction arises.

*Case (2).* Either  $v_{i-2}$  or  $v_{i+2}$  has level  $j - 2$ . Suppose  $v_{i-2}$  has level  $j - 2$ . If  $(v_{i-2}, v_{i+1})$  is not in  $E$  then the same considerations as in Case (1) can be done. Otherwise, again by (8) in Theorem 1,  $(v_{i-2}, v_{i+3}) \in E$ ; since  $c$  has at most one chord, this implies that  $q = i + 3 = 6$ . The cycle  $c$  cannot have any of the following chords:

- $(v_2, v_5)$  and  $(v_3, v_6)$ ; in fact,  $c$  cannot have a chord distinct from  $(v_1, v_4)$  in  $V(G, h_v)$
- $(v_3, v_5)$ ; in fact, if  $(v_3, v_5) \in E$  then  $(v_2, v_5), (v_3, v_6) \in E$ .

Hence, we have that  $(v_2, v_6) \in E$ . Consider the cycle  $(v_2, v_3, v_4, v_5, v_6)$ ; this cycle has no pair of crossing chords (contradiction).

*Case (3).* Both  $v_{i-2}$  and  $v_{i+2}$  have level  $j$ . If  $q = i + 3 = 6$ , we have that the only admissible chords in  $G$  are  $(v_2, v_4)$ ,  $(v_2, v_6)$ ,  $(v_4, v_6)$  and at most one chord in  $\{(v_1, v_4), (v_2, v_5), (v_3, v_6)\}$  (otherwise  $c$  would have two chords in  $V(G, h_v)$ ). In fact, by Theorem 1, the existence of  $(v_1, v_3)$  or  $(v_1, v_5)$  or  $(v_3, v_5)$  would imply the existence of two chords in  $\{(v_1, v_4), (v_2, v_5), (v_3, v_6)\}$ . Therefore, there exists a cycle of length five in  $G$  without crossing chords (contradiction).

If  $q > 6$  we have that  $v_{i-3}$  and  $v_{i+3}$  have level  $j - 1$ . Since  $v_{i-1}$  and  $v_{i+1}$  have level  $j - 1$  in  $h_v$ , again by Theorem 1, there exists a vertex  $v'$  having level  $j - 2$  and adjacent to  $v_{i-3}$ ,  $v_{i+3}$ ,  $v_{i-1}$ , and  $v_{i+1}$ .

If neither  $(v_{i-3}, v_i)$  nor  $(v_{i-2}, v_{i+1})$  is in  $E$ , consider the cycle  $c' = (v_{i-3}, v_{i-2}, v_{i-1}, v_i, v_{i+1}, v', v_{i+3})$ . We have that  $(v_{i-2}, v_i)$  cannot be a chord of  $c'$ , in fact if  $(v_{i-2}, v_i) \in E$ , then  $(v_{i-3}, v_i)$ ,  $(v_{i-2}, v_{i+1}) \in E$ . Therefore  $(v_{i-3}, v_{i+1})$  must be in  $E$ . Consider the cycle  $(v_{i-3}, v_{i-2}, v_{i-1}, v_i, v_{i+1}, v_{i+3})$ ; this cycle has no pair of crossing chords (contradiction).

If either  $(v_{i-3}, v_i)$  or  $(v_{i-2}, v_{i+1})$  is in  $E$ , analogous considerations can be done by considering the cycle  $(v_{i-1}, v_i, v_{i+1}, v_{i+2}, v_{i+3}, v', v_{i-1})$ .  $\square$

**5. Recognition, Steiner tree, and connected domination problems.** The recognition problem (i.e., the problem of deciding in polynomial time whether or not a graph belongs to a given class) is open for the class of perfect graphs, but it has been solved for several of its subclasses, such as the ones discussed in the previous sections. In particular, Burlet and Fonlupt [4] provide an algorithm, based on a decomposition process, that recognizes in polynomial time Meyniel graphs; an easy adaptation of this algorithm (obtained by restricting the decomposition operation and the type of final inseparable components) allows us to also recognize parity graphs (see [4] and [5]).

Two recognition algorithms for distance-hereditary graphs are now presented. The first is a procedure that reduces a distance-hereditary graph into the empty graph, by recursively eliminating vertices with particular properties (similarly to what is done for cographs in [7]); the second one is based on connectivity tests in the graph hangings. Such algorithms are a straightforward consequence of characterizations (7) and (8) in Theorem 1 and it is easy to see that they work in polynomial time.

ALGORITHM 1.

```

input a graph  $G = (V, E)$ ;
output the variable succeeds (true if and only if  $G$  is a distance-hereditary graph);
procedure delete ( $G, succeeds$ );
begin
   $W := \emptyset$ ;
  for every  $v \in G$  do
    if  $v$  is an isolated vertex or a leaf or if  $v$  has a twin in  $G$  then
      begin  $G := \langle V - \{v\} \rangle$ ;  $W := W \cup \{v\}$  end;
    if  $G \neq \emptyset$  then if  $W = \emptyset$  then  $succeeds := false$  else delete ( $G, succeeds$ )
  end;
begin  $succeeds := true$ ; delete ( $G, succeeds$ ) end.

```

ALGORITHM 2

```

input and output as above;
begin
   $succeeds := true$ ;

```

```

for every connected component  $C$  of  $G$  do
  for every  $w \in C$  do
    begin
      determine  $h_w$ ;
      for every pair  $u, v \in C$  do
        if  $h_w(u) = h_w(v) = i$ 
          and  $u$  and  $v$  are connected in  $\langle V - L^{i-1}(h_w) \rangle$ 
          and  $L^{i-1}(h_w) \cap N(u) \neq L^{i-1}(h_w) \cap N(v)$  then
            begin  $succeeds := false$ ; exit end
          end
        end
      end
    end.

```

We now present two polynomial algorithms to find Steiner trees and connected dominations in distance-hereditary graphs, respectively. First, characterization (3) in Theorem 1 suggests a simple greedy technique to find Steiner trees in distance-hereditary graphs in polynomial time by using the following algorithm:

ALGORITHM 3.

```

Input a connected graph  $G = (V, E)$  and a subset  $S$  of its vertices
output a subtree of  $G$  that includes all vertices in  $S$ ;
begin
  for every  $v \in V - S$ 
    if  $\langle V - \{v\} \rangle$  is a connection of  $S$  then  $G := \langle V - \{v\} \rangle$ ;
  determine a spanning tree of  $G$ 
end.

```

THEOREM 3. *If  $G$  is a distance-hereditary graph Algorithm 3 determines a Steiner tree of  $S$  in  $O(|V| * |E|)$  time.*

*Proof.* In fact, the first step in Algorithm 3 determines a nonredundant connection of  $S$  that, by Theorem 1, is also minimum. Since such a step can be done in time  $O(|V| * |E|)$  (in fact, it requires a connectivity test for every vertex in  $V - S$ ) and a spanning tree can be found in linear time, the overall complexity is  $O(|V| * |E|)$ .  $\square$

Also characterization (8) in Theorem 1 has an interesting algorithmic impact. In fact, we now prove that a polynomial algorithm based on this characterization exists to solve the connected domination problem in distance-hereditary graphs.

In order to approach the problem, let us provide some technical lemmas.

LEMMA 1. *Let  $G = (V, E)$  be a distance-hereditary graph,  $h_w$  be a hanging of  $G$ ,  $k$  be  $\max(h_w(v) | v \in V)$ , and  $D \subseteq V$  such that  $\langle D \rangle$  is connected and  $V = N[D]$ . We have that:*

- (a) *For every  $i, 0 < i < k, D \cap L^i \neq \emptyset$ ;*
- (b) *Every vertex  $v \in L^i, 2 \leq i \leq k$ , is adjacent to a vertex  $u \in D \cap L^{i-1}$ ;*
- (c) *If  $w \notin D$ , then  $\langle D \cap (L^1 \cup L^2) \rangle$  is connected.*

*Proof.* In order to prove (a), let us suppose, by contradiction, that there exists  $i, 0 < i < k$ , such that  $D$  does not contain any vertex in  $L^i$ . Three cases arise:

— There exists a pair of vertices  $u, v \in D$  such that  $h_w(u) < i$  and  $h_w(v) > i$ ; in this case  $D$  cannot be connected;

— For every  $v \in D$ , we have  $h_w(v) < i$ ; this implies that no vertex in  $L^k$  could be in  $N[D]$ ;

— For every vertex  $v \in D, h_w(v) > i$ ; in this case  $w$  could not be in  $N[D]$ .

Since in all cases a contradiction arises, (a) is proved.

In order to prove (b), let  $v$  be a vertex of  $G$  such that  $h_w(v) = i \geq 2$  and let  $z \in D$  be adjacent to  $v$ . Let us suppose that  $h_w(z) = i$ . By (a) and by the hypothesis that  $\langle D \rangle$

is connected, there exists a vertex  $u \in D$  with  $h_w(u) = i - 1$  connected in  $\langle D \rangle$  to  $z$  via a path  $p$ . Let  $u'$  be the nearest vertex to  $z$  on  $p$  having level  $i - 1$ ; by (8) in Theorem 1,  $v$  is adjacent to  $u'$ . Since similar considerations can be done if  $h_w(z) = i + 1$ , (b) is proved.

Finally, due to (a) and (b), every vertex in  $D \cap L^2$  is adjacent to a vertex in  $D \cap L^1$ . Furthermore, if  $w \notin D$  then for every pair of nonadjacent vertices in  $D \cap L^1$ , by (3) in Theorem 1, there exists in  $\langle D \rangle$  a path of length two connecting them; the only internal vertex in such a path must be either in  $D \cap L^1$  or in  $D \cap L^2$ ; this proves (c).  $\square$

LEMMA 2. *Let  $G = (V, E)$  be a nontrivial distance-hereditary graph, and let  $h_w$  and  $k$  be as in Lemma 1. There exists a connected dominating set of  $G$  that does not contain any vertex in  $L^k$ .*

*Proof.* Let  $D$  be a subset of  $V$  such that  $\langle D \rangle$  is connected and  $V = N[D]$ , and let  $v \in D$  be a vertex such that  $h_w(v) = k$ .

If  $k = 1$ , then  $\{w\}$  is a connected dominating set of  $G$ .

If  $k = 2$ , then every vertex in  $L^1$  is adjacent to  $w$  and every vertex in  $L^2$  is adjacent to a vertex in  $D \cap L^1$  (because of (b) in Lemma 1). Therefore, if  $D$  is a connected dominating set of  $G$ , then  $D - \{v\} \cup \{w\}$  is a connected dominating set of  $G$ .

If  $k > 2$  then, by Lemma 1, there exist a vertex  $u \in D \cap L^{k-1}$  adjacent to  $v$  and a vertex  $z \in D \cap L^{k-2}$  adjacent to  $u$ ; due to (8) in Theorem 1, every vertex  $v'$  adjacent to  $v$ , with  $h_w(v') = k$ , is also adjacent to  $u$ , and every vertex  $v''$  adjacent to  $v$ , with  $h_w(v'') = k - 1$ , is also adjacent to  $z$ ; therefore  $D$  cannot be a connected dominating set of  $G$ .  $\square$

LEMMA 3. *Let  $G, h_w$  and  $k > 2$  be as in Lemma 1. Let  $G_i, 1 \leq i \leq 2$ , be the subgraph of  $V(G, h_w)$  induced by  $L^i \cup L^{i+1}$ ,  $S_1$  and  $S'_1$  be two subsets of  $L^1$  that are dominations of  $L^2$  in  $G_1$ , and  $S_2$  and  $S'_2$  be two subsets of  $L^2$  that are dominations of  $L^3$  in  $G_2$ . We have that:*

(a)  $N(S_2) \cap L^1 = N(S'_2) \cap L^1$ ;

(b) *If  $S'_1 \subseteq N[S_1]$ , then  $S_1 \subseteq N[S'_1]$ ;*

(c) *If  $\langle S_1 \cup S_2 \rangle$  is connected, then  $\langle S'_1 \cup S'_2 \rangle$  is connected;*

(d) *If  $\langle S_1 \cup S_2 \rangle$  is connected and  $|S_1| = |S'_1| > 1$ , then  $N[S_1 \cup S_2] \cap L^1 = N[S'_1 \cup S'_2] \cap L^1$ .*

*Furthermore, the properties above hold if  $k = 2$ , by assuming  $S_2 = S'_2 = \emptyset$ .*

*Proof.* In order to prove (a), let us suppose, by contradiction, that there exists a vertex  $v \in L^1$  adjacent to a vertex  $u \in S_2$  and not adjacent to any vertex in  $S'_2$ . Let  $z \in L^3$  be a vertex adjacent to  $u$ ; since  $S_2$  and  $S'_2$  are dominations of  $L^3$  in  $G_2$ , there exists a vertex  $y \in S'_2$  (hence distinct from  $u$ ) adjacent to  $z$ . Due to (8) in Theorem 1,  $y$  must be adjacent to  $v$  and a contradiction arises.

In order to prove (b), let us suppose, by contradiction, that there exists a vertex  $v \in S_1 - S'_1$  that is not adjacent to any vertex in  $S'_1$ . Since  $S_1$  is a domination of  $L^2$  in  $G_1$ , there exists a vertex  $z \in L^2$  adjacent to  $v$  and not adjacent to any vertex in  $S_1$  distinct from  $v$ . Let  $y$  be a vertex in  $S'_1 - S_1$  adjacent to  $z$ . Since  $S'_1 \subseteq N[S_1]$ , there exists a vertex  $u \in S_1$  adjacent to  $y$ .

If  $u \in S_1 \cap S'_1$ , let us consider the cycle  $c_1 = (w, v, z, y, u, w)$ . We have that  $c_1$  cannot have any of the following chords:

—  $(w, z)$ , in fact,  $h_w(z) = 2$ ;

—  $(v, u)$  and  $(v, y)$ , since  $v$  is not adjacent to any vertex in  $S'_1$ ;

—  $(z, u)$ , since  $z$  is not adjacent to any vertex in  $S_1$  distinct from  $v$ .

Therefore,  $c_1$  has no pair of crossing chords and a contradiction arises.

If  $u \in S_1 - S'_1$ ,  $u$  must be adjacent to  $v$ , otherwise  $c_1$  would have no pair of crossing chords. Let  $z' \in L^2$  be adjacent to  $u$  and not adjacent to any vertex in  $S_1$  distinct from

$u$ , and let  $y' \in S'_1 - S_1$  be adjacent to  $z'$ . Consider the cycle  $c_2 = (w, v, u, z', y', w)$ . We have that  $c_2$  cannot have any of the following chords:

- $(w, z')$ , in fact,  $h_w(z') = 2$ ;
- $(v, z')$ , since  $z'$  is not adjacent to any vertex in  $S_1$  distinct from  $u$ ;
- $(v, y')$ , since  $v$  is not adjacent to any vertex in  $S'_1$ .

Therefore,  $c_2$  has no pair of crossing chords and a contradiction arises that proves (b).

In order to prove (c), let  $|S_1| = |S'_1| \geq 2$  (in fact (c) trivially holds if  $|S_1| = |S'_1| = 1$ ) and let  $u'$  and  $u''$  be a pair of nonadjacent vertices in  $S'_1$ ; we prove that there exists a vertex  $u \in S'_1 \cup S'_2$  adjacent to both  $u'$  and  $u''$ . First we prove that:

- (i) There exists a vertex  $v \in S_1 \cup S_2$  adjacent to both  $u'$  and  $u''$ .

Three cases arise.

*Case (1)  $u', u'' \in S_1$ .* Due to the connectivity of  $\langle S_1 \cup S_2 \rangle$  and to (3) in Theorem 1, (i) trivially holds.

*Case (2)  $u'' \in S_1$ .* Since  $S'_1$  is a domination of  $L^2$  in  $G_1$ , there exists a vertex  $z' \in L^2$  such that  $N(z') \cap S'_1 = \{u'\}$ . Let  $v' \in S_1 - S'_1$  be adjacent to  $z'$ ;  $v'$  cannot be adjacent to  $u''$ , otherwise the cycle  $(w, u', z', v', u'', w)$  would have no pair of crossing chords. Hence, due to the connectivity of  $\langle S_1 \cup S_2 \rangle$  and to (3) in Theorem 1, there exists a vertex  $v \in S_1 \cup S_2$  adjacent to both  $v'$  and  $u''$ . The cycle  $c = (w, u'', v, v', z', u', w)$  must have the chord  $(v, u')$ . In fact  $c$  cannot have the following chords:

- $(w, z')$ , since  $h_w(z') = 2$ ;
- $(u, u'')$ , by hypothesis;
- $(z', u'')$ , because of the choice of  $z'$ ;
- $(v', u'')$ , see above.

*Case (3)  $u', u'' \in S'_1 - S_1$ .* Since  $S'_1$  is a domination of  $L^2$  in  $G_1$ , there exist two vertices  $z', z'' \in L^2$  such that  $N(z') \cap S'_1 = \{u'\}$  and  $N(z'') \cap S'_1 = \{u''\}$ . Let  $v'$  and  $v''$  be two vertices in  $S_1$  adjacent to  $z'$  and  $z''$ , respectively. Due to the connectivity of  $\langle S_1 \cup S_2 \rangle$  and to (3) in Theorem 1, there exists in  $\langle S_1 \cup S_2 \rangle$  an induced path from  $v'$  to  $v''$  whose length is less than or equal to two. We will show that:

- (ii)  $v'$  is distinct from  $v''$ , and
- (iii)  $v'$  and  $v''$  are not adjacent.

In fact, if  $v'$  and  $v''$  coincide then the cycle  $c_1 = (w, u', z', v', z'', u'', w)$  cannot have any of the following chords:

- $(w, z')$  and  $(w, z'')$ , in fact  $h_w(z') = h_w(z'') = 2$ ;
- $(u', u'')$ ,  $(u', z'')$  and  $(u'', z')$ , because of the choice of  $u', u'', z'$ , and  $z''$ ;
- $(z', z'')$ , in fact its existence should imply the existence of both  $(u', z'')$  and  $(u'', z')$ .

Therefore, the only admissible chords in  $c_1$  should be those starting from  $v'$  and a contradiction arises which proves (ii). If  $v'$  and  $v''$  are adjacent then, analogously, the cycle  $c_2 = (w, u', z', v', v'', z'', u'', w)$  cannot have the chords  $(w, z')$ ,  $(w, z'')$ ,  $(u', u'')$ ,  $(u', z'')$ ,  $(u'', z')$  and  $(z', z'')$ . Furthermore,  $c_2$  cannot have the chords:

- $(z', v'')$  and  $(z'', v')$ , because of (ii);
- $(u', v'')$  and  $(u'', v')$ , otherwise the cycle  $(w, u', v'', z'', u'', w)$  and the cycle  $(w, u', z', v', u'', w)$  would have no pair of crossing chords, respectively.

Therefore the only admissible chords in  $c_2$  are  $(w, v')$ ,  $(w, v'')$ ,  $(u', v')$ , and  $(u'', v'')$  (no pair crosses), and (iii) is proved.

Therefore, let  $v \in S_1 \cup S_2$  be adjacent to both  $v'$  and  $v''$ . Consider the cycle  $c_3 = (w, u', z', v', v, v'', z'', u'', w)$ ; analogously to  $c_2$ ,  $c_3$  cannot have the chords  $(w, z')$ ,  $(w, z'')$ ,  $(u', u'')$ ,  $(u', z'')$ ,  $(u'', z')$ ,  $(z', z'')$ ,  $(u', v'')$ , and  $(u'', v')$ ; furthermore it cannot have the chords  $(z', v'')$ ,  $(z'', v')$  and  $(v', v'')$ , because of (ii) and (iii), and the chords  $(z', v)$  and  $(z'', v)$ , for the same reason, if  $v \in S_1$ , and due to the absence of  $(v'', z')$  and

$(v', z'')$  and (8) in Theorem 1, if  $v \in S_2$ . Therefore the only admissible chords in  $c_3$  are  $(u', v')$ ,  $(u', v)$ ,  $(w, v')$ ,  $(w, v)$ ,  $(w, v'')$ ,  $(u'', v)$ , and  $(u'', v'')$ . If  $(u', v)$  or  $(u'', v)$  were not in  $c_3$  then one of the following cycles should have no pair of crossing chords:

- $c_3$ ;
- $(w, u', z', v', v, u'', w)$ ;
- $(w, u'', z'', v'', v, u', w)$ .

Therefore,  $v$  is adjacent to both  $u'$  and  $u''$  and (i) is proved.

Now, if  $v \in S'_1 \cup S'_2$ , such a vertex is the connecting vertex  $u$  and (c) is proved; otherwise, two cases arise.

Case (1)  $v \in S_1 - S'_1$ . Since  $S_1$  is a domination of  $L^2$  in  $G_1$ , there exists a vertex  $z \in L^2$  adjacent to  $v$  and not adjacent to any vertex in  $S_1$  distinct from  $v$ ; such a vertex cannot be adjacent to  $u'$  or to  $u''$ . In fact, suppose that  $z$  is adjacent to  $u'$  and not adjacent to  $u''$ ; then the cycle  $(w, u', z, v, u'', w)$  should have no pair of crossing chords (contradiction). Analogously, a contradiction arises if  $z$  is adjacent to  $u''$  and not adjacent to  $u'$ . Therefore, suppose that  $z$  is adjacent to both  $u'$  and  $u''$ ; in this case, due to the choice of  $z$ , we have that both  $u'$  and  $u''$  are in  $S'_1 - S_1$ . Let  $z'$ ,  $z''$ ,  $v'$ , and  $v''$  be as in Case 3 above. Since  $z'$  and  $z''$  cannot be adjacent to  $v$ , then they cannot be adjacent to  $z$  (due to (8) in Theorem 1); this implies that the cycles:

- $(u', z', v', v, z, u')$  and
- $(u'', z'', v'', v, z, u'')$ ,

cannot have any pair of crossing chords (contradiction).

Since  $S'_1$  is a domination of  $L^2$  in  $G_1$ , there exists a vertex  $u \in S'_1$  adjacent to  $z$ ; by the considerations above, such a vertex must be distinct from both  $u'$  and  $u''$ . Furthermore, the cycle  $(w, u, z, v, u', w)$  must have the chord  $(u, u')$ ; analogously,  $u$  is adjacent to  $u''$  and (c) is proved.

Case (2).  $v \in S_2 - S'_2$ . Since  $S_2$  is a domination of  $L^3$  in  $G_2$ , there exists a vertex  $z \in L^3$  adjacent to  $v$  and not adjacent to any vertex in  $S_2$  distinct from  $v$ . Since  $S'_2$  is a domination of  $L^3$  in  $G_2$ , there exists a vertex  $u \in S'_2 - S_2$  adjacent to  $z$ . Because of (8) in Theorem 1,  $u$  must be adjacent to both  $u'$  and  $u''$ , and (c) is proved.

In order to prove (d), by (a) and (b), it is sufficient to prove that  $(N(S_1 \cup S_2) \cap L^1) - S'_1 \supseteq (N(S'_1 \cup S'_2) \cap L^1) - S_1$  (due to (c), the same argument can be used to prove that  $(N(S_1 \cup S_2) \cap L^1) - S'_1 \subseteq (N(S'_1 \cup S'_2) \cap L^1) - S_1$ ). Let us suppose, by contradiction, that there exists a vertex  $x \in L^1$  adjacent to a vertex  $u \in S'_1 \cup S'_2$  and not adjacent to any vertex in  $S_1 \cup S_2$ ; due to (a),  $x$  cannot be adjacent to any vertex in  $S'_2$ , then  $u$  is in  $S'_1 - S_1$ . First we will prove that:

- (i) every vertex  $z$  in  $L^2$  adjacent to  $u$  is adjacent to  $x$ .

In fact, let  $v \in S_1$  be a vertex adjacent to  $z$  (since  $u \notin S_1$ ,  $v$  is distinct from  $u$ ); since  $x$  is not adjacent to any vertex in  $S_1$ , if  $z$  were not adjacent to  $x$  then the cycle  $(w, x, u, z, v, w)$  should have no pair of crossing chords; hence (i) is proved.

We now prove that:

- (ii)  $S_1$  and  $S'_1$  are disjoint.

Let  $x$  and  $u$  be as above; let  $z$  be a vertex in  $L^2$  such that  $N(z) \cap S'_1 = \{u\}$  and  $v$  be a vertex in  $S_1 - S'_1$  adjacent to  $z$  and let us suppose that  $u'$  is a vertex in  $S'_1$  distinct from  $u$ . If, by contradiction,  $u'$  were in  $S_1$ , due to the connectivity of  $\langle S_1 \cup S_2 \rangle$  and to (3) in Theorem 1, either  $v$  is adjacent to  $u'$  or there exists a vertex  $y \in S_1 \cup S_2$  adjacent to both  $v$  and  $u'$ . In the first case, the cycle  $(w, x, z, v, u', w)$  has no pair of crossing chords. In the second case, let us consider the cycle  $c = (w, x, z, v, y, u', w)$ ; the only admissible chords in  $c$  are  $(w, v)$ ,  $(w, y)$  and  $(z, y)$ ; hence, if  $z$  is not adjacent to  $y$ ,  $c$  has no pair of crossing chords; otherwise  $(w, x, z, y, u', w)$  has no pair of crossing chords (contradiction); hence (ii) is proved.

Since  $S'_1$  is a domination of  $L^2$  in  $G_1$  and (i) holds, there exists a vertex  $z \in L^2$  adjacent to  $u$  and  $x$  and not adjacent to  $u'$ . Furthermore, there exists a vertex  $z'$  adjacent to  $u'$  and not adjacent to  $x$ ; otherwise (by (i) and by the fact that every vertex in  $L^2$  is adjacent to a vertex in  $S'_1$ ) every vertex in  $L^2$  should be adjacent to a vertex in  $(S'_1 - \{u, u'\}) \cup \{x\}$  and, hence,  $S'_1$  should not be a domination of  $L^2$ . Let  $v, v' \in S_1$  be two vertices adjacent to  $z$  and  $z'$ , respectively (such vertices exist because of (ii)). Since  $\langle S_1 \cup S_2 \rangle$  is connected, there exists an induced path in  $\langle S_1 \cup S_2 \rangle$  connecting  $v$  and  $v'$  whose length, due to (3) in Theorem 1, is less than or equal to two. Three cases arise.

Case (1).  $v$  and  $v'$  coincide. In this case, the cycle  $c_1 = (w, x, z, v, z', u', w)$  cannot have the chords:

- $(w, z)$  and  $(w, z')$ , in fact  $h_w(z) = h_w(z') = 2$ ;
- $(u', z)$  and  $(x, z')$ , because of the choice of  $z$  and  $z'$ ;
- $(z, z')$ , because its existence should imply the existence of both  $(u', z)$  and  $(x, z')$ ;
- $(x, v)$ , since  $x$  is not adjacent to any vertex in  $S_1$ .

Hence, if  $c_1$  does not have the chord  $(x, u')$  then it has no pair of crossing chords; otherwise, the cycle  $(x, z, v, z', u', x)$  has no pair of crossing chords.

Case (2).  $v$  and  $v'$  are adjacent. In this case, the cycle  $c_2 = (w, x, z, v, v', w)$  has no pair of crossing chords, since it cannot have the chords:

- $(w, z)$  and  $(x, v)$ , as above;
- $(x, v')$ , because  $x$  is not adjacent to any vertex in  $S_1$ ;
- $(z, v')$ , because of Case (1).

Case (3). There exists a vertex  $y \in S_1 \cup S_2$  adjacent to both  $v$  and  $v'$ . If  $y \in S_1$  then the cycle  $c_3 = (w, x, z, v, y, w)$  has no pair of crossing chords, since it cannot have the chords:

- $(w, z)$  and  $(x, v)$ , as above;
- $(x, y)$ , because  $x$  is not adjacent to any vertex in  $S_1$ ;
- $(z, y)$ , because of Case (2).

If  $y \in S_2$  then the cycle  $c_4 = (w, x, z, v, y, v', w)$  has no pair of crossing chords, since it cannot have the chords:

- $(w, z)$ ,  $(w, y)$ ,  $(x, v)$ ,  $(x, v')$ , and  $(z, v')$ , as above;
- $(v, v')$ , because of Case (2);
- $(z, y)$ , since its existence would imply the existence of  $(z, v')$ ;
- $(x, y)$ , since  $x$  cannot be adjacent to any vertex in  $S_2$ .

Finally, since in all cases a contradiction arises, (d) is proved.  $\square$

The basic idea, underlying the following algorithm to solve in polynomial time the connected domination problem in distance-hereditary graphs, consists in the reduction of the global optimization problem into this local optimization problem:

Given a hanging of the distance-hereditary graph, find a minimum set of vertices with level  $i$  which is a domination of all vertices with level  $i + 1$  (see (b) in Lemma 1).

In fact, the reduction is based on the following facts:

- The local problem is solvable in polynomial time (due to Proposition (a) in the proof of Theorem 4);
- To obtain a connected dominating set, the vertices with maximum level  $k$  in the hanging can be ignored (by Lemma 2);
- Let  $S_1$  and  $S_2$  be solutions of the local optimization problem at the first and second level, respectively. By Lemma 3, if  $S_1$  contains more than one vertex then both the connectivity of the graph induced by the union of the solutions of the same problem



from the first level to the level  $k-1$  and the adjacency to such union of all vertices at the first level, are independent from the choice of  $S_1$  and  $S_2$ .

ALGORITHM 4.

**input** a connected graph  $G = (V, E)$ ;

**output** a subset  $D$  of  $V$ ;

**function**  $DOMINA(i)$ ; [determine a minimum subset  $DOMINA$  of  $L^{i-1}$   
such that  $N(DOMINA) \supseteq L^i$ ]

**begin**

$DOMINA := L^{i-1}$ ;

**for every**  $v \in DOMINA$  **do**

**if**  $N(DOMINA - \{v\}) \supseteq L^i$  **then**  $DOMINA := DOMINA - \{v\}$

**end**;

**begin**

determine any hanging  $h_w$  of  $G$ ;

$D := \emptyset$ ;  $k := \max(h_w(v) | v \in V)$ ;

**for**  $i := k$  **downto** 3 **do**  $D := D \cup DOMINA(i)$ ;

**if** there exists a vertex  $z \in L^1$  such that  $N[z] \supseteq L^1 \cup L^2$

**then**  $D := D \cup \{z\}$

**else begin**

$D := D \cup DOMINA(2)$ ;

**if**  $N[D] \not\supseteq L^1$  **or**  $\langle D \rangle$  is not connected

**then**  $D := D \cup \{w\}$

**end**

**end.**

**THEOREM 4.** *If  $G$  is a distance-hereditary graph then Algorithm 4 determines a connected dominating set of  $G$  in polynomial time.*

*Proof.* It is easy to see that Algorithm 4 works in polynomial time, that the set  $D$  of vertices obtained by it induces a connected subgraph of  $G$  and that  $V = N[D]$ . Therefore, we have to prove that if  $G$  is a distance-hereditary graph, then  $D$  is a minimum set of vertices enjoying the properties above.

In the following let  $h_w$  be a hanging of  $G$  and let  $k$  be  $\max(h_w(v) | v \in V)$ . If  $k \leq 1$ , then either  $D = \{z\}$  or  $D = \{w\}$ ; hence  $D$  is a connected dominating set of  $G$ . Therefore, let  $k \geq 2$  and let  $G_i$ ,  $1 \leq i < k$ , be the subgraph of  $V(G, h_w)$  induced by  $L^i \cup L^{i+1}$ . First we will prove that:

(a) If  $S$  is a subset of  $L^i$ ,  $1 \leq i < k$ , such that  $N(S) \supseteq L^{i+1}$  and no subset of it has the same property then  $S$  is a domination of  $L^{i+1}$  in  $G_i$ .

Let us assume, without loss of generality, that  $G_i$  is connected (otherwise the following argument can be applied to every connected component of  $G_i$ ). First we will prove that the subgraph of  $G_i$  induced by  $S \cup L^{i+1}$  is also connected. In fact, if  $u$  and  $v$  are two distinct vertices in  $S$ , by (8) in Theorem 1, there exists a vertex in  $L^{i-1}$  adjacent to both  $u$  and  $v$ ; therefore, due to (3) in Theorem 1, every induced path between  $u$  and  $v$  in  $G_i$  has length two. This implies that there exists a vertex in  $L^{i-1}$  adjacent to both  $u$  and  $v$ ; hence, the subgraph of  $G_i$  induced by  $S \cup L^{i+1}$  is a nonredundant connection of  $L^{i+1}$  in  $G_i$ . Since, by Theorem 2,  $V(G, h_w)$  is a distance-hereditary graph and an induced subgraph of a distance-hereditary graph is distance-hereditary,  $G_i$  is also a distance-hereditary graph. Due to (4) in Theorem 1, (a) is proved.

Let us suppose, by contradiction, that there exists  $D' \subseteq V$  such that  $\langle D' \rangle$  is connected,  $V = N[D']$  and  $|D'| < |D|$ . Due to Lemma 2, we may assume that no vertex in  $L^k$  is in  $D'$ . If there exists  $i$ ,  $0 < i < k$ , such that  $|D' \cap L^i| < |D \cap L^i|$  then, due to (a),

there exists a vertex in  $L^{i+1}$  that is not adjacent to any vertex in  $D' \cap L^i$  and, due to (b) in Lemma 1, a contradiction arises.

Therefore, we have that for every  $i$ ,  $0 < i < k$ ,  $|D \cap L^i| = |D' \cap L^i|$ ; then  $w \in D - D'$ . We will prove that this implies that  $|DOMINA(2)| > 1$ .

Let us suppose, by contradiction, that  $|DOMINA(2)| = 1$ . Hence  $\langle D - \{w\} \rangle$  is connected; therefore, since  $w \in D$ , by Algorithm 4 we have that  $D - \{w\}$  is not a domination of  $L^1$ . If  $k > 2$ , let  $DOMINA(2) = \{v\}$ ,  $D' \cap L^1 = \{u\}$ ,  $z$  be a vertex in  $L^1$  that is not in  $N[D - \{w\}]$  and  $y$  be a vertex in  $D' \cap L^2$ ; due to (a) in Lemma 3,  $z$  cannot be adjacent to  $y$  and, hence, the cycle  $(w, z, u, y, v, w)$  has no pair of crossing chords (contradiction). If  $k = 2$ , since  $D'$  does not contain any vertex in  $L^k$ ,  $N[u]$  must contain  $L^1 \cup L^2$ ; but in this case, by Algorithm 4,  $w$  would not be in  $D$  (contradiction).

Therefore, if  $w \in D$  then  $|DOMINA(2)| > 1$ . Furthermore, by Algorithm 4, two cases arise:

- (i)  $\langle D - \{w\} \rangle$  is not connected; or
- (ii) There exists a vertex in  $L^1$  that is not in  $N[D - \{w\}]$ .

Let us suppose that (i) holds. In this case  $\langle DOMINA(2) \rangle$  and  $\langle DOMINA(2) \cup DOMINA(3) \rangle$  are not connected. In fact, due to (b) in Lemma 1, each vertex in  $DOMINA(i+1)$ ,  $1 < i < k$ , is adjacent to at least one vertex in  $DOMINA(i)$ , then there exists a path between any vertex in  $DOMINA(i+1)$ ,  $1 < i < k$ , and a vertex in  $DOMINA(2)$ . Therefore, if it were a path in  $\langle DOMINA(2) \rangle$  or in  $\langle DOMINA(2) \cup DOMINA(3) \rangle$  between any pair of vertices in  $DOMINA(2)$ ,  $\langle D - \{w\} \rangle$  would be connected. On the other hand, since  $w \notin D'$ , due to (c) in Lemma 1,  $\langle D' \cap (L^1 \cup L^2) \rangle$  is connected; hence due to (c) in Lemma 3, a contradiction arises.

Let us now suppose that (ii) holds. Since  $L^1 \subseteq N[D' \cap (L^1 \cup L^2)]$ , due to (b) and (d) in Lemma 3, a contradiction arises.  $\square$

**Acknowledgments.** The authors thank Antonio Sassano for his helpful comments and suggestions and the referees for their careful reading and detailed comments.

#### REFERENCES

- [1] G. AUSIELLO, A. D'ATRI, AND M. MOSCARINI, *Chordality properties on graphs and minimal conceptual connections in semantic data models*, J. Comput. System Sci., 33 (1986), pp. 179-202.
- [2] H-J. BANDELT AND H. M. MULDER, *Distance-hereditary graphs*, J. Combin. Theory, 41 (1986), pp. 182-208.
- [3] C. BERGE, *Graphs*, North-Holland, Amsterdam, 1985.
- [4] M. BURLET AND J. FONLUPT, *Polynomial algorithm to recognize a Meyniel graph*, Ann. Discrete Math., 21 (1984), pp. 225-252.
- [5] M. BURLET AND J. P. UHRY, *Parity Graphs*, Ann. Discrete Math., 21 (1984), pp. 253-277.
- [6] C. J. COLBOURN AND L. K. STEWART, *Permutation graphs: connected domination and Steiner trees*, Research Report CS-85-02, 1985.
- [7] D. G. CORNEIL, H. LERCHS, AND L. STEWART BURLINGHAM, *Complement reducible graphs*, Discrete Appl. Math., 3 (1981), pp. 163-174.
- [8] D. G. CORNEIL AND Y. PERL, *Clustering and domination in perfect graphs*, Discrete Appl. Math., 7 (1984), pp. 27-40.
- [9] A. D'ATRI AND M. MOSCARINI, *Recognition algorithms and design methodology for acyclic databases schemes*, in *Advances in Computing Research*, Vol. 3, P. Kanellakis, ed., JAI Press, Greenwich, CT, 1986, pp. 43-67.
- [10] ———, *Cross graphs, Steiner trees and connected domination*, IASI-CNR Research Report R.147, Rome, 1986, presented to Théorie des Graphes et Combinatoire (3ème Colloque International), Marseille, 1986.
- [11] ———, *On hypergraph acyclicity and graph chordality*, IASI-CNR Research Report R.170, Rome, 1986.
- [12] R. DUKE, *Types of cycles in hypergraphs*, Ann. Discrete Math., 27 (1985), pp. 399-418.

- [13] R. FAGIN, *Degree of acyclicity for hypergraphs and relational database schemes*, J. Assoc. Comput. Mach., 30 (1983), pp. 514–550.
- [14] M. FARBER, *Characterizations of strongly chordal graphs*, Discrete Math., 43 (1983), pp. 173–189.
- [15] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability*, W. H. Freeman, San Francisco, 1979.
- [16] M. C. GOLUMBIC, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
- [17] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.
- [18] E. HOWORKA, *A characterization of distance-hereditary graphs*, Quart. J. Math. Oxford, 28 (1977), pp. 417–420.
- [19] ———, *A characterization of ptolemaic graphs*, J. Graph Theory, 5 (1981), pp. 323–331.
- [20] D. S. JOHNSON, *The NP-completeness column: An ongoing guide*, J. Algorithms, 6 (1985), pp. 434–451.
- [21] H. MEYNIEL, *The graphs whose odd cycles have at least two chords*, Ann. Discrete Math., 21 (1984), pp. 115–119.
- [22] K. WHITE, M. FARBER, AND W. PULLEYBLANK, *Steiner trees, connected domination and strongly chordal graphs*, Network, 15 (1985), pp. 109–124.

## A POLYNOMIAL APPROXIMATION SCHEME FOR SCHEDULING ON UNIFORM PROCESSORS: USING THE DUAL APPROXIMATION APPROACH\*

DORIT S. HOCHBAUM† AND DAVID B. SHMOYS‡

**Abstract.** We present a polynomial approximation scheme for the minimum makespan problem on uniform parallel processors. More specifically, the problem is to find a schedule for a set of independent jobs on a collection of machines of different speeds so that the last job to finish is completed as quickly as possible. We give a family of polynomial-time algorithms  $\{A_\varepsilon\}$  such that  $A_\varepsilon$  delivers a solution that is within a relative error  $\varepsilon$  of the optimum. This is a dramatic improvement over previously known algorithms; the best performance guarantee previously proved for a polynomial-time algorithm ensured a relative error no more than 40 percent. The technique employed is the dual approximation approach, where infeasible but superoptimal solutions for a related (dual) problem are converted to the desired feasible but possibly suboptimal solution.

**Key words.** scheduling, approximation algorithms

**1. Introduction.** We will consider a fundamental problem of scheduling theory. Suppose that we have a set of jobs  $J$  with independent processing times  $p_1, \dots, p_n$  that are to be executed on  $m$  nonidentical machines; these machines run at different speeds  $s_1, \dots, s_m$ . More precisely, if job  $j$  is executed on machine  $i$  it takes  $p_j/s_i$  time units to be completed. The objective is to assign the jobs to machines so as to minimize the total execution time required to run the jobs assigned to the most heavily loaded machine. In other words, it is the minimum time needed to complete the processing of all of the jobs. In the classification scheme of [GLLR] this problem is denoted  $Q//C_{\max}$ , the *minimum makespan problem on uniform parallel machines*. In this paper, we will present a family of algorithms for this problem where these algorithms are, in a sense to be indicated below, the best possible algorithms for this problem.

As is true for most scheduling problems, this problem is likely to be intractable since it is  $NP$ -complete, and therefore the existence of a polynomial-time algorithm for it would imply that  $P = NP$ . As a result, the algorithm designer must be willing to settle for a solution somewhat worse than the optimal schedule. One natural approach is to consider *approximation algorithms* for the problem which deliver a schedule with makespan that is guaranteed to be within a specified relative error of the optimum schedule length. This approach was first considered by Graham [G], who showed that if all of the machines have the same speed, then the simple "on-line" procedure of scheduling *any* job when a machine becomes idle always delivers a schedule that finishes within at most  $1 + (1 - 1/m)$  times the optimal schedule length. We shall call such a polynomial-time procedure a  $(1 - 1/m)$ -approximation algorithm. Work on this special case of identical machines culminated in the recent work of Hochbaum and Shmoys [HS], that showed that for *any* fixed  $\varepsilon > 0$  there exists an  $\varepsilon$ -approximation algorithm. This was a dramatic difference from previously known work on the more general problem with different processing speeds. Although much work had been done

---

\* Received by the editors October 20, 1986; accepted for publication May 13, 1987.

† School of Business Administration, University of California, Berkeley, California 94720. The work of this author was supported in part by the National Science Foundation under grant ECS-85-01988.

‡ Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The work of this author was supported in part by the National Science Foundation under grants ECS-85-01988 and MCS-81-20790, by a Presidential Young Investigator Award, and by Air Force contract AFOSR-86-0078.

on this harder problem (e.g., [CS], [GIS], [HoS]) the best algorithm published to date delivers a solution that could be up to 40 percent more than the optimum [FL].

In this paper, we present a family of polynomial-time algorithms  $\{A_\varepsilon\}$ , such that the algorithm  $A_\varepsilon$  is an  $\varepsilon$ -approximation algorithm for the more general problem. Such a family of algorithms is traditionally called a *polynomial approximation scheme*. Notice that the algorithm  $A_\varepsilon$  is polynomial in the size of the input, but not in the value of  $1/\varepsilon$ . If the family of algorithms has the property that  $A_\varepsilon$  is polynomial in  $1/\varepsilon$  and the size of the instance, then the family is known as a *fully polynomial approximation scheme*. Since this problem is strongly *NP*-complete, our results are, however, the best possible in the sense that if there were a fully polynomial approximation scheme for this problem, then  $P = NP$  [GJ].

Due to the exponential dependence on  $1/\varepsilon$  in the running time of our algorithm, it is not particularly practical for small values of  $\varepsilon$ . However, the result shows that there do exist polynomial-time algorithms that produce solutions with far superior guarantees to the previously known algorithms, and thus one might hope for practical algorithms with better guarantees than are known today. Note that the discussion above implies that there are limits to the amount of improvement that is possible; still one might hope for an  $O(n \log n)$  or  $O(n^2)$  algorithm that is guaranteed to have relative error no more than, for example, 5 percent. In addition to this existential sort of practical implication, we also believe that the framework around which our algorithm is built can lead to efficient algorithms with extremely good guarantees. As an example of this, we give an extremely efficient but exceedingly naive algorithm which is an adaptation of our framework, and is guaranteed to deliver a solution that is within 50 percent of the optimum. In addition, the analysis of this algorithm is similarly transparent.

**2. A framework for approximation algorithms for scheduling problems.** In this section we will describe the basic structure of our polynomial approximation scheme for the minimum makespan problem with uniform processors. Consider for the moment the related question of deciding whether there exists a schedule for a given instance of this problem where all of the jobs are completed by time  $T$ . If we think of the units of the processing times  $p_j$  as steps, and the units of the speeds as steps per unit of time, then machine  $i$  can process  $Ts_i$  steps before the deadline  $T$ . The decision problem can then be viewed as a *bin-packing problem with variable bin sizes*. Furthermore, notice that the notation for this problem can be simplified by rescaling both the processing requirements and the speeds by a factor of  $1/T$ : in this bin-packing variant the aim is to decide whether a set of jobs (which we will interchangeably call pieces) of sizes  $p_1, \dots, p_n$  can be packed into a set of bins of sizes  $s_1, \dots, s_m$ .

Suppose that we had an efficient procedure for solving the bin-packing problem with variable bin sizes. Then it is possible to solve the minimum makespan problem with uniform machines by a simple binary search procedure. For the midpoint  $T$  of the current range of possible optimum makespan values, we run the decision procedure; if a schedule is found then the upper bound is updated to the midpoint, otherwise the lower bound is updated. To initialize the binary search we need to obtain easy upper and lower bounds on the length of the optimum schedule. One such upper bound is  $U = \sum_j p_j / \max_i s_i$  (schedule all jobs on the fastest machine) and one such lower bound is  $L = \sum_j p_j / (m \cdot \max_i s_i)$  (even if all machines are as fast as the fastest, there must be enough total processing capacity for the machines to process the total processing requirement). Notice that these bounds have a ratio bounded by  $m$ . Thus after  $\log m + l$  iterations of a binary search procedure the difference between the upper bound and lower bound on the optimum makespan value is at most  $2^{-l}$  times the optimum value.

Unfortunately, this bin-packing problem is also *NP*-complete, so it seems unlikely that we will find an efficient procedure to solve it. Instead we will argue that solving a *relaxed* version of the bin-packing problem will be sufficient for our purposes. We first introduce some useful terminology. For each collection of jobs  $J$  with processing times  $\{p_1, \dots, p_n\}$ , and set of bin sizes  $S = \{s_1, \dots, s_m\}$ , a *truly feasible packing* is a partition of the job set into  $m$  parts,  $B_i$ ,  $i = 1, \dots, m$  where the total processing requirement of jobs in  $B_i$  is at most  $s_i$  for  $i = 1, \dots, m$ . Similarly, we define an  $\varepsilon$ -*relaxed feasible packing* to be a similar partition, but one that need only satisfy the weaker (or relaxed) condition that the total processing requirement of jobs in  $B_i$  be at most  $(1 + \varepsilon)s_i$ .

An  $\varepsilon$ -*relaxed decision procedure* is a procedure which, given a collection  $J$  of jobs with processing times  $\{p_1, \dots, p_n\}$ , and a set of bin sizes  $S = \{s_1, \dots, s_m\}$ , outputs one of two outcomes:

- (1) An  $\varepsilon$ -relaxed feasible packing; or
- (2) Some certificate that no truly feasible packing exists.

Consider now the binary search procedure described above with an  $\varepsilon$ -relaxed decision procedure in place of the algorithm assumed to solve the bin-packing problem with variable sizes. Notice that when an update of the lower bound is done, the new value must still be a valid lower bound on the optimum makespan length, since the  $\varepsilon$ -relaxed decision procedure fails to produce a packing only when no truly feasible packing exists. Similarly, if the upper bound is updated to  $t$ , then a schedule has been obtained of length at most  $(1 + \varepsilon)t$ . From these observations it is not hard to obtain the following result.

**THEOREM 1.** *An  $\varepsilon$ -approximation algorithm for the minimum makespan problem with uniform parallel machines can be obtained by executing the binary search procedure using an  $\varepsilon/2$ -relaxed decision initialized with upper and lower bounds,  $U$  and  $L$ , respectively, for  $\log m + \log(3/\varepsilon)$  iterations.*

For the complete details of the proof of this, plus a more general setting in which the same basic ideas are applicable, the reader is referred to [HS]. It is also useful to note that by continuing for more iterations (but only polynomially many) it is possible to convert an  $\varepsilon$ -relaxed decision procedure into an  $\varepsilon$ -approximation algorithm (as opposed to the result given above which uses an  $\varepsilon/2$ -relaxed decision procedure). From Theorem 1, we get the following immediate corollary.

**COROLLARY 1.** *If for all fixed  $\varepsilon > 0$  there exists a polynomial-time  $\varepsilon$ -relaxed decision procedure for the bin-packing problem with variable bin sizes, then there is a polynomial approximation scheme for the minimum makespan problem with uniform parallel machines.*

**3. An  $\varepsilon$ -relaxed decision procedure for bin packing with variable bin sizes.** In this section we will show how to construct an  $\varepsilon$ -relaxed decision procedure for the bin-packing problem with variable bin sizes for any  $\varepsilon > 0$ . For the remainder of the paper we will assume that the pieces (or jobs) have sizes  $p_1, \dots, p_n$ , and the bins have sizes  $s_1, \dots, s_m$ , where  $s_1 \geq s_2 \geq \dots \geq s_m$ .

For convenience we shall assume that  $1/\varepsilon$  is a positive integer. The description of the algorithm and the proof of its correctness will proceed in a few phases. We first construct a certain layered directed graph with two nodes designated "initial" and "success." We prove that if there is a *truly feasible packing*, then there is a directed path from "initial" to "success." Furthermore, the existence of such a path provides a means of efficiently constructing an  $\varepsilon$ -relaxed feasible packing. Hence, the procedure consisting of constructing the graph, identifying if there is a path from "initial" to "success," and then deriving the respective packing is indeed an  $\varepsilon$ -relaxed decision procedure.

An intuitive outline of the algorithm relies on the analogy to the special case of bin packing  $m$  bins of equal size ( $=1$ ). Such an algorithm is given in [HS], but its presentation, however, does not lend itself to the required generalization. Here we modify the description of the algorithm to clarify the analogous procedure in the variable-size bins case. For the equal-size case, all pieces lie in the interval  $(0, 1]$ , and the attempt is to  $\varepsilon$ -relaxed pack them in at most  $m$  bins. The first pieces to be packed are of size greater than  $\varepsilon$ ; these *large* pieces will be denoted by  $J_{\text{large}} = \{j \mid p_j > \varepsilon\}$ . The phase of the algorithm where these pieces are packed is called *large-pack*. Since these pieces are large, fewer than  $1/\varepsilon$  of them can fit in one bin. These large pieces are further partitioned according to their size in subintervals of length  $\varepsilon^2$  each. All piece sizes in such subintervals are all rounded down to the lower end of the subinterval, which is the nearest multiple of  $\varepsilon^2$  no more than the original piece size. After this rounding, the number of large piece sizes is at most  $w = (1 - \varepsilon)/\varepsilon^2$ . Thus, the packing of large pieces in a bin can be uniquely described by an array of the distribution of piece sizes that go into that bin. It is an array with one entry for each subinterval, and an integer value between 0 and  $1/\varepsilon$  in each entry. Such an array, or a *configuration*, specifies how many pieces of each subinterval go into each bin. A configuration  $(x_1, \dots, x_w)$  is called *feasible* if each  $x_i \geq 0$  and the total sum of the rounded sizes of pieces in the configuration is at most 1, the size of the bin.

The distribution of the remaining large pieces to be packed, the *state vector*, is described by a similar array, except that each entry may contain a nonnegative integer no more than  $n$ . Therefore, the total number of possible state vectors is at most  $n^w$ . A state vector  $(n_1, n_2, \dots, n_w)$  is *reachable* from a state vector  $(n'_1, n'_2, \dots, n'_w)$  if there is a feasible configuration  $(x_1, \dots, x_w)$  such that  $n_i = n'_i - x_i$  for  $i = 1, \dots, w$ . The first step of the procedure is to construct a layered directed graph where the nodes correspond to state vectors in the following way. Let  $V_0, \dots, V_m$  be the nodes in the 0th through  $m$ th layers, respectively. For  $i = 1, \dots, m-1$ ,  $V_i$  contains a vertex  $(i, \mathbf{n})$  for each possible state vector  $\mathbf{n}$ .  $V_0$  contains only one vertex, the “initial” node, and is labeled with the state vector corresponding to the initial distribution of rounded piece sizes. Similarly  $V_m$  contains only the “success” node, which is labeled with the zero state vector (corresponding to the case that all pieces are packed). From each node  $(i, \mathbf{n})$ , there is an arc directed towards the node  $(i+1, \mathbf{n}')$  if and only if the state vector  $\mathbf{n}'$  is reachable from the state vector  $\mathbf{n}$ . Given any truly feasible packing of the original instance, it is easy to see that the induced packing on the (rounded) large pieces implies that there is a path from the initial node to the success node. We next show that from any path from initial to success we can compute an  $\varepsilon$ -relaxed feasible packing of the large unrounded pieces. The path clearly specifies a packing of the rounded large pieces. If we now restore the large pieces to their original sizes (arbitrarily selecting them from the appropriate subintervals), this “inflating” process may either result in a packing that is truly feasible, or the pieces may exceed the capacity ( $=1$ ) of some bins. However, the rounding was done in such a way so that it is easy to bound the amount by which the inflated pieces will exceed the bin capacity. To round each piece it was necessary to subtract at most  $\varepsilon^2$  from its actual size, and there are no more than  $1/\varepsilon$  pieces per bin. Thus, the total difference between actual and rounded piece sizes in a bin is at most  $\varepsilon$ , and so the total actual piece size cannot exceed  $1 + \varepsilon$ . In summary, the procedure *large-pack* constructs the layered graph, finds a path from initial to success, which then yields an  $\varepsilon$ -relaxed feasible packing of the large pieces.

We now must show how to extend this  $\varepsilon$ -relaxed feasible packing to include the small pieces, in a way that will always succeed if there is a truly feasible packing of the original instance. The existence of a truly feasible packing induces a truly feasible

packing of the large pieces, with sufficient total slack in the bins in order to accommodate all the small pieces. Moreover, the *total slack* in a truly feasible packing of the large pieces,  $V^t = m - \sum_{j \in J_{\text{large}}} p_j$ , can be no more than the total slack remaining in bins packed *under capacity* in an  $\varepsilon$ -relaxed feasible packing of the large pieces, the *relaxed slack*,  $V^{\text{rel}} = \sum_{i=1}^m \max \{0, 1 - \sum_{j \in C_i} p_j\}$  where  $C_i$  is the set of pieces assigned by *large-pack* to bin  $i$ . Now let  $J_{\text{small}} = J - J_{\text{large}}$ . Since  $V^t \geq \sum_{j \in J_{\text{small}}} p_j$  and  $V^{\text{rel}} \geq V^t$ , it follows that  $V^{\text{rel}} \geq \sum_{j \in J_{\text{small}}} p_j$ . If this inequality is not satisfied then there can be no truly feasible packing; otherwise we will be able to *small-pack* the remaining (small) pieces. This is done by assigning one piece at a time to any bin with positive slack (that is, filled with less than its unit capacity), even if this slack is less than the size of the piece. This procedure guarantees that:

- (1) So long as there is positive slack, small pieces can be packed;
- (2) By packing a small piece of size  $p_j$ , the total remaining slack is reduced by at most  $p_j$ ; and
- (3) Bins that become packed over capacity in the small-pack phase are packed with at most  $1 + \varepsilon$  times their true capacity (since the capacity is exceeded only by adding a small piece (i.e.,  $< \varepsilon$ ), to a bin that was previously truly feasibly packed).

Therefore, if there is a truly feasible packing, the *large-pack* and *small-pack* procedures will find an  $\varepsilon$ -relaxed feasible packing. The complexity of the algorithm is dominated by the *large-pack* procedure, where we construct the layered graph. The graph has at most  $2 + (m + 1)n^{1/\varepsilon^2}$  nodes and each node has at most  $(1/\varepsilon)^{1/\varepsilon^2}$  arcs originating at it. The construction of each arc amounts to checking the feasibility of the corresponding packing of a bin. This is done with at most  $1/\varepsilon$  additions and one comparison. The complexity of the  $\varepsilon$ -relaxed procedure is hence  $O((m/\varepsilon)(n/\varepsilon)^{1/\varepsilon^2})$ , which is a polynomial for any fixed positive  $\varepsilon$ .

In the generalization of the equal-size bin  $\varepsilon$ -relaxed procedure to the variable-size case we come across a major obstacle. The size of the subintervals in which the pieces are partitioned depends on the size of the bins in which the pieces are to be packed. Moreover, the definition of large and small pieces depends on the size of the bin in which the pieces are to be packed. Let the largest bin size  $s_1$  be normalized to 1. The piece sizes are hence in the interval  $(0, 1]$ . We round down piece sizes as follows; for a piece in the interval  $(\varepsilon^{k+1}, \varepsilon^k]$  we round its size down to the nearest multiple of  $\varepsilon^{k+2}$ . Formally, define

$$\bar{p}_j = \lfloor p_j / \varepsilon^{k+2} \rfloor \cdot \varepsilon^{k+2}, \quad \text{where } k = \max \{q \geq 0 \mid p_j \leq \varepsilon^q\}.$$

Consider now a bin of size  $s_i$  in the interval  $(\varepsilon^{k+1}, \varepsilon^k]$ . In the previous case, the large pieces for this bin were those with sizes in the interval  $(\varepsilon s_i, s_i]$  since  $s_i = 1$  for all  $i$ . This interval can intersect both of the intervals  $(\varepsilon^{k+2}, \varepsilon^{k+1}]$  and  $(\varepsilon^{k+1}, \varepsilon^k]$  and as a result, we will slightly modify the notion of large in generalizing the algorithm to the case of variable-size bins. The pieces in the interval  $(\varepsilon^{k+1}, \varepsilon^k]$  are *large* for this bin whereas the pieces in the interval  $(\varepsilon^{k+2}, \varepsilon^{k+1}]$  are called *medium* pieces. The pieces of size less than or equal to  $\varepsilon^{k+2}$  are *small* for this bin. Notice that the definition of large, medium, and small pieces did not depend on the precise size of the bin, but only on the interval of the form  $(\varepsilon^{l+1}, \varepsilon^l]$  that contained the bin size; for convenience we will often refer to this as interval  $l$ . As before, it will always be true that a small piece for a bin is no more than  $\varepsilon$  times the bin size. The strategy will be to execute the packing of the bins  $1, 2, \dots, m$  in two phases:

- (1) Large- and medium-piece packing (*l&m-pack*);
- (2) Small-piece packing (*small-pack*).



Note that a piece can be large, medium, or small depending on the size of the bin in which it is packed. It will also be useful to classify bin sizes for a given interval of piece sizes. For pieces in interval  $k$ ,  $(\epsilon^{k+1}, \epsilon^k]$ , a bin is *large* if it is in interval  $k$ , *huge* if it is in interval  $k-1$ , and *enormous* if it is in interval  $k-2$ .

We construct a directed, layered graph where each node is labeled with a state vector describing the remaining pieces to be packed as large or medium pieces. Since each bin may generate different classes of large or medium pieces, a straightforward representation of such an array leads to roughly  $n^n$  such possible state vectors. Instead, the graph will be grouped into *stages*, where a stage will specify the *l&m-pack* of bins in one interval  $(\epsilon^{k+1}, \epsilon^k]$ . Each layer within a stage corresponds to packing a bin in the corresponding interval. Both the bins within the stage and the stages are arranged in order of decreasing bin size. At the end of the *l&m-pack* of the bins of the stage corresponding to interval  $k$ , we will provide for the packing of the remaining unpacked pieces in interval  $k$ ; these pieces will be packed as small pieces. (Note that these pieces must therefore be packed in bins that are enormous for them.)

The treatment of the bins and pieces in this conglomerate way will make it possible to reduce the amount of information encoded in the state vector. The state vector associated with each node is of the form  $(\mathbf{L}; \mathbf{M}; V_1, V_2, V)$ , where  $\mathbf{L}$  and  $\mathbf{M}$  are vectors each describing a distribution of pieces in the subintervals of  $(\epsilon^{k+1}, \epsilon^k]$  and  $(\epsilon^{k+2}, \epsilon^{k+1}]$ , respectively. The subinterval division is of length  $\epsilon^{k+2}$  and  $\epsilon^{k+3}$  respectively, so each of these vectors is of length  $w = (1 - \epsilon)/\epsilon^2$  and the value of each entry is at most  $n$ . For simplicity of notation, let  $l_1, \dots, l_w$  and  $\bar{l}_1, \dots, \bar{l}_w$  denote the values of the lower endpoints of the subintervals of  $(\epsilon^{k+1}, \epsilon^k]$  and  $(\epsilon^{k+2}, \epsilon^{k+1}]$ , respectively. (See Fig. 1.)

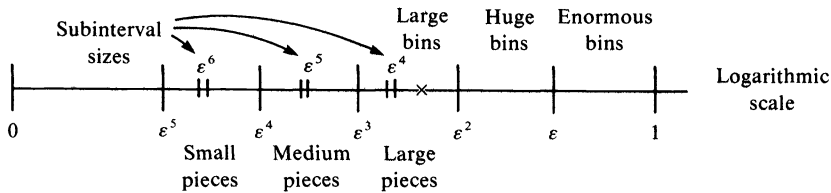


FIG. 1. The  $\times$  represents the size of a bin to be packed in interval 2. The identity of pieces and other bins is relative to this interval.

As was mentioned above, after the *l&m-pack* of the bins in interval  $k$ , we must allow for the packing of the pieces in interval  $k$  that will be packed as small pieces. These pieces must be packed in enormous bins (those from interval  $k-2$ ) and so we need to know that there is sufficient unused capacity in the enormous bins to at least contain the total size of these unpacked pieces. This is the function of the value  $V_1$  in the state vector; it records the slack, or unused capacity in the partial packing of the enormous bins with large and medium pieces. For stages corresponding to intervals greater than  $k$ , we will also need to know the unused capacity in the huge and large bins, and this is the function of  $V_2$  and  $V$ , respectively. However, there is a crucial point in that each node is labeled with a possible value for  $V_1$ ,  $V_2$ , and  $V$ , so we must be able to represent the possible values in some compact way. Consider the sizes of the pieces that will be packed as small pieces into this as-yet-unused capacity. For  $V_1$ , pieces in interval  $k$  are small, and thus all pieces to be packed into this unused capacity have rounded sizes that are multiples of  $\epsilon^{k+2}$ ; as a result, it will be sufficient to represent  $V_1$  as an integer multiple of  $\epsilon^{k+2}$ . Similarly,  $V_2$  and  $V$  will be represented as integer

multiples of  $\epsilon^{k+3}$  and  $\epsilon^{k+4}$ , respectively. Furthermore, we will argue below (in Lemma 1) that for the purposes of the algorithm, it will be sufficient to have one node for all multiples greater than  $n/\epsilon^2$  for any of these three parameters. As a result, the number of possible state vectors in each layer is bounded from above by  $n^{2/\epsilon^2} \cdot (n/\epsilon^2)^3$ .

Suppose that there are  $m_k > 0$  bins with size in  $(\epsilon^{k+1}, \epsilon^k]$ . The stage corresponding to this interval will actually have  $m_k + 1$  layers of nodes. In each of these layers, there is one node for every possible state vector  $(n_1, \dots, n_w; \bar{n}_1, \dots, \bar{n}_w; V_1, V_2, V)$ . A node in the  $(l+1)$ st layer of the stage labeled  $(n_1, \dots, n_w; \bar{n}_1, \dots, \bar{n}_w; V_1, V_2, V)$  is *reachable* from a node in the  $l$ th layer of the stage  $(n'_1, \dots, n'_w; \bar{n}'_1, \dots, \bar{n}'_w; V_1, V_2, V')$ , if:

- (1) There is a configuration  $(x_1, \dots, x_w; \bar{x}_1, \dots, \bar{x}_w)$  where  $x_i$  and  $\bar{x}_i$  are nonnegative integers for  $i = 1, \dots, w$  such that  $n_i = n'_i - x_i$  and  $\bar{n}_i = \bar{n}'_i - \bar{x}_i$ ;
- (2) The configuration is feasible; that is, the total sum of rounded piece sizes,  $\sum_{i=1}^w x_i l_i + \sum_{i=1}^w \bar{x}_i \bar{l}_i$  may not exceed  $s$ , the size of the  $l$ th bin in that stage; and
- (3)  $V = V' + \lceil \text{slack}_l / \epsilon^{k+4} \rceil$ , where  $\text{slack}_l = s - (\sum_{i=1}^w x_i l_i + \sum_{i=1}^w \bar{x}_i \bar{l}_i)$ .

We define here the concept of the usable slack in bin  $l$  as

$$\text{usable slack}_l = \lceil \text{slack}_l / \epsilon^{k+4} \rceil \cdot \epsilon^{k+4}.$$

Note that in the first layer of the first stage we keep only the node with the state vector corresponding to the distribution of pieces in  $(\epsilon, 1]$  and  $(\epsilon^2, \epsilon]$  with  $V_1 = V_2 = V = 0$ .

Intuitively, a node  $z$  is reachable from an initial state vector for the stage if there is an  $l$ & $m$ -pack of the first  $l$  bins in that group using feasible configurations, leaving a remaining piece distribution as in node  $z$ , and accumulating in all  $l$  bins a total of  $V \cdot \epsilon^{k+4}$  in usable slack. (See Fig. 2.)

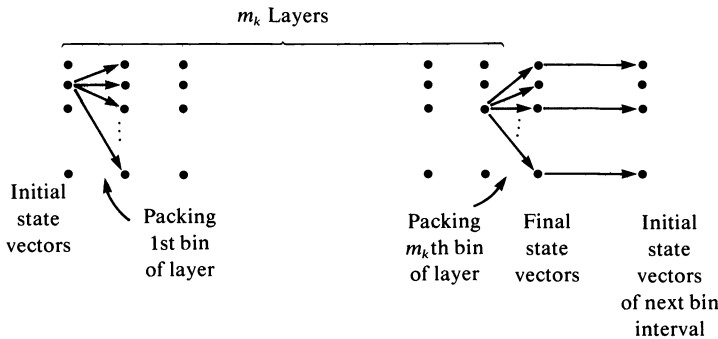


FIG. 2. One stage of the layered graph (for interval  $k$ ).

Note that the number of feasible configurations is at most  $(1/\epsilon^2)^{2/\epsilon^2}$ , since no more than  $1/\epsilon^2$  large or medium pieces can fit in a bin. This number is polynomial for any fixed positive value of  $\epsilon$  (or actually just a large constant).

The state vectors in the final layer of one stage have to be updated to the proper form of initial state vectors for the subsequent interval of higher index that contains a bin. Suppose that the next such interval is  $q$  intervals away; that is, it is the interval  $(\epsilon^{k+q+1}, \epsilon^{k+q}]$ . Let  $l_i^{(t)}$  denote the value of the lower end of the  $i$ th subinterval of  $(\epsilon^{k+t+1}, \epsilon^{k+t}]$  and let  $n_i^{(t)}$ ,  $t \geq 2$ , be the number of pieces contained in that subinterval. (For  $t = 0, 1$ ,  $n_i^{(t)}$  will be used to denote the large and medium pieces remaining after the  $l$ & $m$ -pack for interval  $k$ .) We will set  $q = \infty$  to indicate that there are no more bins.

```

function update ( $q, (n_1^{(0)}, \dots, n_w^{(0)}; n_1^{(1)}, \dots, n_w^{(1)}; V_1, V_2, V)$ )
begin
   $V_1^{(1)} := V_1 \cdot \varepsilon^{k+2} - \sum_{i=1}^w n_i^{(0)} l_i^{(0)}$ 
  if  $V_1^{(1)} < 0$  then return (“failure”)
  if  $q \geq 2$  then
    begin
       $V_1^{(2)} := V_1^{(1)} + V_2 \cdot \varepsilon^{k+3} - \sum_{i=1}^w n_i^{(1)} l_i^{(1)}$ 
      if  $V_1^{(2)} < 0$  then return (“failure”)
    end
  if  $q \geq 3$  then
    begin
       $V_1^{(q)} := V_1^{(2)} + V \cdot \varepsilon^{k+4} - \sum_{t=2}^{q-1} \sum_{i=1}^w n_i^{(t)} l_i^{(t)}$ 
      if  $V_1^{(q)} < 0$  then return (“failure”)
    end
  if  $q < \infty$  then  $V_1 := V_1^{(q)} \cdot \frac{1}{\varepsilon^{k+q+2}}$ 
  case ( $q$ ) of
    1:  $V_1 := V_1 + V_2; V_2 := V$ 
    2:  $V_1 := V_1 + V; V_2 := 0$ 
    [3,  $\dots, \infty$ ):  $V_1 := V_1; V_2 := 0$ 
     $\infty$ :  $V_1 := 0; V_2 := 0; V := 0$ 
  endcase
  return ( $(n_1^{(q)}, \dots, n_w^{(q)}; n_1^{(q+1)}, \dots, n_w^{(q+1)}; V_1, V_2, 0)$ )
end

```

The procedure *update* may result in “failure,” in which case there is no arc originating at the node examined. Otherwise, the output of the function gives the state vector of the node into which the arc will be directed. It is important to notice that in implementing the procedure we will not scan each subinterval on the line, only those with  $n_i^{(t)} > 0$ . This is readily done, since the pieces are sorted. Also note that in the updating layer there is at most one arc originating at each node.

Following the updating process of the last interval in which bins are found, we add one more arc level to the final node, “success.” A node will have an arc from it to “success” if and only if its state vector is the zero vector  $(0, 0, \dots, 0; 0, \dots, 0; V_1, V_2, V)$ . It is somewhat simpler to avoid altogether the update layer at the end of the last interval and have arcs going directly to “success” based on the outcome of “update” only if the update procedure results in the zero state vector.

LEMMA 1. *The graph constructed has at most  $O(2m \cdot n^{2/\varepsilon^2+3} \cdot 1/\varepsilon^6)$  nodes.*

*Proof.* Each of the first and last layers contains exactly one node. All other layers of nodes correspond either to the beginning or the end of a bin interval, or to packing a bin within an interval. For each interval that contains at least one bin, the number of layers is equal to one more than the number of bins in the interval. Since there are  $m$  bins the total number of layers is no more than  $2m$ .

Each layer may contain all possible state vectors. The first  $2w \leq 2/\varepsilon^2$  entries describe the distribution of large and medium pieces remaining in the interval. Every subinterval may contain  $r \in \{0, 1, \dots, n\}$  pieces, so the number of such distributions is  $O(n^{2/\varepsilon^2})$ .

Finally,  $V_1, V_2, V$  express the unused capacities that may be used to pack small pieces. Consider first  $V_1$ ; since each small piece for these bins is of size at most  $\epsilon^k$ , and there are no more than  $n$  small pieces, any volume beyond  $n\epsilon^k$  is irrelevant. The value  $V_1$  is actually given as the scalar multiple of  $\epsilon^{k+2}$ , so multiples beyond  $n \cdot \epsilon^k / \epsilon^{k+2} = n/\epsilon^2$  may be equivalently represented. Identical calculations show that for  $V_2$  and  $V$  as well, it is sufficient to consider only  $n/\epsilon^2$  possible values, where the largest value represents all remaining positive volumes. The total number of state vectors is hence  $O(2m \cdot n^{2/\epsilon^2} (n/\epsilon^2)^3)$ , as claimed.

LEMMA 2. *The number of arcs originating at each node is  $O((1/\epsilon^2)^{2/\epsilon^2})$ .*

*Proof.* Each arc corresponds to a feasible configuration  $(x_1, \dots, x_w; \bar{x}_1, \dots, \bar{x}_w)$ . The number of large pieces from a subinterval  $x_i$  that can go in a bin is in  $\{0, 1, \dots, 1/\epsilon\}$ , whereas the number of medium pieces  $\bar{x}_i$  can vary from 0 up to  $1/\epsilon^2$ . The total number of feasible configurations is therefore bounded by  $(1/\epsilon + 1)^{1/\epsilon^2} (1/\epsilon^2 + 1)^{1/\epsilon}$  which is  $O((1/\epsilon^2)^{2/\epsilon^2})$ .

COROLLARY 2. *The total number of arcs in the graph is  $O(2m(n/\epsilon^2)^{(2/\epsilon^2)+3})$ .*

Once the graph is available, the existence of a path from initial to success can be verified in time linear in the number of arcs. This computation also yields a specific path. We now prove that if there is a truly feasible packing then such a path indeed exists, and if such a path exists, then we can construct a  $(2\epsilon + \epsilon^2)$ -relaxed feasible packing. These two claims are sufficient to yield a  $(2\epsilon + \epsilon^2)$ -relaxed decision procedure.

LEMMA 3. *If there is a truly feasible packing, then there is a path in the multilayered graph from “initial” to “success.”*

*Proof.* Given a truly feasible packing, it is possible to label every piece in the  $k$ th interval  $(\epsilon^{k+1}, \epsilon^k]$  as  $L^{(k)}, M^{(k)}$ , or  $S^{(k)}$ , depending on the size of the bin in which it is packed:

- $j \in L^{(k)}$  if piece  $j$  is packed as large—in a bin in interval  $k$ ;
- $j \in M^{(k)}$  if piece  $j$  is packed as medium—in a bin in interval  $k - 1$ ;
- $j \in S^{(k)}$  if piece  $j$  is packed as small—in a bin in one of the intervals  $0, 1, \dots, k - 2$ .

In addition, let  $P^{(k)}$  denote all of the pieces in interval  $k$ .

Each node in the first layer of a stage is labeled with a state vector that specifies pieces as well as unused capacities. We will show by induction that certain induced partial packings of the truly feasible packing yield paths in the layered graph. The lemma will follow as a corollary to this stronger inductive assertion, which we will give below. Given a truly feasible packing, consider the partial packing induced by the pieces,  $M^{(k)} \cup (\cup_{i=0}^{k-1} P^{(i)})$ . This leaves a certain amount of slack  $v_1^{(k)}$  and  $v_2^{(k)}$  in the bins in the intervals 0 through  $k - 2$  (enormous bins) and in interval  $k - 1$  (huge bins), respectively.

We claim that there is a path from initial to some node in the first layer of stage  $k$  (so that interval  $k$  must contain bins) such that the state vector of that node has  $V_1 \geq v_1^{(k)}, V_2 \geq v_2^{(k)}$  and has the piece distribution of  $(L^{(k)} \cup S^{(k)}; P^{(k+1)})$ .

The claim is certainly true for  $k = 0$ , where there is a trivial path from the initial node to itself, and the initial node does have the appropriate state vector. (Note that  $M^{(0)}$  must be the empty set.)

Inductively, we can then assume that at the beginning of stage  $k$  (which contains bins) we have a path to a state vector labeled with the as-yet-unpacked pieces from interval  $k$ , all of the pieces from interval  $k + 1$ , and appropriate upper bounds on the slacks. Focus now on the packing of the bins in interval  $k$ . For each bin we have a truly feasible packing of the large and medium pieces, and this specifies a feasible

configuration. This configuration corresponds to an arc in the graph. Now we consider the change in the value of  $V$  between the tail and head of this arc. This change is the usable slack (properly represented), which is at least the true slack—for two reasons. First, we effectively round the size of the bin to the next largest multiple of  $\epsilon^{k+4}$ , and second we use the rounded (down) sizes of the pieces to compute the usable slack. Thus at each layer within the stage, the value of  $V$  in the state vector is at least as large as the actual slack in the truly feasible packing. The reader can verify with little difficulty that following the arcs specified by the feasible configurations generated by the given packing must lead to a node in the final layer of the stage that is labeled with a state vector with piece distribution corresponding to  $(S^{(k)}; L^{(k+1)} \cup S^{(k+1)})$  and where the total usable slack value  $V$  is at least the true slack.

It remains only to show that the update arcs are also present as needed. Consider the case where interval  $k + 1$  also contains bins. In this case, we need only show that the volume represented by  $V_1$  is at least  $\sum_{j \in S^{(k)}} \bar{p}_j$ , since then there is an update arc, and it leads to a node that satisfies the induction hypothesis. However, this inequality is easy to verify, since  $v_1^{(k)} \geq \sum_{j \in S^{(k)}} p_j$  and we know that  $V_1 \geq v_1^{(k)}$  (by induction) and  $p_j \geq \bar{p}_j$  (by definition). The remaining cases for the size of the next bin (corresponding to  $q = 2$ ,  $q \in [3, \infty)$  and  $q = \infty$ ) follow by similar calculations.

LEMMA 4. *If there is a path in the graph from initial to success, then there is a  $(2\epsilon + \epsilon^2)$ -relaxed feasible packing.*

*Proof.* There are two types of arcs in the graph, *l&m-pack* arcs that correspond to a feasible configuration, and the update arcs. The relaxed packing is done by tracing the path from initial to success as follows. Each bin has an arc along the path that specifies the *l&m-pack* of pieces in the bin. We pack bins in order of decreasing size (as they are encountered on the path) according to the feasible configuration specified by the arc (arbitrarily choosing pieces from each subinterval). It will be convenient to view the piece as having its rounded size, and later we will consider the effect of inflating it to its true size. In the updating phase at the end of the stage for interval  $k$ , we pack the remaining pieces from interval  $k$  as small pieces. An unpacked piece  $j$  with size in  $(\epsilon^{k+1}, \epsilon^k]$  is packed in any enormous bin  $l$  (in intervals  $1, \dots, k - 2$ ), with *positive* usable slack $_l$ . Recall that for a bin interval  $k$  its usable slack is computed as if the volume of the bin were rounded to the next highest multiple of  $\epsilon^{k+4}$ . We then update usable slack $_l := \max\{\text{usable slack}_l - \bar{p}_j, 0\}$ . This *small-pack* phase is always successfully completed since there is an update arc if and only if there is sufficient total slack to accommodate all pieces to be packed as small pieces, and the total usable slack is at least as large. Therefore, we are also able to construct a packing from the path.

Consider the (rounded) pieces packed into bin  $i$  where the size of bin  $i$  is  $s_i$ , which is in interval  $k$ . Focus on the small piece  $j$  that, when added to the bin, exhausts the usable slack; suppose that piece  $j$  is in interval  $k + 2$ . (It certainly cannot be in an interval of smaller index, since it must be small for bin  $i$ .) In this case, all pieces in the bin have rounded sizes that are multiples of  $\epsilon^{k+4}$ , and in fact, before piece  $j$  was added, the rounded piece sizes did not exceed  $s_i$ . Hence, the bin contains at most  $s_i + \epsilon^{k+2}$  in terms of rounded sizes. If piece  $j$  comes from an interval greater than  $k + 2$ , then the bin clearly contains total rounded size no more than  $s_i + \epsilon^{k+4} + \epsilon^{k+3}$ , which is less than  $s_i + \epsilon^{k+2}$  (since  $\epsilon \leq 1/2$ ). Therefore, if  $B_i$  is the set of pieces packed in bin  $i$ ,

$$\sum_{j \in B_i} \bar{p}_j \leq s_i + \epsilon^{k+2} \leq s_i + \epsilon s_i = (1 + \epsilon)s_i.$$

Furthermore, for any piece  $j$ ,  $p_j \leq (1 + \epsilon)\bar{p}_j$ , since if  $p_j \in (\epsilon^{k+1}, \epsilon^k]$ ,  $p_j \leq \bar{p}_j + \epsilon^{k+2} \leq \bar{p}_j + \epsilon\bar{p}_j$ . Combining these inequalities, we see that bin  $j$  is packed with (unrounded)

pieces of total size at most  $(1 + \varepsilon)(1 + \varepsilon)s_i$ . This is therefore a  $(2\varepsilon + \varepsilon^2)$ -relaxed feasible packing.

Thus, the procedure *all-pack* consisting of building the layered graph, finding a path from initial to success, and then (if possible) converting the path to a packing as described above, is a  $(2\varepsilon + \varepsilon^2)$ -relaxed decision procedure. By observing that  $2\varepsilon + \varepsilon^2 \leq \frac{5}{2}\varepsilon$  when  $\varepsilon \leq \frac{1}{2}$ , we see that the following theorem is a corollary of the previous discussion.

**THEOREM 2.** *For  $\varepsilon \leq \frac{1}{2}$ , the procedure *all-pack* delivers an  $\varepsilon$ -relaxed packing in  $O(m \cdot n^{(10/\varepsilon^2)+3})$  steps. Each step consists of one arc evaluation—at most  $(2/\varepsilon^2) - 1$  additions and one comparison, or for the update—at most  $n$  additions and 3 comparisons.*

**4. A  $\frac{1}{2}$ -relaxed decision procedure for bin packing with variable sizes.** In this section we consider a special case of the problem considered in the previous section: we fix  $\varepsilon = \frac{1}{2}$ . In other words, we wish to construct a procedure which, given an instance  $I = (J, S)$ , either concludes that no truly feasible packing exists, or else finds a  $\frac{1}{2}$ -relaxed feasible packing. Unlike the algorithm of the previous section, this algorithm is extremely simple and efficient. Once again, we assume that the bin sizes are  $s_1 \geq s_2 \cdots \geq s_m$ .

Consider the following recursive procedure:

```

procedure pack ( $J, S, m$ )
begin
  if  $\sum_{j \in J} p_j \leq \sum_{i \in S} s_i$  then
    begin
       $J_{\text{sml}} := \{j \mid p_j \leq s_m/2\}$ 
       $J_{\text{new}} := J - J_{\text{sml}}$ 
       $J_{\text{fit}} = \{j \in J_{\text{new}} \mid p_j \leq s_m\}$ 
      if  $J_{\text{fit}} \neq \text{empty}$  then
        begin
          choose  $\bar{j}$  such that  $p_{\bar{j}} = \max_{j \in J_{\text{fit}}} p_j$ 
          pack  $\bar{j}$  in bin  $m$  (+)
           $J_{\text{new}} := J_{\text{new}} - \{\bar{j}\}$ 
        end
         $S_{\text{new}} := S - \{m\}$ 
        if  $J_{\text{new}} \neq \text{empty}$  then call pack ( $J_{\text{new}}, S_{\text{new}}, m - 1$ )
        while there exists unpacked piece  $j \in J$ 
          find bin  $i$  packed with  $\leq s_i$  add piece  $j$  to bin  $i$  (*)
        end
      else
        output “no truly feasible packing”
      end
    end
  end

```

**LEMMA 5.** *If an instance  $I = (J, S)$  has a truly feasible packing then the instance  $I_{\text{new}} = (J_{\text{new}}, S_{\text{new}})$  created by procedure pack ( $J, S, m$ ) has a truly feasible packing.*

*Proof.* If  $(J, S)$  has a truly feasible packing, then certainly so does  $(J - J_{\text{sml}}, S)$ . Consider any such truly feasible packing. Since all of the pieces in  $J - J_{\text{sml}}$  are greater than  $s_m/2$  only one piece  $\bar{j}$  can be contained in bin  $m$ . If  $j \neq \bar{j}$  then form a new packing  $\tilde{}$  by interchanging these pieces. By the choice of  $\bar{j}$ ,  $p_j \leq p_{\bar{j}}$ , so the bin that contained  $\bar{j}$  before the interchange remains truly feasibly packed after the swap. Thus, by considering bins 1 to  $m - 1$ , we see that there is a truly feasible packing for the instance  $I_{\text{new}}$ .

**LEMMA 6.** *If the procedure pack ( $J, S, m$ ) outputs “no truly feasible packing” then there is no truly feasible packing.*

*Proof.* Suppose for a contradiction that there were a truly feasible packing. Then, by Lemma 5, for each recursive call of *pack* there is a truly feasible packing of the specified instance. However, for the failure message to be printed, the last of these instances must have  $\sum_j p_j > \sum_i s_i$ . This is clearly a contradiction, since no instance that has greater total piece size than total bin size can have a truly feasible packing.

LEMMA 7. *If the procedure  $\text{pack}(J, S, m)$  does not output “no truly feasible packing” then it successfully packs all pieces in a  $\frac{1}{2}$ -relaxed feasible packing.*

*Proof.* In considering the procedure *pack*, there is only one statement in which it could conceivably fail, and this is the statement indicated by (\*). Why should it always be possible to find a bin that is packed within its true capacity? If this were not possible, then all bins are packed beyond their true capacity, and then surely  $\sum_j p_j > \sum_i s_i$ . But this is precisely the situation we have excluded in this case of the **if** statement.

To show that the packing produced is  $\frac{1}{2}$ -relaxed feasible is also quite simple. Consider the two steps in the procedure in which pieces are packed. In the statement indicated by (+), we ensure that the piece fits within the true bin capacity. In statement (\*), we always add to some bin  $i$  a piece of size  $\leq s_m/2 \leq s_i/2$ , and since bin  $i$  previously contained  $\leq s_i$ , afterwards it contains no more than  $(3/2)s_i$ .

We now discuss an efficient implementation of the procedure  $\text{pack}(J, S, m)$ . We shall assume that the piece and bin sizes are given in sorted order. Note that the recursive procedure packs “large” pieces in bins of decreasing bin size, and then packs “small” pieces in bins of increasing size. It will be convenient to maintain two pointers to the sorted list of piece sizes: one to the largest piece no larger than the current bin size, and one to the smallest piece at least half the current bin size. By the monotonicity property just mentioned, only  $O(n)$  time is required to maintain these pointers, amortized over the running time of the procedure. Furthermore, given these pointers it is easy to see that the procedure can be implemented in linear time, since no piece need be “touched” more than a constant number of times. By combining these ideas with Lemmas 6 and 7, we get the following result.

THEOREM 3. *The procedure  $\text{pack}(J, S, m)$  is a  $\frac{1}{2}$ -relaxed decision procedure for the bin-packing problem with variable bin sizes. Furthermore, given the bin and piece sizes in sorted order, the algorithm runs in linear time.*

**5. Conclusions.** In considering the framework employed in the polynomial approximation scheme, it is important to note that this framework is not particular to this scheduling problem. As was discussed in [HS], the key notion in the success of this approach is a *dual approximation algorithm*. For the ordinary bin-packing problem, for example, an  $\varepsilon$ -dual approximation algorithm delivers a solution where the number of bins used is at most the optimum number, but is possibly infeasible. This infeasibility is bounded: each bin can contain no more than  $(1 + \varepsilon)$  times the original bin capacity.

The  $\varepsilon$ -relaxed decision procedure is essentially the same notion, except for the fact that there is no optimization involved. This can be fixed by considering the following generalized problem: given a set of pieces  $\{p_1, \dots, p_n\}$  and a profile of bin sizes  $s_1, \dots, s_m$  find the minimum number of copies of this profile needed to pack all of the pieces. It is quite simple to see how to convert the  $\varepsilon$ -relaxed decision procedure into an  $\varepsilon$ -dual approximation algorithm by using binary search.

We believe that this “dual approach” to approximation will continue to yield strong results in constructing approximation algorithms for problems for which good (traditional) approximation algorithms have been, heretofore, elusive. The work presented here certainly adds further confirming evidence.

## REFERENCES

- [CS] Y. CHO AND S. SAHNI, *Bounds for list schedules on uniform processors*, this Journal, 9 (1980), pp. 91-103.
- [FL] D. K. FRIESEN AND M. A. LANGSTON, *Bounds for multifit scheduling on uniform processors*, this Journal, 12 (1983), pp. 60-70.
- [GJ] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, San Francisco, 1979.
- [GIS] T. GONZALEZ, O. H. IBARRA, AND S. SAHNI, *Bounds for LPT schedules on uniform processors*, this Journal, 6 (1977), pp. 155-166.
- [G] R. L. GRAHAM, *Bounds for certain multiprocessing anomalies*, Bell System Tech. J., 45 (1966), pp. 1563-1581.
- [HS] D. S. HOCHBAUM AND D. B. SHMOYS, *Using dual approximation algorithms for scheduling problems: practical and theoretical results*, J. Assoc. Comput. Mach., 34 (1987), pp. 144-162.
- [HoS] R. HOROWITZ AND S. SAHNI, *Exact and approximate algorithms for scheduling non-identical processors*, J. Assoc. Comput. Mach., 23 (1976), pp. 317-327.
- [LL] J. W. S. LIU AND C. L. LIU, *Bounds on scheduling algorithms for heterogeneous computing systems*, in Information Processing 74, J. L. Rosenfeld, ed., North-Holland, Amsterdam, 1974, pp. 349-353.
- [GLLR] R. J. GRAHAM, E. L. LAWLER, J. K. LENSTRA, AND A. H. G. RINNOOY KAN, *Optimization and approximation in deterministic sequencing and scheduling: a survey*, Ann. Discrete Math., 5 (1979), pp. 287-326.



## A GRAPH THEORETIC APPROACH TO STATISTICAL DATA SECURITY\*

DAN GUSFIELD†

**Abstract.** In this paper we study the problem of protecting sensitive data in an  $n$  by  $n$  two-dimensional table of statistics, when the nonsensitive data are made public along with the row and column sums for the table. A sensitive cell is considered unprotected if its exact value can be deduced from the nonsensitive cell values and the row and column sums. We give an efficient algorithm to identify all unprotected cells in a table. The algorithm runs in linear time if the sensitive values are known, and in  $O(n^3)$  time if they are not known. We then consider the problem of suppressing the fewest additional cell values to protect all the sensitive cells, when some cells are initially unprotected. We give a linear time algorithm for this problem in the useful special case that all cell values are strictly positive. We next consider the problem of computing the tightest upper and lower bounds on the values of sensitive cells. We show that each cell bound can be computed in  $O(n^3)$  time, but all  $\Theta(n^2)$  values can be computed in  $O(n^4)$  total time. In the case that all the cells are sensitive, we show that trivial methods compute the tightest bounds, and in fact, the  $n^2$  lower bounds can be computed in  $O(n)$  arithmetic and comparison operations. Although these problems at first appear to be problems in integer linear algebra, the solutions are based on computational graph theory, and the worst case times are significantly faster than for approaches based on linear algebraic computations.

**Key words.** data security, graph algorithms, network flow

**AMS(MOS) subject classifications.** 68R10, 90B10, 68Q25

**1. Introduction.** In this paper we study the problem of protecting sensitive data in a two-dimensional table of statistics when the nonsensitive data is made public. Work in this area was begun by Fellegi, Cox, and Sande [FEL], [COX75], [COX77], [COX78], [COX80], [SAN], and reported in Denning [DEN]. The general problem is motivated by concerns for privacy and security, and is a problem of practical importance and active interest for both the U.S. Census Bureau [USDC], and Statistics Canada [BCS]. For a more complete discussion of background and motivation see [DEN]. The problems considered in the paper all appear at first to be problems in integer linear algebra, but in this paper they are reduced to graph theoretical problems where they are more efficiently solved.

**1.1. Problem statements, definitions, and main results.** The basic setting for the paper is that one party (the Census Bureau, say) has a two-dimensional table,  $D$ , of cross-tabulated integer statistics; each entry  $D(i, j)$  is a nonnegative integer in cell  $(i, j)$  of  $D$ ;  $R(i)$  is the sum of the cell values in the  $i$ th row of  $D$ , and  $C(j)$  is the sum of the cell values in the  $j$ th column of  $D$ . We assume that  $D$  has at least two rows and at least two columns, and that each  $R(i)$  and  $C(j)$  is strictly positive. All the row and column sums are to be made public (disclosed) along with *some* of the cell values, but the remaining cell values, called *sensitive* values, are to be *suppressed* (not disclosed). Unless care is taken, however, the disclosure of the nonsensitive values might allow an adversary to deduce the *exact* value of one or more of the sensitive values. Hence, to avoid disclosing the exact value of any sensitive cell, the values of some nonsensitive cells may also need to be suppressed; these suppressions are called *complementary suppressions*. In a table  $D$ , the set of *suppressed cells* consists of the sensitive cells along with any complementary suppressed cells.

---

\* Received by the editors August 25, 1986; accepted for publication (in revised form) June 12, 1987. This research was supported in part by National Science Foundation grant MCS-81/05894 and by U.S. Census Bureau grant JSA 86-9.

† Division of Computer Science, University of California, Davis, California 95616.

Given a table with suppressions, a suppressed cells is called *protected* if its exact value is not determinable from knowledge of only the row and column totals, and the values of the unsuppressed cells. To make this precise, let  $X$  be the set of all suppressed cells and for cell  $(i, j) \in X$ , let  $x(i, j)$  be a variable denoting the value of the cell. For each row  $i$  and column  $j$ , let  $R^*(i)$  and  $C^*(j)$  be, respectively, the sum of the undisclosed values in row  $i$  of  $D$ , and the sum of the undisclosed values in column  $j$  of  $D$ .

DEFINITION. Let  $R$  be the set of row indices and  $C$  be the set of column indices. For  $i \in R$ , let  $X(i)$  be the set of indices  $j \in C$  such that  $(i, j)$  is a suppressed cell. Similarly, for  $j \in C$ , let  $X(j)$  be the set of indices  $i \in R$  such that  $(i, j)$  is a suppressed cell.

DEFINITION. We define a *legal solution* of  $D$  as a nonnegative integer solution to the following system of linear equalities:

For each fixed  $i \in R$ ,  $\sum_{j \in X(i)} x(i, j) = R^*(i)$ , and

For each fixed  $j \in C$ ,  $\sum_{i \in X(j)} x(i, j) = C^*(j)$ .

DEFINITION. A cell  $(i, j)$  in  $X$  is *protected* if and only if there exists two legal solutions  $x_1$  and  $x_2$ , such that  $x_1(i, j) \neq x_2(i, j)$ .

DEFINITION. A table with suppressions is *protected* if and only if all of its suppressed cells are protected.

Notice that this definition of protection essentially implies that side information such as correlations between cell values or special bounds on cell values etc. are unknown to the adversary, or at least do not restrict the set of legal solutions. Most of the results in this paper can easily be extended to the cases when given upper and lower bounds on the values of individual cells are known to the adversary (provided that the Census Bureau knows what bounds the adversary is assuming); we will briefly discuss these extensions later. Notice also that the assumption that each  $R(i)$  and  $C(j)$  is strictly positive is reasonable, since without the assumption there is no way to protect the (all zero) entries in such a row or column.

In this paper we discuss four problems:

(a) (Census Bureau problem) Given knowledge of the complete, original table,  $D$ , and given a set of cells  $X$ , determine whether table  $D$  would be protected after the suppression of the values in set  $X$ .

We cast this problem as a graph theoretic problem and give an algorithm that identifies all unprotected cells and runs in linear time in the size of  $X$ .

(a') (Adversary problem) Given a table  $D$ , where the values in the set of cells  $X$  are suppressed, hence unknown, determine whether  $D$  is protected, and if not, determine the exact value of all the unprotected cells.

We reduce this problem to the Census Bureau problem, using a single maximum flow computation on a graph with  $|X|$  edges. Hence, after one maximum flow computation, the adversary problem can be solved, as above, in time linear in  $|X|$ .

(b) (Complementary Suppression problem) Given the full  $n \times m$  table  $D$ , and a set of sensitive cells  $X$  to be suppressed, determine the minimum number (and selection) of complementary suppressions needed so that all the sensitive cells are protected. This is a problem that the Census Bureau faces when choosing which cells to disclose in the case that additional suppressions are needed to protect the sensitive cells.

We solve this problem in time  $O((n + m)|X|)$ , for the case that all cell values are strictly positive. The solution is based on a computational problem in graph theory studied by Eswaran and Tarjan [ET]. In [G] we sketch a more complex linear time solution to this special case.

(c) (General Interval Estimation) Given a table with a set  $X$  of cell suppressions, determine the tightest upper and lower bounds on the values of each suppressed cell.

This is a generalization of problem (a') since a cell is protected if and only if the upper and lower bounds on its value are unequal.

We show how to compute each bound of a suppressed cell with a single  $O(n^3)$  time network flow computation. In an  $n \times n$  table with  $\Theta(n^2)$  sensitive cells, this yields an  $O(n^5)$  time algorithm to compute all the bounds. We then improve this running time to  $O(n^4)$ , by reducing the problem of computing all the upper bounds to a problem studied in [SC]. We also show the surprising result that in an  $n \times m$  table, there can never be more than  $n + m - 1$  distinct upper bound values, no matter how many cell values are suppressed.

(d) (Special Interval Estimation) In the case that all the cell values are suppressed, determine the tightest upper and lower bounds on the cell values. We show that trivial methods suffice here, and in fact, the lower bounds can be computed with  $O(\max [n, m])$  arithmetic operations and comparisons in an  $n \times m$  table, although the output has size  $nm$ .

In the published literature on these problems, problem (c) is solved by minimum cost flow or linear programming methods applied independently for each bound [COX77], [COX78], [COX80], [DEN]; problem (b) is solved by trial and error, iteratively using the upper and lower bounds computed by (c) [COX80]; problem (a) is solved by computing upper and lower bounds and checking for equality [COX80]; and no special, efficient method is discussed for problem (d). Hence the results here provide substantial theoretical and practical improvements for problems (a), (b), and (d), and a theoretical improvement in problem (c). It is an empirical question whether the simpler and faster (in worst case) network flow methods for problem (c) are better in practice than the linear programming methods suggested in the literature [COX78].

In actual systems concerned with statistical security, problems (a) and (c) are often in the inner loop of larger programs that repeatedly modify the table and then need to know if the cells are secure, or what the tightest bounds on their values are. For example, problem (b) is of that type. Hence any speedup in solving these two problems is particularly important.

**2. Determining if  $D$  is protected.** In this section, we discuss problem (a), giving a linear time algorithm to determine if  $D$  is protected. Although the Census Bureau problem was posed above as a problem in integer linear algebra, we will see that it is more effectively treated when viewed as a problem in computational graph theory.

**DEFINITION.** Graph  $H$  is a bipartite graph with node set  $R$  on one side representing the rows of table  $D$ , and node set  $C$  on the other side representing the columns of  $D$ . There is an edge  $(i, j)$  between node  $i$  on  $R$  and node  $j$  on  $C$  if and only if cell  $(i, j)$  is suppressed in  $D$ . An edge  $(i, j)$  in  $H$  is undirected if and only if  $D(i, j) > 0$ , and is directed from node  $i$  in  $R$  to node  $j$  in  $C$  if and only if  $D(i, j) = 0$ .

Figure 1 shows an example of a table  $D$  and Fig. 2 shows the graph  $H$  derived from  $D$ .

**DEFINITION.** We use the notation  $\langle i, j \rangle$  to indicate that edge  $(i, j)$  is directed from node  $i$  to node  $j$ . Where the distinction does not matter, we will use the term "edge" and the notation  $(i, j)$  to refer to either the undirected edge  $(i, j)$  or the directed edge  $\langle i, j \rangle$ .

**DEFINITION.** A cycle in  $H$  is called *traversable* if it is a simple cycle, and it is possible to walk around it without ever going opposite to the direction of any of the directed edges on it. Clearly any directed or undirected simple cycle is traversable. Note that an undirected edge in  $H$  can be used in one direction in one traversable cycle, and in the other direction in another traversable cycle.

T = 31		9	5	4	6	7	C*(j)
	73	18	6	17	15	17	C(j)
7	21	5	2	6	0	8	
4	10	4	0	0	4	2	
7	15	3	3	4	5	0	
7	13	4	0	2	0	7	
6	14	2	1	5	6	0	
R*(i)	R(i)	table D					

FIG. 1. The squared numbers indicate the sensitive cells.

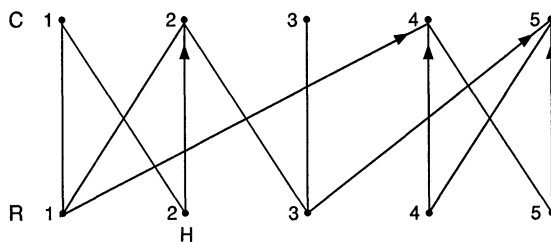


FIG. 2

The following theorem reduces the Census Bureau problem to a graph theoretic problem.

**THEOREM 1.** *A suppressed cell  $(i^*, j^*)$  is protected if and only if its associated edge in  $H$  is contained in at least one traversable cycle in  $H$ .*

*Proof.* We first show sufficiency. Let  $SC$  be a traversable cycle in  $H$ . Arbitrarily pick a directed edge  $e$  in  $SC$ , and if there are none, then pick any edge  $e$  in  $SC$ . Starting with edge  $e$ , alternately label the edges around  $SC$  positive and negative. Since  $H$  is bipartite, the labeling is well defined, and since  $SC$  is traversable, all the directed edges will have positive labels. Let  $f$  be the minimum of the  $D(i, j)$  values of the negative labeled edges in  $SC$ . Adding  $f$  to  $D(i, j)$  for any positive labeled edge  $(i, j)$ , and subtracting  $f$  from  $D(i, j)$  for the negative labeled edge  $(i, j)$  gives a legal solution  $x$  to  $D$ , such that for any edge  $(i, j)$  on  $SC$ ,  $x(i, j) \neq D(i, j)$ . Hence all cells associated with edges in  $SC$  are protected.

To prove necessity, suppose cell  $(i^*, j^*)$  is protected and let  $x$  be a legal solution to  $D$  such that  $x(i^*, j^*) \neq D(i^*, j^*)$ . Define  $F(i, j) = x(i, j) - D(i, j)$ , for each suppressed cell in  $D$ . We represent  $F$  as a weighted bipartite graph,  $F^*$ , containing directed edge  $\langle i, j \rangle$ ,  $i \in R, j \in C$ , with weight  $F(i, j)$ , if  $F(i, j) > 0$ , and containing directed edge  $\langle j, i \rangle$ ,  $i \in R, j \in C$ , with weight  $F(i, j)$ , if  $F(i, j) < 0$ . Clearly, if there is no edge between nodes  $i$  and  $j$  in  $H$ , then there is none in  $F^*$ . Further, if  $\langle i, j \rangle$  is an edge in  $H$ , then edge  $\langle j, i \rangle$  cannot appear in  $F^*$ , since  $D(i, j) = 0$  and so  $F(i, j) = x(i, j) - D(i, j) \geq 0$ . Since  $x(i^*, j^*) \neq D(i^*, j^*)$ , either  $\langle i^*, j^* \rangle$  or  $\langle j^*, i^* \rangle$  is an edge in  $F^*$ . We will suppose that  $F^*$  contains edge  $\langle i^*, j^* \rangle$  and show that  $\langle i^*, j^* \rangle$  is contained in a traversable cycle in  $H$ . The case when  $F^*$  contains edge  $\langle j^*, i^* \rangle$  is symmetric.

First, note that for any fixed node  $i$  in  $F^*$ , the weights of all the edges incident with  $i$  sum to zero. Hence any node  $i$  in  $F^*$  that is incident with at least one edge is incident with at least one edge directed into  $i$  and one directed out of  $i$ ; we call this the *balance* property. It follows easily from the balance property, and from the fact

that  $\langle i, j \rangle$  is an edge in  $F^*$  only if  $\langle j, i \rangle$  is not, that there exists a traversable cycle,  $SC$ , in  $F^*$ , with at least four edges. Notice that in  $SC$  the signs on the edges alternate positive, negative around the cycle. Cycle  $SC$  must also be a traversable cycle in  $H$ , since every edge in  $SC$  is either in  $H$  with the same direction as in  $F^*$ , or is directed in  $F^*$  and undirected in  $H$ . Let  $f$  be the minimum absolute value of the weights of the edges on  $SC$ . We subtract  $f$  from each positive weight on  $SC$ , add  $f$  to each negative weight on  $SC$ , and remove any edges with resulting weight zero (there is at least one); the result is a subgraph of  $F^*$  which again has the balance property. We can therefore continue finding traversable cycles in  $F^*$ , and hence also in  $H$ , changing weights and removing edges each time, until there are no more edges left. Edge  $\langle i^*, j^* \rangle$  must be in at least one of these cycles, hence must be in a traversable cycle in  $H$ .  $\square$

Theorem 1 shows that for the question of protection, only the directions of the edges in  $H$  matter. This means that the only distinction that matters (in determining protection) is whether  $D(i, j)$  is zero or not. Notice also that the Census Bureau problem can be answered without needing to explicitly know  $R^*(i)$  or  $C^*(j)$ .

**2.1. Graph  $CG$ .** By Theorem 1, problem (a) can be solved by determining which (if any) of the edges of  $H$  are in no traversable cycles in  $H$ . We next solve this problem with a linear time algorithm that constructs the mixed graph  $CG$ ; the edges of  $CG$  are exactly those edges of  $H$  that are in no traversable cycles in  $H$ . Graph  $CG$  will also be central in the problem of minimizing the number of complementary suppressions. The construction of  $CG$  uses the graph theoretic objects of *bridges* and *strong components*. Definitions of these objects and algorithms to find them are fully developed in a number of texts, such as [AHU] and [EVE].

#### CONSTRUCTION OF GRAPH $CG$ .

(1) Find the strongly connected components of  $H$ . Let  $A$  be the set of edges running between these components. Clearly all the edges of  $A$  are directed edges in  $H$ .

(2) Consider each of the strong components as a separate undirected graph, ignoring the directions on any directed edges. Find all (if any) bridges in each of the separate graphs, and let  $B$  be the set of bridges found. It is easily proved that any edge in  $B$  is an undirected edge in  $H$ .

(3) Let  $K$  be the graph induced by the edges of  $H$  minus  $A \cup B$ , and let  $K(1), \dots, K(r)$  be the connected components of  $K$ . Graph  $CG$  is formed by condensing each  $K(i)$  in  $H$ . Hence each node in  $CG$  is in 1-1 correspondence with some  $K(i)$ , and the edges of  $CG$  are exactly  $A \cup B$ .

In Fig. 3, graph  $CG$  is constructed from  $H$  shown in Fig. 2.

**THEOREM 2.** *Edge  $\langle i, j \rangle$  is in no traversable cycle in  $H$  if and only if  $\langle i, j \rangle$  is in  $A \cup B$ . That is, cell  $\langle i, j \rangle$  is unprotected if and only if nodes  $i$  and  $j$  are in different components of  $K$ .*

*Proof.* First any edge which is in a traversable cycle in  $H$  is in some strong component of  $H$ , and so no edge in set  $A$  is in a traversable cycle. Second, any directed edge  $\langle u, v \rangle$  in a strong component is traversable, since, by definition of strong component, there must be a traversable path  $P$  from  $v$  to  $u$  in  $H$ . So edge  $\langle u, v \rangle$  followed by path  $P$  is a traversable cycle in  $H$ . Hence  $A$  exactly contains the directed edges that are on no traversable cycles. Now we consider which undirected edges of  $H$  are traversable. Since no edge in  $A$  is in a traversable cycle, we can delete set  $A$  from  $H$  without affecting which undirected edges are traversable. Any bridges in the resulting graph (considered undirected) are clearly nontraversable, so all edges in  $B$  are nontraversable. What remains to be shown is that any undirected edge  $\langle u, v \rangle$  not in  $B$  is in some traversable cycle in  $H$ . Since  $\langle u, v \rangle$  is not a bridge in the strong component it

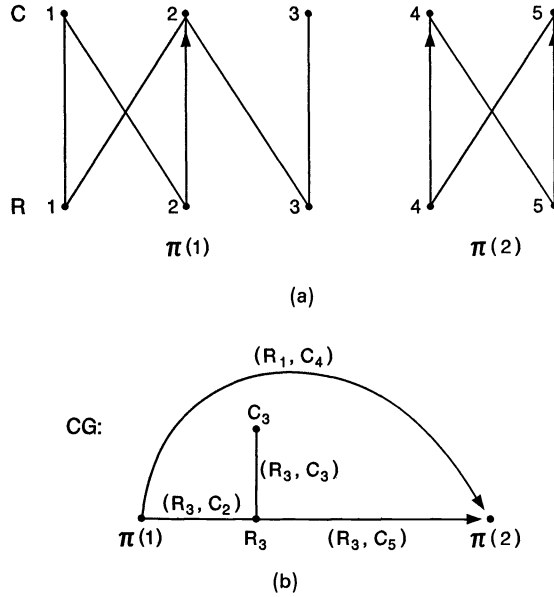


FIG. 3. Construction of graph  $CG$  from  $H$ . Figure 3(a) shows the two strong components  $\pi(1)$  and  $\pi(2)$  of  $H$ . In  $\pi(1)$  the bridges are  $(R_3, C_2)$  and  $(R_3, C_3)$ . There are no bridges in  $\pi(2)$ . The edges of set  $A$  are  $(R_1, C_4)$  and  $(R_3, C_5)$ . Figure 3(b) shows graph  $CG$ .

is contained in, there must be a path  $P$  in  $H$ , not containing edge  $(u, v)$ , connecting (ignoring the directions of the edges on  $P$ ) node  $u$  and node  $v$ . If  $P$  is traversable in either direction, we are done. If  $P$  is not traversable, let  $\langle x, y \rangle$  be the first directed edge on  $P$  such that the walk from  $u$  to  $v$  on  $P$  goes in the wrong direction, i.e., from  $y$  to  $x$ . Since  $\langle x, y \rangle$  is not in set  $A$ , it is on some traversable cycle in  $H$ , and so there is a traversable path,  $P'$ , from  $y$  to  $x$  in  $H$ . Now the path consisting of  $P$  to  $y$ ,  $P'$  to  $x$ , and  $P$  to  $v$  is a path from  $u$  to  $v$  which traverses one less edge in the wrong direction than does  $P$ . By repeating this argument, a traversable path in  $H$  from  $u$  to  $v$  can be found, and hence  $(u, v)$  is in a traversable cycle in  $H$ .  $\square$

Hence problem (a) can be solved in time linear in the size of  $X$ , as it is well known [AHU], [EVE] that strong components and bridges can be found in time linear in the number of edges and nodes in the graph.

Figure 3 shows that cells  $(1, 4)$ ,  $(3, 2)$ ,  $(3, 3)$ , and  $(3, 5)$  of table  $D$  in Fig. 1 are unprotected.

**2.2. Solving the adversary problem (a').** Let  $x$  be any legal solution to  $D$ , and let  $D(x)$  be the complete table formed by assigning  $x(i, j)$  to each suppressed cell  $(i, j)$  in  $D$ . It follows by the definition of protection that any suppressed cell is unprotected in  $D(x)$  if and only if it is unprotected in  $D$ . Hence, although the adversary does not have the full original table  $D$ , if he knows *any* legal solution  $x$  to  $D$ , then he can input  $D(x)$  to the above algorithm and determine which cells in  $D$  are unprotected. If  $(i, j)$  is such a cell, then  $D(i, j) = x(i, j)$ , and hence the adversary knows its value.

We now show that a legal solution to  $D$  can be found in time  $O(n^3)$  using network flow.

**2.2.1. Finding a legal solution.**

**DEFINITION.** Network  $G$  is constructed from graph  $H$  as follows: first direct each edge  $(i, j)$  in  $H$  from  $i \in R$  to  $j \in C$ , and give each such edge a capacity of infinity; then add a node  $s$  to  $H$  and a directed edge from  $s$  to each node of  $R$ , and a node  $t$

to  $H$  and a directed edge from each node of  $C$  to  $t$ . The capacity of each edge  $\langle s, i \rangle$  is set to  $R^*(i)$ , and the capacity of each edge  $\langle j, t \rangle$  is set to  $C^*(j)$ . Let  $T$  be the sum of the capacities of the edges out of  $s$  ( $T$  is also the sum of the edge capacities of the edges into  $t$ ).

DEFINITION. A flow function  $F$  is an assignment of a nonnegative real number,  $F(i, j)$ , to each directed edge  $\langle i, j \rangle$  of  $G$  such that  $F(i, j)$  is less than or equal to the capacity of edge  $\langle i, j \rangle$  and so that for every node  $i \neq s, t, \sum_j F(i, j) = \sum_j F(j, i)$ . The quantity  $\sum_i F(s, i)$  is called the *value* of the flow, and it also equals  $\sum_i F(i, t)$ . A flow is called a *maximum flow* if its value is maximum over all flows in  $G$ . A flow is called *integral* if  $F(i, j)$  is an integer for every edge  $\langle i, j \rangle$ .

It is known [FF] that a maximum integral flow exists if the capacities on all the edges are integers. The following fact is immediate.

FACT 1. Let  $D$  and  $G$  be as above and let  $x$  be any legal solution of  $D$ . If  $F(i, j)$  is set to  $x(i, j)$  for each  $(i, j)$  in  $X$ , then  $F(i, j)$  is an integral maximum flow in  $G$  of value  $T$ , i.e.,  $F$  is an integral flow that saturates all edges out of  $s$ , and these edges form an  $s-t$  cut in  $G$ . Conversely, any integral maximum flow in  $G$  (of value  $T$ ) defines a solution to  $D$  in the analogous way:  $x(i, j) \leftarrow F(i, j)$  for  $(i, j) \in X$ . Hence a suppressed cell  $(i, j)$  in  $D$  is protected if and only if there exists two integral flows  $F_1$  and  $F_2$  in  $G$ , each of value  $T$ , such that  $F_1(i, j) \neq F_2(i, j)$ .

The example in Fig. 4 illustrates the above definitions and fact.

It is well established that the maximum flow can be found in time  $O(n^3)$  in a graph with  $O(n)$  nodes, and for sparse graphs, the maximum flow can be found in

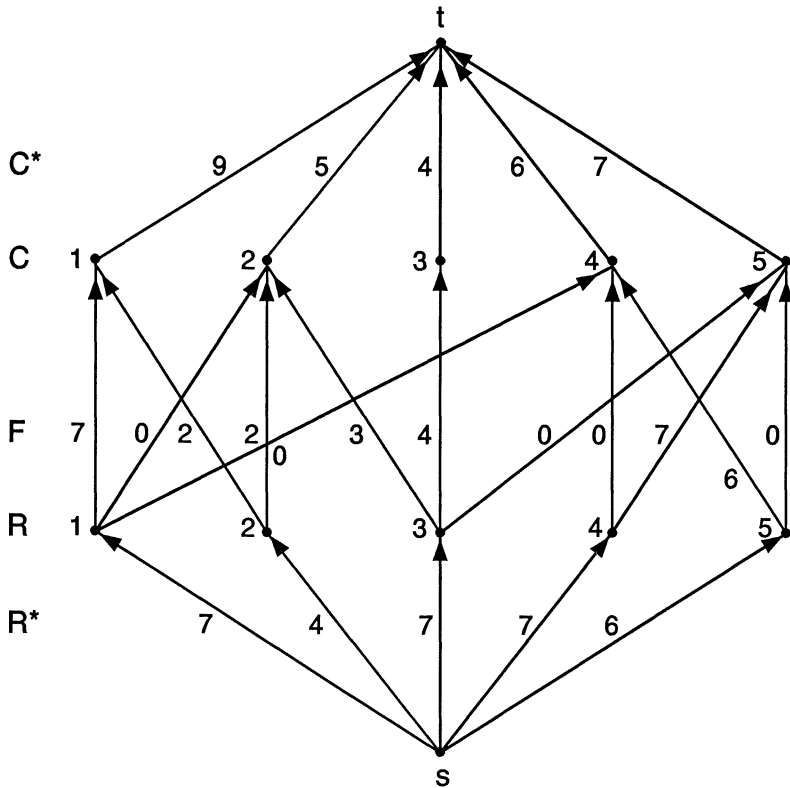


FIG. 4. Graph  $G$  derived from  $D$ . A maximum flow  $F$  is indicated on the center edges.

time  $O(nE \log n)$ , where  $E$  is the number of edges in the graph [ST]. Recent papers on network flow have improved these time bounds to  $O(nE \log(n^2/E))$  [GT], and  $O(nE + n^2 \log(c_{\max}))$  [AO], where  $c_{\max}$  is the largest capacity in the network. It has also been established [GMB] that for an  $n \times m$  table where  $n < m$ , the flow can be found in  $O(n^2 m)$  time rather than  $O((n+m)^3)$ ; this distinction is important when  $n \ll m$ .

Note that problem (a) is solved in linear time while problem (a') first requires the computation of a maximum flow. These two solutions are essentially the same, but in problem (a), the Census Bureau has the advantage that it knows a particular legal solution for free, namely the one corresponding to  $D$  itself.

**3. Optimizing complementary suppressions.** In this section, we will use  $H$  and  $CG$  to represent and help determine the optimal complementary suppressions in  $D$ ; we will see that this problem can be cast as a graph augmentation problem on  $CG$ .

Each complementary suppression in  $D$  charges graph  $H$ : if cell  $(i, j)$  is suppressed and  $D(i, j) = 0$  then the directed edge  $\langle i, j \rangle$ ,  $i \in R, j \in C$ , is added to  $H$ , and if  $D(i, j) > 0$  then the undirected edge  $(i, j)$  is added to  $H$ . This direct correspondence between cell suppressions in  $D$  and edge additions in  $H$ , combined with Theorem 1, implies that the complementary suppression problem can be solved by taking any  $H$  and adding the fewest number of edges corresponding to complementary suppressions in  $D$ , so that every edge in the resulting graph is in some traversable cycle. Problems of this type are called *graph augmentation* problems.

It is easy to see (and prove) that the complementary suppression problem can be solved using  $CG$  in place of  $H$ , i.e., the smallest set of edges that must be added to  $CG$  so that every edge is traversable, is also the smallest set that must be added to  $H$  to make every edge in it traversable, and vice versa. What makes this augmentation problem difficult, is that only those edges corresponding to unsuppressed cells in  $D$  can be added to  $CG$ , and if  $(i, j)$  is an unsuppressed cell where  $D(i, j) = 0$ , then the edge corresponding to cell  $(i, j)$  is the *directed* edge  $\langle i, j \rangle$ ,  $i \in R, j \in C$ .

The above complementary suppression problem has been shown to be NP-complete [KG], but the graph  $CG$  is still useful, since one can quickly see the effect of any proposed complementary suppressions. This leads to efficient implementation of heuristics (such as those suggested in [COX80]) based on hill climbing and incremental optimization, i.e., incrementally selecting a suppression that makes the most edges traversable, updating  $CG$ , and repeating until every edge is in some traversable cycle. Further, there are useful special cases of the complementary suppression problem that can be efficiently solved. In the following section, we solve the case when  $D(i, j) > 0$  for every cell in  $D$ . This assumption is valid for a large class of problems, and is of practical interest. We can also apply the solution to the more general class of problem where  $D(i, j) > 0$  for every sensitive cell, but where other cells in  $D$  may have zero value.

**3.1. Special case:  $D$  strictly positive.** In this section, we optimally and efficiently solve the complementary suppression problem under the assumption that  $D(i, j) > 0$  for every cell in  $D$ . This condition is often satisfied in certain classes of Census Data, and the agent with the complete table can exploit the condition when it arises. We should note, however, that for the definition of protection to remain relevant, the adversary must not know when these special cases arise. That is, the adversary must work in the solution space that includes zero, while the Census Bureau can exploit more restrictive assumptions when appropriate. In this section, we present a simple  $O((n+m)|X|)$  time method based on a related problem and solution studied by Eswaran and Tarjan [ET]. In [G], we examine a more general class of graph augmenta-



tion problems (containing the special case discussed here) and sketch a solution to those problems that solves the special case in  $O(|X|)$  time. The faster solution is more involved than the one given here.

*Unlabeled graph augmentation.* In [ET], the following problem is solved: Given a forest,  $FG$ , of undirected trees, add the fewest number of undirected edges to  $FG$  so that every edge is in a simple cycle (hence of length at least three), where it is permitted to add an edge between *any* pair of nodes in  $FG$ . In our applications, we will only need the solution to this problem when  $FG$  is a single tree and contains at least three leaves (isolated nodes never arise in our applications, and the complementary suppression problem for two leaves is easy to solve directly). We sketch the solution of the unlabeled graph augmentation problem given in [ET], for the case of  $FG$  a single tree.

#### ALGORITHM ET

1. Number the leaves of  $CG$  in pre-order, i.e., traversing the tree by depth-first search, the leaves are numbered in the order that they are reached. Let  $v(i)$  be the  $i$ th leaf reached.

2. Suppose  $FG$  has  $p$  leaves and  $k = \lfloor p/2 \rfloor$ . Add an edge between leaves  $v(i)$  and  $v(k+i)$ , for  $i = 1$  through  $k$ . If  $p$  is odd ( $p = 2k + 1$ ), then add an edge between  $v(p)$  and any other leaf in  $FG$ .

For a proof that this solution is correct, see [ET]. Note that Algorithm ET clearly runs in time linear in the number of edges of  $FG$ .

*Labeled augmentation: modifying ET.* When table  $D$  is strictly positive,  $CG$  is a forest of undirected trees. If we take  $CG$  in place of  $FG$  and use Algorithm ET (adding  $\lfloor p/2 \rfloor$  edges and yielding graph  $CG^*$ ) we may not end up with a solution to the complementary suppression problem since “illegal” edges (those not corresponding to unsuppressed cells in  $D$ ) may have been added. We give here an  $O((n+m)|X|)$  time algorithm that takes  $CG^*$  and transforms it to obtain an optimal solution to the suppression problem. We first need some definitions and observations.

**DEFINITION.** In  $CG$  we give a node the label of  $R$  if it is a node on the  $R$  side of  $G$ , we give it the label  $C$  if it is on the  $C$  side of  $G$ , and we give it the label  $B$  if it is a condensed node and therefore contains nodes of  $G$  from both  $R$  and  $C$ . Let  $r$ ,  $c$  and  $b$  be the number of leaves of  $CG$  labeled  $R$ ,  $C$ , and  $B$ , respectively.

Given this notation, the complementary suppression problem for the case that  $D$  is strictly positive is a graph augmentation problem on  $CG$  with the constraint that no  $(R, R)$  edges or  $(C, C)$  edges may be added. We will refer to this graph augmentation problem as the *labeled augmentation problem*.

**FACT 2.** Assuming  $r \leq c$ , a simple lower bound on the size of the optimal solution of the labeled augmentation problem is  $c$  if  $r + b \leq c$ , and is  $\lfloor (r + c + b)/2 \rfloor$  otherwise. We will see that this bound can be exactly met.

*Assumption.* We will assume that  $CG$  has  $p = r + b + c > 2$  leaves, that  $p$  is even, that  $r \leq c$ , and until § 3.1.4, that  $r + b \geq c$ . The complementary suppression problem is trivial when  $p = 2$ . When  $p$  is odd, we reduce the analysis to the even case as follows: If  $r + b \geq c$  and  $p$  (defined as  $r + b + c$ ) is odd, it follows that  $r + b > c$ ; we remove from  $CG$  any  $B$  leaf and the unique path from it to a node of degree three or more, creating a forest with an even number of leaves where  $r + b \geq c$ . The solution to the labeled augmentation problem when  $p$  is odd is then obtained by taking the solution to the reduced case ( $p$  even) and adding an edge from the deleted  $B$  leaf to any other leaf. In § 3.1.4 we will briefly discuss the case that  $r + b < c$ .

*CG a single tree.* To explain the general idea of transforming the solution given by Algorithm ET into a solution of the labeled augmentation problem, we first assume that *CG* is a single tree; later we discuss how to handle the case of *CG* a forest.

DEFINITION. Let  $E^*$  initially be the set of edges added to *CG* by Algorithm ET, and let  $CG^*$  be the graph resulting from adding  $E^*$  to *CG*.

We note the following consequence of Algorithm ET, which is an easy extension of the correctness proof of ET given in [ET].

FACT 3. When *CG* is a single tree, every edge in *CG* is in a cycle in  $CG^*$  containing exactly one edge of  $E^*$ . That is, let  $C(e)$  be the unique cycle created by adding an edge  $e$  of  $E^*$  to *CG*, and let  $CC$  be the set of all such cycles, then every edge in *CG* is contained in at least one cycle of  $CC$ . When edge  $e'$  of *CG* is in  $C(e)$ , we say that edge  $e$  covers edge  $e'$ .

THEOREM 3.  $E^*$  can be transformed into an optimal solution to the labeled augmentation problem in time  $O((n + m)|X|)$ .

*Proof.* If  $E^*$  contains no  $(R, R)$  edge and no  $(C, C)$  edge then it is a solution to the labeled augmentation problem. However, if there is an  $(R, R)$  edge,  $e$ , in  $E^*$ , then since  $r \leq c$ , there must also be either a  $(C, C)$  edge or a  $(C, B)$  edge in  $E^*$ . Suppose there is a  $(C, C)$  edge,  $e'$ , in  $E^*$ ; the case of  $(C, B)$  is similar. Let  $S$  be the unique smallest subgraph of *CG* that connects the endpoints of  $e$  and  $e'$ . These four endpoints are leaves of  $S$ , and the topology of  $S$  is one of the three cases shown in Fig. 5. In each of the three cases there is a way to add two  $(R, C)$  edges between the endpoints of  $e$  and  $e'$  so that every edge of  $S$  is in a cycle consisting of edges of  $S$  plus exactly one of the two new edges, i.e., every edge in  $S$  is covered by one of the new edges. For example, in the first case we add an edge between the upper left  $C$  and lower right  $R$ , and an edge between the upper right  $C$  and the lower left  $R$ . Now in  $CG^*$ , all edges covered by  $e$  and  $e'$  are in  $S$ , and so by Fact 3, every edge in *CG* not in  $S$  is covered by some edge of  $E^* - \{e, e'\}$ . Hence we can delete  $e$  and  $e'$  from  $E^*$ , and add the appropriate two new  $(R, C)$  edges to  $E^*$ , so that every edge in *CG* is again covered by some edge in  $E^*$ . Note also that  $E^*$  remains disjoint from the edges of *CG*. We call such a set of edge deletions and insertions an *edge exchange*. By successive edge exchanges we can find a set of edges,  $E^*$ , containing no  $(R, R)$  edges, such that every edge in *CG* is covered by an edge in  $E^*$ . Note that in these edge exchanges, the size of  $E^*$  remains constant.

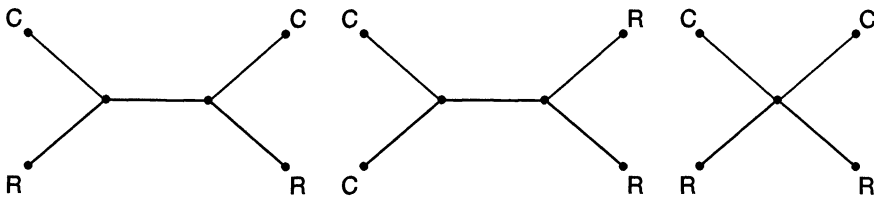


FIG. 5. Cases for edge exchanges. Each line represents a path between the endpoints.

Suppose now that  $E^*$  contains no  $(R, R)$  edges; we will remove all the  $(C, C)$  edges in  $E^*$  using edge exchanges similar to those above. If there is a  $(C, C)$  edge in  $E^*$ , but no  $(R, R)$  edge, then there must be either an  $(R, B)$  or a  $(B, B)$  edge in  $E^*$  (this follows by arithmetic from the assumptions that  $r \leq c$  and  $r + b \geq c$ ). We can again do an edge exchange to replace a  $(C, C)$  and, say, a  $(B, B)$  edge with two appropriate  $(C, B)$  edges, maintaining the size of  $E^*$ , and the property that every edge of *CG* is covered by an edge of  $E^*$ . We repeat until  $E^*$  contains no  $(R, R)$  or  $(C, C)$  edges, and

note that  $|E^*|$  has remained constant, and that  $E^*$  has throughout been disjoint from the edges of  $CG$ .

The set  $E^*$  is now a solution to the labeled augmentation problem. It is easy to see that it is optimal, since  $|E^*| = p/2 = (r + c + b)/2$ , the lower bound noted in Fact 2.

To analyze the time needed for the transformation, note that at most  $(r + c)/2$  edge exchanges are needed, and the cost of each is bounded by the number of edges in  $CG$ . Both  $(r + c)$  and the number of edges in  $CG$  are bounded by  $(n + m)$  and by  $|X|$ , hence  $O((n + m)|X|)$  time suffices. Note that this could be reduced to  $O(|X|)$  time if, in every edge exchange, both of the possible pairs of edge additions maintain the cover of the edges of  $S$ . However, as shown in the first case of Fig. 5, it is not true that both pairs of edge additions maintain the cover.  $\square$

*Labeled augmentation when  $CG$  is a forest.* In general,  $G$  may not be connected and so  $CG$  will not be a single tree. We now solve the labeled augmentation problem in the case that  $CG$  is a forest. The idea is that we first add a set of node disjoint edges (where no node is incident with more than one edge in the set) between leaves to connect the trees of  $CG$  into a single tree,  $T^*$ , and then solve the problem as before. We must be careful in the way that we add edges to form  $T^*$ . We call the first edge additions (forming  $T^*$ ) phase one, and the next edge additions (done by Algorithm ET) phase two, which is then followed by any needed edge exchanges. We let  $E^*$  now start with only those edges added in phase two. However, we must be careful about which edges are added in phase one, because when  $CG$  is a forest, Fact 3 is no longer true, i.e., it is not true that in  $CG^*$  every edge of  $CG$  is in some cycle containing *only one* edge of  $CG^* - CG$  (see Fig. 6).

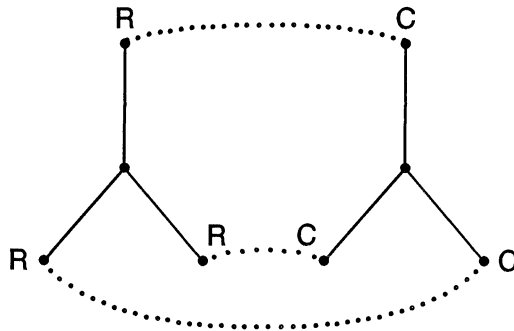


FIG. 6.  $CG$  is shown in solid and the optimal augmentation is dashed. No edge in  $CG$  is covered by just a single edge of the optimal solution.

Fact 3 was crucial in the proof that all edges remain traversable after an edge exchange, and the edge exchange operation and argument become much more complicated if we ever remove edges added in phase one. To get around this problem, we want to be sure that the edges added in phase one can appear together in an optimal solution to the labeled augmentation problem. If so, then these edges become fixed after phase one, and the remaining problem is exactly the labeled augmentation problem for the single tree  $T^*$ .

**THEOREM 4.** *The trees of  $CG$  can be connected (forming  $T^*$ ) by node disjoint, phase one, edges such that there exists an optimal solution of the labeled augmentation problem for forest  $CG$ , containing all of those phase one edges.*

*Proof.* We first arbitrarily relabel  $(b+c-r)/2$  of the  $B$  leaves in  $CG$  to be  $R$  leaves, and relabel the remaining  $(b+r-c)/2$   $B$  leaves to be  $C$  leaves. The effect is that  $CG$  now contains an equal number of  $R$  and  $C$  leaves and no  $B$  leaves. We now claim that it is possible to connect  $CG$  into a single tree,  $T^*$ , using a set of node disjoint  $(R, C)$  edges. In fact, this can be done myopically: as long as  $CG$  is not connected there must exist two components in  $CG$  such that one contains an  $R$  leaf and one contains a  $C$  leaf. To see this, suppose that  $CG$  is not connected, and let  $CC$  be a component containing an  $R$  leaf. Since each component is a tree, each component has a leaf, and if a component other than  $CC$  has a  $C$  leaf, the claim is proved. If there are no  $C$  leaves outside of  $CC$ , then all leaves outside of  $CC$  are  $R$  leaves, and one of these can be connected to a  $C$  leaf in  $CC$ . There must be such  $C$  leaves, since the number of  $R$  and  $C$  leaves are equal before any phase one edge additions and this equality is maintained with each edge addition.

After  $T^*$  is formed, the number of  $R$  and  $C$  leaves are still equal, hence the number of  $(R, R)$  edges added by Algorithm ET (in solving the unlabeled augmentation problem on  $T^*$ ) must equal the number of  $(C, C)$  edges added. Edge exchanges strictly between these (phase two) edges can then be done to form a legal solution to the labeled augmentation problem for  $CG$ . This solution contains all phase one edges, and is optimal since it adds exactly  $p/2$  edges. Hence there exists an optimal solution to the labeled augmentation problem for  $CG$ , containing all the phase one edges.  $\square$

There are other ways of connecting the trees in phase one, and it is often useful to have several choices available. This is particularly true for the case when  $D(i, j) > 0$  for the sensitive cells, but nonsensitive cells can be zero (see [G] for a more complete discussion of this).

*Labeled augmentation when  $r+b < c$ .* We briefly discuss the labeled augmentation problem in the case when  $r+b < c$ . We assume that  $r+b \geq 1$  since the complementary suppression problem is simple to solve directly when  $CG$  only has  $C$  leaves. When  $r+b < c$ , at least  $c$  edges must be added. If  $CG$  is a single tree, we can achieve this bound by taking the solution given by ET (which contains  $p/2$  edges) and make edge exchanges until the only edges in  $E^*$  are  $(R, C)$ ,  $(C, B)$  and  $(C, C)$  edges. This can always be done since  $r+b < c$ . At this point we can arbitrarily pick an  $R$  or  $B$  node,  $v$ , in  $CG$ , delete all  $(C, C)$  edges from  $E^*$  and attach each exposed  $C$  leaf to  $v$ . In the case that  $CG$  is a forest, a similar argument to those above shows a way to connect the forest with phase one edges which have the property that they can be together in an optimal solution. As before, the problem then reduces to the case of a single tree. Unlike the previous case, however, the phase one edges need not all be node disjoint, since even in the case of a single tree, as many as  $c-(r+b)$  edges may touch nodes that are touched by other edges of the augmentation.

Figure 7 shows a complete example of complementary suppression when  $D$  is strictly positive. Since cycles in  $H$  are traversable when  $D$  is strictly positive, we only indicate which cells have been suppressed, and can ignore all numerical values.

*Labeled augmentation and necessary conditions for protection.* Cox in [COX80] discusses the utility of using simple necessary conditions for protection in heuristic methods for complementary suppression. In particular, he discusses the necessary condition that every row or column containing a sensitive cell must contain at least two suppressed cells. This is the condition that  $H$  contains no leaves. A stronger necessary condition that subsumes this is that  $H$  contains no bridges, i.e., that every edge in  $H$  is in a simple cycle (ignoring the directions of the edges). When  $D$  is strictly positive, every cycle in  $H$  is traversable, and so the necessary condition is also sufficient. In the general case this is not true, but the above labeled augmentation methodology

D

×	×	×	×				
			×				
				×	×		
				×	×	×	
						×	×
						×	×

FIG 7(a). Example of complementary suppression.  $D$  is assumed strictly positive; the cells with  $x$ 's are suppressed.

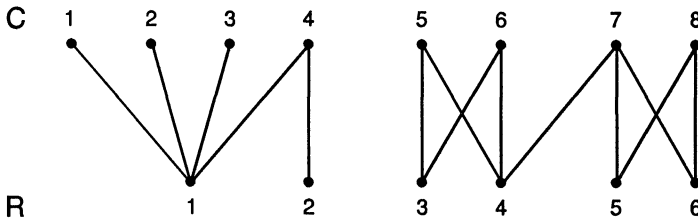


FIG. 7(b). Graph  $H$  derived from  $D$ .

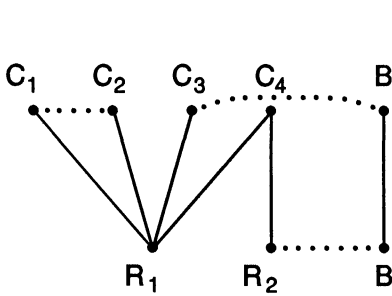


FIG. 7(c). Graph  $CG$  derived from  $H$  with phase one and two edges (dashed) added. Edge  $(C_3, B)$  is the phase one edge.

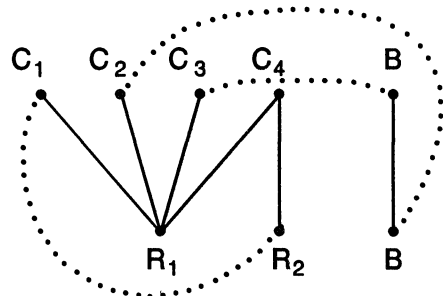


FIG. 7(d). Solution to the labeled augmentation problem after edge exchange.

D

×	×	×	×				
○			×				
	○			×	×		
				×	×	×	
						×	×
		○				×	×

FIG. 7(e). Table  $D$  with complementary suppressions added (shown as circles).

can still be used to add the fewest number of complementary suppressions to satisfy the necessary condition that every edge in  $H$  be in a simple cycle (ignoring directions). Hence this stronger necessary condition can be efficiently and optimally imposed, and therefore may be useful in practice.

*Special case:  $X$  strictly positive.* Above we solved the complementary suppression problem when  $D$  is assumed to be strictly positive. A less restrictive and more useful assumption is that the original sensitive cells are strictly positive (no zero valued data is sensitive), but other cells in  $D$  may have zero value. We do not know how to solve this problem efficiently, but under the assumption that the zero valued cells are not extremely dense, the above methodology may often be used to efficiently produce optimal solutions, or near optimal solutions. Full details on this problem are contained in [G1].

**4. Interval estimation: tightest general upper and lower bounds.**

DEFINITION. If cell  $(i, j)$  is suppressed in  $D$  then the tightest upper bound on its value is the largest value that  $x(i, j)$  can take on in any legal solution of  $D$ . Similarly the tightest lower bound on its value is the smallest value that  $x(i, j)$  can take on in a legal solution of  $D$ .

In this section, we first show how to compute each tightest bound with a single network flow computation. This improves previous methods (cited in the introduction) which were based on *minimum cost* network flow or linear programming. Each network flow computation takes  $O(n^3)$  time, hence, in an  $n$  by  $n$  table if  $\Theta(n^2)$  bounds need to be computed, the best time guarantee based on this method would be  $O(n^5)$ . We next show that this time bound can be reduced to  $O(n^4)$ , and that  $O(n \log n)$  maximum flow computations suffice to find all the tightest upper bound values in any table. A biproduct is that in any table (even those with  $\Theta(n^2)$  missing values), there are never more than  $2n - 1$  distinct upper bound values. These results are obtained by reducing the problem of computing tightest upper bounds to a problem discussed by Schnorr [SC]. The problem of computing all the lower bounds is more directly seen to take only  $O(n^4)$  time.

**4.1. Computing a single bound.**

DEFINITION. Given the complete table  $D$ , construct the graph  $G(H)$  from  $H$  and  $D$  as follows: replace every edge in  $H$  with two edges between the same nodes, with one edge in each direction; for each edge  $\langle i, j \rangle$  from  $R$  to  $C$ , set the capacity of  $\langle i, j \rangle$  to  $M$ , a finite number larger than the largest  $R^*(i)$ ; for each edge  $\langle j, i \rangle$  from  $C$  to  $R$ , set the capacity of  $\langle j, i \rangle$  to  $D(i, j)$ .

Graph  $G(H)$  can be thought of as the augmentation graph [FF] of the flow in network  $G$  given by the original values in  $D$ .

DEFINITION. Let  $FG(i, j)$  be the value of the maximum flow from node  $i$  to node  $j$  in  $G(H)$ . Note that  $FG(i, j)$  is not just the flow on the single edge  $\langle i, j \rangle$ .

THEOREM 5. If  $(i^*, j^*)$  is a suppressed cell in  $D$ , with  $i^* \in R$  and  $j^* \in C$ , then  $FG(j^*, i^*)$  is the tightest upper bound on the value of cell  $(i^*, j^*)$  in  $D$ , and  $\max [0, D(i^*, j^*) - FG(i^*, j^*) + M]$  is the tightest lower bound on the value of cell  $(i^*, j^*)$  in  $D$ .

*Proof.* Let  $F$  be the maximum flow from  $j^*$  to  $i^*$  in  $G(H)$ , and recall that  $F(i, j)$  is the flow from node  $i$  to node  $j$  along the single edge  $\langle i, j \rangle$ .

We first show that  $F$  defines a legal solution  $x$  of  $D$  where  $x(i^*, j^*)$  has value  $FG(j^*, i^*)$ . Define  $x(i^*, j^*) = FG(j^*, i^*)$ , and for every other suppressed cell  $(i, j) \neq (i^*, j^*)$ ,  $i \in R$ ,  $j \in C$ , define  $x(i, j) = D(i, j) + F(i, j) - F(j, i)$ . Since  $F(j, i) \leq D(i, j)$ ,  $x(i, j) \geq 0$ . Now consider any node  $i$  in  $R$  other than  $i^*$ . Then with the above assignment,

the sum in row  $i$  is

$$\sum_{j \in X(i)} x(i, j) = \sum_{j \in X(i)} D(i, j) + \sum_{j \in X(i)} F(i, j) - \sum_{j \in X(i)} F(j, i).$$

But the flow into  $i$  must equal the flow out of  $i$ , so

$$\sum_{j \in X(i)} x(i, j) = \sum_{j \in X(i)} D(i, j) = R^*(i).$$

For  $i^*$ ,

$$\sum_{j \in X(i^*)} x(i^*, j) = FG(j^*, i^*) + \sum_{j \in X(i^*)-j^*} D(i^*, j) + \sum_{j \in X(i^*)-j^*} [F(i^*, j) - F(j, i^*)].$$

But  $\sum_{j \in X(i^*)-j^*} [F(i^*, j) - F(j, i^*)]$  is the negative of the net flow into  $i^*$  from all nodes other than  $j^*$ , hence it is  $-(FG(j^*, i^*) - D(i^*, j^*))$ . So

$$\begin{aligned} \sum_{j \in X(i^*)} x(i^*, j) &= FG(j^*, i^*) + \sum_{j \in X(i^*)} D(i^*, j) - FG(j^*, i^*) \\ &= \sum_{j \in X(i^*)} D(i^*, j) = R^*(i^*). \end{aligned}$$

The argument for any column is similar, so  $x$  is indeed a legal solution to  $D$ .

We now show that there is no legal solution where cell  $(i, j)$  has a value larger than  $FG(j^*, i^*)$ . Suppose there is a solution  $x$  such that  $x(i^*, j^*) > FG(j^*, i^*)$ . Consider the differences  $x(i, j) - D(i, j)$ , for each suppressed cell  $(i, j) \neq (i^*, j^*)$ ,  $i \in R, j \in C$ ; assign  $F'(i, j)$  to be  $x(i, j) - D(i, j)$  when the difference is positive, and  $F'(j, i)$  to be  $|x(i, j) - D(i, j)|$  when the difference is negative; and assign  $F'(j^*, i^*)$  to be  $D(i^*, j^*)$ . We first show that  $F'$  is a flow in  $G(H)$  from  $j^*$  to  $i^*$ . Clearly, all values in  $F'$  are positive, and no capacity constraints are exceeded. Now for any  $i \neq i^*, i \in R$ ,

$$\begin{aligned} \sum_{j \in X(i)} [F'(i, j) - F'(j, i)] &= \sum_{j \in X(i)} [x(i, j) - D(i, j)] \\ &= \sum_{j \in X(i)} x(i, j) - \sum_{j \in X(i)} D(i, j) = 0, \end{aligned}$$

since  $x$  and  $D$  are both legal solutions, and hence both terms add to  $R^*(i)$ . So  $F'$  is a flow in  $G(H)$ . To calculate its value, consider the net flow into  $i^*$ . That flow value is

$$\begin{aligned} D(i^*, j^*) + \sum_{j \in X(i^*)-j^*} [D(i^*, j) - x(i^*, j)] &= R^*(i^*) - \sum_{j \in X(i^*)-j^*} x(i^*, j) \\ &= x(i^*, j^*) > FG(j^*, i^*). \end{aligned}$$

This contradicts the assumption that  $F$  is the maximum flow from  $j^*$  to  $i^*$  in  $G(H)$ . Hence  $FG(j^*, i^*)$  is the tightest upper bound on the value of cell  $(i, j)$ .

The formal proof for the lower bound is very similar to that for the upper bound, and is omitted. The intuition behind the lower bound formula is that the flow from  $i^*$  to  $j^*$  is  $M$  plus the largest amount by which  $D(i^*, j^*)$  can be decreased while keeping the row and column sums correct. So to find the largest possible decrement, subtract  $M$  from  $FG(i^*, j^*)$ ; then to find the tightest lower bound, subtract the decrement from the initial value of the cell. The result will be the tightest lower bound, unless the decrement is so large that the result is negative. A negative result is not meaningful for our problem, and so in this case the tightest lower bound is zero.  $\square$

**4.2. Speeding up the computation of tightest bounds.** If each bound is computed independently as above, and there are  $\Theta(n^2)$  suppressed cells in an  $n$  by  $n$  table, the best implied time bound for finding all the bounds would be  $O(n^5)$ . However, there

is a great deal of interdependence between the bounds, and it is not necessary to compute each one independently. It is possible to exploit this interdependence to obtain an  $O(n^4)$  time method to compute all bounds in an  $n \times n$  table, no matter how many cells are suppressed. Another reflection of the interdependence of the bounds is that there can never be more than  $n + m - 1$  distinct upper bound values in any  $n \times m$  table, no matter how many cells are suppressed; similarly there are never more than  $n + m$  distinct lower bound values.

DEFINITION. For two nodes  $i$  and  $j$  in  $G(H)$  define  $\beta(i, j)$  as the minimum of the flow values  $FG(i, j)$  and  $FG(j, i)$ .

LEMMA 1. Let  $(i, j)$  be a suppressed cell in  $D$ . Then in  $G(H)$ ,  $FG(i, j) > FG(j, i)$ , where  $i \in R$ , and  $j \in C$ .

Proof. This is trivially true because there is a direct edge from  $i$  to  $j$  with large capacity  $M$ , hence  $FG(i, j) \cong M$ , while by Theorem 5,  $FG(j, i)$  is the tightest upper bound on the value of cell  $(i, j)$  which is certainly bounded by  $R^*(i) < M$ .  $\square$

Lemma 1 and Theorem 5 together imply the following.

LEMMA 2. For a suppressed cell  $(i, j)$  in  $D$ ,  $\beta(i, j)$ , computed on  $G(H)$ , is the tightest upper bound on the value of cell  $(i, j)$ .

The computation of tightest bounds involves many network flow computations on the unchanging graph  $G(H)$ , where the choice of source and sink varies in each computation. Hence, we would like an efficient method (significantly faster than computing each flow separately) to compute  $FG(i, j)$  for all pairs  $(i, j)$  in  $X$ . Such methods are known for undirected graphs [GH], [G2]; in those methods, all  $\Omega(k^2)$  flow values can be computed with only  $k - 1$  flows in a  $k$  node undirected graph. For directed graphs, no such equivalent method is known. However, if  $\beta(i, j)$  is defined to be the minimum of the flow values from  $i$  to  $j$ , and from  $j$  to  $i$ , then it is known how to compute the  $\Omega(k^2)$  values of  $\beta(i, j)$  for all pairs of nodes in a directed graph with  $k$  nodes, using at most  $O(k \log k)$  maximum flows. Further, the total time needed for all the flows together can be bounded by  $O(k^4)$  [SC]. We refer the reader to [SC], but note that these results follow from the structure and interdependence of the beta values. Graph  $G(H)$  has  $n + m$  nodes if the table is  $n \times m$ , hence given Lemma 2, we have the following.

THEOREM 6. All tightest upper bounds on the values of suppressed cells can be computed in  $O((n + m)^4)$  time in an  $n \times m$  table, and hence in  $O(n^4)$  in an  $n \times n$  table.

To compute the tightest lower bounds in  $O(n^4)$  time, note first that there can only be  $2n - 1$  nonzero lower bounds. This follows from viewing the network flow problem on  $G$  (the problem of finding a single legal solution to  $D$ ) as a linear programming problem, and noting that for such a linear programming problem, all but  $2n - 1$  variables are set to zero in any basic feasible solution [MUR]. Such a basic feasible solution can be constructed from any feasible solution in time  $O(n^4)$ , and hence all but  $2n - 1$  lower bounds of zero are identified in that time. The lower bounds for the remaining  $2n - 1$  cells can each be computed independently, in total time  $O(n^4)$ .

In [G1], we present a completely different network flow method for computing each individual bound. In that method, the graph is changed for each cell, so the method is not easily combined with Schnorr's beta method.

The number of distinct upper bound values. We have shown that for any suppressed cell  $(i, j)$  the tightest upper bound on the value of cell  $(i, j)$  is equal to  $\beta(i, j) = \min [FG(i, j), FG(j, i)]$  in  $G(H)$ . It is known [SC] that there can never be more than  $k - 1$  distinct values for  $\beta(i, j)$  in any directed graph with  $k$  nodes. Since  $G(H)$  has  $n + m$  nodes, and the  $\beta$  values define the tightest upper bounds in  $D$ , there can never be more than  $n + m - 1$  distinct tightest upper bounds in  $D$ . This is easily verified on



the table where all cells are suppressed, but holds in general. The proof in [SC] is extracted from the details of the method to compute the betas; for a simpler proof, see [G1]. In summary, we have the following.

**THEOREM 7.** *In any  $n \times m$  table  $D$ , there are at most  $n + m - 1$  distinct tightest upper bounds on the values of the suppressed cells in  $D$ .*

**5. Tightest upper and lower bounds in a totally suppressed table.** There are applications of the estimation problem where all entries in the table are suppressed. Here we show that trivial methods suffice in this case to compute the tightest upper and lower bounds on the cell values.

**THEOREM 8.** *In a totally suppressed table the tightest upper bound on the value of cell  $(i, j)$  is  $\min [R(i), C(j)]$ .*

Before proving Theorem 8, we establish the following lemma.

**LEMMA 3.** *Let  $R(1), \dots, R(n)$  and  $C(1), \dots, C(m)$  be any nonnegative integers such that  $\sum_{i=1}^n R(i) = \sum_{j=1}^m C(j)$ . Then there exists a legal solution to the table  $D$  where all cells are suppressed and  $R(i)$  is the  $i$ th row total and  $C(j)$  is the  $j$ th column total in  $D$ .*

*Proof.* In the network flow interpretation of table  $D$  above, graph  $H$  is a complete bipartite graph, and so in  $G$  any  $s-t$  cut must contain either all edges incident with  $s$ , or all edges incident with  $t$ . Hence the minimum cut and maximum flow have value  $\sum_{i=1}^n R(i)$ , and there is a legal solution to  $D$ .  $\square$

*Proof of Theorem 8.* Clearly  $\min [R(i), C(j)]$  is an upper bound on the value of cell  $(i, j)$ . Assume  $\min [R(i), C(j)] = R(i)$ . To show that the bound can be met we must exhibit a solution to  $D$  in which  $(i, j)$  is given value  $R(i)$ . We simply give cell  $(i, j)$  the value  $R(i)$ , and give all other cells in row  $i$  the value 0; we then remove row  $i$  from  $D$  and set  $C(j)$  to  $C(j) - R(i)$ . The row totals still equal the column totals in the reduced table, and so by Lemma 3, the reduced table has a solution, and hence  $\min [R(i), C(j)]$  is an attainable value for  $(i, j)$ .  $\square$

Now we examine the lower bound.

**THEOREM 9.** *In the totally suppressed table the tightest lower bound on the value of cell  $(i, j)$  is  $\max [0, R(i) + C(j) - T]$ , where  $T = \sum_{i=1}^n R(i)$ .*

*Proof.* We first show that the above is a lower bound. The cell values in row  $i$  must add up to  $R(i)$ , but without cell  $(i, j)$ , the total that all other cells in row  $i$  can contribute is  $T - C(j)$ . Hence cell  $(i, j)$  must have value at least  $R(i) + C(j) - T$ . The same lower bound is obtained by doing the analysis along column  $j$ . To show that this bound is tight, give cell  $(i, j)$  the value  $\max [0, R(i) + C(j) - T]$ , and assign values to the other cells in row  $i$  so that the total assigned in row  $i$  is exactly  $R(i)$  (this is always possible). After deleting row  $i$  and reducing each  $C(k)$  by the amount assigned to cell  $(i, k)$ , the row and column totals are still equal, and so, by Lemma 3, the reduced table has a solution, and the theorem is proved.  $\square$

**COROLLARY 1.** *In a totally suppressed table of size  $n \times m$ , let  $C(k)$  be the largest row or column total. Then only cells in column  $k$  can have a nonzero lower bound. Further, cells in column  $k$  can have a nonzero lower bound only if  $C(k) > T/2$ .*

*Proof.*  $\max [0, R(i) + C(j) - T] = 0$  unless  $R(i) + C(j) > T$ , which can happen only if either  $R(i) > T/2$  or  $C(j) > T/2$ , in which case  $C(k) > T/2$  also. Now for any  $j \neq k$ ,  $C(j) \leq T - C(k)$ , and so  $R(i) + C(j) - T \leq R(i) + [T - C(k)] - T = R(i) - C(k) \leq 0$ . So only cells in column  $k$  can possibly have a nonzero lower bound.  $\square$

A symmetric statement holds when the largest row or column sum is a row sum. Hence we need to compute lower bounds only for the cells in the row or column with maximum sum; all other cells have lower bounds of zero. Therefore  $O(\max [n, m])$

arithmetic operations and comparisons suffice in this case, although the output is of size  $O(nm)$ .

**COROLLARY 2.** Any  $n \times m$  totally suppressed table is protected, for  $n, m > 1$ .

*Proof.* Cell  $(i, j)$  is protected if  $\min [R(i), C(j)] > \max [0, R(i) + C(j) - T]$ . We assumed that every row and column sum was nonzero, so  $(i, j)$  is protected if  $\min [R(i), C(j)] > R(i) + C(j) - T$ . Assume that  $R(i) \leq C(j)$ . If  $R(i) = R(i) + C(j) - T$  then  $C(j) = T$ , but when  $n > 1$  this is a contradiction to the assumption that all column sums are nonzero.  $\square$

**6. Extensions and open questions.** As mentioned in the introduction, the Census Bureau and Adversary problems can also be efficiently solved when there are known independent upper and lower bounds on the value of each suppressed cell. In that case, graph  $H$  is constructed as before, but for cell  $(i, j)$ , the  $(i, j)$  edge in  $H$  is directed from  $i$  to  $j$  if  $D(i, j)$  is at its lower bound, from  $j$  to  $i$  if  $D(i, j)$  is at its upper bound, and undirected otherwise. Cell  $(i, j)$  is then protected if and only if edge  $(i, j)$  is in a traversable cycle in  $H$ , and the linear time algorithm to find the traversable edges presented in this paper works for this case as well.

Several of the ideas in this paper have been extended in nontrivial ways, and appear in [K], [KG]. In particular, one may ask whether a given linear function of a set of suppressed cells is invariant over all the legal solutions of  $D$ . In this paper we essentially considered the simplest linear function, that of a single cell value. In [KG] it is shown that given any linear function of a set of suppressed cells in  $D$ , one can determine in linear time in  $|X|$  if that function is an invariant.

We mention a few of the many interesting open questions that remain.

(1) Generalize the approach in this paper to tables of three or more dimensions. Even the totally suppressed case is open for three dimensions. For example, the three-dimensional analogues of Theorems 8 and 9 do not even hold (see Fig. 8).

u 1	1	0
L 0	1, 0 u L	0, 0 u L
u 5	5	1
L 1	4, 1 u L	1, 0 u L
	u 5	1
	L 1	0

**FIG. 8.** Three-dimensional table of size  $2 \times 2 \times 2$ . The two numbers in each cell below the triangle are the upper and lower table entries respectively, i.e., cell  $(1, 1, 1)$  has entry 1, cell  $(2, 1, 1)$  has value 4, cell  $(2, 1, 2)$  has value 1, etc. The numbers in the triangles are the sums of the cells looking downward. The numbers on the left side are upper and lower sums of the cells looking horizontally along a line, and the numbers on the bottom are upper and lower sums of the cells looking vertically along a line. The minimum of the sums constraining cell  $(2, 1, 1)$  is 5, and so the "analogue" of Theorem 8 suggests that in the totally suppressed table with these sums  $(2, 1, 1)$  could have value 5. But this is not possible since then cell  $(2, 2, 1)$  would be forced to be zero and then cell  $(1, 2, 1)$  would be forced to be 1, which contradicts the fact that cell  $(1, 2, 1)$  is forced to be zero by the zero in the triangle.

(2) In this paper a cell was defined to be protected if the tightest upper and lower bounds differ by at least one. A more useful condition is that they should differ by some  $\delta > 1$ . Can we efficiently (in linear time) determine whether  $D$  is protected for a given  $\delta > 1$  specified ahead of time? What about for  $\delta$  given as input? What about the complementary suppression problem in these cases?

(3) It may be realistic to assume bounds on the sums of a set of cell values, or correlations between the values of certain cells. How can these be handled in the protection, suppression, and estimation problems?

(4) Can the general tightest upper bounds be computed faster than  $O(n^4)$  time? There is an easy reduction from the maximum flow problem to the problem of computing a single bound, hence  $O(n^3)$  seems the best realistic target for the time needed to compute all the bounds. We conjecture that this bound is attainable.

**Acknowledgments.** I would like to thank everyone in the Yale theory group and Dick Karp for listening to parts of this work as it evolved. Special thanks to Gregory Sullivan for reading early drafts of this paper.

#### REFERENCES

- [AHU] A. AHO, J. HOPCROFT AND J. ULLMAN, *Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [AO] R. AHUJA AND J. ORLIN, *A simple  $O(nm + n^2 \log c_{\max})$  sequential algorithm for the maximum flow problem*, Massachusetts Inst. of Technology, Cambridge, MA, unpublished manuscript, 1986.
- [BCS] G. BRACKSTONE, L. CHAPMAN, AND G. SANDE, *Protecting the confidentiality of individual statistical records in Canada*, Statistics Canada, March 1983.
- [COX75] L. COX, *Disclosure analysis and cell suppression*, Proc. American Statistical Association, Social Statistics Section, 1975, pp. 380-382.
- [COX77] ———, *Suppression methodology in statistical disclosure*, Proc. American Statistical Assoc., Social Statistics Section, 1977, pp. 750-755.
- [COX78] ———, *Automated statistical disclosure control*, Proc. American Statistical Association, Survey Research Methods Section, 1978, pp. 177-182.
- [COX80] ———, *Suppression methodology and statistical disclosure control*, J. Amer. Statist. Assoc., 75 (1980), pp. 377-385.
- [DEN] D. DENNING, *Cryptography and Data Security*, Addison-Wesley, Reading, MA, 1982.
- [ET] K. ESWARAN AND R. E. TARJAN, *Augmentation problems*, this Journal, 5 (1976), pp. 653-665.
- [EVE] S. EVEN, *Graph Algorithms*, Computer Science Press, Potomac, Maryland, 1979.
- [FEL] I. P. FELLEGI, *On the question of statistical confidentiality*, J. Amer. Statist. Assoc., 67 (1972), pp. 7-18.
- [FF] L. FORD AND D. FULKERSON, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.
- [G] D. GUSFIELD, *Optimal mixed graph augmentation*, Technical Report Yale/DCS/TR-327, Dept. Comput. Sci., Yale Univ., August 1984, this Journal, 16 (1987), pp. 599-612.
- [G1] ———, *A graph theoretic approach to statistical data security*, Technical Report Yale/DCS/TR-326, Dept. Comput. Sci., Yale Univ., August 1984.
- [G2] ———, *Very simple algorithms and programs for all pairs network flow analysis*, Technical Report ECS87-1, Computer Science Division, Univ. of California, Davis, April 1987.
- [GH] R. E. GOMORY AND T. C. HU, *Multi-terminal network flows*, SIAM J. Appl. Math., 9 (1961), pp. 551-570.
- [GMB] D. GUSFIELD, C. MARTEL, AND D. BACA-FERNANDEZ, *Fast Algorithms for Bipartite Network Flow*, this Journal, 16 (1987), pp. 237-251.
- [GT] A. GOLDBERG AND R. E. TARJAN, *A new approach to the maximum flow problem*, Proc. 18th Annual ACM Symposium on Theory of Computing, 1986.
- [K] M. KAO, *Systematic protection of precise information on two dimensional cross tabulated tables*, Ph.D. dissertation, Yale University, New Haven, CT, 1986.
- [KG] M. KAO AND D. GUSFIELD, *Detection and protection of leaked information in cross-tabulation*, extended abstract, January 1986.

- [MUR] K. MURTY, *Linear and combinatorial programming*, John Wiley, New York, 1976.
- [SAN] G. SANDE, *Automated cell suppression to preserve confidentiality of business statistics*, Statist. J. United Nations ECE 2 (1984) pp. 33-41.
- [SC] C. P. SCHNORR, *Bottlenecks and edge connectivity in unsymmetrical networks*, this Journal, 8 (1979), pp. 265-274.
- [ST] D. SLEATOR AND R. E. TARJAN, *A data structure for dynamic trees*, J. Comput. System Sci., 26 (1983), pp. 362-391.
- [USDC] U.S. DEPARTMENT OF COMMERCE (1978), *Report on statistical disclosure and disclosure-avoidance techniques*, Statistical policy working paper 2, U.S. Government Printing Office, Washington, D.C.

## MINIMUM SPANNING TREES IN $k$ -DIMENSIONAL SPACE\*

PRAVIN M. VAIDYA†

**Abstract.** We study the problem of finding a minimum spanning tree in the complete graph on a set  $V$  of  $n$  points in  $k$ -dimensional space. The points are the vertices of this graph and the weight of an edge between two points is the distance between the points under some  $L_q$  metric. We give an  $O(\varepsilon^{-k} n \log n)$  algorithm for finding an approximate minimum spanning tree in such a graph; the weight of the approximate minimum spanning tree is guaranteed to be at most  $(1 + \varepsilon)$  times the weight of a minimum spanning tree. We also present an algorithm to find a minimum spanning tree in the complete graph on  $V$ . Under the assumption that  $V$  consists of  $n$  random points, independently and uniformly distributed in the unit  $k$ -cube  $[0, 1]^k$ , the expected running time of this minimum spanning tree algorithm is shown to be  $O(n\alpha(cn, n))$  where  $c$  is a constant dependent on  $k$  and  $\alpha$  is the inverse Ackermann function.

**Key words.** minimum spanning trees, approximation algorithms

**AMS(MOS) subject classifications.** 68Q20, 68Q25, 68U05

**1. Introduction.** Given an undirected graph with a weight assigned to each edge, a spanning tree is a connected acyclic subgraph, and a minimum spanning tree (MST) is a spanning tree whose edges have a minimum total weight among all spanning trees. The classical algorithms for finding an MST were given by Dijkstra [4], Kruskal [9], Prim [10], and Sollin [1]. It is well known that for a graph on  $n$  vertices, an MST may be found in  $O(n^2)$  time. For a graph with  $m$  edges and  $n$  vertices, it was shown by Yao [15] that an MST may be found in  $O(m \log \log n)$  time. Further results on MST's may be found in [6], [8].

We study the problem of finding an MST in the complete graph on a given set  $V$  of  $n$  points in  $k$ -dimensional space. The points are the vertices of this graph and the weight of an edge between any two points is the distance between the points under some distance metric  $L_q$ . Each point  $x$  is given as a vector  $(x_1, x_2, \dots, x_k)$ . The  $L_q$ ,  $q = 1, 2, \dots, \infty$ , distance between any two points  $x$  and  $y$  is given by  $(\sum_{i=1}^k |x_i - y_i|^q)^{1/q}$ . (Note that the  $L_\infty$  distance is given by  $\max_i |x_i - y_i|$ .) We assume that the dimension  $k$  and the distance metric  $L_q$  are fixed (so distance is to be always interpreted as  $L_q$  distance).

The problem of finding an MST on a set of points in  $k$ -dimensional space differs from the problem of finding an MST in a general weighted undirected graph in two respects. First, the input consists only of  $kn$  numbers, the edges and the edge weights being implicitly defined. Second, in many applications of MST on points in space, like clustering, pattern recognition [5], [17], and other geometric and statistical applications, a spanning tree whose weight is close to the weight of an MST would serve just as well. So it is useful and interesting to investigate if the geometric nature of the problem can be exploited to obtain fast algorithms for finding a spanning tree on points in space whose weight is minimum or close to minimum.

Shamos and Hoey were the first to utilize the geometric nature of this problem, and in [12] they give an  $O(n \log n)$  algorithm for  $n$  points in the plane ( $k = 2$ ) with Euclidian metric. In [16] Yao gives algorithms which construct an MST in time

---

\* Received by the editors August 18, 1986; accepted for publication April 17, 1987. The work of this author was supported by a fellowship from the Shell Foundation.

† Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801. Present address, AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

$O(n^{2-2^{-(k+1)}}(\log n)^{1-2^{-(k+1)}})$  for any fixed  $k \geq 3$ , and distance metrics  $L_q$ ,  $q = 1, 2, \infty$ . In [7],  $O(n(\log n)^{k-\theta})$ ,  $\theta \leq 2$ , algorithms are given for fixed  $k \geq 2$  and  $L_1, L_\infty$  distance metrics. An algorithm for finding a spanning tree, with weight at most  $(1 + \varepsilon)$  times minimum, for  $k = 3$  and  $L_2$  distance metric is given in [2] but the running time is dependent on the ratio of the maximum to the minimum distance between any two points. In [14], an  $O(n(\log n)^{k+1}\varepsilon^{-(k-1)})$  algorithm is given for finding a spanning tree with weight at most  $(1 + \varepsilon)$  times minimum, for  $\varepsilon > 0$ , and fixed  $k$  and fixed metric  $L_q$ .

We give two algorithms for fixed dimension  $k$  and fixed metric  $L_q$ . The first algorithm, given in § 3, runs in  $O(\varepsilon^{-k}n \log n)$  time and finds an approximate minimum spanning tree whose weight is at most  $(1 + \varepsilon)$  times the weight of an MST. The second algorithm, presented in § 4.1, always finds an MST. For  $n$  random points, independently and uniformly distributed in the unit  $k$ -cube  $[0, 1]^k$ , the expected running time of the second algorithm is shown to be  $O(n\alpha(cn, n))$ , where  $c$  is a constant dependent on  $k$  and  $\alpha$  is the inverse Ackermann function defined in [13]. As  $\alpha$  grows extremely slowly with  $n$ , the expected running time of the second algorithm is almost linear. We note that the constants in the running times of both the algorithms depend on the dimension  $k$ . The probabilistic analysis of the second algorithm is given in § 4.2.

As far as the model of computation is concerned, we assume that all access, arithmetic and comparison operations require constant time. We also assume that some form of indirect addressing is available so that the process of distributing  $n$  numbers into  $m$  buckets can be carried out in  $O(m + n)$  time.

Without loss of generality we assume that all the  $n$  points in the given set  $V$  are located in the unit  $k$ -cube  $[0, 1]^k$ . We let  $d(p, p')$  denote the distance between two points  $p$  and  $p'$ , and we let  $W_{\text{MST}}$  denote the weight (length) of an MST in the complete graph on  $V$ . A box is defined to be the product  $J_1 \times J_2 \times \dots \times J_k$  of  $k$  intervals, or alternatively the set of those points  $x = (x_1, x_2, \dots, x_k)$  such that  $x_i$  is in interval  $J_i$ ,  $i = 1, 2, \dots, k$ . A box is cubical if and only if all the  $k$  intervals defining it have the same length, and the size of a cubical box is the length of each of the  $k$  intervals defining it. For a set of points  $S$ , we let  $d_{\text{max}}(S)$  denote the greatest distance between two points in  $S$ . For sets of points  $S_1$  and  $S_2$ , we let  $d_{\text{min}}(S_1, S_2)$  and  $d_{\text{max}}(S_1, S_2)$ , respectively, denote the minimum and maximum distance between a point in  $S_1$  and a point in  $S_2$ .

**2. A brief overview.** In the approximate minimum spanning tree algorithm we first extract a sparse graph  $G = (V, E)$  from the given set of points  $V$ , and then find an MST in  $G$  using a standard procedure [6], [7], [11]. There are  $O(\varepsilon^{-k}n)$  edges in  $G$  and there is a spanning tree in  $G$  of weight at most  $(1 + \varepsilon)W_{\text{MST}}$ . We obtain the graph  $G$  as follows. Using a collection of grids the region containing the given points is divided into cubical boxes, each grid partitioning the region into boxes of identical size. In each box  $b$ , we select a representative point from among the points in  $V$  that are located in box  $b$ . An edge between the representative in box  $b$  and the representative in box  $b'$  is included in  $G$  if and only if  $b$  and  $b'$  are of identical dimensions and the minimum distance between  $b$  and  $b'$  is below a certain threshold. In addition to edges between representatives,  $G$  contains an adequate number of short edges (of length  $< \varepsilon W_{\text{MST}}/3n^2$ ) suitably chosen to ensure that  $G$  is connected. Now suppose  $V$  is partitioned into  $V_1$  and  $V_2$ , and  $(p_1, p_2)$ ,  $p_1 \in V_1$ ,  $p_2 \in V_2$ , is the unique edge in an MST from  $V_1$  to  $V_2$ . Then either there are points  $p'_1 \in V_1$ ,  $p'_2 \in V_2$ , such that  $(p'_1, p'_2)$  is in  $G$ ,  $p'_1$  and  $p'_2$  are representative points in boxes, and  $d(p'_1, p'_2) \leq (1 + \varepsilon)d(p_1, p_2)$ , or among the short edges in  $G$  there is a path from  $p_1$  to  $p_2$  of length at most  $\varepsilon W_{\text{MST}}/n^2$ . This guarantees the existence of a good spanning tree in  $G$ .

The procedure to extract  $G$  may be implemented in  $O(\varepsilon^{-k}n \log n)$  time, and as  $G$  contains  $O(\varepsilon^{-k}n)$  edges, an MST in  $G$  may be found in  $O(\varepsilon^{-k}n \log n)$  time using a standard procedure [6], [8], [11], [15]. So the overall running time of the approximate minimum spanning tree algorithm is  $O(\varepsilon^{-k}n \log n)$ .

In the exact minimum spanning tree algorithm we first extract a graph  $G' = (V, E')$  from the given set of points  $V$  such that  $G'$  always contains a spanning tree of weight  $W_{\text{MST}}$ . The graph  $G'$  is sparse with high probability, and can be extracted in  $O(|E'|)$  time. We then try to sort the edges in  $G'$  by weight, in  $O(|E'|)$  (linear) time, by running a fixed number of passes of radix (bucket) sort [11], with radix  $2^{\lceil \log_2 n \rceil}$ . If this approach fails to sort the edges in  $G'$ , we sort them in  $O(|E'| \log |E'|)$  time using a standard algorithm [11]. Once a sorted list of edges in  $G'$  is available, utilizing Kruskal's algorithm an MST in  $G'$  may be obtained in  $O(n\alpha(|E'|, n))$  time where  $\alpha$  is the inverse Ackermann function defined in [13]. Suppose the given set  $V$  consists of  $n$  random points, independently and uniformly distributed in the unit  $k$ -cube  $[0, 1]^k$ . Then the probability that  $G'$  has more than  $cn$  edges, where  $c$  is a constant dependent on the dimension  $k$ , is  $o(1/n^2)$ . Also, the probability that a fixed number of passes of radix sort, with radix  $2^{\lceil \log_2 n \rceil}$ , fail to sort the edges is  $o(1/n^2)$ . Then it follows that the expected running time of the minimum spanning tree algorithm is  $O(n\alpha(cn, n))$ .

$G'$  is obtained in a manner similar to  $G$  above. Using a collection of grids the unit  $k$ -cube  $[0, 1]^k$  is divided into cubical boxes, each grid partitioning the unit  $k$ -cube into boxes of identical size. Let  $b_1, b_2$ , be boxes of identical dimensions such that the minimum distance between  $b_1$  and  $b_2$  is above a certain threshold and below another threshold. We test an easy to compute condition such that (i) if the condition is false, then none of the edges between a point in  $b_1 \cap V$  and a point in  $b_2 \cap V$  can be included in an MST in the complete graph on  $V$ , whereas (ii) if the condition is true, then there is an empty region which does not contain a point in  $V$  and whose volume is greater than or equal to the volume of  $b_1$  or  $b_2$ . We include every edge between a point in  $b_1 \cap V$  and a point in  $b_2 \cap V$  in  $G'$  if and only if this condition holds. In addition, we also include most of the short edges (length  $< c'n^{-1/k}$ ,  $c'$  a constant) in  $G'$ . We thereby ensure that  $G'$  always contains an MST in the complete graph on  $V$ . If the number of edges in  $G'$  exceeds  $cn$ , then either there is a large region in the unit  $k$ -cube which does not contain a point in  $V$  or there is some region in the unit  $k$ -cube that contains a concentration of points in  $V$ , and under the assumption that the points in  $V$  are independently and uniformly distributed in the unit  $k$ -cube both these events occur with very low probability.

**3. Approximate minimum spanning tree algorithm.** The algorithm to find an approximate MST in the complete graph on the given set  $V$  of  $n$  points consists of two stages: in the first stage we extract a sparse graph  $G = (V, E)$  from the given set  $V$  of points, and in the second stage we use a standard procedure [6], [7], [11] to find a minimum spanning tree in the graph  $G$ . The graph  $G = (V, E)$  has the following properties.

(1)  $G$  contains a spanning tree whose weight (length) is at most  $(1 + \varepsilon + 1/n)$  times  $W_{\text{MST}}$ .

(2)  $|E| = O(\varepsilon^{-k}n)$ .

We shall require a few definitions before we can describe the algorithm to extract  $G$ . Let  $g_0$  be a smallest cube enclosing all the  $n$  points in  $V$ , and let  $L_0$  be the length of a side of  $g_0$ . Let  $g_i$  be a grid that partitions  $g_0$  into  $2^{ki}$  identical cubical boxes, and let  $\delta = \lceil (\log_2(24(\lceil \varepsilon^{-1} \rceil)^2 k^2 n^2)) \rceil$ . Let  $B_i$  denote the set of those boxes (cubes) in  $g_i$  which contain a point in  $V$ , and let  $B = \bigcup_{i=0}^{\delta} B_i$ . Let  $L_i = c_a \lceil \varepsilon^{-1} \rceil 2^{-i} L_0$ , where  $c_a = 6k^{1/q}$

for  $L_q$  metric,  $q = 1, 2, 3, \dots, \infty$ . We let  $r(b)$  denote the representative point in box  $b$ . We now give an algorithm to extract the graph  $G = (V, E)$  from the set of points  $V$ .

**ALGORITHM SPARSE-GRAPH.**

1. For each box  $b \in B$ , pick a representative point  $r(b)$  from among the points in  $b \cap V$ , and let  $R_i = \{r(b) : b \in B_i\}$ , for  $i = 1, \dots, \delta$ . We pick representatives so that  $R_1 \subseteq R_2 \subseteq \dots \subseteq R_i \subseteq \dots \subseteq R_\delta$ .
2. Let  $X = \bigcup_{i=1}^\delta X_i$  where

$$X_i = \{(r(b_1), r(b_2)) : b_1 \in B_i, b_2 \in B_i, d_{\min}(b_1, b_2) \leq L_i\}.$$

3. Let  $Y = \{(p, r(b)) : p \in b, b \in B_\delta\}$ .
  4. Let  $E = X \cup Y$ .
- end Sparse-Graph

The edges in  $X_i$  connect representatives in boxes  $b_1, b_2$ , in  $B_i$  such that  $d_{\min}(b_1, b_2)$  lies between two thresholds  $L_i$  and  $L_i/3$ . The edges in  $Y$  are small in length compared to  $W_{\text{MST}}$  and they ensure that  $G = (V, E)$  is connected.

We need some additional definitions before we can show that  $G$  has the desired properties. Let  $Z$  denote the set of all the edges in the complete graph on  $V$ , and let

$$Z_i = \{(p_1, p_2) : p_1 \in b_1, p_2 \in b_2, b_1 \in B_i, b_2 \in B_i, d_{\min}(b_1, b_2) \geq L_i/3\}.$$

The following two lemmas follow directly from the definitions.

**LEMMA 1.** For  $1 \leq i \leq \delta$ , if  $b_1 \in B_i, b_2 \in B_i$ , and  $d_{\min}(b_1, b_2) \geq L_i/3$  then  $d_{\max}(b_1) = d_{\max}(b_2) \leq (\varepsilon/2)d_{\min}(b_1, b_2)$  and  $d_{\max}(b_1, b_2) \leq (1 + \varepsilon)d_{\min}(b_1, b_2)$ .

**LEMMA 2.**  $Z_0 \subseteq Z_1 \subseteq \dots \subseteq Z_i \subseteq \dots \subseteq Z_\delta \subseteq Z$ .

**LEMMA 3.** For  $1 \leq i \leq \delta$ , if  $(p_1, p_2) \in (Z_i - Z_{i-1})$  then  $p_1, p_2$ , are located in boxes  $b_1, b_2$ , which satisfy  $b_1 \in B_i, b_2 \in B_i$ , and  $L_i/3 \leq d_{\min}(b_1, b_2) \leq L_i$ .

*Proof.* Suppose  $p_1, p_2$  are located in boxes  $b_1, b_2$ , in  $B_i$ , respectively. Since  $(p_1, p_2) \in Z_i, d_{\min}(b_1, b_2) \geq L_i/3$ . Let  $b'_1 \in B_{i-1}, b'_2 \in B_{i-1}$ , and let  $b_1 \subseteq b'_1, b_2 \subseteq b'_2$ . Since  $(p_1, p_2) \in (Z_i - Z_{i-1})$ , we have  $d_{\min}(b'_1, b'_2) \leq L_{i-1}/3$ . Then

$$\begin{aligned} d_{\min}(b_1, b_2) &\leq d_{\min}(b'_1, b'_2) + d_{\max}(b'_1)/2 + d_{\max}(b'_2)/2 \\ &\leq L_{i-1}/3 + k^{1/q} 2^{-(i-1)} L_0 \\ &\leq L_i. \end{aligned} \quad \square$$

We now show that  $G = (V, E)$  contains a spanning tree whose weight is at most  $(1 + \varepsilon + 1/n)W_{\text{MST}}$ . Let  $T$  be an MST in the complete graph on  $V$ . We shall give a function  $f: T \rightarrow 2^E$  such that the graph  $(V, \bigcup_{e \in T} f(e))$  is connected and the sum of the weights of the edges in  $\bigcup_{e \in T} f(e)$  is at most  $(1 + \varepsilon + 1/n)W_{\text{MST}}$ . The function  $f(e)$  is defined as follows.

- (1) If  $e \in T$  and  $e \in E$ , then we let  $f(e) = \{e\}$ .
- (2) Suppose  $e \in ((T - E) \cap (Z - Z_\delta))$ . Let  $e = (p_1, p_2)$ , and let  $b_1, b_2$  be the boxes in  $B_\delta$  which contain  $p_1, p_2$ , respectively. Then  $f(e) = \{(p_1, r(b_1)), (r(b_1), r(b_2)), (r(b_2), p_2)\}$  and  $f(e) \subseteq (Y \cup X_\delta)$ . The length of an edge in  $Y$  is at most  $\varepsilon W_{\text{MST}}/3n^2$  and  $d(r(b_1), r(b_2)) \leq \varepsilon W_{\text{MST}}/3n^2$ , so the sum of the lengths of the edges in  $f(e)$  is at most  $\varepsilon W_{\text{MST}}/n^2$ .
- (3) Let  $e \in ((T - E) \cap (Z_i - Z_{i-1}))$ ,  $i \leq \delta$ , and let  $b_1, b_2$  be the boxes in  $B_i$  which contain the endpoints  $p_1, p_2$ , of  $e$ . Then by Lemma 3,  $(r(b_1), r(b_2)) \in X_i$ , and hence  $(r(b_1), r(b_2)) \in E$ . We let  $f(e) = \{(r(b_1), r(b_2))\}$ . By Lemma 1,  $d(r(b_1), r(b_2)) \leq (1 + \varepsilon) d(p_1, p_2)$ .

It remains to be shown that the graph  $(V, \bigcup_{e \in T} f(e))$  is connected. Consider a partition of  $V$  into  $V_1$  and  $V_2$ . Let  $e = (p_1, p_2)$ , where  $p_1 \in V_1$  and  $p_2 \in V_2$ , be the unique edge in  $T$  from  $V_1$  to  $V_2$ .



(1) Suppose  $e \in (Z - Z_\delta)$ . Then  $f(e)$  defines a path from  $p_1$  to  $p_2$ .  
 (2) Suppose  $e \in (Z_i - Z_{i-1})$ ,  $i \leq \delta$ , and boxes  $b_1, b_2$  in  $B_i$  contain  $p_1, p_2$ , respectively. Then we must have that  $(b_1 \cap V) \subseteq V_1$  and  $(b_2 \cap V) \subseteq V_2$ . This is seen as follows. Assume that there is a point  $p'$  in  $b_1 \cap V_2$ . As  $d_{\min}(b_1, b_2) > d_{\max}(b_1)$ , replacing  $(p_1, p_2)$  by  $(p_1, p')$  in  $T$  gives a spanning tree on  $V$  of smaller length which cannot happen. So  $(b_1 \cap V) \subseteq V_1$ . It follows similarly that  $(b_2 \cap V) \subseteq V_2$ . Then the edge  $(r(b_1), r(b_2))$  connects  $V_1$  and  $V_2$ .

We now show that  $E$  contains a linear number of edges. We note that  $R_1 \subseteq R_2 \subseteq \dots \subseteq R_i \subseteq \dots \subseteq R_\delta$ . If there is an edge in  $X_i$  between two points in  $R_{i-1}$  then the same edge is also present in  $X_{i-1}$ , and hence every edge in  $X_i - X_{i-1}$  is incident on a representative point in the set of representatives  $R_i - R_{i-1}$ . There are at most  $O((2c_a \varepsilon^{-1})^k)$  edges in  $X_i$  incident on any point in  $V$ , and so we get

$$\begin{aligned} |X| &= \sum_{i=1}^{\delta} |X_i - \bigcup_{j=0}^{i-1} X_j| + |X_0| = O((2c_a \varepsilon^{-1})^k \sum_{i=1}^{\delta} (|R_i| - |R_{i-1}|)) \\ &= O((2c_a \varepsilon^{-1})^k |R_\delta|) = O((2c_a \varepsilon^{-1})^k n). \end{aligned}$$

Then since  $|Y| \leq n$ , and  $E = X \cup Y$  we get that  $|E| = O((2c_a \varepsilon^{-1})^k n) = O(\varepsilon^{-k} n)$  for fixed  $k$ .

To obtain a fast implementation of *Algorithm Sparse-Graph* we construct a data structure which is best described as a *tree-of-boxes*.

(1) The root of the tree is the box  $g_0$ , and the children of each box  $b$  in  $B_i$  are those boxes in  $B_{i+1}$  which are sub-boxes of  $b$ .

(2) The leaf boxes are the boxes in  $B_\delta$ , and each leaf box contains a list of points in  $V$  that are in the box.

(3) The boxes at each level  $i$ , i.e., the boxes in  $B_i$  are linked together in a doubly linked list.

(4) From each box  $b$  in  $B_i$  there are pointers to (i) its father in  $B_{i-1}$ , (ii) its sons in  $B_{i+1}$ , (iii) each box  $b'$  in  $B_i$  satisfying  $d_{\min}(b, b') \leq L_i$ , and (iv) the leftmost leaf box in the subtree rooted at box  $b$ .

The *tree-of-boxes* has  $O(\log n)$  levels and at most  $n$  boxes per level. For each box  $b \in B_i$ , there are at most  $O((2c_a \varepsilon^{-1})^k)$  boxes  $b'$  in  $B_i$  such that  $d_{\min}(b, b') \leq L_i$ . So the *tree-of-boxes* requires  $O((2c_a \varepsilon^{-1})^k n \log n)$  storage, and can be constructed in  $O((2c_a \varepsilon^{-1})^k n \log n)$  time by starting from the root and proceeding towards the leaves level by level.

Once the *tree-of-boxes* is available, the representatives in boxes may be chosen in time proportional to  $|B| = O(n \log n)$ . For boxes  $b_1$  and  $b_2$ , we can find points  $p_1 \in b_1, p_2 \in b_2$ , such that  $d(p_1, p_2) = d_{\min}(b_1, b_2)$  in  $O(k)$  time. Utilizing the *tree-of-boxes*, each of the edge sets  $X_i$  can be extracted in  $O(k(2c_a \varepsilon^{-1})^k n)$  time, and so  $G$  may be extracted in  $O(k(2c_a \varepsilon^{-1})^k n \log n)$  time.

Once we have the sparse graph  $G = (V, E)$  one of the standard algorithms in [6], [8], [11], [15] may be used to find an MST in  $G$ . We thus have an algorithm for finding an approximate minimum spanning tree on a set  $V$  of  $n$  points in  $k$ -dimensional space. The running time is  $O(\varepsilon^{-k} n \log n)$  for fixed  $k$ , and the weight of the approximate minimum spanning tree obtained is at most  $(1 + \varepsilon)$  times the weight of a minimum spanning tree.

#### 4. MST algorithm with almost linear expected running time.

**4.1. Description.** Like the approximate minimum spanning tree algorithm in the previous section, the algorithm to find a minimum spanning tree in the complete graph on  $V$  also consists of two stages. In the first stage we extract a graph  $G' = (V, E')$

which is guaranteed to contain a spanning tree of weight  $W_{\text{MST}}$ , and in the second stage we find a minimum spanning tree in  $G'$ . Unlike the graph  $G$  in the previous section,  $G'$  is not necessarily sparse; however,  $G'$  is sparse with high probability under the assumption that the given points are uniformly and independently distributed in  $[0, 1]^k$ .

Let  $g_0$  be the unit  $k$ -cube  $[0, 1]^k$  and let  $g_i$  be a grid that divides  $g_0$  into  $2^{ki}$  identical cubical boxes. Let  $B_i$  be the set of all the boxes in  $g_i$  (rather than just the set of occupied boxes) and let  $B = \bigcup_{i=0}^{\delta} B_i$ . Define  $\delta$  to be  $\lceil \log_2 n/k \rceil$ . Let  $L_i = c_e 2^{-i}$  where  $c_e = 12.1k^{1/q}$  for  $L_q$  metric,  $q = 1, 2, 3, \dots, \infty$ . Let  $Z$  be the set of all the edges in the complete graph on  $V$  and let the edge sets  $Z_i$  be defined as in § 3. We note that Lemma 2 and Lemma 3 in § 3 are still valid under these definitions.

For a box  $b$ , let  $\mu(b)$  denote the singleton set containing the center of box  $b$ . For a pair of boxes  $b_1, b_2$ , we define  $\pi(b_1, b_2, i)$  to be the union of those boxes  $b_j$  which are such that

- (i)  $b_j \in B_i$ ,
- (ii)  $d_{\max}(b_j, \mu(b_1)) < d_{\min}(b_1, b_2) - d_{\max}(b_1, \mu(b_1))$ , and
- (iii)  $d_{\max}(b_j, \mu(b_2)) < d_{\min}(b_1, b_2) - d_{\max}(b_2, \mu(b_2))$ .

We now give an algorithm to extract  $G'$ .

ALGORITHM PROBABLY-SPARSE-GRAPH.

1. Let  $C = \bigcup_{i=1}^{\delta} C_i$  where

$$C_i = \{(p_1, p_2): p_1 \in b_1, p_2 \in b_2, b_1 \in B_i, b_2 \in B_i, L_i/3 \leq d_{\min}(b_1, b_2) \leq L_i, \pi(b_1, b_2, i) \cap V = \emptyset\}.$$

2. Let  $D = \{(p_1, p_2): p_1 \in b_1, p_2 \in b_2, b_1 \in B_{\delta}, b_2 \in B_{\delta}, d_{\min}(b_1, b_2) \leq L_{\delta}\}$ .
3. Let  $E' = C \cup D$ .

end Probably-Sparse-Graph

We have to show that  $G' = (V, E')$  contains a spanning tree which is an MST in the complete graph on  $V$ . Let  $T$  be an MST in the complete graph on  $V$  and let  $e = (p_1, p_2)$  be an edge in  $T$ .

- (1) Suppose  $e \in (Z - Z_{\delta})$ . We have that  $(Z - Z_{\delta}) \subseteq D$  and so  $e \in D$ .

(2) Suppose  $e \in (Z_i - Z_{i-1})$ ,  $i \leq \delta$ , and  $p_1, p_2$  are located in boxes  $b_1, b_2$  in  $B_i$ . We shall show that  $\pi(b_1, b_2, i)$  is nonempty, so the condition  $\pi(b_1, b_2, i) \cap V = \emptyset$  is not vacuously true. By Lemma 3 in § 3,  $L_i/3 \leq d_{\min}(b_1, b_2) \leq L_i$ . Let  $\hat{p}$  be the center of the line segment joining the centers of boxes  $b_1$  and  $b_2$ . It is easily shown that there is a cubical box  $\hat{b}$  of size  $2^{-(i-1)}$  centered at  $\hat{p}$ , such that  $d_{\max}(\hat{b}, \mu(b_1)) < d_{\min}(b_1, b_2) - d_{\max}(b_1, \mu(b_1))$ , and  $d_{\max}(\hat{b}, \mu(b_2)) < d_{\min}(b_1, b_2) - d_{\max}(b_2, \mu(b_2))$ . As  $\hat{b}$  must contain a box in  $B_i$  we conclude that  $\pi(b_1, b_2, i)$  is nonempty. Since  $(p_1, p_2) \in T$ , for each point  $p$  in  $V$  either  $d(p, p_1) \geq d(p_1, p_2)$  or  $d(p, p_2) \geq d(p_1, p_2)$ , and it then follows that  $\pi(b_1, b_2, i) \cap V$  must be empty. So  $e \in C_i$ . Hence  $T \subseteq C \cup D \subseteq E'$ .

*Algorithm Probably-Sparse-Graph* can be implemented in  $O(|E'| + k(2c_e)^k n)$  time by using a *tree-of-boxes*, similar to the one in § 3 for the boxes in  $B$ . We first build a skeleton for the *tree-of-boxes*. The skeleton differs from the *tree-of-boxes* described in § 3 in that there are no points in  $V$  stored at any of the leaf boxes. The skeleton has  $O(n)$  boxes and can be constructed in  $O((2c_e)^k n)$  time. Once the skeleton is available, we utilize a radix (bucket) sort [11], with radix  $2^{k\delta}$  ( $= O(n)$ ), to distribute the  $n$  points into the  $2^{k\delta}$  leaf boxes in  $O(n)$  time. We note that at this stage the *tree-of-boxes* contains all the boxes in  $B$  irrespective of whether they do or do not contain a point in  $V$ . Then in  $O(n)$  time we delete from the *tree-of-boxes* all those boxes which do not contain a

point in  $V$ , and add to each occupied box  $b$  a pointer to the leftmost leafbox in the subtree rooted at box  $b$ .

Utilizing the *tree-of-boxes*, we can check whether  $\pi(b_1, b_2, i) \cap V$  is empty in  $O(k(2c_e)^k)$  time if  $b_1 \in B_i, b_2 \in B_i$ , and  $d_{\min}(b_i, b_2) \leq L_i$ . The edge set  $C_i$  may be extracted in  $O(|C_i| + k(2c_e)^k n)$  time, for  $i = 1, \dots, \delta$ , and  $D$  may be obtained in  $O(|D| + k(2c_e)^k n)$  time. So *Algorithm Probably-Sparse-Graph* may be implemented in  $O(|E'| + k(2c_e)^k n)$  time.

We shall now describe a procedure for finding an MST in the graph  $G' = (V, E')$ . We first try to sort the edges in  $G'$  by employing a fixed number of passes of radix (bucket) sort [11], with radix (base)  $2^{\lceil \log_2 n \rceil}$ . If this approach fails to sort the edges in  $G'$  we sort them in  $O(|E'| \log |E'|)$  time using a standard algorithm [11]. After sorting the edges, we utilize Kruskal's algorithm [9], [11] to find an MST in  $G'$ . The edges are examined in increasing order of weight and an edge is chosen if and only if it does not form a cycle with the previously chosen edges. The chosen edges form an MST in  $G'$ . Once a sorted list of edges is available, Kruskal's algorithm may be implemented in  $O(|E'| \alpha(|E'|, n))$  time, where  $\alpha$  is the inverse Ackermann function defined in [13].

The algorithm to find an MST in the complete graph on  $V$  consists of first extracting  $G'$  using *Algorithm Probably-Sparse-Graph*, and then finding an MST in  $G'$  using the above described procedure. Let us assume that  $V$  consists of  $n$  random points, independently and uniformly distributed in the unit  $k$ -cube  $[0, 1]^k$ . Let  $P_L$  be the probability that for some  $i, 0 \leq i \leq kn^7 - 1$ , the lengths of at least two edges lie in the interval  $[in^{-7}, (i+1)n^{-7}]$ . In § 4.2 we show that  $P_L$  is  $o(1/n^2)$ . So the probability that eight passes of radix sort, with radix  $2^{\lceil \log_2 n \rceil}$ , fail to sort the edges in  $G'$  is  $o(1/n^2)$ ; thus the probability that the edges in  $G'$  cannot be sorted in  $O(|E'|)$  time, using a fixed number of passes of radix sort, is  $o(1/n^2)$ . Let  $P_{G'}$  be the probability that  $G'$  contains more than  $cn$  edges, where  $c$  is a constant dependent on the dimension  $k$ . In § 4.2 we also show that  $P_{G'}$  is  $o(1/n^2)$ . So with probability  $1 - o(1/n^2)$ , the running time of the algorithm to find an MST in the complete graph on  $V$  is  $O(cn\alpha(cn, n) + k(2c_e)^k n)$ . The worst case running time of the same algorithm is  $O(n^2 \log n)$ . Therefore, when  $k$  is fixed, the expected running time of the algorithm for finding an MST in the complete graph on  $V$  is  $O(n\alpha(cn, n))$ .

**4.2. Probabilistic analysis.** We assume that the set  $V$  consists of  $n$  random points which are independently and uniformly distributed in the unit  $k$ -cube  $[0, 1]^k$ . Let  $P_L$  be the probability that for some  $i, 0 \leq i \leq kn^7 - 1$ , the lengths of at least two edges lie in the interval  $[in^{-7}, (i+1)n^{-7}]$ . Let  $P_{G'}$  be the probability that  $G'$  contains more than  $cn$  edges, where  $c$  is a constant dependent on the dimension  $k$ . We have to show that both  $P_{G'}$  and  $P_L$  are  $o(1/n^2)$ .

We shall first bound  $P_L$ . Fix an ordering on the points in the unit  $k$ -cube. Assume that there are at least two edges  $e_1$  and  $e_2$  whose lengths lie in the interval  $[in^{-7}, (i+1)n^{-7}]$ , for some  $i$ . Suppose  $e_1, e_2$  have an endpoint in common and  $e_1 = (x, y), e_2 = (y, z)$ . Then the points  $x$  and  $z$  are restricted to lie in a shell, of thickness  $n^{-7}$  and radius at most  $k$ , around point  $y$ , and there are at most  $n^3$  possibilities for triples  $(x, y, z)$ . Suppose  $e_1, e_2$  do not have an endpoint in common and  $e_1 = (x, y), e_2 = (z, w)$ . Without loss of generality let  $x \leq y$  and  $z \leq w$ . Then  $x$  is restricted to lie in a shell, of thickness  $n^{-7}$  and radius at most  $k$ , around  $y$ ;  $z$  must lie in a similar shell around  $w$ ; and there are at most  $n^4$  choices for tuples  $(x, y, z, w)$ . We thus have

$$P_L \leq kn^7 \left( \left( \frac{c_1}{n^7} \right)^2 n^3 + \left( \frac{c_2}{n^7} \right)^2 n^4 \right) = o\left( \frac{1}{n^2} \right)$$

where  $c_1$  and  $c_2$  are constants dependent on dimension  $k$ .

We shall now estimate  $P_{G'}$ . Let us assume that  $n$  is a power of  $2^k$ . Let  $m_i = 2^{ki}$ ,

$$\beta_i = \{b: b \in B_i, (p_1, p_2) \in C_i, \text{ either } p_1 \in b \text{ or } p_2 \in b\},$$

and

$$\Pi_i = \{\pi(b_1, b_2, i): b_1 \in B_i, b_2 \in B_i, p_1 \in b_1, p_2 \in b_2, (p_1, p_2) \in C_i\}.$$

We note that the volume of each region in  $\Pi_i$  is at least  $m_i^{-1}$ . There exists a constant  $c_5 \geq 1$ , dependent on  $k$ , such that for each  $i, i = 1, \dots, \delta$ , each region in  $\Pi_i$  intersects at most  $c_5$  other regions in  $\Pi_i$ . Let  $c_3 = (36k + 3)^k$ . There are at most  $c_3 m_i$  possibilities for regions in  $\Pi_i$ . We let  $c_6$  be the smallest positive integer greater than 16 such that  $2^{c_6}/c_5 > 2 + \log_e(c_3 c_5) + 6c_6$  and  $c_6 \geq 1 + 2^{c_6} \log_e(1 + 2^{-c_6})$ .

We define  $P_i$  as follows.

- (1) For  $n/m_i > 6c_3 \log_2 n$ , let  $P_i$  be the probability that  $C_i$  is not empty.
- (2) For  $2^{c_6} < n/m_i \leq 6c_3 \log_2 n$ , let  $P_i$  be the probability that  $C_i$  contains more than  $c_3 m_i$  edges.
- (3) For  $1 \leq n/m_i \leq 2^{c_6}$ , let  $P_i$  be the probability that the number of edges in  $C_i$  exceeds  $c_3 c_7 m_i$  where  $c_7 = 2e^4 2^{2c_6}$ .

Let  $P_D$  be the probability that the number of edges in  $D$  exceeds  $c_3 c_7 n$ .

Let the constant  $c$  in the definition of  $P_{G'}$  be  $4c_3 c_7$ . If the number of edges in  $G'$  exceeds  $4c_3 c_7 n$  then either  $D$  contains more than  $c_3 c_7 n$  edges, or for some  $i, 1 \leq i \leq \delta$ ,  $C_i$  contains more than  $c_3 c_7 m_i$  edges. Hence,

$$P_{G'} \leq \sum_{i=1}^{\delta} P_i + P_D.$$

We shall show that each of  $P_i$  and  $P_D$  is  $o(1/n^4)$  and it then follows that  $P_{G'} = o(1/n^2)$  as  $\delta \leq \log_2 n$ . In order to bound  $P_i$  we consider three cases depending on the value of  $n/m_i$ .

*Case 1.*  $n/m_i > 6c_3 \log_2 n$ . In this case we show that there is a large empty region in the unit  $k$ -cube  $[0, 1]^k$ . If  $C_i$  is nonempty there is at least one region in  $\Pi_i$  and this region does not contain a point in  $V$ . There are at most  $c_3 m_i$  possibilities for regions in  $\Pi_i$  and each of these regions has volume at least  $(m_i)^{-1}$ . Hence,

$$P_i \leq c_3 m_i \left(1 - \frac{1}{m_i}\right)^n = o\left(\frac{1}{n^4}\right).$$

We shall now give a lemma that will be useful for Case 2 and Case 3. Let  $P_{ij}$  be the probability that the number of boxes in  $B_i$ , which contain at least  $e^{2j}$  and at most  $e^{2j+1}$  points in  $V$ , exceeds  $m_i/2^{3j}$ .

LEMMA 4. Let  $2^{j_0} \geq (n/m_i) \geq 1$ , and  $(n/m_i) \leq 6c_3 \log_2 n$ . Then  $\sum_{j=j_0}^{\log_2 n} P_{ij} = o(1/n^4)$ .

*Proof.* Suppose  $2^j \geq 6c_3 \log_2 n$ , and  $2^j \geq (n/m_i)$ . As  $P_{ij}$  is bounded by the probability that there is at least one box in  $B_i$  with at least  $e^{2j}$  points, we have

$$\begin{aligned} P_{ij} &\leq m_i \binom{n}{e^{2j}} (m_i)^{-e^{2j}} \\ &\leq m_i \frac{n^{e^{2j}}}{(e^{2j})!} (m_i)^{-e^{2j}} \\ &\leq O\left(m_i \left(\frac{n}{m_i 2^j}\right)^{e^{2j}} e^{-e^{2j}}\right) \quad (\text{using Stirling's approx.}) \\ &= o\left(\frac{1}{n^5}\right). \end{aligned}$$

Suppose  $1 \leq (n/m_i) \leq 2^j < 6c_3 \log_2 n$ . We can get a bound on  $P_{ij}$  by first choosing  $m_i/2^{3j}$  boxes in  $B_i$ , then choosing  $e^2 m_i/2^{2j}$  points to be located in the chosen boxes and letting the remaining points lie anywhere in the unit  $k$ -cube. So

$$\begin{aligned} P_{ij} &\leq \binom{m_i}{m_i/2^{3j}} \binom{n}{e^2 m_i/2^{2j}} 2^{-3je^2 m_i 2^{-2j}} \\ &\leq \frac{(m_i)^{m_i 2^{-3j}} n^{e^2 m_i 2^{-2j}}}{(m_i 2^{-3j})! (e^2 m_i 2^{-2j})!} 2^{-3je^2 m_i 2^{-2j}} \\ &\leq O\left( (e2^{3j})^{m_i 2^{-3j}} e^{-e^2 m_i 2^{-2j}} \left(\frac{n}{m_i 2^j}\right)^{e^2 m_i 2^{-2j}} \right) \quad (\text{Stirling's approx.}) \\ &= o\left(\frac{1}{n^5}\right). \end{aligned}$$

The proof of the lemma then follows.  $\square$

*Case 2.*  $2^{c_6} < n/m_i \leq 6c_3 \log_2 n$ . This case is broken down into two subcases depending on the number of boxes in  $\beta_i$ . We have

$$\begin{aligned} P_i &\leq \Pr(|\beta_i| > 8m_i^7 n^{-6}) + \Pr(|\beta_i| \leq 8m_i^7 n^{-6} \text{ and } |C_i| > c_3 m_i) \\ &= o(1/n^4) + o(1/n^4) = o(1/n^4). \end{aligned}$$

If  $|\beta_i| > 8m_i^7 n^{-6}$  then there is a large empty region, whereas if  $|\beta_i| \leq 8m_i^7 n^{-6}$  and  $|C_i| > c_3 m_i$  then there is a concentration of points in some region.

*Case 2.1.* Suppose  $|\beta_i| > 8m_i^7 n^{-6}$ . There are at least  $|\beta_i|/2$  regions in  $\Pi_i$  and as a region in  $\Pi_i$  intersects at most  $c_5$  other regions in  $\Pi_i$ , we can find at least  $c_5^{-1} m_i^7 n^{-6}$  disjoint regions in  $\Pi_i$  each of which does not contain any point in  $V$ . As there are at most  $c_3 m_i$  choices for regions in  $\Pi_i$ , and each region in  $\Pi_i$  has volume at least  $m_i^{-1}$ , we have

$$\begin{aligned} \Pr(|\beta_i| > 8m_i^7 n^{-6}) &\leq \binom{c_3 m_i}{c_5^{-1} m_i^7 n^{-6}} (1 - c_5^{-1} m_i^6 n^{-6})^n \\ &\leq \frac{(c_3 m_i)^{c_5^{-1} m_i^7 n^{-6}}}{(c_5^{-1} m_i^7 n^{-6})!} (1 - c_5^{-1} m_i^6 n^{-6})^n. \end{aligned}$$

Using Stirling's approximation for factorials, taking logarithms and noting that  $\log_e(1-x) \leq -x$ , for  $0 \leq x < 1$ , we get

$$\begin{aligned} \log_e (\Pr(|\beta_i| > 8m_i^7 n^{-6})) &\leq nc_5^{-1} \left(\frac{m_i}{n}\right)^6 \left(-1 + \frac{m_i}{n} \left(1 + \log_e(c_3 c_5) + 6 \log_e\left(\frac{n}{m_i}\right)\right)\right) \\ &\leq -\frac{n}{c_5 2^{c_6} (6c_3 \log_2 n)^6}. \end{aligned}$$

Thus

$$\Pr(|\beta_i| > 8m_i^7 n^{-6}) = o(1/n^4).$$

*Case 2.2.* Suppose  $|\beta_i| \leq 8m_i^7 n^{-6}$  and  $|C_i| > c_3 m_i$ . Let  $j_0 = 2 \log_2(n/m_i) - 1$ . A point in any box  $b$  in  $B_i$  is joined to points in at most  $c_3$  other boxes in  $B_i$  by edges in  $C_i$ , and so we have that

$$c_3 \sum_{b \in \beta_i} (|b \cap V|)^2 \geq |C_i| > c_3 m_i$$

and hence

$$\sum_{b \in \beta_i, |b \cap V| \geq 2^{j_0}} (|b \cap V|)^2 > (1 - 2^{-c_6+1}) m_i.$$

Then for some  $j, j = j_0 (= 2 \log_2(n/m_i) - 1), \dots, \log_2 n$ , the number of boxes in  $\beta_i$  (and hence  $B_i$ ), which contain at least  $e^{2^j}$  and at most  $e^{2^{j+1}}$  points in  $V$ , exceeds  $m_i/2^{3j}$ . Then from Lemma 4,

$$Pr(|\beta_i| \leq 8m_i^7 n^{-6} \text{ and } |C_i| > c_3 m_i) \leq \sum_{j=j_0}^{\log_2 n} P_{ij} = o\left(\frac{1}{n^4}\right).$$

Case 3.  $1 \leq n/m_i \leq 2^{c_6}$ . Suppose  $|C_i| > c_3 c_7 m_i$ . A point in box  $b$  in  $B_i$  is joined to points in at most  $c_3$  other boxes in  $B_i$  by edges in  $C_i$ , and so we get

$$c_3 \sum_{b \in B_i} (|b \cap V|)^2 \geq |C_i| > c_3 c_7 m_i = 2e^4 2^{2c_6} c_3 m_i;$$

hence,

$$\sum_{b \in B_i, |b \cap V| \geq 2^{c_6}} (|b \cap V|)^2 > (2e^4 - 1) 2^{2c_6} m_i.$$

Then for some  $j, j = c_6, \dots, \log_2 n$ , the number of boxes in  $B_i$ , which contain at least  $e^{2^j}$  points and at most  $e^{2^{j+1}}$  points in  $V$  exceeds  $m_i/2^{3j}$ . Then from Lemma 4 we can conclude that

$$P_i \leq \sum_{j=c_6}^{\log_2 n} P_{ij} = o\left(\frac{1}{n^4}\right).$$

By reasoning in the same manner as in Case 3 above we can show that  $P_D = o(1/n^4)$ .

**5. Conclusions.** We have given an  $O(\varepsilon^{-k} n \log n)$  algorithm for finding an approximate minimum spanning tree on a set  $V$  of  $n$  points in  $k$ -dimensional space, the weight of the approximate minimum spanning tree is guaranteed to be at most  $(1 + \varepsilon)$  times the weight of a minimum spanning tree. We have also presented an algorithm for finding a minimum spanning tree on a set  $V$  of  $n$  points in  $k$ -dimensional space. Under the assumption that the set  $V$  consists of  $n$  random points, independently and uniformly distributed in the unit  $k$ -cube  $[0, 1]^k$ , the expected running time of this minimum spanning tree algorithm is shown to be  $O(n\alpha(cn, n))$  where  $c$  is a constant dependent on  $k$  and  $\alpha$  is the inverse Ackermann function.

**Acknowledgments.** The author would like to thank Professor C. L. Liu, S. Kapoor, and P. Ramanan for helpful discussions.

REFERENCES

- [1] C. BERGE AND A. GHOUILA-HOURI, *Programming, Games and Transportation Networks*, John Wiley, New York, 1965.
- [2] K. CLARKSON, *Fast expected-time and approximation algorithms for geometric minimum spanning trees*, Proc. 16th Annual ACM Symposium on the Theory of Computing, 1984, pp. 342-348.
- [3] D. CHERITON AND R. E. TARJAN, *Finding minimum spanning trees*, this Journal, 5 (1976), pp. 724-742.
- [4] E. W. DIJKSTRA, *A note on two problems in connection with graphs*, Numer. Math., 1 (1959), pp. 269-271.
- [5] R. O. DUDE AND P. E. HART, *Pattern Classification and Science*, John Wiley, New York, 1973.
- [6] M. L. FREDMAN AND R. E. TARJAN, *Fibonacci heaps and their uses in network optimization algorithms*, Proc. 25th Annual IEEE Symposium on the Foundations of Computer Science, 1984, pp. 338-346.
- [7] H. GABOW, J. BENTLEY, AND R. TARJAN, *Scaling and related techniques for geometric problems*, Proc. 16th Annual ACM Symposium on the Theory of Computing, 1984, pp. 135-143.

- [8] H. N. GABOW, Z. GALIL, AND T. H. SPENCER, *Efficient implementation of graph algorithms using contraction*, Proc. 25th Annual IEEE Symposium on the Foundations of Computer Science, 1984, pp. 347–357.
- [9] J. B. KRUSKAL, *On the shortest spanning subtree of a graph and the travelling salesman problem*, Amer. Math. Soc., 7 (1956), pp. 48–50.
- [10] R. C. PRIM, *Shortest connection networks and some generalizations*, Bell System Tech. J., 36, pp. 1388–1401.
- [11] E. M. REINGOLD, J. NIEVERGELT, AND N. DEO, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ.
- [12] M. I. SHAMOS AND D. J. HOEY, *Closest-point problems*, Proc. 16th Annual IEEE Symposium on the Foundations of Computer Science, 1975, pp. 151–162.
- [13] R. E. TARJAN, *Efficiency of good but not linear set union algorithm*, J. Assoc. Comput. Mach., 22 (1975), pp. 215–225.
- [14] P. M. VAIDYA, *A fast approximation algorithm for minimum spanning trees in  $k$ -dimensional space*, Proc. 25th Annual IEEE Symposium on the Foundations of Computer Science, 1984, pp. 403–407.
- [15] A. C. YAO, *A  $O(|E|\log \log V)$  algorithm for finding minimum spanning trees*, Inform. Process. Lett., 4 (1975), pp. 21–23.
- [16] A. C. YAO, *On constructing minimum spanning trees in  $k$ -dimensional spaces and related problems*, this Journal, 11 (1982), pp. 721–736.
- [17] C. T. ZAHN, *Graph-theoretical method for detecting gestalt clusters*, IEEE Trans. Comput., C-20 (1970), pp. 68–70.

## SOME GEOMETRY FOR GENERAL RIVER ROUTING\*

ALAN SIEGEL† AND DANNY DOLEV‡

**Abstract.** Efficient solutions are given to compute the optimal placement for a pair of VLSI modules interconnected by river routing. Specifically, let the (perpendicular) distance between the two modules be the **separation**, and call the (transverse) displacement the **offset**. This paper principally considers the separation problem: Given an offset and a wiring rule, find the minimum separation permitting a legal wiring. The design rules might use wires which are exclusively rectilinear, polygonal with a finite number of slopes, or possibly restricted to some other class of shapes such as circular arcs plus linear pieces.

Techniques are developed which unify a variety of different placement problems, and give efficient solutions under extremely general conditions. The advantage of these generalizations is not only their theoretical framework; the results extend naturally to more precise models of real river routing, and the theory is applicable to placement problems for collections of modules.

**Key words.** VLSI, river routing, placement

**AMS(MOS) subject classifications.** F.2.2, B.7.2

**1. Introduction.** VLSI circuits are often designed in pieces analogous to macros or subroutines of computer programs. Each piece, called a **module**, has **terminals**, which are connection points for wires distributing signals and power among the modules. **River routing** is wiring that has no crossovers; it can essentially be implemented on a single layer. Although such a routing cannot be used to make arbitrary connections, it is commonly used wherever practical, since designs with higher percentages of river routing are often easier to lay out, more compact, and faster than circuits with more complex interconnections. Two rows of terminals, for example, may be connected by river routing when the corresponding terminals for each row are in the same order (but are not necessarily in perfect alignment). Input pads, arrays of identical cells, PLAs, and bristle block constructions (as defined in [J]) are likely to have interconnections free from crossovers.

We analyze the problem of efficiently computing optimal placements for a pair of interconnected modules. The specific definitions of optimality will be given in the next section. The requirement of efficiency precludes taking the quadratic time necessary to determine all possible  $O(n^2)$  coordinates of the interconnection wires; the goal is to have fast primitives to compute the space required for the wiring.

These utilities would be useful in automated hierarchical design systems. They can also be used to find optimal solutions to more complicated problems such as placement for two rows of modules interconnected by river routing [Si1]. Furthermore, the iteration of fast primitives, such as the ones presented here, ought to provide satisfactory solutions to more general placement problems, which are computationally too complex to solve optimally. Finally, the algorithms and VLSI properties developed in this paper have application to other problems of placement and routing as well as to the theory of computational geometry.

---

\* Received by the editors August 4, 1986; accepted for publication April 21, 1987. This work was done at Stanford University, Stanford, California 94305 and was partially supported by Defense Advanced Research Projects Agency contract MDA-903-80-C-0107 and National Science Foundation grant MCS-80-12907.

† New York University, New York, New York 10012.

‡ Hebrew University, Jerusalem, Israel.



**2. The wiring problem.** The wiring problem is illustrated in Fig. 2.1. Two rows of  $n$  corresponding terminals,  $\{P_i\}$  and  $\{Q_i\}$ , are to be interconnected by wires. The rows are parallel, and each row represents a rigid module; the inter-terminal spacings between the  $P_i$  are fixed, as are the  $Q_i$  spacings. A relative placement of the terminals is thus completely determined by two values: the vertical distance between the rows, which is the **separation**, and the horizontal distance between the first pair of terminals  $P_1$  and  $Q_1$ , which is the **offset**. The connecting wires will be subject to various constraints (design rules) described later. These constraints determine the permissible shapes for wires and the spacing between adjacent wires. Consequently they determine which offsets and separations are permissible for the modules. As shown, the rows are taken to be horizontal, with the  $\{Q\}$  above the  $\{P\}$ .

We examine the following.

(1) *The Separation Problem.* Given an offset and a wiring rule, find the minimum separation permitting a legal wiring.

The principal conclusion is:

For general curvilinear wiring rules, the separation can be found in  $O(n \log n)$  time.

Related problems include:

(2) *The Optimal Offset Problem.* Given a wiring rule, find an offset minimizing the separation.

(3) *The Offset Range Problem.* Given a wiring rule and a separation, find all offsets permitting a legal wiring.

In [Si2], the separation problem is solved in  $O(n)$  time for polygonal wiring. In [Si3], the optimal offset problem is solved in time<sup>1</sup>  $O(\Psi(n) \log n)$  where  $\Psi(n)$  is the time to find the separation. The results in this paper, therefore, are directly applicable to the optimal offset problem. In [Si1], the offset range problem is solved in  $O(n \log n)$  time for general curvilinear wiring rules, and in time  $O(n)$  for polygonal wiring; the results are then extended to solve placement problems for ensembles of modules interconnected by river routing channels.

Tompa examined river routing for two parallel rows of terminals in [T1]. His work gives a quadratic solution for locating (if possible) the interconnection wires when the terminals are given fixed positions. This solution is optimal since the wiring description may have  $O(n^2)$  coordinates. Although the solution is optimal, it does not give efficient methods to answer the important questions of finding good module placements solving the separation or optimal offset problems. Tompa [T2] reports the

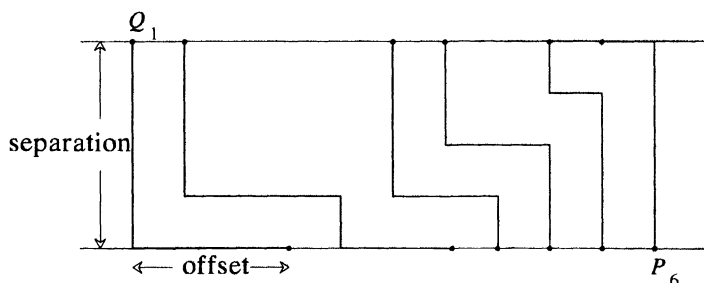


FIG. 2.1

<sup>1</sup> Recently, a very elegant  $O(n)$  time solution was discovered for the optimal offset problem with rectilinear wiring [M].

existence of an  $O(n \log n)$  algorithm by Kirkpatrick and Seidel to solve the separation problem for linear plus circular wiring and states the optimal offset problem as an open question. The offset range problem is related to the work by Leiserson and Pinter [LP] which gives an  $O(n)$  time solution for rectilinear wiring on an integer grid, and solves placement problems for two rows of interconnected modules. [Si1] includes generalizations and refinements of results originating in [LP].

In this paper, we develop some of the techniques which unify these problems and provide efficient solutions under extremely general conditions. The advantage of these generalizations is not only their theoretical framework; the results extend naturally to more precise models of real river routing, and the techniques are applicable to other types of wiring problems and to problems in computational geometry as well. See [Si1], [Si2], and [Si3] for more of the routing theory.

**3. The models.** In VLSI design, wires on a given layer satisfy specific constraints. Wires are frequently required to be composed exclusively of rectilinear segments; sometimes 45-degree pieces are also permitted. Other fabrication processes allow a variety of segment slopes. These polygonal wiring rules and, for theoretical interest, more general wiring shapes will be included in our methods. The minimum distance between wires is set to 1, and the wire width is taken as 0 for mathematical convenience.

The limitations in the resolution of a fabrication process also affect wire shape. A circular figure, for example, can be approximated arbitrarily well by sufficiently small rectilinear segments, but a VLSI wire must be composed of pieces with a fixed minimum size. Our model can include this restriction by requiring all wire segments to have endpoints lying on a lattice (which is hereafter called a grid). We examine integer grids, which are lattices with unit spacing, and fractional grids, which have fractional spacing.<sup>2</sup> Continuous grids will also be considered.

Given  $n$  pairs of terminals,  $(P_1, Q_1), \dots, (P_n, Q_n)$ , define  $P_i$  to be both the name of a terminal on the bottom row and the horizontal position of the terminal, and take  $Q_j$  to be a similar terminal on the upper row.

Since wires must be separated by unit distance, every routing problem has forbidden regions restricting the wiring flow. Specifically, for every terminal  $P_j$  and index  $s$ , there is an  $s$ th region (specified later) around  $P_j$ , which cannot be entered by a wire connecting  $P_{j+t}$  with  $Q_{j+t}$  for  $|t| \geq s$ . See Fig. 3.1.

With a scheme permitting unrestricted wire shapes, these regions will be concentric disks centered at the terminals [T1]. We require the boundaries to conform to the shapes

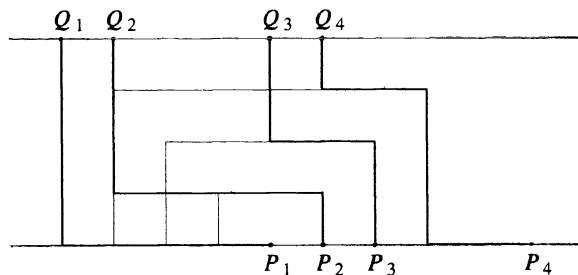


FIG. 3.1. Wires restricted by rectangular barriers around  $P_1$ .

<sup>2</sup> More precisely, our models include a variety of wiring schemes defined on lattices. The specific models are in fact gridless.

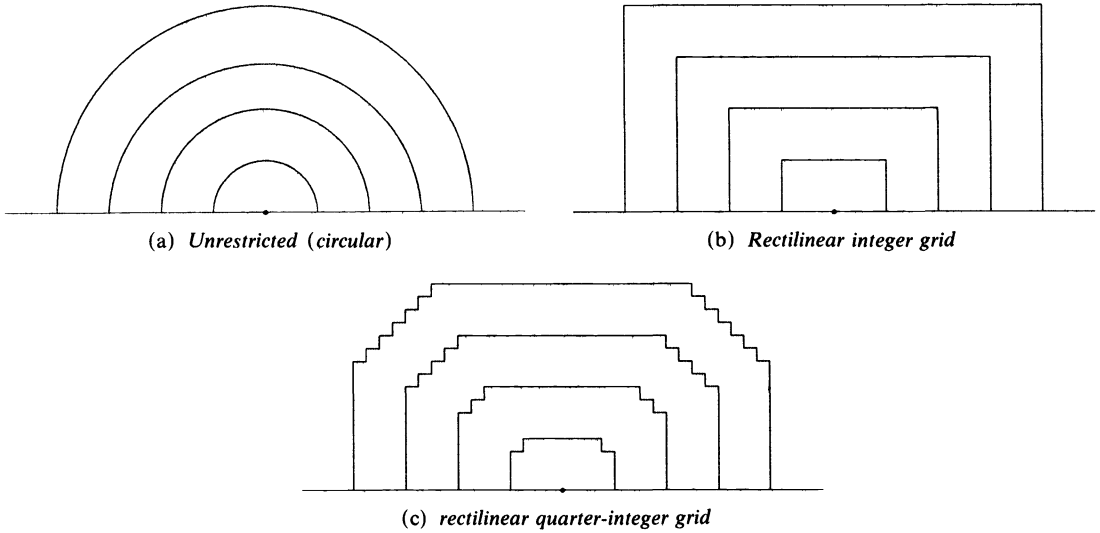


FIG. 3.2. Separation barriers.

permitted by the wiring scheme and grid restrictions.<sup>3</sup> In the rectilinear case with an integer grid they are concentric rectangles. On a quarter-integer grid the separation barriers are no longer rectangular. See Fig. 3.2. We remark that the boundaries of the barriers in Fig. 3.2(a), for example, are concave, that is, the curves have a negative second derivative. The region lying below such a boundary, however, is geometrically convex, that is, any pair of points lying inside the region defines a line segment which is contained within the region. To avoid confusion, we shall always refer to this latter property as geometrical convexity.

In §§ 5 and 6, the separation problem is analyzed principally for families of similar concave barriers. In § 7, the requirements of concavity and similarity will be relaxed somewhat and the wiring models extended to include virtually all known wiring schemes. Sections 8 and 9 apply the results to more realistic wiring models.

**4. The separation function.** It has already been observed that the separation must be sufficient to allow each terminal  $Q_j$  to lie outside the  $|j-i|$ th barrier emanating from  $P_i$  for all  $i$ . The separation, of course must also be at least as high as the highest point on any wire. These two facts may be used to modify the separation barriers. Define the  $j$ th **right separation barrier** to be the curve obtained by extending the  $j$ th barrier to the left of its origin continuously as a horizontal line with height (which in most cases equals)  $j$ . **Left separation barriers** are defined analogously. These new barriers are monotone, and if the original regions are geometrically convex and similar, then

<sup>3</sup> It is not difficult to show that the boundaries do conform to the wiring shapes for, say, unrestricted curvilinear, integer-grid-rectilinear, quarter-integer-grid-rectilinear, and half-integer-grid-rectilinear-plus-45-degree wiring schemes. However, this criterion is not satisfied by design rules having an arbitrary collection of permissible wire slopes and requiring segments to have endpoints on a given grid. Furthermore, the solution to the separation problem for these arbitrary wiring rules may require wires which are not even monotone in  $x$ . These and related difficulties are eliminated in § 4 by defining wiring rules in terms of separation barriers. Another advantage of this barrier formulation is that quarter-integer rectilinear wiring, for example, can be defined for problems where the terminals are not located on a quarter-integer grid. Moreover, rectilinear-plus-45-degree wiring can be defined not only as in the half-integer-grid scheme, but also with barriers packed, say, more densely than any model having rational coordinates.

the new ones are also. Figure 4.1 shows a family of modified right separation barriers. All barriers are now assumed to be so modified. Now the **separation function**  $W(i, j)$  can be defined as the height at  $Q_j$ , for  $j > i$   $\{j < i\}$ , of the  $|j - i|$ th right  $\{left\}$  barrier emanating from  $P_i$ . See Fig. 4.2.

Let the family of right barrier curves emanating from the origin be represented by the functions  $H_\rho(x)$ , where  $\rho$  is the index. ( $H_\rho$  is taken to be lower semicontinuous, that is,  $H_\rho(x) = \min \{y \mid (x, y) \text{ is on the curve}\}$ .) Note that the convex separation regions as shown in Fig. 4.2 have concave functions defining their boundaries. The family of right barriers emanating from  $P_i$  are defined by  $H_\rho(x - P_i)$ , and the separation function (for both left and right barriers) is:

$$(4.1) \quad W(i, j) = \begin{cases} H_{j-i}(Q_j - P_i) & \text{if } j > i, \\ 0 & \text{if } j = i, \\ H_{i-j}(P_i - Q_j) & \text{if } j < i. \end{cases}$$

It follows that the separation is not less than  $\max_{i,j} W(i, j)$ . We now show the following lemma.

LEMMA 1. *The solution to the separation problem is  $\max_{i,j} W(i, j)$ .*

*Proof.* It remains to prove that  $P$  and  $Q$  can be legally connected with separation width  $\max_{i,j} W(i, j)$ . Suppose  $Q_j \leq P_j$  and  $Q_{j-1} \leq P_{j-1}$ . Connect  $P_j$  with  $Q_j$  by running wire  $j$  along the bounding envelope formed from the  $(j - i)$ th right barrier emanating from  $P_i$ ,  $i = 1, 2, \dots, j - 1$ . An envelope is illustrated in Fig. 4.3(a). The spacing requirement between wires is satisfied because any point on wire  $j - 1$  (with the exception of those on the vertical segment ending at  $Q_{j-1}$  or the horizontal segment starting at  $P_{j-1}$ ) is by definition on the  $(j - i - 1)$ th barrier emanating from  $P_i$  for some  $i < j$  while wire  $j$  is beyond the  $(j - i)$ th barrier. It is easy to see that the vertical segment terminating at  $Q_{j-1}$  and the horizontal segment terminating at  $P_{j-1}$  are at least unit

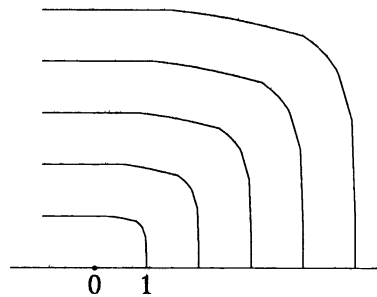


FIG. 4.1. Similar concave barriers.

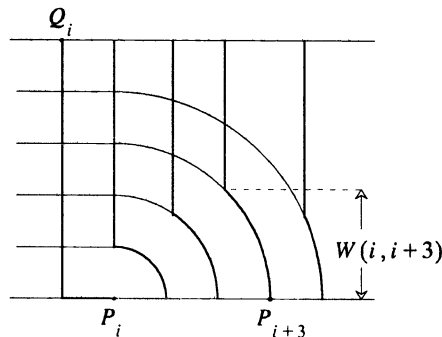
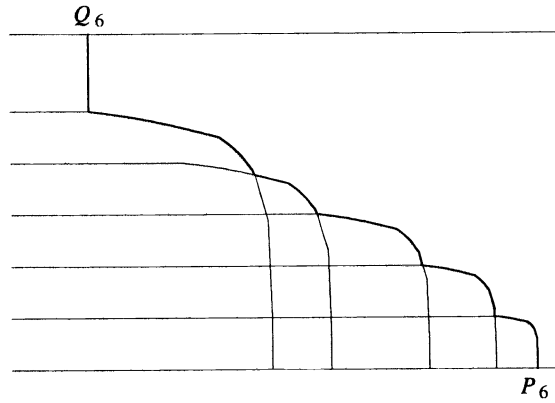
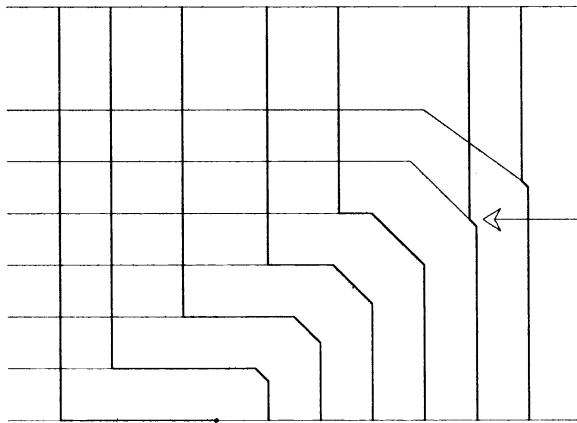


FIG. 4.2. A separation from circular barriers.

(a) *Interacting barriers*(b) *Nonmonotonic separation*FIG. 4.3. *The complications of wiring.*

distance from wire  $j$ . The argument is the same if both inequalities are reversed. The two remaining possibilities cause wires  $j$  and  $j-1$  to run in opposite directions; the wires are then closest at  $Q_{j-1}$  and  $Q_j$  or  $P_{j-1}$  and  $P_j$ .  $\square$

Note that a barrier envelope may have barrier pieces with endpoints which are not on a discrete grid for, say, curvilinear barriers. The use of such an envelope as a wire path, however, is reasonable when positive wire thicknesses are included in the model, or when practical wiring shapes are considered. Consequently we shall assume that a wire can indeed follow the boundary of intersecting barriers and rise vertically when necessary to connect to a terminal  $Q_j$ . Thus a wiring scheme is determined by its separation barriers. An alternative assumption is that the wiring scheme permits legal connections when the placement satisfies the separation constraints. Tompa, for example, shows how to compute interconnections with wires as short as possible using unrestricted curvilinear (i.e., circular) barriers [T]. In this case the wires might follow a path which differs from the envelope, but the legal placements for the two wiring schemes are the same.

Define a **left block** to be a maximal sequence of pairs of terminals  $(P_i, Q_i), \dots, (P_j, Q_j)$  such that  $Q_k \leq P_k$ , for  $i \leq k \leq j$ . This condition says that all the connection points in the upper row of a *left block* are located to the left of the corresponding positions on the lower row. The previous lemma shows that a left block,

if legally wireable, can be wired so that each wire in the block decreases monotonically in height from its  $Q$ -terminal  $Q_i$  to its  $P$ -terminal  $P_i$ . Clearly analogous conclusions hold for a **right block**. It follows that a wireable left or right block can be wired within its rectangular boundary. Consequently, for a fixed offset, the separation is determined by the worst block. Therefore we may assume that the separation problem is to be solved for a single left block (which uses right separation barriers). The separation is in fact determined by the worst **strict block**, where a strict left block is a maximal sequence of pairs  $(P_i, Q_i), \dots, (P_j, Q_j)$  such that  $Q_k < P_k$  for  $i \leq k \leq j$ .

Lemma 1 shows that separation problem can be solved in  $O(n^2)$  time. For rectilinear wiring on an integer grid the complexity can be shown to be  $O(n)$  [DKSSU]. Figure 4.3 illustrates the difficulties in determining the separation for more complex wiring schemes; one example shows many wiring barriers interacting with one wire ( $P_6$  to  $Q_6$  in this case); the other shows the nonmonotonic behavior of the separation function.

**5. The partitioning property.** The barrier families discussed in this paper all satisfy some fundamental geometric properties. These properties will enable the use of a divide-and-conquer approach to reduce the time needed to find the separation. The idea behind the Partitioning Theorem is essentially the following. Consider two line segments that connect  $Q_{i_1}$  with  $P_{i_1}$  and  $Q_{i_2}$  with  $P_{i_2}$ . Suppose  $i_1$  and  $i_2$  maximize, respectively,  $W(*, j_1)$  and  $W(*, j_2)$ . The Partitioning Property then says that the segments cannot cross. It is then a simple matter to exploit this property to find the maximum separation in  $O(n \log n)$  time (Theorem 9): Set  $j = \lceil n/2 \rceil$ , and find the index  $i^*$ , which is the largest  $i \leq \lceil n/2 \rceil$  maximizing  $W(i, \lceil n/2 \rceil)$ . The separation problem will then split into two subproblems, to find  $\max_{i \in [1, i^*], j \in [1, \lceil n/2 \rceil]} W(i, j)$ , and to find  $\max_{i \in [i^*, n], j \in [\lceil n/2 \rceil + 1, n]} W(i, j)$ . The splitting is applied recursively. In this section, we identify the basic barrier properties which guarantee that the Partitioning Property is satisfied.

Define  $l(\rho, x)$ , for  $x \leq \rho$ , to be the horizontal line segment that is bounded by the curves  $H_\rho$  and  $H_{\rho-1}$  and that intersects  $H_\rho$  at  $(x, H_\rho(x))$ . Let  $l(\rho, \rho)$  have endpoints  $(\rho - 1, 0)$  and  $(\rho, 0)$ ,  $\rho > 1$ . Take  $l$  to be a half-infinite line if it does not intersect  $H_{\rho-1}$  or if  $l$  intersects the barrier along an infinite ray. (Properties P0c, P0d below will ensure that  $l$  must otherwise intersect  $H_{\rho-1}$  in exactly one point.) See Fig. 5.1. Denote the length of  $l \equiv l(\rho, w)$  by  $|l|$ , and let  $Lx(\rho, w)$  be the  $x$ -coordinate of  $l$ 's left endpoint. We first consider barrier families satisfying the following properties.

- P0a:  $H_\rho(x) = 0$  for  $x \geq \rho$ . This says if terminal  $Q_j$  is at least  $j - i$  units to the right of  $P_i$ , then the wiring from  $Q_j$  to  $P_j$  is unaffected by the presence of  $P_i$ .
- P0b:  $H_\rho(x) > H_{\rho-1}(x)$  for  $x < \rho$ . The barriers emanating from a common point do not intersect since different wires must not touch.
- P0c:  $H_\rho(x) \geq H_\rho(y)$  for  $x \leq y \leq \rho$ . The barrier functions are nonincreasing.
- P0d:  $H_\rho(x + \varepsilon) - 2H_\rho(x) + H_\rho(x - \varepsilon) \leq 0$  for  $\varepsilon > 0, x + \varepsilon \leq \rho$ . The  $H_\rho(x)$  are concave for  $x \leq \rho$ .
- P1: If  $x \leq y \leq \rho$ , then  $|l(\rho, x)| \geq |l(\rho, y)|$ . The length of the horizontal line segment between  $H_\rho$  and  $H_{\rho-1}$  does not decrease as a function of the segment's height. In other words, the left derivative of  $H_\rho$  at  $x$  is no greater than the left derivative of  $H_{\rho-1}$  at  $Lx(\rho, x)$ .
- P2: If  $x \leq \rho$ , then  $|l(\rho - 1, x - 1)| \geq |l(\rho, x)|$ . As a weak consequence, the curves  $H_\rho$  cut any horizontal line into a sequence of segments having lengths which do not increase as the sequence is traversed from left to right.

It should be noted that generalizations of these properties will be the basis of Theorems 15, 17, and 18. We avoid unnecessary complication by using properties P0, P1, and

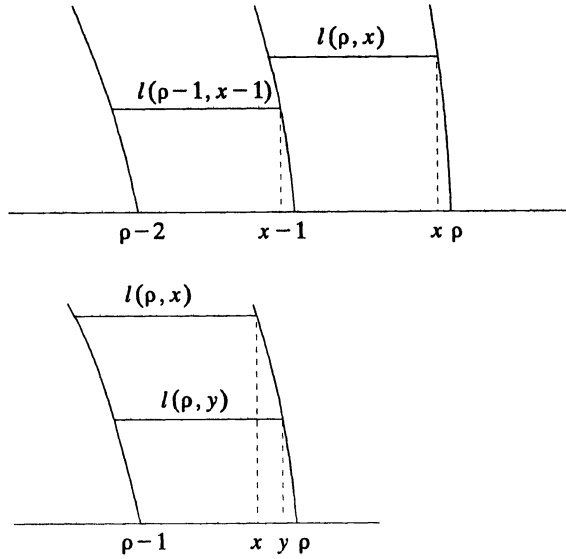


FIG. 5.1

P2 in this section, and escape repetition by giving proofs, which, mutatis mutandis, also hold for the more general barriers.

LEMMA 2. *Let  $d > 0$  and suppose the  $H_\rho$ 's satisfy P0, P1, and P2. Then  $H_{\rho+1}(\nu) > H_\rho(u)$  implies  $H_{\rho+1}(\nu - d) > H_\rho(u - d)$ .*

*Proof.* Suppose  $H_{\rho+1}(\nu) > H_\rho(u)$ . Then  $\nu < \rho + 1$ , and we may assume that  $u \leq \rho$ . Let  $l$  be the horizontal segment with left endpoint  $(u, H_\rho(u))$  and bounded on the right by  $H_{\rho+1}$ . See Fig. 5.2. Let the  $x$ -coordinate of  $l$ 's right endpoint be  $z$ . It is not necessarily true that  $l = l(\rho + 1, z)$  since  $z$  could equal  $\rho + 1$  and the barrier  $H_{\rho+1}$  might be a vertical segment at  $\rho + 1$ . Let  $x = (z + \nu)/2$ . Then  $\nu < x < z$ ,  $Lx(\rho + 1, x) < u$  and  $|l(\rho + 1, x)| \geq |l|$ . By P0c,  $H_{\rho+1}(\nu - d) \geq H_{\rho+1}(x - d)$ . Moreover,  $Lx(\rho + 1, x - d) = x - d - |l(\rho + 1, x - d)| < z - d - |l| = u - d$ . So  $H_{\rho+1}(\nu - d) \geq H_{\rho+1}(x - d) \geq H_\rho(Lx(\rho + 1, x - d)) \geq H_\rho(u - d)$ . It is not possible that  $H_{\rho+1}(\nu - d) = H_{\rho+1}(x - d) = H_\rho(u - d)$  since P0c and P0d would then contradict P0b by saying that  $H_{\rho+1}$  must be flat to the left of  $x - d$ , which implies that  $H_{\rho+1}(s)$  and  $H_\rho(s)$  intersect at  $s = \min\{u - d, x - d\}$ .  $\square$

LEMMA 3. *Let  $r > 0$ ,  $d > 0$ , and suppose the  $H_\rho$ 's satisfy P0, P1, and P2. Then  $H_{\rho+r}(\nu) > H_\rho(u)$  implies  $H_{\rho+r}(\nu - d) > H_\rho(u - d)$ .*

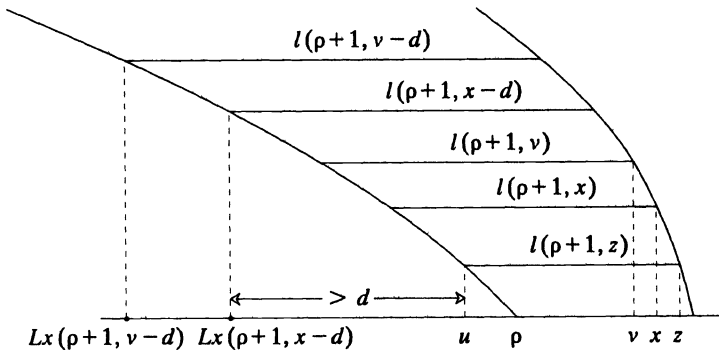


FIG. 5.2

*Proof.* It suffices to assume that  $\nu < \rho + r$ , and  $u \leq \rho$ . The proof is by induction on  $r$ . Lemma 2 proves the case  $r = 1$ . Suppose Lemma 3 is true for  $r = k - 1$ , and let  $r = k$ . Let  $x_0 = Lx(\rho + k, \nu)$  and let  $x_1 = \max_x \{x: H_{\rho+k-1}(x) > H_\rho(u)\}$ . Let  $x = (x_1 + x_0)/2$ . Then  $H_{\rho+k}(\nu) > H_{\rho+k-1}(x) > H_\rho(u)$ . Thus by the induction hypothesis,  $H_{\rho+k-1}(x - d) > H_\rho(u - d)$ . Moreover,  $H_{\rho+k}(\nu - d) > H_{\rho+k-1}(x - d)$ . Combining these two inequalities shows that  $H_{\rho+k}(\nu - d) > H_\rho(u - d)$ .  $\square$

LEMMA 4. *Let  $\rho > 1$  and suppose the  $H_\rho$ 's satisfy P0, P1, and P2. Then  $H_{\rho+1}(\nu) > H_\rho(u)$  implies  $H_\rho(\nu - 1) > H_{\rho-1}(u - 1)$ .*

*Proof.* Again take  $\nu < \rho + 1$ , and  $u \leq \rho$ . Let  $l$  be the horizontal segment with left endpoint  $(u, H_\rho(u))$  and bounded by  $H_{\rho+1}$  on the right. Let  $s = Lx(\rho + 1, \nu)$ . See Fig. 5.3. Clearly  $s \in [-\infty, u)$ . P2 says  $|l(\rho, \nu - 1)| \geq |l(\rho + 1, \nu)|$ , so  $Lx(\rho, \nu - 1) = \nu - 1 - |l(\rho, \nu - 1)| \leq \nu - |l(\rho + 1, \nu)| - 1 = s - 1 < u - 1$ , and  $H_\rho(\nu - 1) \geq H_{\rho-1}(Lx(\rho, \nu - 1)) \geq H_{\rho-1}(u - 1)$ . So

$$(5.1) \quad H_\rho(\nu - 1) \geq H_{\rho-1}(u - 1).$$

The inequality is made strict with the same trick as Lemma 2. Let  $z$  be the  $x$ -coordinate of  $l$ 's right endpoint. Let  $x = (z + \nu)/2$ . Then  $x > \nu$ , but  $H_{\rho+1}(x) > H_\rho(u)$ . Substituting  $x$  for  $\nu$  in (5.1) and noting that  $x > \nu$  gives  $H_\rho(\nu - 1) \geq H_\rho(x - 1) \geq H_{\rho-1}(u - 1)$ . If equality holds everywhere,  $H_\rho(\nu - 1) = H_\rho(x - 1) = H_{\rho-1}(u - 1)$ . This says that  $H_\rho$  is flat to the left of  $x - 1$  and must intersect  $H_{\rho-1}$  at  $\min(x - 1, u - 1)$ , which contradicts P0b. Hence  $H_\rho(\nu - 1) > H_{\rho-1}(u - 1)$ .  $\square$

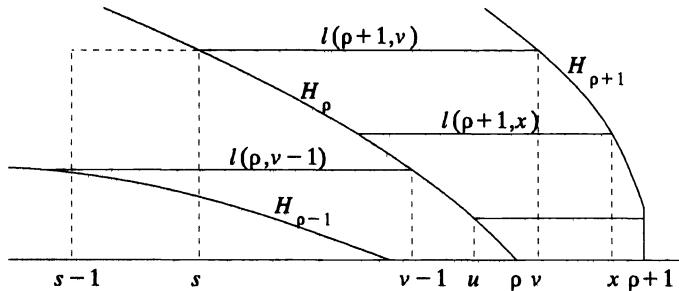


FIG. 5.3

LEMMA 5. *Let  $r > 0$ , and suppose the  $H_\rho$ 's satisfy P0, P1, and P2. Then  $H_{\rho+r}(\nu) > H_\rho(u)$  implies  $H_{\rho+r-1}(\nu - 1) > H_{\rho-1}(u - 1)$ .*

*Proof.* The proof uses an induction argument similar to that for Lemma 3. Once more  $\nu < \rho + r$ , and we may assume  $u \leq \rho$ . Lemma 4 proves the case  $r = 1$ . Suppose Lemma 5 is true for  $r = k - 1$ , and let  $r = k$ .  $H_{\rho+k-1}$  must have values greater than  $H_\rho(u)$  since it cannot intersect the curve  $H_\rho(x)$  for  $x < \rho$ . Since  $|l(\rho + k, \nu)| \geq 1$ ,  $H_{\rho+k-1}$  must have values less than  $H_{\rho+k}(\nu)$ . Choose  $x$  so that  $H_\rho(u) < H_{\rho+k-1}(x) < H_{\rho+k}(\nu)$ . Then by the induction hypothesis,  $H_{\rho+k-2}(x - 1) > H_{\rho-1}(u - 1)$ . Moreover  $H_{\rho+k-1}(\nu - 1) > H_{\rho+k-2}(x - 1)$ . Combining these two inequalities shows that  $H_{\rho+k-1}(\nu - 1) > H_{\rho-1}(u - 1)$ .  $\square$

We define the **Partitioning Property** as follows:

Let  $W(i, j)$  be the separation function defined by a wiring scheme.

Suppose  $P_i, P_{i+q}, Q_j$ , and  $Q_{j+r}$  are in a left block, where  $q \geq 0, r \geq 0$ , and  $j + r > i + q$ .

$$(5.2) \quad \text{If } W(i, j + r) > W(i, j), \text{ then } W(i + q, j + r) > W(i + q, j).$$

Lemmas 2, 3, 4, and 5 are summarized in the following.



**THEOREM 6 (The Partitioning Theorem).** *Let  $W(i, j)$  be the separation function defined by a barrier family satisfying properties P0, P1, and P2. Then  $W(i, j)$  has the Partitioning Property.*

*Proof. Case 1:  $j > i + q$ . We must show that*

$$(5.3) \quad \text{if } W(i, j+r) > W(i, j), \text{ then } W(i+1, j+r) > W(i+1, j),$$

since induction on  $q$  then gives Theorem 6 for all nonnegative integral values of  $q$  where  $j > i + q$ . See Fig. 5.4. According to equation (4.1), (5.3) says, suppose  $H_{j+r-i}(Q_{j+r} - P_i) > H_{j-i}(Q_j - P_i)$ ; then  $H_{j+r-i-1}(Q_{j+r} - P_{i+1}) > H_{j-i-1}(Q_j - P_{i+1})$ . Let  $\rho = j - i$ ,  $\nu = Q_{j+r} - P_i$ ,  $u = Q_j - P_i$ , and  $d + 1 = P_{i+1} - P_i$ . The unit separation requirement between terminals ensures that  $d \geq 0$ . We must show that if  $H_{\rho+r}(\nu) > H_{\rho}(u)$ , then  $H_{\rho+r-1}(\nu - 1 - d) > H_{\rho-1}(u - 1 - d)$ . Since  $H_{\rho+r}(\nu) > H_{\rho}(u)$ , Lemma 5 says that  $H_{\rho+r-1}(\nu - 1) > H_{\rho-1}(u - 1)$ . Then by Lemma 3,  $H_{\rho+r-1}(\nu - 1 - d) > H_{\rho-1}(u - 1 - d)$ .

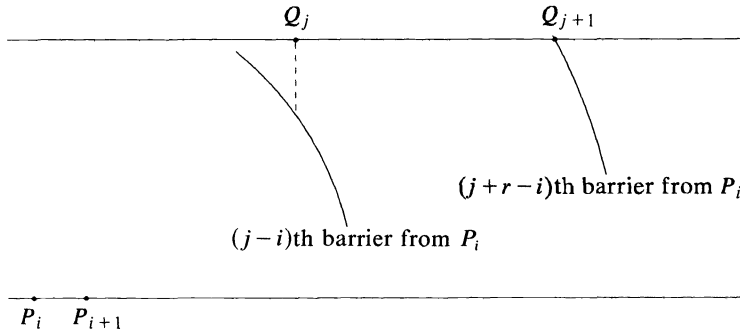


FIG. 5.4

*Case 2:  $j \leq i + q$ . Then  $W(i + q, j) = 0$ , since  $P_{i+q}$  and  $Q_j$  belong to a left block. Thus it suffices to show if  $W(i, j+r) > 0$  then  $W(i + q, j+r) > 0$ . Now  $W(i, j+r) > 0$  says that  $Q_{j+r} - P_i < j+r - i$ . Therefore  $Q_{j+r} - P_{i+q} < j+r - i - q$ , whence  $Q(i + q, j+r) > 0$ .  $\square$*

Evidently the separation needed between  $P_i$  and  $Q_j$  is the same if determined by barriers emanating from  $P_i$  or (upside down) barriers emanating from  $Q_j$ . Hence rotating the plane 180 degrees gives an equivalent inequality by interchanging  $P$  with  $Q$ ,  $i$  with  $j+r$  and  $q$  with  $-r$ . This observation, contraposition, and renaming the variables gives the following corollary.

**COROLLARY 7.** *Suppose  $P_i, P_{i+q}, Q_j$ , and  $Q_{j+r}$  are in a left block, where  $q \geq 0, r \geq 0$ . If  $w$  is defined by separation barriers satisfying P0, P1, and P2, then:*

(1) For  $j+r > i+q$ ,

$$W(i+q, j) \geq W(i+q, j+r) \Rightarrow W(i, j) \geq W(i, j+r).$$

(2) For  $j > i$ ,

$$W(i, j+r) > W(i+q, j+r) \Rightarrow W(i, j) > W(i+q, j).$$

(3) For  $j > i$ ,

$$W(i+q, j) \geq W(i, j) \Rightarrow W(i+q, j+r) \geq W(i, j+r).$$

*Proof.* (1) Follows from contraposition. (2) Follows from rotation. (3) Follows from rotation and contraposition.  $\square$

Consequently, we have the following.

**COROLLARY 8.** *Let  $W$  be a separation function having the Partitioning Property, and suppose that all terminals are in one left block. Let  $I = [x_1, x_2]$  and  $J = [y_1, y_2]$  be arbitrary subintervals of  $[1, n]$ . Given  $j_0 \in J$ , let  $i_0 \in I$  be the maximum value in  $I \cap [1, \max(x_1, j_0)]$  satisfying  $W(i_0, j_0) = \max_{i \in I} W(i, j_0)$ . Then the separation function attains its maximum on  $I \times J$  at some  $(i_{\max}, j_{\max})$  where either  $(i_{\max}, j_{\max}) \cong (i_0, j_0)$  or  $(i_{\max}, j_{\max}) \cong (i_0, j_0)$ . The inequalities are understood to hold coordinatewise.*

*Proof.* **Case 1:**  $W(i_0, j_0) > 0$ . Suppose  $i_{\max} < i_0$  and  $j_{\max} > j_0$ . Then by Corollary 7(2)

$$W(i_{\max}, j_{\max}) > W(i_0, j_{\max}) \Rightarrow W(i_{\max}, j_0) > W(i_0, j_0).$$

Hence the requirement that  $W(i_0, j_0)$  be maximal for  $j_0$  fixed is contradicted.

Similarly, if  $i_{\max} > i_0$  and  $j_{\max} < j_0$ , then

$$W(i_{\max}, j_{\max}) \cong W(i_0, j_{\max}) \Rightarrow W(i_{\max}, j_0) \cong W(i_0, j_0)$$

by Corollary 7(3). The requirement that  $i_0$  be maximum is thus contradicted. We have assumed that  $i_{\max} < j_{\max}$  since otherwise  $W(i_{\max}, j_{\max})$  would be zero for this left block.

**Case 2:**  $W(i_0, j_0) = 0$ . Evidently  $i_0$  equals the middle value of  $x_1, j_0$ , and  $x_2$ . If  $x_1 < x_2 < j_0$ , for example, then the definition of  $i_0$  requires that it equal  $x_2$ .

If  $i_0 = x_1$ , then for  $h$  and  $k$  such that  $k < j_0 \cong x_1 < h$ , it follows that  $W(h, k) = 0$  since  $k < h$ .

Otherwise  $W(i_0 - 1, j_0) = 0$ , so  $Q_{j_0} - P_{i_0-1} \cong j_0 - i_0 + 1 > 0$  whence the unit spacing requirement for terminals ensures that  $Q_k - P_h \cong k - h$  for  $k \cong j_0$ , and  $h \cong i_0$ . For these values of  $k$  and  $h$ ,  $W(h, k) = 0$ . Similarly,  $W(h, k) = 0$  if  $k < j_0$  and  $h \in [i_0 + 1, x_2]$  since the interval of  $h$ 's (is empty or) contains no value smaller than  $j_0$ .  $\square$

It should be noted that Theorem 6 has an alternate formulation:

*Let  $W(i, j)$  be the separation function defined by a barrier family satisfying properties P0, P1, and P2. Suppose  $P_i, P_{i+q}, Q_j$ , and  $Q_{j+r}$  are in a left block, where  $q \cong 0, r \cong 0$ , and  $j + r > i + q$ . Suppose further, that  $W(i, j + r) > 0$ .*

$$\text{If } W(i, j + r) \cong W(i, j), \text{ then } W(i + q, j + r) \cong W(i + q, j).$$

The purpose of the strict inequality in Theorem 6 is to establish the positivity of  $W(i, j + r)$ . This reformulation can be proved from analogous versions of Lemmas 2-5. The alternative version of Theorem 6 allows the requirement that  $i_0$  be maximum in Corollary 8 to be replaced by the condition that it be maximum if  $\max_{i \in I} W(i, j_0) = 0$ . This reformulation, when combined with its analogue to Corollary 7, is also sufficient to give divide-and-conquer schemes for computing the separation.

**6. General wiring.** This section contains an  $O(n \log n)$  time algorithm to solve the separation problem for wiring schemes having the Partitioning Property. In particular, wiring rules defining similar concave separation barriers will be shown to satisfy this property.

**THEOREM 9.** *Suppose  $W(i, j)$  is a separation function which satisfies the Partitioning Property, and for any  $i$  and  $j$ ,  $W(i, j)$  can be computed in unit time. Then the separation can be found in  $O(n \log n)$  time.*

*Proof.* Observe that  $P$  and  $Q$  can be partitioned into their left and right blocks in linear time, and the separation problem can be solved independently for each block. Hence it suffices to assume that  $P$  and  $Q$  constitute one left block.

Let  $j = \lceil n/2 \rceil$ , and find  $i_{\lceil n/2 \rceil}$ , which is the largest  $i \leq \lceil n/2 \rceil$  which maximizes  $W(i, \lceil n/2 \rceil)$ . Corollary 8 ensures that the maximum separation is among those separations restricted to the intervals  $i \in [1, i_{\lceil n/2 \rceil}]$  and  $j \in [1, \lceil n/2 \rceil]$ , or  $i \in [i_{\lceil n/2 \rceil}, n]$ , and  $j \in [\lceil n/2 \rceil + 1, n]$ . Repeating this divide-and-conquer step of setting  $j$  to be each  $Q$ -subinterval midpoint and allowing  $i$  to vary over the corresponding  $P$ -subinterval requires  $\lceil \log(n+1) \rceil$  levels and uses a total of  $O(n \log n)$  comparisons. The separation is then  $\max_j W(i_j, j)$ .  $\square$

Theorems 6 and 9 show that the separation problem can be solved in  $O(n \log n)$  time for barrier families satisfying P0, P1, and P2. We now show that families of similar separation barriers have these properties. A family of geometrically similar separation barriers is defined by its first barrier. For a concentric family emanating from the origin, the similarity requirement says that

$$(6.1) \quad H_\rho(x) = \rho H_1\left(\frac{x}{\rho}\right).$$

Unrestricted curvilinear wiring, rectilinear wiring on an integer grid, and rectilinear plus 45-degree wiring on a half-integer grid, for example, have barrier families which are similar, concentric, and concave. It should be noted that we shall assume the function  $H_1$ , (and hence  $H_\rho$ ) can be computed in  $O(1)$  time. This is certainly the case if, for example,  $H_1$  represents a barrier that is circular, or that is defined piecewise by a fixed collection of functions, each of which can be evaluated in  $O(1)$  time. It then follows that  $W(i, j)$  can be computed in constant time.

**THEOREM 10.** *Let  $W$  be defined by a family of similar separation barriers  $H_\rho(x) = \rho h(x/\rho)$ , where  $h(x) = 0$  for  $x \geq 1$ , and  $h(x)$  is concave and nonincreasing for  $x \leq 1$ . Then the separation problem can be solved in  $O(n \log n)$  time.*

*Proof.* It suffices to show that  $H_\rho$  satisfy P0, P1, and P2.

P0: These properties are trivially satisfied.

P1: Figure 6.1 shows  $H_\rho$ ,  $H_{\rho-1}$ , and  $l(\rho, y)$ . By similarity,  $H_\rho$  and  $H_{\rho-1}$  have parallel tangents at their intersections with the ray from the origin through  $(y, H_\rho(y))$ .<sup>4</sup> By concavity of  $H_{\rho-1}$ , the left endpoint of  $l(\rho, y)$  must have a left tangent with slope

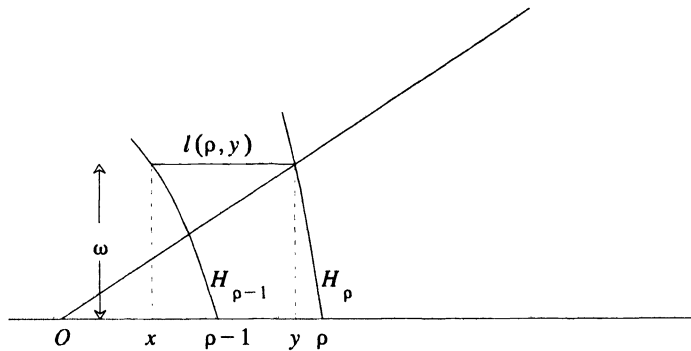


FIG. 6.1

<sup>4</sup> A more precise statement includes the fact that the separation curves might not have tangents at a denumerable number of points. Concave curves are not everywhere differentiable, but are everywhere left differentiable and right differentiable. Define the left tangent of  $H_\rho$  at  $x$  to be the line through  $(x, H_\rho(x))$  with a slope as large (close to 0) as possible which has no point  $(z, H_\rho(z))$  for  $z \leq \rho$  on its right. This slope equals the left derivative of  $H_\rho$  at  $x$ . More precisely, then, the ray from the origin to  $(y, H_\rho(y))$  has parallel left tangents at its intersection points with  $H_{\rho-1}$  and  $H_\rho$ .

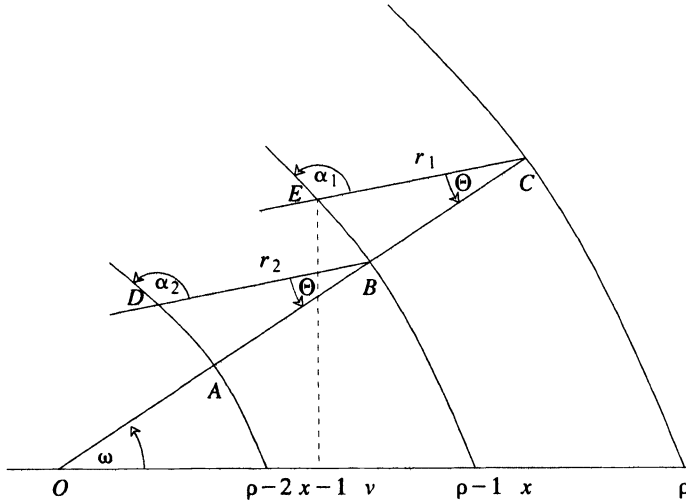


FIG. 6.2

no less (i.e., no steeper) than the slope of the left tangent at  $y$ . Consequently  $|l(\rho, y)|$  cannot decrease as  $y$  decreases. Formally, let  $l$  have endpoints  $(x(w), w)$  and  $(y(w), w)$ . Then

$$\frac{d}{dw}|l| = \frac{d}{dw}(y(w) - x(w)) = \frac{1}{H'_\rho(y)} - \frac{1}{H'_{\rho-1}(x)} \geq 0.$$

P2: Let, as in Fig. 6.2, the ray from the origin intersect  $H_{\rho-2}$ ,  $H_{\rho-1}$ , and  $H_\rho$  at  $A$ ,  $B$  and  $C$ , respectively, and intersect the positive  $x$  axis at an angle  $\omega$ . Let  $D$  and  $E$  be on  $H_{\rho-2}$  and  $H_{\rho-1}$  so that  $\angle ECB = \angle DBA = \theta$ . Let  $|\overline{DB}| = r_2$ ,  $|\overline{EC}| = r_1$ .

CLAIM. The ray  $R$  which runs from the origin through  $D$  lies above  $E$ .

Proof. If triangle  $OBD$  is magnified by  $\rho/(\rho - 1)$ , with  $O$  as the fixed point, and with no rotation, then  $B$  is mapped to  $C$ , and the ray along  $\overline{BD}$  is mapped onto the ray along  $\overline{CE}$ . Ray  $R$  is mapped onto itself. Note that  $D$  is not sent to  $E$ , but instead becomes some point inside barrier  $H_{\rho-1}$  because the magnification factor is less than  $(\rho - 1)/(\rho - 2)$ .  $R$  must therefore intersect  $H_{\rho-1}$  somewhere above the line through  $C$  and  $E$ .

The same reasoning as that used for property P1 shows that  $H_{\rho-2}$  is no steeper at  $D$  than  $H_{\rho-1}$  is at  $E$ . It is tempting to conclude that  $(d/d\theta)r_2 \geq (d/d\theta)r_1$ , but this inequality need not hold. As shown in Fig. 6.2, let  $\alpha_2$  be the angle between the left tangent to  $H_{\rho-2}$  at  $D$  and  $DB$ . Let  $\alpha_1$  be the analogous angle at  $E$ . As observed above,  $\alpha_2 \geq \alpha_1$ . In fact,  $\pi > \alpha_2 \geq \alpha_1 > 0$ . Elementary trigonometry gives  $(d/d\theta)r_1 = -r_1 \cot \alpha_1$ ,  $(d/d\theta)r_2 = -r_2 \cot \alpha_2$ . When  $\alpha_2 < \pi/2$ , it is possible that  $(d/d\theta)r_2 < (d/d\theta)r_1$ . In any case,  $(dr_2/r_2 d\theta) = -\cot \alpha_2 \geq -\cot \alpha_1 = (dr_1/r_1 d\theta)$ . Integrating the differential inequality as  $\theta$  goes from 0 to  $\omega$  gives  $\log(|l(\rho - 1, \nu)|/|\overline{AB}|) \geq \log(|l(\rho, x)|/|\overline{BC}|)$ , where  $\nu$  is the  $x$ -coordinate of  $B$ , and  $x$  is the  $x$ -coordinate of  $C$ . Exponentiating shows that

$$(6.2) \quad \frac{|l(\rho - 1, \nu)|}{|\overline{AB}|} \geq \frac{|l(\rho, x)|}{|\overline{BC}|}.$$

Since the similarity properties of  $H_\rho$  guarantee that the lengths of  $\overline{AB}$  and  $\overline{BC}$  are equal,  $|l(\rho - 1, \nu)| \geq |l(\rho, x)|$ . The similarity properties also ensure that  $x - \nu = x/\rho \leq 1$ . Hence  $x - 1 \leq \nu$ , so  $|l(\rho - 1, x - 1)| \geq |l(\rho - 1, \nu)|$ , whence  $|l(\rho - 1, x - 1)| \geq |l(\rho, x)|$ .  $\square$

The explicit form of the inequality in (6.2) makes the proof directly applicable to Theorem 18.

Theorem A in the Appendix gives a direct proof that the separation function defined by similar concave barrier curves enjoys the Partitioning Property. The proof is slightly stronger since it shows quantitatively “by just how much” the Partitioning Property is satisfied, that is, it gives a formula for  $W(i + q, j + r) - W(i + q, j)$ .

COROLLARY 11. *The separation for a wiring scheme with circular barriers can be found in  $O(n \log n)$  time.  $\square$*

COROLLARY 12. *The separation for a wiring scheme with similar convex polygonal barriers can be found in  $O(n \log n)$  time.  $\square$*

**7. Weakening the similarity and curvature restrictions.** Evidently any algorithm which finds the separation for a particular wiring scheme will, with one proviso, find the separation for the same wiring scheme where the notion of distance is now redefined. The proviso is that the new measure of separation distance be a nondecreasing function  $\Phi$  of the original distance measure. Then

$$(7.1) \quad \max \Phi(W(i, j)) = \Phi(\max W(i, j)).$$

Under the map  $\Phi(x) = \frac{1}{4} \lfloor 4x \rfloor$ , rectilinear wiring on a quarter-integer grid turns out to be equivalent to wiring defined by a particular family of similar separation barriers which are composed of rectilinear plus 45-degree segments. The first barrier has a 45-degree segment connecting rectilinear pieces terminating at  $(1, \frac{3}{4})$  and  $(\frac{3}{4}, 1)$ . In concrete terms, given a separation problem for rectilinear wiring on a quarter-integer grid, one way to find the solution is by using rectilinear plus 45-degree wiring (as defined by the oversized barrier family described above) to connect  $P$  and  $Q$ . Then apply the map  $(x', y') = (x, \frac{1}{4} \lfloor 4y \rfloor)$  to each point of the wiring pictorial. The new pictorial (with added requisite vertical segments) is an optimal solution to the original problem. In particular, the separation is  $\frac{1}{4} \lfloor 4w \rfloor$ , where  $w$  is the separation found for the particular rectilinear plus 45-degree wiring. See Fig. 7.1. It should be noted that this barrier family is oversized so that the resulting rectilinear barriers are separated by distances of at least 1.

We have proven the following.

COROLLARY 13. *The separation problem for rectilinear wiring on a quarter-integer grid can be solved in  $O(n \log n)$  time.  $\square$*

For completeness, it should be noted that rectilinear plus 45-degree wiring, in this model, is the same for quarter-integer and for half-integer grids; the separation barriers turn out to be identical for the two cases because of the need for neighboring barriers

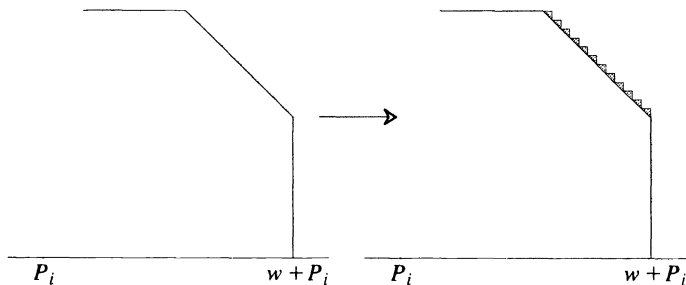


FIG. 7.1

to be separated by at least unit distance. Similarly, one-third and one-half integer grid rectilinear wiring schemes turn out to be the same as integer grid wiring.

It is clear that the separation problem can be solved in  $O(n \log n)$  time for any separation problem that can be decomposed into a fixed number of subfamilies which satisfy the Partitioning Property. The decomposition should be of the form  $W(i, j) = \max_k W_k(i, j)$  where  $W_k(i, j)$  is the separation induced by the  $k$ th subfamily. In addition, if there is a strictly monotone map  $\Phi$  which transforms one barrier subfamily into another, then the two subfamilies induce separation problems having the same time complexity.

The Partitioning Property holds for some barrier families which fail to satisfy P0, P1, and P2. When all terminals are located on a quarter-integer grid, for example, the separation function defined by rectilinear wiring on a quarter-integer grid satisfies the Partitioning Property. Evidently there ought to be criteria for barrier subfamilies which induce separation problems solvable in  $O(n \log n)$  time, and the criteria seemingly ought to be invariant over monotone maps. However, the Partitioning Property need not hold for a quarter-integer grid rectilinear wiring scheme if the terminals are not located on gridpoints. More generally, if  $W$  has the Partitioning Property, and  $\Phi$  is nondecreasing, but not necessarily strictly increasing, then  $\Phi(W)$  only satisfies a Weak Partitioning Property: if  $\Phi(W(i, j+r)) > \Phi(W(i, j))$  then  $\Phi(W(i+q, j+r)) \cong \Phi(W(i+q, j))$ . It is not evident how this weaker property can be used to give an  $O(n \log n)$  time algorithm to find the maximum of  $\Phi(W)$ . Of course  $\max \Phi(W)$  can be found efficiently by applying Theorem 9 to  $W$ .

The inadequacy of the Weak Partitioning Property is made more precise in the following.

**THEOREM 14.** *Let  $W$  be the separation function defined by separation barriers composed of quarter-integer rectilinear segments. Then the worst-case behavior of any optimal algorithm to find  $\max_{i,j} W(i, j)$  based solely on the values of  $W$  is  $\Theta(n^2)$  comparisons.*

*Proof.* It suffices to construct a family of separation barriers that has the Weak Partitioning Property, and that satisfies the conclusion of this theorem. We use the barrier family defined by quarter integer rectilinear segments, but allow terminal locations to be on a finer grid. The barrier function for a left block is  $H_\rho(x) = \lfloor 4\rho h(x/\rho) \rfloor / 4$ , where  $h(x) = 1$  if  $x < 3/4$ ,  $h(x) = 7/4 - x$  if  $3/4 \leq x < 1$ , and  $h(x) = 0$  if  $x \geq 1$ .

Let  $P_i = (7/4)i$  and  $Q_i = (7/4)i - (3/4)n + 1/8$  for  $i = 1, 2, \dots, n$ . Then the solution to the separation problem is  $W(1, n) = (3/4)n - 1/4$ . In fact, for all  $i$  and  $j$  where  $1 \leq i, j \leq n$ , and  $j - i \geq (3/4)n$ ,  $W(i, j) = (1/4) \lfloor 4((7/4)(j - i) - Q_j + P_i) \rfloor = (1/4) \lfloor 4((3/4)n - 1/8) \rfloor = (3/4)n - 1/4$ . Pick any  $i_0$  and any  $j_0$  where  $j_0 - i_0 \geq (3/4)n$ . Change the problem by letting  $P_{i_0} = (7/4)i_0 + 3/32$  and  $Q_{j_0} = (7/4)j_0 - (3/4)n + 1/32$ . Call the separation function induced by this new problem  $\tilde{W}(i, j)$ . Then  $W(i, j) = \tilde{W}(i, j)$  if  $i \neq i_0$  or  $j \neq j_0$ . Finally,  $\tilde{W}(i_0, j_0) = (3/4)n$ . Therefore no algorithm can determine if a given separation function is  $W$  rather than a  $\tilde{W}$  without testing  $\Omega(n^2)$  pairs of  $i$  and  $j$ . Clearly  $n^2$  is a trivial upper bound on the number of comparisons necessary to find the separation.  $\square$

It should be noted that the separation problem for quarter-integer rectilinear separation barriers can be solved in linear time; see [Si2] for the details.

Evidently properties P0, P1, and P2 ought to be extended to characterize a class of wiring schemes which is invariant under strictly increasing maps. The proof technique for Theorem 6 shows the following.

THEOREM 15. Let  $H_\rho$  satisfy:

Q0a:  $H_\rho(x) = 0$  for  $x \geq \rho$ .

Q0b:  $H_\rho(x) > H_{\rho-1}(x)$  for  $x < \rho$ .

Q0c:  $H_\rho(x) \geq H_\rho(y)$  for  $x \leq y \leq \rho$ .

Q0d: If  $x < y < \rho$  and  $H_\rho(x) = H_\rho(y)$ , then  $H_\rho(v) = H_\rho(y)$  for  $v \leq y$ . In other words, if  $H_\rho$  is flat at  $y$ , then it must be constant to the left of  $y$ .

Q1: If  $x \leq y \leq \rho$ , then  $|l(\rho, x)| \geq |l(\rho, y)|$ .

Q2: If  $x \leq \rho$ , then  $|l(\rho - 1, x - 1)| \geq |l(\rho, x)|$ .

Then the separation function  $W$  defined by barriers  $H_\rho$  satisfies the Partitioning Property.  $\square$

**8. A better wiring model.** All constructions considered so far implicitly permit wires to terminate at  $P$  or  $Q$  in a horizontal direction. If, for example, the wiring in the integer rectilinear case is required to leave rows  $P$  and  $Q$  vertically for at least one unit in length, then the physical separation  $w$  is

$$w = \begin{cases} 1 & \text{if } P_i = Q_i \text{ for all } i, \\ 2 + \max W(i, j) & \text{otherwise.} \end{cases}$$

Figure 8.1 illustrates rectilinear wiring with this restriction. The purpose is to avoid

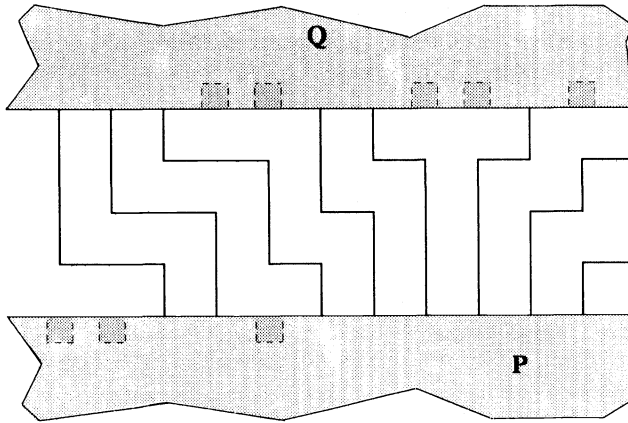


FIG. 8.1. A physical separation—rectilinear wiring avoiding internal wires.

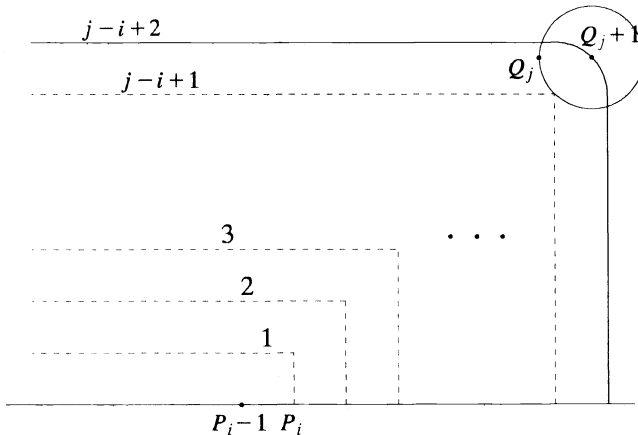


FIG. 8.2. Barrier modifications.

unknown wires inside modules  $P$  and  $Q$  (the black boxes denote possible internal wires).

More generally, we may imagine, as shown in Fig. 8.2, wiring internal to modules  $P$  and  $Q$  located (for left blocks) one unit to the left of  $P_i$  and to the right of  $Q_j$ . Now the separation induced by terminals  $P_i$  and  $Q_j$  is influenced by a first barrier around  $P_i - 1$  and a final point at  $Q_j + 1$ . The separation function for a left block is then<sup>5</sup>

$$(8.1) \quad W(i, j) = \begin{cases} \max \{1, \hat{H}_{j+2-i}(Q_j + 2 - P_i)\} & \text{if } j \geq i, \\ 0 & \text{if } j < i. \end{cases}$$

The reason the curves  $\hat{H}_\rho$  are different from the  $H_\rho$ 's is that the imaginary wire terminating at  $Q_j + 1$  need not obey the wiring limitations. Its only requirement is it must be at least one unit away from the preceding wire. Consequently its curve is described by the locus of points one unit beyond the wireable barrier curve  $H_{j+1-i}(x)$  emanating from  $P_i - 1$ . For polygonal wiring, the locus is a new polygonal barrier (corresponding in some sense to  $H_{j+2-i}$ ) with corners replaced by tangential circular arcs of radius 1. For families of similar concave wiring barriers, the unit distance locus curves are still concave but no longer similar. The Partitioning Property, however, is nevertheless applicable.

**THEOREM 16.** *Let  $H_\rho$  be the locus of points unit distance beyond barrier  $K_{\rho-1}$ ,  $\rho = 2, 3, \dots$ , where the  $K_\rho$ 's are a family of similar barriers satisfying P0. Then the separation function defined by the  $H_\rho$  satisfies the Partitioning Property.*

*Proof.* It suffices to show that  $H_\rho$  satisfies P0, P1, and P2. The proof, of course, is similar to the proof of Theorem 10.

P0: These properties are trivially satisfied.

P1: Figure 8.3 shows  $K_{\rho-2}$ ,  $H_{\rho-1}$ ,  $K_{\rho-1}$ , and  $H_\rho$ . Let  $A, B, C, D, E,$  and  $F$  be as shown. Let  $\overline{AC}$ ,  $\overline{BD}$  be of unit length and perpendicular to  $H_{\rho-1}$  and (to some local support line of)  $K_{\rho-2}$ .  $\overline{EF}$  is parallel to  $\overline{AC}$ , of unit length, and perpendicular to  $H_\rho$  and (to some local support line of)  $K_{\rho-1}$ .  $A$  and  $E$  are on a ray from the origin, and  $\overline{AE}$  intersects the  $x$ -axis at a nonzero angle.  $D$  and  $F$  are on a horizontal segment;  $l = \overline{DF}$ . Since  $\overline{EF}$  has nonnegative slope,  $F$  is at least as high as  $E$ .  $E$  is above  $A$ , and

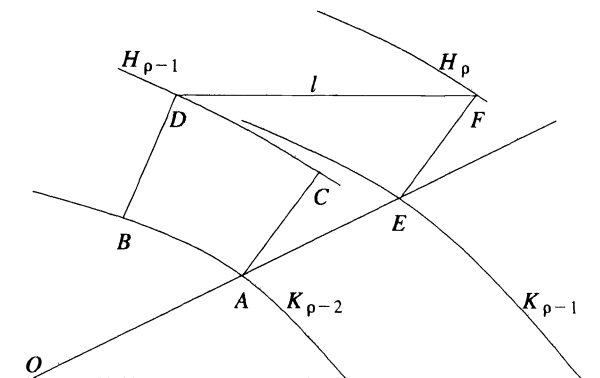


FIG. 8.3

<sup>5</sup> We have assumed that the presence of unknown wires on other layers within modules  $P$  and  $Q$  prevents the modules from being separated by less than unit distance. It is not difficult to include other minimum separation requirements in this model, or to include additional separation requirements due to interactions between different layers.



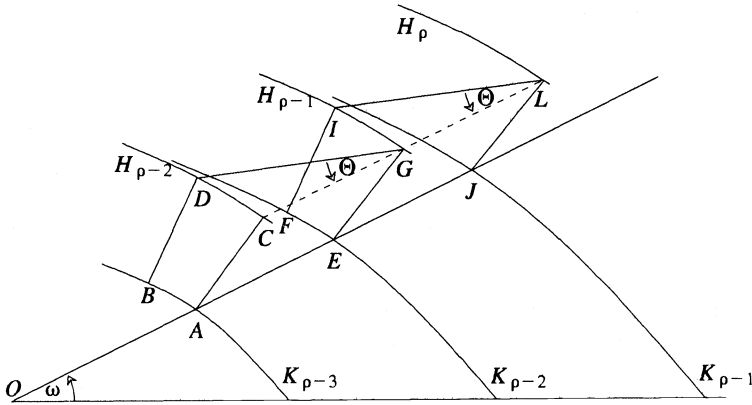


FIG. 8.4

since  $\overline{AC}$  is parallel to  $\overline{EF}$ ,  $D$  is above  $C$ . By P0d,  $\overline{AC}$  cannot cross  $\overline{BD}$ , so  $B$  must be above  $A$ . Hence  $\overline{BD}$  must have a slope at least as large as  $\overline{AC}$  by P0d. Thus  $H_{\rho-1}$  is no steeper at  $D$  than  $H_{\rho}$  at  $F$ . Consequently  $|I|$  cannot decrease as  $F$  moves up  $H_{\rho}$ .

P2: Let  $A, B, C, D, E, F, G, I, J,$  and  $L$  be as shown in Fig. 8.4.  $K_{\rho-3}, H_{\rho-2}, K_{\rho-2}, H_{\rho-1},$  and  $K_{\rho-1}, H_{\rho}$  are as shown.  $A, E,$  and  $J$  are on a ray from the origin, which intersects the positive  $x$ -axis at an angle  $\omega$ . The symmetry of  $K_{\rho}$  and the orthogonality of unit segments  $\overline{AC}, \overline{EG},$  and  $\overline{JL}$  ensure that  $C, G,$  and  $L$  are a translation of  $A, E,$  and  $J$  by  $\overline{AC}$ .  $\overline{BD}$  and  $\overline{FI}$  are unit perpendiculars such that  $B$  is on  $K_{\rho-3}, D$  is on  $H_{\rho-2}, F$  is on  $K_{\rho-2},$  and  $I$  is on  $H_{\rho-1}$ . Let  $\angle DGC = \angle ILG = \theta$ . Then the ray from the origin through  $B$  lies above  $F$ . Consequently  $H_{\rho-1}$  is at least as steep at  $I$  as  $H_{\rho-2}$  is at  $D$ . Now  $|\overline{CG}| = |\overline{GL}|$ . Integrating, over  $\theta \in [0, \omega]$ , the same kind of differential inequality as in the proof of Theorem 10 gives  $\log(|\overline{GD}|/|\overline{GC}|) \geq \log(|\overline{LI}|/|\overline{LG}|)$ , whence P2 must hold.  $\square$

**9. Modeling wire thickness.** More accurate wiring models should also include wire thickness. Evidently the solutions to the separation problem are applicable if, say, the barrier curves represent the outer edges of densely packed wires with positive thickness. If, as illustrated in Fig. 9.1, these curves  $H_{\rho}$  are similar, and concave, the wire thickness is  $2\mu$ , and the minimum separation space between wires is 1, the separation function still has the Partitioning Property. The separation algorithm for these cases need not be changed. These results also apply to barrier families which model a wire as the set of points within a fixed distance  $\mu$  of a skeleton curve. In particular, if the skeleton curves are similar and convex, Theorem 16 is applicable.

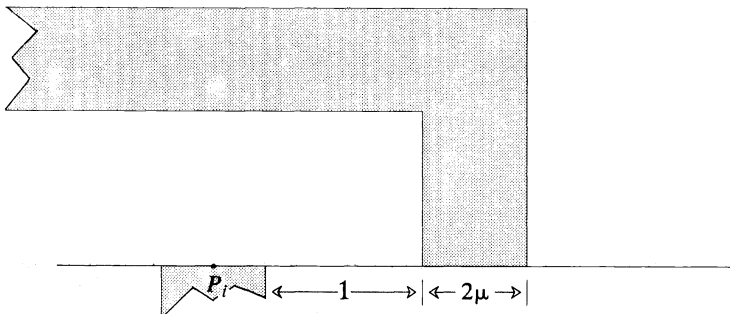


FIG. 9.1. A model including wire thickness.

Separation barriers might also be defined by similar subfamilies when wire thickness depends on the power load. This case is best handled by using real barrier and terminal indices. Let the terminal indices be  $\rho_1, \rho_2, \dots, \rho_n$ , where the  $\rho_i$ 's are real. Let the thickness of wire  $\rho_i$  be  $\rho_i - \rho_{i-1} - 1$ , where  $\rho_0$  is set to 0. Then if the thicknesses are zero, the indices  $\rho_i$  are just 1, 2,  $\dots$ ,  $n$ . The physical location of a "thick" terminal  $P_{\rho_i}$  is the right endpoint of the connection interval for the terminal. This location value is assigned to  $P_{\rho_i}$ . The unit spacing requirement for terminals is formalized by

$$(9.1) \quad P_{\rho_i} - P_{\rho_{i-1}} \geq \rho_i - \rho_{i-1}, \quad Q_{\rho_i} - Q_{\rho_{i-1}} \geq \rho_i - \rho_{i-1}.$$

Let  $l(i, x)$  be the horizontal line segment bounded on the left by  $H_{\rho_{i-1}}$  and with right endpoint  $(x, H_{\rho_i}(x))$ . We define the following properties for the separation barriers of a left block:

- R0a:  $H_{\rho}(x) = 0$  for  $x \geq \rho$ .
  - R0b:  $H_{\rho}(x) > H_{\delta}(x)$  for  $\delta < \rho, x < \rho$ .
  - R0c:  $H_{\rho}(x) \geq H_{\rho}(y)$  for  $x \leq y \leq \rho$ .
  - R0d:  $H_{\rho}(x + \varepsilon) - 2H_{\rho}(x) + H_{\rho}(x - \varepsilon) \leq 0$  for  $\varepsilon > 0, x + \varepsilon \leq \rho$ .
  - R1: If  $x \leq y \leq \rho$ , then  $|l(\rho, x)| \geq |l(\rho, y)|$ .
  - R2: If  $x \leq \rho_i$ , then  $|l(\rho_{i-1}, x - \rho_i + \rho_{i-1})| / (\rho_{i-1} - \rho_{i-2}) \geq |l(\rho_i, x)| / (\rho_i - \rho_{i-1})$ .
- This is just the direct generalization of P2.

Suppose, for example, that  $P_{\rho_i}$ 's barriers emanate from location  $P_{\rho_i}$ , and the indexing for these barriers is given by  $H_{\rho_{i+1}-\rho_i}, H_{\rho_{i+2}-\rho_i}, H_{\rho_{i+3}-\rho_i}, \dots$ . Then the proofs of Lemmas 2, 3, 4, 5 and Theorem 6 show the following.

**THEOREM 17.** *Let  $\{P, Q\}$  be a separation problem satisfying (9.1), and  $H_{\rho}$  the separation barriers satisfying R0, R1, and R2. Then the separation function  $W$  defined by the separation barriers has the Partitioning Property.  $\square$*

If the routing wires must avoid unknown wires inside modules  $P$  and  $Q$ , then  $P_{\rho_i}$ 's barrier family will have an origin at  $P_{\rho_i} - \rho_i + \rho_{i-1}$ . The barriers will be the locus of points unit distance beyond the similar family  $K_{\rho_i - \rho_{i-1}}, K_{\rho_{i+1} - \rho_{i-1}}, K_{\rho_{i+2} - \rho_{i-1}}, \dots$ , which emanates from  $P_{\rho_i}$ 's family origin. The separation function for a left block will be given by

$$(9.2) \quad W(i, j) = \begin{cases} \max \{1, H_{\rho_{j+1}-\rho_{i-1}}(Q_{\rho_j} + 1 - P_{\rho_i} + \rho_i - \rho_{i-1})\} & \text{if } j \geq i, \\ 0 & \text{if } j < i. \end{cases}$$

It is worth noting that if the wire thickness is zero, i.e., if  $\rho_j = j$ , then (9.2) reduces to (8.1).

**THEOREM 18.** *Let  $H_{\rho}$  be the locus of points unit distance beyond the similar barrier family  $K_{\rho-1} = (\rho - 1)k(x / (\rho - 1))$  for  $\rho \geq 2$ , where  $k(x)$  is nonincreasing and concave for  $x \leq 1$ , and  $k(x) = 0$  for  $x \geq 1$ . Then for terminals satisfying (9.1) the separation function  $W$  defined in (9.2) by the  $H_{\rho}$  has the Partitioning Property.  $\square$*

Theorem 15 can also be extended to the case of wires with thicknesses.

Notice that the wiring models have in a sense come around full circle; in the most realistic models of the separation problem as characterized by Theorem 18, the "terminal origins" are idealized points, the wiring barriers have zero width and shapes which cannot be fabricated, and the barrier indices may not even be integer valued.

**10. Appendix.** The idea behind the Partitioning Theorem is essentially the following. Consider two line segments which connect  $Q_{j_1}$  with  $P_{i_1}$  and  $Q_{j_2}$  with  $P_{i_2}$ . Suppose  $i_1$  and  $i_2$ , respectively, maximize  $W(*, j_1)$  and  $W(*, j_2)$ . The Partitioning Property then says that the segments cannot cross. Figure 10.1 illustrates this result in the context of Theorem A.

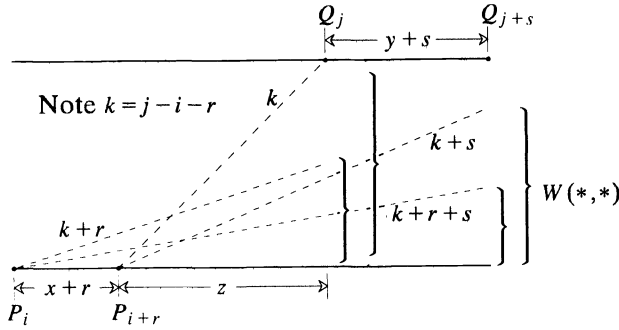


FIG. 10.1. A left-block separation. Variables in the proof are as shown. Variables between the horizontal lines are barrier number ( $\rho$ ) while  $x, y, z, r,$  and  $s$  are measures of distance. Note that only  $z$  can be negative. We note that  $k$  must be positive because of the supposition that the term  $W(i+r, j)$ , in (10.1), is positive.

The following is a direct proof that the Partitioning Property holds for a separation function defined by similar concave barriers. Although not as general as the techniques of § 5, the proof may give better insight into the behavior of separation barriers and their intrinsic properties.

**THEOREM A.** *Let  $W(i, j)$  be the separation function defined by similar concave separation barriers  $H_\rho$ , where  $H_1 = h(x)$ . Suppose  $P_i, P_{i+r}, Q_j,$  and  $Q_{j+s}$  are in a left block, where  $r \geq 0, s \geq 0,$  and  $j+s > i+r$ .*

$$(10.1) \quad \text{If } W(i, j+s) > W(i, j), \text{ then } W(i+r, j+s) > W(i+r, j).$$

*Proof.* Let

$$x = P_{i+r} - P_i - r, \quad y = Q_{j+s} - Q_j - s, \quad z = Q_j - P_{i+r}, \quad k = j - i - r.$$

The wire separation restrictions imply that  $x \geq 0$  and  $y \geq 0$ .

Now suppose that all four terms in (10.1) are positive. It suffices to show that

$$\{W(i, j) - W(i, j+s)\} - \{W(i+r, j) - W(i+r, j+s)\} \geq 0.$$

Substituting the definition for  $W$  gives

$$\begin{aligned} \text{L.H.S.} = & \left\{ (j-i)h\left(\frac{Q_j - P_i}{j-i}\right) - (j+s-i)h\left(\frac{Q_{j+s} - P_i}{j+s-i}\right) \right\} \\ & - \left\{ (j-i-r)h\left(\frac{Q_j - P_{i+r}}{j-i-r}\right) - (j+s-i-r)h\left(\frac{Q_{j+s} - P_{i+r}}{j+s-i-r}\right) \right\}. \end{aligned}$$

This expression is just a weighted difference of  $h$  at four points.

If  $x$  and  $y$  are zero,

$$(10.2) \quad \begin{aligned} \text{R.H.S.} = & \left\{ (k+r)h\left(\frac{z+r}{k+r}\right) - (k+r+s)h\left(\frac{z+r+s}{k+r+s}\right) \right\} \\ & - \left\{ kh\left(\frac{z}{k}\right) - (k+s)h\left(\frac{z+s}{k+s}\right) \right\}. \end{aligned}$$

Suppose  $h$  is twice differentiable. Then the concavity of  $h$  says that  $h''(x) \leq 0$  for  $x < 1$ . The Fundamental Theorem of Calculus says

$$\int_0^s f'(v) \, dv = f(s) - f(0).$$

Using it to express each inner difference in (10.2) as the integral of a derivative gives:

$$\begin{aligned} \text{R.H.S.} &= - \int_0^s \left\{ h\left(\frac{z+r+v}{k+r+v}\right) + \frac{k-z}{k+r+v} h'\left(\frac{z+r+v}{k+r+v}\right) \right\} dv \\ &\quad + \int_0^s \left\{ h\left(\frac{z+v}{k+v}\right) + \frac{k-z}{k+v} h'\left(\frac{z+v}{k+v}\right) \right\} dv \\ (10.3) \quad &= - \int_0^s \left\{ \left[ h\left(\frac{z+r+v}{k+r+v}\right) - h\left(\frac{z+v}{k+v}\right) \right] \right. \\ &\quad \left. + (k-z) \left[ \frac{1}{k+r+v} h'\left(\frac{z+r+v}{k+r+v}\right) - \frac{1}{k+v} h'\left(\frac{z+v}{k+v}\right) \right] \right\} dv. \end{aligned}$$

Applying the Fundamental Theorem of Calculus to (10.3) gives

$$\begin{aligned} \text{R.H.S.} &= - \int_0^r \int_0^s \left\{ \frac{k-z}{(k+u+v)^2} h'\left(\frac{z+u+v}{k+u+v}\right) \right. \\ &\quad \left. + (k-z) \left[ \frac{-1}{(k+u+v)^2} h'\left(\frac{z+u+v}{k+u+v}\right) \right. \right. \\ (10.4) \quad &\quad \left. \left. + \frac{k-z}{(k+u+v)^3} h''\left(\frac{z+u+v}{k+u+v}\right) \right] \right\} du \, dv \\ &= - \int_0^r \int_0^s \left\{ \frac{(k-z)^2}{(k+u+v)^3} h''\left(\frac{z+u+v}{k+u+v}\right) \right\} du \, dv. \end{aligned}$$

The concavity of  $h$  and the nonnegativity of  $r, s$ , and  $k$  ensure that (10.4) is nonnegative.

However,  $x, y$  can be positive. The complete proof uses auxiliary points which correspond to  $\tilde{P}_i = P_i + x$  and  $\tilde{Q}_{j+s} = Q_{j+s} - y$ . There follows:

$$\begin{aligned} &W(i, j) - W(i, j+s) - W(i+r, j) + W(i+r, j+s) \\ &= \left\{ (k+r)h\left(\frac{z+r}{k+r}\right) + \left[ -(k+r)h\left(\frac{z+r}{k+r}\right) + (k+r)h\left(\frac{z+r+x}{k+r}\right) \right] \right. \\ &\quad \left. - (k+r+s)h\left(\frac{z+r+s}{k+r+s}\right) \right. \\ &\quad \left. + \left[ (k+r+s)h\left(\frac{z+r+s}{k+r+s}\right) - (k+r+s)h\left(\frac{z+r+s+x+y}{k+r+s}\right) \right] \right. \\ &\quad \left. - kh\left(\frac{z}{k}\right) + (k+s)h\left(\frac{z+s}{k+s}\right) - \left[ (k+s)h\left(\frac{z+s}{k+s}\right) - (k+s)h\left(\frac{z+s+y}{k+s}\right) \right] \right\} \end{aligned}$$

The inner terms (in square brackets) are differences, and the remaining terms are exactly as in (10.2). The Fundamental Theorem of Calculus and (10.4) give

$$\begin{aligned}
 \text{R.H.S.} &= \int_0^x \left\{ h' \left( \frac{z+r+\nu}{k+r} \right) \right\} d\nu \\
 &\quad + \int_0^{y+x} \left\{ -h' \left( \frac{z+r+s+\nu}{k+r+s} \right) \right\} d\nu + \int_0^y \left\{ h' \left( \frac{z+s+u}{k+s} \right) \right\} du \\
 &\quad - \int_0^r \int_0^s \left\{ \frac{(k-z)^2}{(k+u+\nu)^3} h'' \left( \frac{z+u+\nu}{k+u+\nu} \right) \right\} du d\nu. \\
 (10.5) \quad &= \int_0^x \left\{ h' \left( \frac{z+r+\nu}{k+r} \right) - h' \left( \frac{s+z+r+\nu}{s+k+r} \right) \right\} d\nu \\
 &\quad + \int_0^y \left\{ h' \left( \frac{z+s+u}{k+s} \right) - h' \left( \frac{x+r+z+s+u}{r+k+s} \right) \right\} du \\
 &\quad + \int_0^r \int_0^s \left\{ \frac{-(k-z)^2}{(k+u+\nu)^3} h'' \left( \frac{z+u+\nu}{k+u+\nu} \right) \right\} du d\nu.
 \end{aligned}$$

We have already seen that the last term in (10.5) is nonnegative. The Mean Value Theorem says that  $f(b) - f(a) = f'(\xi)(b - a)$  for some  $\xi$  where  $\min(a, b) \leq \xi \leq \max(a, b)$ . It follows that the first two integrals in (10.5) can each be replaced by expressions of the form  $\int h''(\xi) \Delta \arg d\nu$ . It remains to show that  $\Delta \arg$  is not positive, that is,

$$\left( \frac{z+r+\nu}{k+r} \right) - \left( \frac{s+z+r+\nu}{s+k+r} \right) \leq 0 \quad \text{and} \quad \left( \frac{z+s+u}{k+s} \right) - \left( \frac{x+r+z+s+u}{r+k+s} \right) \leq 0.$$

The assumption that  $W(i, j)$  and  $W(i+r, j+s)$  in (10.1) are positive says that  $(k+r)h((z+r+x)/(k+r))$  and  $(k+s)h((z+s+y)/(k+s))$  are positive. Since  $h(t) = 0$  for  $t \geq 1$ , it follows that  $(z+r+x)/(k+r) < 1$ , and  $(z+s+y)/(k+s) < 1$ . The denominators are positive. Hence  $(z+r+\nu)/(k+r) < 1$  for  $0 \leq \nu \leq x$ , and  $(z+s+u)/(k+s) < 1$  for  $0 \leq u \leq y$ . Since the fractions are less than one, adding a nonnegative quantity such as  $r$  or  $s$  to both the numerator and denominator of  $((z+r+\nu)/(k+r))$  or  $((z+s+u)/(k+s))$  cannot decrease their values. An additional  $x$  in the numerator has the same result. These operations give  $((s+z+r+\nu)/(s+k+r)) \geq ((z+r+\nu)/(k+r))$  and  $((x+r+z+s+u)/(r+k+s)) \geq ((z+s+u)/(k+s))$ . Consequently each term in (10.5) is nonnegative.

There remains the possibility that some of the terms in (10.1) could be equal to zero. These cases are trivial.

It should be noted that the derivation and the nonnegativity of (10.5) hold even if  $h$  is not twice differentiable. As explained in, say, [CH], even discontinuous functions possess generalized derivatives which are usually called distributional derivatives. The Dirac delta function  $\delta(x) = (d/dx)H(x)$ , where  $H(x) \equiv (0 \text{ if } x < 0 \text{ else } 1)$ , for example, is a distributional derivative (which happens to be nonnegative). The concavity requirement for separation barriers ensures that on  $(-\infty, 1]$ ,  $h'' \leq 0$  in the distributional sense, and this is sufficient for our purposes. Alternatively, it suffices to observe that Theorem A is essentially a statement about four points on a concave curve, and any such four points can be interpolated by a smooth concave function  $h$ .  $\square$

## REFERENCES

- [AHU] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison Wesley, Reading, MA, 1974.
- [CH] R. COURANT AND D. HILBERT, *Methods of Mathematical Physics*, Vol. II, Interscience, New York, 1966, pp. 766-788.
- [DKSSU] D. DOLEV, K. KARPLUS, A. SIEGEL, A. STRONG, AND J. D. ULLMAN, *Optimal wiring between rectangles*, Proc. 13th Annual ACM Symposium on the Theory of Computing, 1981, pp. 312-317.
- [FP] M. J. FISCHER AND M. S. PATERSON, *Optimal tree layout*, Proc. 12th Annual ACM Symposium on the Theory of Computing, 1980, pp. 177-189.
- [GJ] M. GAREY AND D. JOHNSON, *Computers and Intractability: A Guide to NP-Completeness*, Freeman, San Francisco, 1978.
- [GKT] L. J. GUIBAS, H. T. KUNG, AND C. THOMPSON, *Direct VLSI implementation of combinatorial algorithms*, Caltech Conf. on VLSI, January, 1979.
- [J] D. JOHANNSEN, *Bristle blocks: A silicon compiler*, Caltech Conf. on VLSI, January 1979, pp. 303-310. See also 16th Design Automation Proceedings, June 1979, pp. 310-313.
- [L] A. S. LAPAUGH, *A polynomial time algorithm for optimal routing around a rectangle*, Proc. 21st Annual IEEE Symposium on Foundations of Computer Science, 1980, pp. 282-293.
- [LP] C. LEISERSON AND R. PINTER, *Optimal placement for river routing*, Carnegie-Mellon Conference on VLSI Systems and Computations, Computer Science Press, Rockville, MD, October 1981, pp. 126-142.
- [Lat] B. LATTIN, *VLSI design methodology*, Caltech Conf. on VLSI, January 1979.
- [M] A. MIRZAIAN, *River Routing in VLSI*, J. Comp. Syst. Sci., 34, 1987, pp. 43-54.
- [MC] C. MEAD AND L. CONWAY, *Introduction to VLSI Systems*, Addison Wesley, Reading, MA, 1980.
- [S] M. SHAMOS, *Geometry and statistics*, in Algorithms and Complexity, J. F. Traub, ed. Academic Press, New York, 1976, pp. 256-259.
- [SD] A. SIEGEL AND D. DOLEV, *The separation for general single-layer wiring barriers*, Carnegie-Mellon Conference on VLSI Systems and Computations, Computer Science Press, Rockville, MD, October 1981, pp. 143-152.
- [Si1] A. SIEGEL, *Fast optimal placement for river routing*, in preparation.
- [Si2] ———, *The separation problem for river routing with polygonal wires*, in preparation.
- [Si3] ———, *The optimal offset problem for river routing*, in preparation.
- [St] J. A. STORER, *The node cost measure for embedding graphs on the planar grid*, Proc. 12th Annual ACM Symposium on the Theory of Computing, 1980, pp. 201-210.
- [T1] M. TOMPA, *An optimal solution to a wire-routing problem*, J. Comput. System Sci., (1980), pp. 127-150.
- [T2] ———, Private communication.
- [V] L. VALIANT, *Universality considerations in VLSI circuits*, IEEE Trans. Comput., (1981), pp. 135-140.

## RELATIONS BETWEEN CONCURRENT-WRITE MODELS OF PARALLEL COMPUTATION\*

FAITH E. FICH†, PRABHAKAR RAGDE‡, AND AVI WIGDERSON‡

**Abstract.** Shared memory models of parallel computation (e.g., parallel RAMs) that allow simultaneous read/write access are very natural and already widely used for parallel algorithm design. The various models differ from each other in the mechanism by which they resolve write conflicts. To understand the effect of these communication primitives on the power of parallelism, we extensively study the relationship between four such models that appear in the literature, and prove nontrivial separations and simulation results among them.

**Key words.** parallel computation, lower bounds, parallel random access machines

**AMS(MOS) subject classification.** 68Q10

**1. Introduction.** Parallel computation has been the object of intensive study in recent years. Many models of synchronous parallel computation have been proposed. One important model is the CRCW PRAM (concurrent-read concurrent-write parallel random access machine, sometimes denoted WRAM). Not only have numerous algorithms been designed for the CRCW PRAM (examples include [Ga], [KMR], [SV], and [TV]), but it has also been shown to be closely related to unbounded fan-in circuits and alternating Turing machines ([CSV], [LY2]).

Specifically, a CRCW PRAM consists of a set of processors (i.e., random access machines)  $P_1, P_2, \dots, P_n$  together with a shared memory. One step consists of three phases. In the read phase, every processor may read one shared memory cell. In the compute phase, every processor may perform computation. In the write phase, every processor may write into one shared memory cell. Any number of processors can simultaneously read from the same memory cell, and any number may attempt to simultaneously write into the same memory cell.

An arbitrary amount of computation will be allowed in each compute phase. Although this is unrealistic, it enables us to concentrate on communication between processors. For all the problems we consider, communication rather than computation is the limiting factor. In fact, the algorithms presented in this paper actually perform very little computation at each step. Furthermore, the powerfulness of the model makes the lower bounds we present very strong.

A fundamental question concerning CRCW PRAMs is how to resolve write conflicts. One method is to assign priorities to processors and, if more than one processor attempts to write to the same memory cell, then the one with the highest priority will succeed. Without loss of generality (by reordering processors), we can assume that priorities are assigned in order of processor index, with highest priority given to the processor of lowest index [Go]. We call this the PRIORITY model.

Other mechanisms for conflict resolution appear in the literature. In the ARBITRARY model, if more than one processor attempts to write to the same memory cell, an arbitrary one will succeed [V]. Algorithms for the ARBITRARY model must

---

\* Received by the editors October 7, 1986; accepted for publication (in revised form) April 22, 1987. This work was supported by National Science Foundation grants MCS-8120790, MCS-8402676, and ECS-8110684, Defense Advanced Research Projects Agency contract N00039-82-C-0235, an IBM Faculty Development Award, the University of Washington Graduate School Research Fund, and a Natural Sciences and Engineering Research Council of Canada Postgraduate Scholarship.

†Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4.

‡The Hebrew University, Jerusalem, Israel.

work regardless of who wins the competition to write at each step. The COMMON model allows simultaneous writes to the same memory cell only if all processors doing so are writing a common value [Ku].

When more than one processor attempts to write to the same memory cell in the COLLISION model, a special collision symbol will appear in that cell. No information is given about which processors were involved in the collision nor what values they were trying to write. This write-conflict resolution scheme is a synchronous version of that used by Ethernet and other multiple access channels [Gr].

Write conflicts can also be avoided by not allowing them; in the concurrent-read exclusive-write (CREW) PRAM, at most one processor can attempt to write to a given memory cell at each time step [FW]. An even more restrictive model is the exclusive-read exclusive-write (EREW) PRAM, in which both reads and writes are restricted in this manner.

Any algorithm that runs on the ARBITRARY model will run unchanged on the PRIORITY model; if an algorithm works regardless of who wins a competition to write, then it will certainly work if the processor of lowest index always wins. Thus the PRIORITY model is at least as powerful as the ARBITRARY model. Similarly, the ARBITRARY model is at least as powerful as the COMMON model, the COMMON and COLLISION models are at least as powerful as the CREW PRAM, and the CREW PRAM is at least as powerful as the EREW PRAM.

One step of the COLLISION model can be simulated by two steps on the ARBITRARY model, using the same number of processors and shared memory cells. First, each processor in the ARBITRARY model writes where the corresponding processor in the COLLISION model wrote. However, it writes its index in addition to the value originally written. Then each processor reads from the cell to which it has just written. If the index written there is not its own, a collision must have occurred during the previous write step. In this case, the processor writes the collision symbol to the memory cell.

Our aim is to understand the relative power of these models. Algorithms running on these models have appeared in the literature, and their expositions often include attempts to implement them on the most restrictive model possible. Such attempts are of little value without knowing which of the inclusions described above are strict.

Cook, Dwork, and Reischuk have shown that the CREW PRAM is strictly less powerful than the CRCW PRAM. In particular, their work [CDR] shows that the  $n$ -way OR function, which can be computed in one step on the COMMON model, requires  $\Omega(\log n)$  steps using a CREW PRAM. By considering the problem of searching in a sorted list of distinct elements, Snir [S] has shown that the EREW PRAM is strictly less powerful than the CREW PRAM.

In this paper, we obtain separation results for the four CRCW models as a function of the number of shared memory cells  $m$  (called the *communication width* [VW]) when the number of processors is held fixed at  $n$ . This is an important restriction, since one step on the PRIORITY model is easily simulated by two steps on the COMMON or COLLISION model if the number of processors is squared and sufficient common memory is allowed [Ku]. When width is restricted, however, the four models are not equivalent. Restricting width has a meaning in a practical as well as theoretical sense; a bus or a satellite relay may be considered to be a CRCW PRAM with width 1.

Table 1 summarizes our results on simulations and separations. A particular model is denoted by its name followed by the number of shared memory cells in parentheses (e.g., COMMON(1)). The time bound given is the number of steps on the weaker machine required to simulate one step on the more powerful machine. All logarithms are to the base 2. The results in §2 and §3 are, for the most part, easy



adversary arguments; those in the remaining sections are harder and more revealing. Among the results we consider particularly significant is an information-theoretic lower bound for computation on COMMON(1) which is applicable in a more general setting (Theorem 6). The characterization of the global state of information proven in that theorem also allows us to prove a surprising constant time simulation of COMMON(1) by COLLISION(1) (Theorem 10).

TABLE 1

Simulated Machines	Simulating Machines	Time Bounds	Sections
PRIORITY(1)	ARBITRARY( $m$ ) COLLISION( $m$ ) COMMON( $m$ )	$\Theta\left(\frac{\log n}{\log(m+1)}\right)$	2
PRIORITY( $m$ )	ARBITRARY( $m$ ) COLLISION( $m$ ) COMMON( $m$ )	$O(\log n)$	2
PRIORITY( $m$ ) $m = O(n/c)$	ARBITRARY( $cm$ ) COLLISION( $cm$ ) COMMON( $cm$ )	$O\left(\frac{\log n}{\log(c+1)}\right)$	2
ARBITRARY(1)	COLLISION( $m$ ) COMMON( $m$ )	$\Theta\left(\frac{\log n}{\log(m+1)}\right)$	3
PRIORITY( $km$ ) ARBITRARY( $km$ ) COLLISION( $km$ )	COMMON( $m$ )	$\Omega(k \log(n/km))$	4
PRIORITY( $km$ )	ARBITRARY( $m$ )	$O\left(\frac{k \log n}{\log(k+1)}\right)$ $\Omega\left(\frac{k \log(n/km)}{\log(k+1)}\right)$	6
COLLISION(1)	COMMON( $m$ )	$\Theta\left(\frac{\log n}{\log(m+1)}\right)$	2,5
COMMON(1)	COLLISION( $m$ )	$\Theta\left(\frac{\log n}{\log(m+1)}\right)$	2,5
COMMON(1) on domain $\{0, 1\}^n$	COLLISION(1)	$O(1)$	5

New lower bound techniques are developed to obtain the results below. We consider this work as another step (following [S], [CDR], and [VW]) in forming a foundation of lower bound techniques for parallel computation. Also, as our lower bounds concern the communication between processors, we believe these techniques may be applied to distributed (asynchronous) computation as well (e.g., in the Ethernet model). Recently, results have been obtained using more powerful techniques for models with infinite shared memory ([FMW], [MW]) and an infinite number of processors [B]. Li and Yesha ([LY1],[LY2]) have extended many of these results to models with the input in read-only memory (ROM) and have proved other results on this related model.

**2. Simulating PRIORITY(1) by weaker models.** Let us consider how to simulate one step of an algorithm for PRIORITY( $m$ ) on a machine with a weaker write

conflict resolution method, but with at least as much shared memory. Each processor in the  $\text{PRIORITY}(m)$  machine will be simulated by one processor in the simulating machine. Likewise, the contents of each shared memory cell in the  $\text{PRIORITY}(m)$  will appear in a specific shared memory cell. Simulation of the read phase is trivial. However, in the write phase on the simulating machine, processors must know if they are the processor of lowest index writing into the cell that they wish to write into. This requires some extra computation and leads to the definition of the following problem.

**m-colour MINIMIZATION.**

Before: Each processor  $P_i$ , for  $i = 1, \dots, n$ , has a colour  $x_i \in \{0, \dots, m\}$  known only to itself.

After: Each processor  $P_i$  knows the value  $a_i$ , where

$$a_i = \begin{cases} 1 & \text{if for all } j < i, x_i \neq x_j \text{ and } x_i > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Thus  $a_i = 1$  if and only if  $P_i$  is the processor of lowest index with colour  $x_i$  and  $x_i \neq 0$ .

In the simulation,  $x_i$  represents the memory cell into which the simulated processor  $P_i$  wishes to write;  $x_i = 0$  if  $P_i$  does not wish to write. Once the problem is solved,  $P_i$  will write if and only if  $a_i = 1$ , thus resolving the write conflict in the fashion that  $\text{PRIORITY}$  machine would.

Clearly, the  $m$ -colour  $\text{MINIMIZATION}$  problem takes only one step to solve on  $\text{PRIORITY}(m)$ .

**THEOREM 1.** *On  $\text{COMMON}(m)$ , the 1-colour  $\text{MINIMIZATION}$  problem can be solved in  $O(\frac{\log n}{\log(m+1)})$  steps.*

*Proof.* Without loss of generality, we may assume  $m \leq \sqrt{n}$ . If  $m > \sqrt{n}$  we only use the first  $\sqrt{n}$  cells of memory. This is because with  $m = \sqrt{n}$  we already achieve  $O(1)$  running time.

Throughout the algorithm, memory cells will contain only 1's and 0's. Note that for the 1-colour  $\text{MINIMIZATION}$  problem,  $x_i \in \{0, 1\}$ . We call the processor of lowest index whose colour is 1 the *winner*.

The algorithm repeatedly performs the following sequence of steps. First, all shared memory cells are set to 0 by having processor  $P_i$ , for  $i = 1, \dots, m$ , write 0 into cell  $M_i$ . The processors are divided into  $m + 1$  nearly equal groups, where each group contains a set of consecutively numbered processors. The first  $n \bmod (m + 1)$  groups contain  $\lceil \frac{n}{m+1} \rceil$  processors and the rest contain  $\lfloor \frac{n}{m+1} \rfloor$ . A processor  $P_i$  in the  $j$ th group, where  $1 \leq j \leq m$ , will write 1 into  $M_j$  if and only if  $x_i = 1$ . At this point, if all memory cells are unchanged (i.e., contain the value 0), the winner is in the  $(m + 1)$ st group; otherwise it is in the group corresponding to the memory cell of lowest index containing a 1.

We note that a processor does not have to know which group contains the eventual winner, only whether its group wins. The following subroutines are used to decide which of the above two cases holds, in constant time.

**LEFTMOST ONE IN MEMORY.**

Before: Cells  $M_i$ , for  $i = 1, \dots, m$ , each contain 1 or 0.

After:  $M_i$  contains 1 if and only if all  $M_j$  for  $j < i$  were initially 0, and  $M_i$  was initially 1.

Procedure: Processor  $P_i$  forms the ordered pair  $(j, k)$  from its name by setting  $j \leftarrow (i \bmod m) + 1$  and  $k \leftarrow i - m(j - 1)$ . If  $j < k \leq m$  and  $M_j$  contains 1,  $P_i$  writes 0 into  $M_k$ .

**EMPTY MEMORY (m-way OR).**

Before: Cells  $M_i$ , for  $i = 1, \dots, m$ , each contain 1 or 0.

After:  $M_1$  contains 0 if and only if all  $M_i$  were initially 0.

Procedure: Processor  $P_i$  will read  $M_i$  and, if it contains 1,  $P_i$  writes 1 into  $M_1$ .

After the LEFTMOST ONE IN MEMORY algorithm is applied, the processors in group  $i$  look at  $M_i$  to see if they are in the winning group or not (depending on whether  $M_i$  contains 1 or 0, respectively). Note that this algorithm uses  $m^2$  processors; the assumption  $m \leq \sqrt{n}$  ensures that sufficient processors are available. Application of EMPTY MEMORY will then allow the  $(m + 1)$ st group to decide if it is the winning group or not, by looking at  $M_1$ .

All processors except the ones in the winning group set  $a_i = 0$  and stop; the ones in the winning group repeat the above procedure with  $n$  replaced by the size of the group. This continues until the size of the winning group is equal to 1; at this point, the winner is determined.

Intuitively, the algorithm cuts the size of the winning group by a factor of  $m + 1$  each time. More precisely, if  $g_t$  is the size of the set of processors that may still be the winner after the  $t$ th step, then

$$g_0 = n \quad \text{and} \\ g_t \leq \left\lceil \frac{g_{t-1}}{m+1} \right\rceil.$$

If  $T \geq \log n / \log(m + 1)$ , then  $g_T \leq \lceil \frac{n}{(m+1)^T} \rceil \leq 1$ . Thus the algorithm takes at most  $\lceil \frac{\log n}{\log(m+1)} \rceil$  iterations. Since each iteration takes a constant number of steps, we have the desired upper bound.  $\square$

**COROLLARY 1.1.** *On ARBITRARY( $m$ ) and COLLISION( $m$ ), the 1-colour MINIMIZATION problem can be solved in  $O\left(\frac{\log n}{\log(m+1)}\right)$  steps.*

*Proof.* The algorithm described in the proof of Theorem 1 will run on ARBITRARY( $m$ ). It will also run on COLLISION( $m$ ) provided that, before performing LEFTMOST ONE IN MEMORY, each processor  $P_i$  for  $i = 1, \dots, m$ , reads memory cell  $M_i$  and, if  $M_i$  contains the collision symbol, writes 1 into  $M_i$ .  $\square$

It follows that ARBITRARY( $m$ ), COLLISION( $m$ ), and COMMON( $m$ ) can simulate one step of PRIORITY(1) in  $O\left(\frac{\log n}{\log(m+1)}\right)$  steps. One cell of the simulating machine (say  $M_1$ ) is designated as being equivalent to the single cell of PRIORITY(1). Processor  $P_i$  then sets  $x_i = 1$  if it wishes to write at that step, and sets  $x_i = 0$  otherwise. The algorithm for 1-colour MINIMIZATION is then followed, with the following change: the value 0 is replaced by the value presently in  $M_1$ , and the value 1 is replaced by some other value. At the end, the winner writes into  $M_1$  the value that the corresponding processor would have written in the PRIORITY(1) algorithm.

**COROLLARY 1.2.** *COMMON( $m$ ), COLLISION( $m$ ), and ARBITRARY( $m$ ) can simulate one step of PRIORITY( $m$ ) in  $O(\log n)$  steps.*

*Proof.* The write conflict resolution problems for each of the  $m$  memory cells of the PRIORITY machine can be treated as separate 1-colour MINIMIZATION problems that are solved simultaneously.

In the write conflict resolution problem for memory cell  $M_j$ , let  $x_i = 1$  if the simulated processor  $P_i$  wishes to write into memory cell  $M_j$ ; otherwise, let  $x_i = 0$ . This subproblem is solved via the algorithm described in Theorem 1 or Corollary 1.1, using the single cell  $M_j$ .

In the algorithm, a processor only writes if its colour is 1. Each processor will have colour 1 for at most one of the subproblems, namely the subproblem corresponding to where the processor it is simulating wished to write. In fact, this is the only subproblem that the processor can win. Thus the processor can ignore the rest of the subproblems. Although many subproblems are being solved simultaneously, each processor is required to participate in the solution of at most one.  $\square$

**COROLLARY 1.3.** *If  $m = O(n/c)$ , then COMMON( $cm$ ), COLLISION( $cm$ ), and ARBITRARY( $cm$ ) can simulate one step of PRIORITY( $m$ ) in  $O(\frac{\log n}{\log(c+1)})$  steps.*

*Proof.* As in the proof of Corollary 1.2, the write conflict resolution problems for each of the  $m$  memory cells of the PRIORITY machine can be treated as separate 1-colour MINIMIZATION problems. Each of these  $m$  problems will be solved simultaneously, with  $c$  memory cells devoted to each problem.

Essentially, the algorithm described in Theorem 1 or Corollary 1.1 is used. However, the allocation of processors is done somewhat differently for the computation of LEFTMOST ONE IN MEMORY and EMPTY MEMORY. The  $n$  processors are divided into groups of  $c$  consecutively numbered processors. Since  $m = O(n/c)$ , each group is responsible for the solution of a constant number of subproblems.

Unfortunately, the LEFTMOST ONE IN MEMORY algorithm requires  $c^2$  processors. Instead, each processor reads one of the  $c$  cells of memory and then they use the 1-colour MINIMIZATION algorithm again to decide which processor read the lowest index cell containing the value 1. This takes a constant number of steps, since the number of memory cells and the number of processors are the same.  $\square$

Unlike the proof of Corollary 1.2, it is not sufficient to have only those processors that wish to write into memory cell  $M_j$  participate in the solution of the 1-colour MINIMIZATION problem associated with  $M_j$ . There may be far fewer than  $c$  processors wanting to write into some cells. Even if there were a sufficient number of processors wanting to write into each shared memory cell, a processor would not necessarily know the identities of the other processors working with it. Therefore, it would not know what to do during the execution of the LEFTMOST ONE IN MEMORY algorithm.

When  $m = O(n^{1-\epsilon})$  for some constant  $\epsilon > 0$ , choosing  $c = n^\epsilon$  in Corollary 1.3 provides us with constant time simulations of PRIORITY( $m$ ), without increasing the number of processors.

The following lower bound shows that the algorithm in Theorem 1 is optimal, to within a constant factor. The proof uses a fairly simple adversary argument, which we present in detail, as it serves as a paradigm for subsequent proofs.

**THEOREM 2.** *The 1-colour MINIMIZATION problem requires at least  $\frac{\log(n+1)-1}{\log(m+1)}$  steps to solve on ARBITRARY( $m$ ).*

Consider an algorithm solving this problem. An *input* to the algorithm is a specification of the values of all the colours  $x_i$ . (Recall that  $x_i \in \{0, 1\}$ .) We will use an adversary argument, constructing an input on which the algorithm requires this many steps.

The write action of a particular processor  $P_i$  at step  $t$  depends only on  $x_i$  and the sequence of contents  $(H_0, H_1, \dots, H_{t-1})$  of the  $m$  shared memory cells. Here  $H_i$  is a specification of the contents of memory cells  $M_1, M_2, \dots, M_m$  immediately before step  $i + 1$ . We call this sequence the *history* through time  $t$ . Given a fixed history, we say  $P_i$  writes into  $M_j$  on value  $v$  if it attempts to write to memory cell  $M_j$  when  $x_i = v$ .

At each step we will “fix” the values of certain input variables and maintain a set of *allowable* inputs. The set of *allowable* inputs will consist of those inputs in which each fixed variable has the value to which it was fixed. This will be done in a manner such that all *allowable* inputs will produce the same history up to that step.

The term *position* is used to denote the index of an input variable. A position  $i$  is said to be fixed to a certain value if the corresponding variable  $x_i$  is fixed to that value. If the variable  $x_i$  is not fixed, the position  $i$  is said to be *free*. We will maintain a set  $S$  of fixed positions and a set  $F$  of free positions such that  $S \cup F = \{1, \dots, n\}$ . Associated with each position in  $S$  will be the value to which that position is fixed. Furthermore, we will ensure that, for each free position in  $F$ , there will be no lower numbered positions fixed to 1.

Suppose there are at least two free positions  $i$  and  $j$  after some step  $T$ , and  $i < j$ . Consider the following two inputs:  $I_{ij}$ , in which  $x_i$  and  $x_j$  are the only variables in free positions that equal 1, and  $I_j$ , in which  $x_j$  is the only variable in a free position that equals 1. Both these inputs put  $P_j$  in the same state at the end of step  $T$ , since in both  $P_j$  sees the same input value and the same history. But, for  $I_j$ ,  $a_j = 1$ , and for  $I_{ij}$ ,  $a_j = 0$ . Thus  $P_j$  cannot know the value  $a_j$  after the  $T$ th step. We can conclude that, when the algorithm terminates, there is at most one free position.

Initially,  $S = \phi$ ,  $H_0$  is the initial contents of memory, and the conditions stated above are satisfied. Now suppose that, after the  $t$ th step, we have fixed a set of positions  $S$  so that all allowable inputs produce the same history  $(H_0, H_1, \dots, H_t)$  through step  $t$ . Furthermore, for any free position in  $F$ , there is no lower numbered position fixed to 1. Let  $f_t$  denote the number of free positions remaining after step  $t$ .

We determine  $H_{t+1}$  by fixing certain free positions, as follows.

- 1) If position  $i$  is fixed before step  $t + 1$  and processor  $P_i$  writes to some memory cell  $M_j$  at step  $t + 1$ , then the contents of  $M_j$  in  $H_{t+1}$  can be fixed by declaring  $P_i$  to win the competition to write into  $M_j$ . Notice that we do not need to fix any additional free positions in this case.
- 2) For all cells  $M_j$  into which some processor writes on 0 at step  $t + 1$  given history  $(H_0, \dots, H_t)$ , choose one processor  $P_{i_j}$  doing so, fix position  $i_j$  to 0, and declare  $P_{i_j}$  to win the competition to write into  $M_j$  at step  $t + 1$  for all inputs consistent with  $S$ . This fixes the contents of  $M_j$  in  $H_{t+1}$  to a unique value.
- 3) Suppose there are  $r$  memory cells not taken care of above. Processors only write on value 1 into these remaining cells. Let  $f$  be the number of remaining free positions. Divide these free positions into  $r + 1$  nearly equal groups; the first group will contain the lowest  $\lfloor \frac{f}{r+1} \rfloor$  positions, the second the next  $\lfloor \frac{f}{r+1} \rfloor$  positions, and so on. The last  $f \bmod (r + 1)$  groups will contain  $\lceil \frac{f}{r+1} \rceil$  free positions. For each such cell  $M_j$ , let  $P_{i_j}$  denote the processor of *highest* index writing on 1 into  $M_j$  and associate this cell with the group containing  $P_{i_j}$ . Since there are  $r$  cells and  $r + 1$  groups, at least one group will have no cells associated with it. Let  $G$  be such a group and suppose that it comprises free positions  $k$  through  $l$ . The idea is that, for each memory cell, by either forcing no processor to write to it or forcing the highest index processor wishing to write to it to do so, we can provide *no information* about group  $G$ .
  - i) Fix all free positions with index less than  $k$  to 0. Consider any cell  $M_j$  associated with a group consisting of free positions less than  $k$ . All processors that write into  $M_j$  at step  $t + 1$  only do so on 1. However, they will have all had their colours  $x_i$  set to 0. Hence, no processor will write into these cells at step  $t + 1$  for any allowable input and the contents of these cells in  $H_{t+1}$  will remain as they were in  $H_t$ .

- ii) Fix all free positions with index greater than  $l$  to 1. For any cell  $M_j$  associated with a group consisting of free positions greater than  $l$ , we declare  $P_{i_j}$  to win the competition to write into  $M_j$  at step  $t+1$  for every allowable input. This fixes the contents of  $M_j$  in  $H_{t+1}$  to a unique value.
- 4) The remaining cells have no processor writing into them on any allowable input at step  $t+1$ . The contents of these cells in  $H_{t+1}$  will be the same as they were in  $H_t$ .

Intuitively, the number of free positions is cut down by at most a factor of  $m+1$  at each step. More precisely, if  $f_t$  is the number of free positions at time  $t$ , then  $f \geq f_t - (m-r)$  and

$$f_{t+1} \geq \min_{0 \leq r \leq m} \left\lfloor \frac{f_t - m + r}{r + 1} \right\rfloor = \left\lfloor \frac{f_t}{m + 1} \right\rfloor.$$

Let  $T$  be the total number of steps taken by the algorithm. Since  $f_0 = n$  and  $f_T \leq 1$ , it follows that  $T \geq \frac{(\log(n+1)-1)}{\log(m+1)}$ .  $\square$

It has been shown [Ra] that  $\Omega\left(\frac{\log n}{\log(m+1)}\right)$  steps are required to solve 1-colour MINIMIZATION on ARBITRARY( $m$ ) even if processors are allowed to make random choices to determine their behaviour at each step.

**3. Simulating ARBITRARY(1) by weaker models.** The preceding section demonstrated a separation between PRIORITY(1) and ARBITRARY( $m$ ). We can show a similar separation between ARBITRARY(1) and COMMON( $m$ ) and between ARBITRARY(1) and COLLISION( $m$ ) by considering the following problem.

**$m$ -colour REPRESENTATIVE.**

Before: Each processor  $P_i$ , for  $i = 1, \dots, n$  has a colour  $x_i \in \{0, \dots, m\}$  known only to itself.

After: Each processor  $P_i$  knows the value  $a_i \in \{0, 1\}$ , where  $a_i = 1$  for exactly one processor among those with each particular nonzero colour  $c$ .

Notice that the  $m$ -colour REPRESENTATIVE problem requires only one step on ARBITRARY( $m$ ).

**THEOREM 3.** *On COMMON( $m$ ) or COLLISION( $m$ ), the 1-colour REPRESENTATIVE problem can be solved in  $O\left(\frac{\log n}{\log(m+1)}\right)$  steps.*

Theorem 3, in fact, follows easily from Corollary 1.1, as any solution to the 1-colour MINIMIZATION problem is also a solution to the 1-colour REPRESENTATIVE problem. The following theorem provides the separation between ARBITRARY(1) and COMMON( $m$ ).

**THEOREM 4.** *On COMMON( $m$ ), the 1-colour REPRESENTATIVE problem requires at least  $\frac{\log n}{\log(m+1)}$  steps to solve.*

*Proof.* The proof is similar to that of Theorem 2; here we merely sketch the differences. As before, we maintain a set  $S$  of fixed positions, a set  $F$  of free positions, and  $H_t$ , a specification of the contents of the  $m$  memory cells after step  $t$ . In this proof, we only fix positions to the value 0, but we do not allow the all-zero input. All other consistent inputs are allowed. The number of free positions is initially  $n$ .

Suppose there are at least two free positions  $i$  and  $j$  after some step  $T$ . Consider the following three inputs:  $I_i$ , in which  $x_i$  is the only variable that has value 1,  $I_j$ , in which  $x_j$  is the only variable that has value 1, and  $I_{ij}$ , in which  $x_i$  and  $x_j$  are the

only variables that have value 1. Both  $I_i$  and  $I_{ij}$  put  $P_i$  in the same state at the end of step  $T$ , since in both  $P_i$  sees the same input value and the same history. Similarly,  $I_j$  and  $I_{ij}$  both put  $P_j$  in the same state at the end of step  $T$ . For  $I_i$ ,  $a_i = 1$  and for  $I_j$ ,  $a_j = 1$ . If step  $T$  is the last step of the algorithm, then, for  $I_{ij}$ ,  $a_i = a_j = 1$ . This contradicts the fact that the algorithm solves the 1-colour REPRESENTATIVE problem. We can conclude that, when the algorithm terminates, there is at most one free position.

Given the set of fixed positions  $S$  and the history  $(H_0, H_1, \dots, H_t)$ , through time  $t$  we need to show how to fix some free positions in a way that defines  $H_{t+1}$ . Those memory cells into which no processors write on either 0 or 1 retain their values. Memory cells that are written into by some processor on 0 (including any processors in a fixed position) are handled as in the proof of Theorem 2.

The difficulty occurs for those cells into which processors write only on 1. Let the set of indices of such cells be denoted  $C$ . For  $j \in C$ , let  $W_j$  be the set of processors in the remaining free positions that, at step  $t + 1$ , write on 1 into  $M_j$  and let  $W_0$  be those that do not write on 1 at this step. Note that none of these processors write into  $M_j$  on 0 for any  $j \in C$ .

Now, the number of remaining free positions is at least  $|F| - (m - |C|)$ . Thus, for some  $j \in C \cup \{0\}$ , it follows that

$$|W_j| \geq \frac{|F| - m + |C|}{|C| + 1} \geq \frac{|F|}{m + 1}.$$

For one such  $j$ , we fix the colours of all processors not in  $W_j$  to 0. This defines the contents of all memory cells  $M_k$  with  $k \in C - \{j\}$ . Specifically, their contents in  $H_{t+1}$  remain as they were in  $H_t$ .

Finally, when  $j \in C$ , consider the processors in  $W_j$ . There is an allowable input in which all these processors have colour 1 and, thus, they must all write the same value on 1. Since we prohibited the all 0 input, all allowable inputs result in at least one processor in  $W_j$  receiving an input value that causes it to write at this step. Thus  $H_{t+1}$  is determined, at the cost of cutting down the number of free positions by at most a factor of  $m + 1$ . As before, we can define a recurrence bounding the number of free positions at step  $t$  and conclude that Theorem 4 is true.  $\square$

A lower bound of  $\Omega\left(\frac{\log n}{\log(m+1)}\right)$  steps holds for the 1-colour REPRESENTATIVE problem on COMMON( $m$ ) even if processors are allowed to make random choices [Ra]. The following theorem provides an analogous separation between ARBITRARY(1) and COLLISION( $m$ ).

**THEOREM 5.** *On COLLISION( $m$ ), the 1-colour REPRESENTATIVE problem requires at least  $\frac{\log(n+1) - \log 3}{\log(m+1)}$  steps to solve.*

*Proof.* The proof of this theorem is very similar to that of Theorems 2 and 4. We must define the set of consistent inputs to be those with at least two 1's in free positions, in order to enable us to fix collisions in a history. It is not difficult to reason that, as long as there are at least three positions left free by the adversary at the end of the algorithm, there is an input on which the algorithm answers incorrectly. (The conclusion follows easily from the fact that any processor  $P_i$  cannot distinguish consistent inputs that agree on the private bit of  $P_i$ .)

In the course of fixing the history in a cell after a particular step, there are three cases to consider: where no processors write into that cell, where exactly one processor writes, and where two or more processors write. The precise details of how to fix positions are thus slightly more complicated than those of Theorem 4, but no new techniques are involved. A recurrence for the number of free positions left after  $t$  steps can be defined, and the result obtained.  $\square$

**4. A lower bound for COMMON(1) and its applications.** We say that a PRAM with one cell  $M$  of shared memory computes a surjective function  $f : \{0, 1\}^n \rightarrow R$  for some range set  $R$  if, at the beginning of the computation,  $P_i$  has the  $i$ th argument (denoted by  $x_i$ ) in its local memory and, at the end of the computation, the value of  $f(x_1, x_2 \dots x_n)$  appears in  $M$ .

Our definition can be thought of as *public computation*, since the answer must appear in shared memory. The REPRESENTATIVE and MINIMIZATION problems defined earlier can be thought of as *private computation*, since each processor is required to compute a private answer bit  $a_i$ . A good lower bound for public computation can lead to a good lower bound for private computation if  $(a_1, a_2, \dots, a_n)$  can be made public in a small number of steps. For example, in 1-colour MINIMIZATION, the unique processor with  $a_i = 1$  can take one more step and write  $i$  into  $M$ .

The following theorem gives a lower bound on the number of steps required to publicly compute any function on COMMON(1). The lower bound depends only on the number of function values that are possible.

**THEOREM 6.** *On COMMON(1), any algorithm that publicly computes a surjective function  $f : \{0, 1\}^n \rightarrow R$  requires at least  $\lceil \log_3 |R| \rceil$  steps for some input.*

It is important to notice that Theorem 6 does not apply to private computation. For example, no communication is required to privately compute the answer bits  $(a_1, \dots, a_n)$  given the input bits  $(x_1, \dots, x_n)$ , where  $a_i = x_i$  for  $i = 1, \dots, n$ . However, by Theorem 6, a linear number of steps must be performed to publically compute the identity function  $id : \{0, 1\}^n \rightarrow \{0, 1\}^n$  on COMMON(1).

Furthermore, Theorem 6 does not apply when the domain is *cleft*, that is, a proper subset of  $\{0, 1\}^n$  [Re]. Consider the case where inputs are restricted so that in all valid input, exactly one input variable has value 1 and the function to be computed is the index of that variable. This function has a range of size  $n$ , but only requires a single step to publicly compute on COMMON(1).

Although the theorem, as stated, applies to the case of a single shared memory cell, it is powerful enough to use in a more general setting.

**LEMMA 6.1.**  *$T$  steps of COMMON( $m$ ) can be simulated by  $mT$  steps of COMMON(1).*

*Proof.* Each processor in COMMON(1) keeps a picture in its local memory of what shared memory on COMMON( $m$ ) would contain at the corresponding step. The simulation of each step of COMMON( $m$ ) proceeds in phases: in the  $i$ th phase, the single cell in the simulating machine takes the role of  $M_i$  in the simulated machine. All processors that would write the value  $v$  into  $M_i$  on COMMON( $m$ ) at this step write the ordered pair  $(v, i)$  into  $M_1$  on COMMON(1). Then all processors read  $M_1$  and update the contents of  $M_i$  in their picture of the shared memory of COMMON( $m$ ). Each phase takes one step and so one step of COMMON( $m$ ) is simulated by  $m$  steps of COMMON(1).  $\square$

**COROLLARY 6.2.** *On COMMON( $m$ ), any algorithm that publicly computes a surjective function  $f : \{0, 1\}^n \rightarrow R$  requires at least  $\lceil \frac{(\log_3 |R|)}{m} \rceil$  steps for some input.*

When COMMON( $m$ ) publicly computes a function, we require the value of the function to appear in shared memory at the end of the computation, but it may be distributed among the  $m$  shared memory cells. Note that this definition of computation is particularly weak when  $m$  is large; consider the identity function, whose value is just an  $n$ -tuple consisting of the input bits. When  $m = n$ , this can be publicly computed in one step under our definition; but Beame [B] has shown that if the tuple must appear in a single cell,  $\Omega(\log n)$  steps are required on PRIORITY( $n^{O(1)}$ ).

By specifying a particular function to be computed, we can separate the ARBITRARY and COMMON models, with the separation varying as a function of the size



of shared memory.

**COROLLARY 6.3.** *Simulating one step of COLLISION( $km$ ) on COMMON(1) requires  $\Omega(km \log(n/km))$  steps; on COMMON( $m$ ), it requires  $\Omega(k \log(n/km))$  steps.*

*Proof.* Partition the input positions into  $m$  groups of size  $\lfloor n/km \rfloor$  or  $\lfloor n/km \rfloor + 1$ , and consider the function  $f$  defined on domain  $\{0, 1\}^n$  whose value is an  $km$ -tuple  $(a_1, a_2, \dots, a_{km})$  such that  $a_i = j$  if  $x_j$  is the only variable in group  $i$  with value 1 and  $a_i = 0$  if either no variable or more than one variable in group  $i$  has value 1.

This function can be publicly computed in two steps on COLLISION( $km$ ), assuming each shared memory cell is initialized to 0. In the first step, each processor  $P_i$  in group  $j$  writes  $i$  into  $M_j$  on value 1. Each processor  $P_i$ , for  $i = 1, \dots, m$ , then reads cell  $M_i$  and, if it sees the collision symbol, writes 0 into  $M_i$  at the second step.

The function  $f$  has at least  $(\lfloor \frac{n}{km} \rfloor)^{km}$  possible values. Applying Theorem 6 and Corollary 6.2 give lower bounds of  $\Omega(km \log(n/km))$  and  $\Omega(k \log(n/km))$  for COMMON(1) and COMMON( $m$ ), respectively.  $\square$

By letting  $k = 1$ , Corollary 6.3 proves a separation between COLLISION( $m$ ) and COMMON( $m$ ) when  $m = o(n)$ . In particular, when  $m = O(n^{1-\epsilon})$  for some constant  $\epsilon > 0$ ,  $\Omega(\log n)$  steps are required.

We introduce some terminology to be used in the proof of Theorem 6. The *tree of possible computations* has nodes that intuitively correspond to the different states that the PRAM can be in during the course of the computation. Formally, with each node  $v$  at depth  $t$ , we associate a history  $(H_0, H_1, \dots, H_t)$  and a set  $I_v$  consisting of all inputs that generate this history through step  $t$ . An input is said to *reach* node  $v$  if it is a member of  $I_v$ . The children of  $v$  correspond to all possible extensions to the history at  $v$ ; each child is labelled with a different extension  $(H_0, H_1, \dots, H_t, H_{t+1})$ . The last entry in the history associated with a leaf of the tree will be the function value for all inputs that reach that leaf.

The statement of the theorem has an “information theory” flavour, and if we could show that the degree of fanout in the computation tree is bounded by a constant, the result would follow easily. Unfortunately, arbitrarily high fanout is possible, as the example with cleft domain showed. The intuition behind this theorem is that a node of high fanout corresponds to a step at which many different values can be written, depending on the input. Since for a particular input, two processors may not attempt to write different values, this implies that some knowledge of “mutual exclusion” can be inferred from the history. In the example with cleft domain, the knowledge that only one processor had an input variable with value 1 allowed  $n$  different values to be written at step 1, depending on the input. We will show that this “mutual exclusion” takes time to set up and is not reusable.

With each node  $v$  in the computation tree, we can associate a formula  $f_v$  in conjunctive normal form, whose variables are the private input bits  $x_i$ . This formula will have the property that the set of inputs  $I_v$  associated with this node is exactly the set of inputs that satisfy the formula  $f_v$ . The construction of these formulas will proceed by induction on the depth of a node. Formulas will have two types of clauses: *trivial* clauses will contain exactly one literal, and *nontrivial* clauses will contain more than one literal.

For the root  $r$  of the computation tree, we define  $f_r$  to be the empty formula. Now suppose we have a node  $w$  with associated history  $(H_0, H_1, \dots, H_{t-1})$  and associated formula  $f_w$ . Suppose, furthermore, that  $w$  has a child  $v$  and that the history at  $v$  is the history at  $w$  extended by the value  $H_t$ . This means that for some inputs in  $I_w$ , the content of  $M$  after the  $t$ th step is the value  $H_t$ . The action of any processor at step  $t$  for an input in  $I_w$  is completely determined by the history through step  $t - 1$  (the history associated with  $w$ , which is the same for all inputs in  $I_w$ ) and by the

processor's private input bit  $x_i$ . Thus, it is possible to determine which private bit values would cause processors to write  $H_t$ .

For inputs that reach  $v$ , at least one processor must have a value that causes it to write  $H_t$ . Thus inputs with history  $(H_0, H_1, \dots, H_{t-1}, H_t)$  must satisfy  $f_w$  and also a clause consisting of the OR of the literals corresponding to these possible bit values. For example, if  $P_1$  writes  $H_t$  when  $x_1 = 1$ , and  $P_2$  does so when  $x_2 = 0$ , then the added clause would be  $(x_1 \vee \bar{x}_2)$ . This is the only (possibly) nontrivial clause that we add. In two cases we do not add such a clause: when one processor  $P_i$  writes regardless of what its private bit is, and when no processor writes, i.e.,  $H_t = H_{t-1}$ . Since there is only one memory cell, each processor reads its content during every read phase. Therefore, we can assume, without loss of generality, that processors write into the memory cell  $M$  only to change its value.

All possible bit values that would have resulted in something other than  $H_t$  being written will result in additional trivial clauses. For example, if  $P_3$  would have written  $H'_t$  (different from  $H_t$ ) if  $x_3 = 1$ , we add the trivial clause  $(\bar{x}_3)$ , since the fact that  $H_t$  and not  $H'_t$  was written implies that  $x_3 = 0$ . We can also substitute these known values into other clauses. In our example, a clause containing the literal  $x_3$  would have that literal removed; a clause containing the literal  $\bar{x}_3$  would be entirely removed. This simplification is crucial to our proof, as it removes nontrivial clauses.

LEMMA 6.4. *If a node  $w$  with  $q$  children has a formula  $f_w$  with  $c$  nontrivial clauses, the formula at each child of  $w$  has at most  $c + 3 - q$  nontrivial clauses.*

*Proof.* If  $q \leq 2$  this follows from the construction, as at most one nontrivial clause is added. Thus we may assume  $q > 2$ . There are  $q$  possible extensions of the computation history at this node. One of them could correspond to the case where no one writes ( $H_t = H_{t-1}$ ), but there are at least  $q - 1$  different values that could be written at the next step.

No processor may write more than one of these values, for otherwise that processor would always write, and  $w$  would have exactly two children. For each value written, we arbitrarily select one processor that writes it; assume without loss of generality that for  $i = 1, 2, \dots, q - 1$ , value  $V_i$  is written by  $P_i$  at this step if literal  $l_i$  is true. (Note that  $l_i$  is either  $x_i$  or  $\bar{x}_i$ .)

The formula  $f_w$  implies that at most one of the literals  $l_1, l_2, \dots, l_{q-1}$  is true. Otherwise, there would exist an input in  $I_w$  for which two different processors would attempt to simultaneously write different values, a violation of the COMMON model.

Now consider the formula  $f_v$  at the child  $v$  of  $w$  that corresponds to  $V_{q-1}$  being written. This is created by first adjoining one nontrivial clause to  $f$ , and also some trivial clauses as a result of the knowledge that  $l_1, l_2, \dots, l_{q-2}$  are all false. This knowledge also results in some substitutions. Let  $\beta = (\beta_1, \beta_2, \dots, \beta_n)$  be an input in  $I_v$  which makes  $l_{q-1}$  true. Since  $I_v$  is a subset of  $I_w$ , the input  $\beta$  satisfies  $f_w$  and makes each of the literals  $l_1, l_2, \dots, l_{q-2}$  false.

For  $j = 1, \dots, q - 2$ , let  $\beta^j$  be the input obtained from  $\beta$  by complementing  $\beta_j$  (i.e.,  $\beta^j$  makes both  $l_j$  and  $l_{q-1}$  true). The input  $\beta^j$  cannot satisfy  $f_w$ , because it makes two literals in  $\{l_1, l_2, \dots, l_{q-1}\}$  true. Let  $C_j$  be some clause in  $f_w$  that  $\beta^j$  does not satisfy. Since there exists an input in  $I_w$  which makes  $l_j$  true, and another that makes  $l_j$  false,  $C_j$  must be nontrivial. The only difference between  $\beta$  and  $\beta^j$  is in the value of the  $j$ th bit. Thus  $C_j$  must contain the literal  $\bar{l}_j$ . Furthermore,  $\bar{l}_j$  is the only literal in  $C_j$  that  $\beta$  makes true.

Note that  $C_j$  contains the literal  $\bar{l}_j$  and  $\beta^i$  makes  $l_j$  false. Thus  $\beta^i$  satisfies  $C_j$ , but not  $C_i$ . Hence, for  $1 \leq i < j \leq q - 2$ , the clauses  $C_i$  and  $C_j$  are distinct.

Consider the creation of  $f_v$ . The substitutions that follow from the knowledge that  $l_1, l_2, \dots, l_{q-2}$  are false will remove the nontrivial clauses  $C_i$ , for  $i = 1, \dots, q - 2$ .

Thus  $f_v$  can have at most  $c - (q - 2) + 1$  nontrivial clauses, as required. A similar argument works for the other children of  $v$ ; in fact, the child that corresponds to the case when no process writes will have at most  $c + 2 - q$  nontrivial clauses.  $\square$

The importance of Lemma 6.4 is that, although we cannot bound the degree of a node in the computation tree, high degree requires accumulating and then destroying nontrivial clauses, and only one nontrivial clause is accumulated per level. We use this idea to prove the following lemma.

LEMMA 6.5. *The maximum number of leaves in a computation tree of height  $h$  is  $3^h$ .*

*Proof.* Let  $L(s, h)$  be the maximum number of leaves in a subtree of height  $h$  whose root formula has  $s$  nontrivial clauses. By Lemma 6.4, we have

$$L(s, 1) \leq s + 3 \text{ and} \\ L(s, h) \leq \max_{2 \leq q \leq s+3} \{q \cdot L(s + 3 - q, h - 1)\}.$$

This can be shown by induction on  $h$  to satisfy the inequality  $L(s, h) \leq (3 + s/h)^h$ . The base case is obvious; suppose the statement is true for  $h < k$ . Then

$$L(s, k) \leq \max_{2 \leq q \leq s+3} \{q \cdot L(s + 3 - q, k - 1)\} \\ \leq \max_{2 \leq q \leq s+3} \left\{ q \left( 3 + \frac{s + 3 - q}{k - 1} \right)^{k-1} \right\}.$$

The quantity inside the curly brackets, considered as a function over real  $q$ , can be shown by elementary calculus to reach its maximum at  $q = 3 + s/k$ . This yields  $L(s, k) \leq (3 + s/k)^k$ , as required.  $\square$

Theorem 6 then follows from the fact that each leaf of the computation tree can be labelled with at most one function value. All function values must appear, so the tree has at least  $|R|$  leaves. By Lemma 6.5, the computation tree must have height at least  $\lceil \log_3 |R| \rceil$ .

We can extend this result and obtain a theorem similar to Theorem 6 for probabilistic algorithms. In the probabilistic COMMON model, each processor is allowed to make random choices to determine its behaviour at each step. We insist that no sequence of choices results in two processors attempting to write different values into the same cell at the same time. Theorem 7 gives a bound on the expected number of steps to compute a function in terms of the size of its range.

THEOREM 7. *In the probabilistic COMMON(1) model, any algorithm that publicly computes a surjective function  $f : \{0, 1\}^n \rightarrow R$  has an expected running time of at least  $\lceil \log_3 |R| \rceil$  steps on some input.*

As in Corollary 6.3, we obtain a logarithmic separation between the probabilistic COMMON( $m$ ) model and the deterministic ARBITRARY( $m$ ) model, for  $m = O(n^{1-\epsilon})$  where  $\epsilon$  is a positive constant. In this case, randomization does not help the COMMON model to simulate the more powerful model.

Theorem 7 is proved using the following two lemmas.

LEMMA 7.1. *The sum of the root-leaf distances to any set  $S$  of leaves in a tree of possible computations is at least  $|S| \cdot \lceil \log_3 |S| \rceil$ .*

*Proof.* Let us define a *tree skeleton* to be a tree whose nodes can be labelled with nonnegative integers, such that the root is labelled with zero, and any node labelled with  $s$  that has  $q$  children has each child labelled no higher than  $s + 3 - q$ . Lemma 6.5 is actually a statement about tree skeletons; any computation tree leads in a natural way to a tree skeleton, where the label of a node is just the number of nontrivial

clauses in its formula. Let  $S$  be our set of chosen leaves and let  $Q$  be the sum of the root-leaf distances. We can prune away everything but the root-leaf paths to leaves in  $S$ . This still leaves a tree skeleton, since deleting a node does not invalidate the labels of its siblings. The pruning also leaves  $Q$  unchanged.

We can then transform the tree skeleton in a way that will never increase the sum of the root-leaf distances to leaves in  $S$ . Suppose we can find two leaves  $v_1$  and  $v_2$ , where  $v_i$  is at depth  $t_i$  and  $t_2 - t_1 \geq 2$ . We add two children  $v'_1, v'_2$  to  $v_1$ , label them with the same number as  $v_1$ , and delete  $v_2$ . We remove  $v_1, v_2$  from  $S$  and add  $v'_1, v'_2$ .

Continuing in this fashion, we can obtain a tree skeleton and a set  $S'$  of leaves, where  $|S| = |S'|$ , and the depths of all leaves in  $S'$  differ by no more than 1. Lemma 6.5 implies that the depth of each leaf is at least  $\lfloor \log_3 |S| \rfloor$ . Since the sum of root-leaf distances to leaves in  $S'$  is less than or equal to  $Q$ , the result follows.  $\square$

LEMMA 7.2. *Let  $T_1$  be the expected running time for a given probabilistic algorithm solving problem  $P$ , maximized over all possible inputs. Let  $T_2$  be the average running time for a given input distribution, minimized over all possible deterministic algorithms to solve  $P$ . Then  $T_1 \geq T_2$ .*

Lemma 7.2 was stated by Yao [Y] in a stronger form; the weak form here can be proved in a few lines. We can consider a probabilistic algorithm as a probabilistic distribution of deterministic algorithms. Let  $A$  be our set of deterministic algorithms, and  $I$  our set of inputs. Let  $r[A_i, I_j]$  be the running time of algorithm  $A_i$  on input  $I_j$ . Suppose our given probabilistic algorithm chooses to run deterministic algorithm  $A_i$  with probability  $p_i$ , and that our given input distribution gives probability  $q_j$  to  $I_j$ . Then

$$\begin{aligned} T_1 &= \max_{I_j \in I} \left\{ \sum_{A_i \in A} p_i r[A_i, I_j] \right\} \\ &\geq \sum_{I_j \in I} q_j \sum_{A_i \in A} p_i r[A_i, I_j] \\ &= \sum_{A_i \in A} p_i \sum_{I_j \in I} q_j r[A_i, I_j] \\ &\geq \min_{A_j \in A} \left\{ \sum_{I_j \in I} q_j r[A_i, I_j] \right\} \\ &= T_2. \end{aligned} \quad \square$$

We wish to bound  $T_1$  from below. By Lemma 7.2, it suffices to bound  $T_2$  from below. To do this, we must specify an input distribution that results in a large average running time for any algorithm to compute  $f$ . This input distribution must depend on  $f$ , but not on a particular program. For each possible value of  $f$ , choose one input that results in that value. This selects a set of  $|R|$  inputs; our chosen distribution will give probability  $1/|R|$  to each of these.

To bound  $T_2$  from below for this distribution, consider the tree of possible computations associated with some deterministic algorithm. Our set of inputs reaches some set of  $|R|$  leaves. Then the expected running time on the given input distribution is the average depth of these leaves which by, Lemma 7.1, is at least  $\lfloor \log_3 |R| \rfloor$ . This proves Theorem 7.

By a more careful analysis, the base of the logarithm in the lower bounds of Theorems 6 and 7 can be reduced to  $1 + \sqrt{2}$ . It is possible to define a somewhat artificial family of functions  $\{f_i\}$  such that  $f_i$  has range  $R_i$  and can be publicly computed in  $\log_\beta |R_i| + O(1)$  steps on COMMON(1), where  $\beta = \frac{1 + \sqrt{13}}{2}$  [Ra].

**5. The relationship between COMMON and COLLISION.** It is not immediately obvious whether COMMON is at least as powerful as COLLISION, whether COLLISION is at least as powerful as COMMON, or whether the power of these two models are incomparable.

To begin with, consider the following problem.

**EXACTLY ONE.**

Before: Each processor  $P_i$ , for  $i = 1, \dots, n$ , has a bit  $x_i$  known only to itself.

After:  $M_1$  contains the value 1 if and only if exactly one bit  $x_i$  is initially one.

If  $M_1$  is initialized to 0, the EXACTLY ONE problem can be solved on COLLISION(1) by having each processor  $P_i$  write the value 1 into  $M_1$  when  $x_i = 1$ . However, in a model without the ability to detect collisions, it can take significantly longer.

**THEOREM 8.** *The EXACTLY ONE problem requires at least  $\frac{\log n}{\log(m+1)}$  steps to solve on COMMON( $m$ ).□*

The proof of this result is essentially identical to the proof of Theorem 4. It then follows that  $\Theta\left(\frac{\log n}{\log(m+1)}\right)$  steps on COMMON( $m$ ) are needed, in the worst case, to simulate one step of COLLISION(1). The lower bound comes from Theorem 8, and the upper bound follows from the simulation of ARBITRARY by COMMON given in Theorem 1 and the fact that ARBITRARY can simulate COLLISION in constant time (as observed in §1).

Conversely, consider the following problem which can be solved in a constant number of steps on COMMON(1).

**k-GROUP IDENTIFICATION.**

Before: The processors are divided into  $k$  groups of size  $s$  or  $s + 1$ , where  $s = \lfloor n/k \rfloor$ .

Each processor knows the indices of all other processors in its group. Each processor  $P_i$ , for  $i = 1, \dots, n$ , has a private bit  $x_i \in \{0, 1\}$  known only to itself. Furthermore, for  $1 \leq i, j \leq n$ , if  $x_i = x_j = 1$ , then  $P_i$  and  $P_j$  are in the same group.

After:  $M_1$  contains the value  $a \neq 0$  if and only if  $x_i = 1$  for some processor  $P_i$  in group  $a$ .  $M_1$  contains the value 0 if and only if all input variables are 0.

**THEOREM 9.** *On COLLISION( $m$ ), the  $\lfloor \sqrt{n} \rfloor$ -GROUP IDENTIFICATION problem requires  $\Omega\left(\frac{\log n}{\log(m+1)}\right)$  steps to solve.*

*Proof.* The proof is similar to that of Theorems 2,4, and 5. In addition to maintaining a set  $S$  of fixed positions, a set  $F$  of free positions, and the history  $(H_0, H_1, \dots, H_t)$  through time  $t$ , we maintain a set  $A \subseteq \{1, \dots, \lfloor \sqrt{n} \rfloor\}$  of groups whose positions have not yet been completely fixed. Positions are fixed only to 0. Allowable inputs are consistent with the fixed positions and also with the restriction mentioned in the statement of the problem.

At each step of the construction, we not only fix individual positions within groups to 0, but fix whole groups to 0 as well. It is possible to do this in such a way that if  $|A| = s$  before step  $t$ , then it has size  $\frac{s}{m+1}$  after step  $t$ . Furthermore, if  $b$  is a lower bound, for each  $a \in A$ , on the number of free positions in group  $a$ , then  $\frac{b-1}{m+1}$  is a lower bound afterwards. This is essentially done by associating with each group the cell into which the most processors in that group write on 1, and then finding the cell that is associated with the most groups. All groups not associated with that cell are fixed to

0 and removed from A; all processors in groups associated with that cell that write on 1 into different cells have their positions fixed to 0. As long as two free positions remain in two different groups, the algorithm cannot answer. Again, the details are omitted.  $\square$

The above result is somewhat unsatisfying. In an essential way, the proof depends on the fact that the  $k$ -GROUP IDENTIFICATION problem has a cleft domain. Recall, this means that the input is a proper subset of  $\{0, 1\}^n$ . Such an unrealistic assumption might never arise in the computation of a function defined on the domain  $\{0, 1\}^n$ . Indeed, the following theorem shows that COLLISION(1) is at least as powerful as COMMON(1) for computing functions on the domain  $\{0, 1\}^n$ . This simulation result uses the same structure that was used in the lower bound result of Theorem 6.

**THEOREM 10.** *The public computation of any function  $f : \{0, 1\}^n \rightarrow R$  requires at most twice as many steps on COLLISION(1) as it does on COMMON(1).*

*Proof.* We show how to convert any algorithm that publicly computes a function  $f : \{0, 1\}^n \rightarrow R$  on COMMON(1) into an algorithm, using at most twice as many steps, that works on COLLISION(1). This new algorithm simulates the original algorithm in a step by step fashion, using two steps to simulate each step of the COMMON(1) algorithm.

For any input, the COLLISION(1) algorithm will produce a history  $(H_0, H_1, \dots, H_{2t})$  through step  $2t$ , with the property that  $(H_0, H_2, \dots, H_{2t})$  is the history through step  $t$  produced by the COMMON(1) algorithm on that input. Thus, the set of inputs  $I_v$  that reach any even depth node  $v$  in the tree of possible computations for the COLLISION(1) algorithm is a subset of  $I_{v'}$  for some node  $v'$  occurring at the half the depth in the tree of possible computations for the COMMON(1) algorithm. The node  $v'$  will be called the COMMON(1) node corresponding to  $v$ .

The COLLISION(1) algorithm is obtained from the COMMON(1) algorithm as follows. At any even depth node in the COLLISION(1) algorithm, each processor  $P_i$  writes its index  $i$ , provided processor  $P_i$  writes on the input  $x_i$  at the corresponding COMMON(1) node. Without loss of generality, we will assume that all the values written in every possible execution of the COMMON(1) algorithm are distinct from the processor indices  $1, \dots, n$ . If no write takes place, then, at the next step, the processors do nothing. If the value  $i$  appears in the shared memory cell, indicating that processor  $P_i$  was the only processor attempting to write, then processor  $P_i$  writes the value it would have written in the COMMON(1) algorithm. Finally, if a collision occurred, then processor  $P_1$  writes the value that would have been written into the shared memory cell in the COMMON(1) algorithm. It remains to show that there is enough information for processor  $P_1$  to determine this value.

As in the proof of Theorem 6, the set of inputs  $I_{v'}$  that reach a node  $v'$  in the COMMON(1) tree can be described by a Boolean formula  $f_{v'}$  in conjunctive normal form. For each even depth node  $v$  in the COLLISION(1) tree, the Boolean formula  $f_{v'}$  associated with the corresponding COMMON(1) node  $v'$ , is satisfied by each input in  $I_v$ . This is because  $I_v \subseteq I_{v'}$ . There is additional information about the input that the COLLISION(1) algorithm also accumulates during the course of the simulation.

**CLAIM.** *For each (nontrivial) clause in  $f_{v'}$ , either every input in  $I_v$  satisfies at least two literals in that clause (although not necessarily the same literals for different inputs) or there is a particular literal in that clause which is true for all inputs in  $I_v$ .*

*Proof.* The proof of this claim proceeds by induction on the depth of the node  $v$ . If  $v$  is the root of the COLLISION(1) tree, then the root of the COMMON(1) tree is its corresponding node. Its associated formula, the empty formula, has no clauses. In this case, the claim is satisfied.

Now suppose  $v$  has depth  $2t$ , where  $t \geq 1$ , and assume that the claim is true for

the grandparent  $w$  of  $v$ . Let  $w'$  and  $v'$  be the COMMON(1) nodes corresponding to  $w$  and  $v$ , respectively.

Consider the history  $(H_0, H_1, H_2, \dots, H_{2t-1}, H_{2t})$  through step  $2t$  produced by the COLLISION(1) algorithm on inputs in  $I_v$ . By construction, no write occurs at step  $2t - 1$  in the COLLISION(1) algorithm (i.e.,  $H_{2t-1} = H_{2t-2}$ ) if and only if no write would have occurred at step  $t$  in the COMMON(1) algorithm. In this case, each nontrivial clause in  $f_{v'}$  is also a nontrivial clause in  $f_{w'}$ . Since  $I_v \subseteq I_w$ , there is nothing to prove.

Otherwise, node  $v'$  is the child of  $w'$  arising when the value  $H_{2t}$  is written at the  $t$ th step in the COMMON(1) algorithm. Compared to  $f_{w'}$ , the formula  $f_{v'}$  contains at most one additional nontrivial clause, consisting of the literals corresponding to those private bit values that cause processors to write the value  $H_{2t}$ .

If  $H_{2t-1}$  is the index of processor  $P_i$ , then, for every input in  $I_v$ ,  $P_i$  is the only processor that writes at step  $2t - 1$  in the COLLISION(1) algorithm. Thus, the literal (either  $x_i$  or  $\bar{x}_i$ ) causing  $P_i$  to write the value  $H_{2t}$  at step  $t$  in the COMMON(1) algorithm is true for all inputs in  $I_v$ . Notice that the additional nontrivial clause in  $f_{v'}$ , if there is one, contains this literal, thereby satisfying the conditions of the claim.

Finally, if  $H_{2t-1}$  is the collision symbol, then, for every input in  $I_v$ , at least two processors write at step  $2t - 1$  in the COLLISION(1) algorithm and, therefore, would have written at step  $t$  in the COMMON(1) algorithm. Hence at least two literals in the additional clause are true. This concludes the proof of the claim.  $\square$

We are now ready to complete the proof of Theorem 10. Let  $v$  be a node of depth  $2t$  in the COLLISION(1) tree and let  $u$  be the child corresponding to a collision occurring at step  $2t + 1$ . We will show that the values written by any processor at the  $(t + 1)$ st step of the COMMON(1) algorithm, for any input in  $I_v$ , are all the same. Hence,  $P_1$  can determine this value from its knowledge of  $I_v$  and the programs of the other processors.

Let  $a$  be an input in  $I_u$  and assume that there is another input in  $I_v$  for which, at the  $(t + 1)$ st step in the COMMON(1) algorithm, a different value is written. Suppose that  $P_j$  is a processor writing this other value and that it does so because literal  $l_j = 1$ .

Consider the input  $b$  obtained from  $a$  by making  $l_j = 1$ . Then  $b \notin I_{v'}$ ; otherwise the COMMON model would be violated. Since a collision occurs, two or more true literals occurring in  $a$  cause a particular value to be written. Thus at least one processor would write that value at step  $t + 1$  on input  $b$ . Since  $l_j = 1$  in  $b$ ,  $P_j$  would simultaneously write another value.

Because  $a \in I_{v'}$  and  $b \notin I_{v'}$ ,  $f_{v'}(a) = 1$  and  $f_{v'}(b) = 0$ . Now  $f_{v'}$  is a formula in conjunctive normal form. Therefore  $f_{v'}$  contains a clause  $g$  such that  $g(a) = 1$  and  $g(b) = 0$ . Since  $b$  is obtained from  $a$  by changing  $l_j$  from 0 to 1,  $\bar{l}_j$  is the only literal in  $g$  satisfied by  $a$ . By the claim,  $\bar{l}_j = 1$  is true for all inputs in  $I_v$ . This is a contradiction.  $\square$

**6. Simulating PRIORITY( $km$ ) by ARBITRARY( $m$ ).** Section 2 considered the simulation of PRIORITY machines by ARBITRARY machines with more memory. Here we study the ‘‘complementary’’ problem of simulating PRIORITY machines by ARBITRARY machines with less memory. As a corollary, we obtain a separation between PRIORITY and ARBITRARY machines with the same amount of memory.

Our goal is to solve the  $km$ -colour MINIMIZATION problem in the ARBITRARY( $m$ ) model. This can clearly be done in time  $O(k \log n)$ , by dividing the colours into  $k$  groups of  $m$  colours and solving one group at a time by the algorithm of Theorem 1. A proof similar to that of Corollary 6.3 shows that the  $km$ -colour MIN-

MINIMIZATION problem requires  $\Omega(km \log(n/km))$  steps on COMMON(1), and thus  $\Omega(k \log(n/km))$  steps on COMMON( $m$ ). In fact, it is possible to do considerably better on ARBITRARY( $m$ ), as the following theorem and corollary demonstrate. Hence ARBITRARY is another example of a computational model in which “mixing” the computation of several functions on disjoint sets of inputs enhances efficiency. Boolean and arithmetic circuits also exhibit this property ([P], [U], [AHU]).

**THEOREM 11.** *On ARBITRARY(1), the  $m$ -colour MINIMIZATION problem can be solved in  $O\left(\frac{m \log n}{\log m}\right)$  steps.*

When  $m = O(n^\epsilon)$  for some constant  $\epsilon > 0$ , this upper bound  $O(m)$  matches the trivial lower bound. In comparison with algorithms that solve the problem one colour at a time, this solution uses an average of  $O(1)$  steps per colour. This disproves a conjecture of Vishkin [V].

The idea is to try to solve the  $m$  different problems concurrently, although we have only one common memory cell. We say that a processor  $P_i$  has colour  $c$  if  $x_i = c$ . For each colour  $c$ , processors maintain an ordered set  $S_c \subset \{1, \dots, n\}$  and a “current winner”  $w_c \in \{1, \dots, n, \infty\} - S_c$ . If  $w_c < \infty$ , then  $w_c$  is the smallest index of any processor globally known to have colour  $c$ . When there is no processor that is globally known to have colour  $c$ , as is the case initially,  $w_c = \infty$ . The set  $S_c$  consists of the indices of those processors that may replace the current winner. In particular,  $i \in S_c$  implies that  $i < w_c$ . Initially,  $S_c = \{1, 2, \dots, n\}$ .

The algorithm proceeds in phases. In a single phase, each set  $S_c$  is shrunk by approximately a factor of  $m$ . When  $S_c = \emptyset$  for all  $c$ , the algorithm can halt.

At the beginning of a phase, each set  $S_c$  is divided into  $m$  pieces  $S_c^1, S_c^2, \dots, S_c^m$  of size at most  $\left\lceil \frac{|S_c|}{m} \right\rceil$ , where the processors in  $S_c^a$  have lower indices than those in  $S_c^b$  for  $a < b$ . The goal in a phase is to publicly determine, for each colour  $c$ , the first group among  $S_c^1, S_c^2, \dots, S_c^m$  containing the index of some processor having colour  $c$ . Then  $S_c$  is updated appropriately.

Conceptually, these sets are arranged in an  $m \times m$  array; the entry in row  $c$  and column  $i$  is  $S_c^i$ . At each step of the phase, we either eliminate a row or the leftmost column of the array. The set  $C$  will consist of those colours whose rows have not yet been eliminated in this phase.

If processor  $P_j$  has colour  $c$  and  $j$  belongs to the group in the leftmost column of the row corresponding to colour  $c$ , then it attempts to write its index and colour into memory cell  $M_1$ . Throughout the phase, the invariant is maintained that for any colour  $c \in C$ , no processor with that colour lies in  $S_c^k$  for any eliminated column  $k$ . Hence, when  $(j, c)$  appears in  $M_1$ , any processor having colour  $c$  and with index lower than  $j$  must be in the group currently in the leftmost column of row  $c$ . In this case, row  $c$  is eliminated from the array. If no write occurs at a given step, then none of the groups in the leftmost column contain the winner for their row and the column can be eliminated. More formally, the phase proceeds as follows.

$C \leftarrow \{1, 2 \dots m\}$

$i \leftarrow 1$

While  $C \neq \emptyset$  and  $i \leq m$  do

    If  $j \in S_{x_j}^i$  and  $x_j \in C$ , processor  $P_j$  will attempt to write  $(j, x_j)$  into  $M_1$ .

    If  $(j, c)$  appears in  $M_1$

        Remove  $c$  from  $C$ , set  $w_c \leftarrow P_j$ , and shrink  $S_c$  to  $\{k \in S_c^i : k < j\}$

    Otherwise set  $i \leftarrow i + 1$ .

At each step, either  $|C|$  is decreased by one or  $i$  is increased by one. A phase thus takes at most  $2m - 1$  steps, and any set  $S_c$  which was of size  $s$  before the phase is of



size at most  $\lceil \frac{s}{m} \rceil - 1 < \frac{s}{m}$  at the end of the phase. Thus  $O\left(\frac{\log n}{\log m}\right)$  phases suffice.  $\square$

**COROLLARY 11.1.** *One step of PRIORITY( $km$ ) can be simulated by  $O\left(\frac{k \log n}{\log k}\right)$  steps of ARBITRARY( $m$ ), for  $k > 1$ .*

*Proof.* Simulating one step of PRIORITY( $km$ ) is equivalent to solving the  $km$ -colour MINIMIZATION problem; divide the colours into  $m$  groups of  $k$  colours each, and use the algorithm above to solve each group in parallel.  $\square$

The following lower bound proves this procedure optimal for  $km = O(n^{1-\epsilon})$ , and proves a separation between PRIORITY( $m$ ) and ARBITRARY( $m$ ).

**THEOREM 12.** *The  $km$ -colour MINIMIZATION requires  $\Omega\left(\frac{k \log(n/km)}{\log(k+1)}\right)$  steps to solve on ARBITRARY( $m$ ).*

**COROLLARY 12.1.** *ARBITRARY( $m$ ) requires at least  $\Omega(\log(n/m))$  steps to simulate one step of PRIORITY( $m$ ).*

To simplify the proof of Theorem 12, we divide the processors into  $km$  groups of size at least  $\lfloor n/km \rfloor$ , and declare that the processors in group  $i$  will have colour either  $i$  or 0. Note that we can define this restricted problem over domain  $\{0, 1\}^n$ .

We maintain a history  $H_0, H_1 \dots$ , a set  $S$  of fixed positions, and a set  $F$  of free positions. Initially, the processor of highest index in each group has its colour fixed to 1, and all other positions are free. We maintain the invariant that for any free position, there are no positions of lower index that are within the group and fixed to 1.

Our measure of the algorithm's progress against the adversary strategy will be by means of a potential function. If there are  $s_i$  free positions in group  $i$ , then this function has value  $\sum_{i=1}^{km} \log_{k+1}(s_i + 1)$ . Initially, then, the function has value at least  $km \log_{k+1}(n/km)$ . We shall show that the adversary can fix positions in such a fashion so that the history is determined through step  $t$  and the value of this function decreases by at most  $ctm$ , for some absolute constant  $c$ .

As long as there is at least one free position (that is, the value of the potential function is nonzero), the algorithm has not solved all colours, thus establishing the lower bound.

Given history  $H_0, H_1 \dots H_t$ , and the fact that group  $i$  contains  $s_i$  free positions, we show how to fix  $H_{t+1}$  and cause a drop in the potential function of at most  $cm$ . Initially, the contents of each cell in  $H_{t+1}$  are unfixed. Suppose the free positions in group  $i$  at any point are  $j_1, j_2, \dots, j_s$ . We define  $lower_i$  to be the lowest  $\frac{1}{k+1}$ st of the free positions in group  $i$ , that is, positions  $j_1$  through  $j_{\lceil s/(k+1) \rceil}$ .  $upper_i$  is defined to be all free positions in group  $i$  not in  $lower_i$ .

- 1) If a processor in any fixed position writes at step  $t+1$  into an unfixed cell, declare one such processor to win the competition to write into that cell for all allowable inputs. The value of the potential function does not change. Repeat this step until all cells are fixed or no such processor exists.
- 2) If any processor in any free position writes on 0 into an unfixed cell, choose one such processor, fix its colour to 0, and declare it to win the competition to write at time  $t+1$ . If it is in group  $i$ , then the potential function drops by

$$\log_{k+1}(s_i + 1) - \log_{k+1} s_i = \log_{k+1} \left(1 + \frac{1}{s_i}\right) \leq \log_{k+1}(2) \leq 1.$$

Repeat this step until all cells are fixed or no such processor exists.

- 3) Once the first two cases are taken care of, then processors write into unfixed cells only if they receive their colour. If there is a processor  $P_j$  in a free position in group  $i$  such that  $P_j$  writes into an unfixed cell on colour  $i$ , and  $j \in upper_i$ , then

fix  $upper_i$  to colour  $i$  and declare  $P_j$  to win the competition to write into that cell for all consistent inputs. The potential function drops by at most

$$\log_{k+1}(s_j + 1) - \log_{k+1}\left(\frac{s_j + 1}{k + 1}\right) = 1.$$

Repeat this step until all cells are fixed or no such processor exists.

- 4) Fix  $lower_i$  to 0 for all groups. This ensures that no processor writes into any unfixed cells at this step, for any consistent input. The drop in the potential function is at most

$$\begin{aligned} \sum_{j=1}^{km} \log_{k+1}(s_j + 1) - \sum_{j=1}^{km} \log_{k+1}\left(s_j - \frac{(s_j + 1)}{k + 1} + 1\right) &= m \log_{k+1}\left(\frac{k + 1}{k}\right) \\ &\leq c' m \text{ for some constant } c'. \end{aligned}$$

Steps 2 and 3 of the adversary argument can be repeated at most  $m$  times, since each step fixes one cell. Hence these steps cause a potential drop of at most  $m$ . The total potential drop is thus at most  $(c' + 1)m$ .  $\square$

The proof of Theorem 12 is a generalization of the argument used for the case  $m = 1$  in [Ra], and of the similar argument used in [LY2] to prove a weaker lower bound for the case  $k = 1$ , when inputs are stored in read-only memory.

**7. Conclusions.** In this paper, we have thoroughly investigated the problem of simulating one step of a machine with one shared memory cell by another with  $m$  shared memory cells. If the domains of the functions being computed can be arbitrary, then either the programs for the first machine can be run directly on the second or  $\Theta\left(\frac{\log n}{\log(m+1)}\right)$  steps are required. When the domain of the function is  $\{0, 1\}^n$ , the results are, for the most part, the same. The only exception is that, in this case, COMMON(1) can be simulated by COLLISION(1), and hence by COLLISION( $m$ ), in constant time.

The converse problem of simulating one step of a machine with  $m$  shared memory cells by another with one shared memory cell is not as well understood. The complexity of simulating PRIORITY( $m$ ) by ARBITRARY(1) is  $\Theta\left(\frac{m \log n}{\log m}\right)$ . For COMMON(1) simulating PRIORITY( $m$ ), ARBITRARY( $m$ ), or COLLISION( $m$ ) the lower bound can be improved to  $\Omega(m \log(n/m))$ . However, our upper bound is  $O(m \log n)$ .

The upper bound for COLLISION(1) simulating PRIORITY( $m$ ), ARBITRARY( $m$ ), or COMMON( $m$ ) is the same, except when COMMON( $m$ ) is computing a function with domain  $\{0, 1\}^n$ . In this case, an  $O(m)$  upper bound follows from Lemma 6.1 and Theorem 10. This is clearly optimal. The  $\Omega\left(\frac{m \log n}{\log m}\right)$  lower bound for simulating PRIORITY( $m$ ) on COLLISION(1) is a direct consequence of the lower bound on ARBITRARY(1).

The case of simulating a model with  $m$  cells by another with  $m$  cells is also not as well understood. We have shown that the complexity of simulating COLLISION( $m$ ) by COMMON( $m$ ) or simulating PRIORITY( $m$ ) by ARBITRARY( $m$ ) is  $\Omega(\log n - \log m)$ . For small values of  $m$ , i.e.  $m = O(n^{1-\epsilon})$  for some constant  $\epsilon > 0$ , these results are tight to within a constant factor. Nothing is known about the complexity of simulating COMMON( $m$ ) by COLLISION( $m$ ) (for  $m > 1$ ) except the upper bounds of  $O(\log n)$  (implied by Corollary 1.1) and  $O(m)$  (implied by Theorem 10). Is it possible to do this simulation in  $O(1)$  time?

No other separation between models with the same (finite) amount of shared memory is known for higher values of  $m$ . More work and probably other techniques are needed to extend these results for all ranges of  $m$ . To improve these results, we must understand in a more fundamental way how processors can use larger amounts of memory to communicate. Li and Yesha [LY1],[LY2] have made a first step in this direction by considering concurrent-read concurrent-write PRAM's with a small shared memory plus  $n$  cells of read-only memory containing the input.

Another relevant result concerns the problem of element distinctness on machines with an infinite amount of shared memory. In this problem,  $n$  integers are stored in the first  $n$  cells of shared memory, and the machine must decide whether or not there exist two which are equal. In [FMW] it is shown that element distinctness requires  $\Omega(\log \log \log n)$  steps on COMMON( $\infty$ ) but  $O(1)$  steps on COLLISION( $\infty$ ). The lower bound has been improved to  $\Omega(\sqrt{\log n})$  steps [RSSW]. Both these results imply the existence of a function  $f(n)$  such that the corresponding separation holds between COMMON( $f(n)$ ) and COLLISION( $f(n)$ ). For the  $\Omega(\log \log \log n)$  result,  $f(n) = 2^{n^{O(1)}}$ , but for the  $\Omega(\sqrt{\log n})$  result,  $f(n)$  grows much more rapidly with  $n$ .

We conclude by mentioning a surprising recent result [FRW2], which shows that one step of PRIORITY( $m$ ) can be simulated by  $O\left(\frac{\log n}{\log \log n}\right)$  steps of COMMON( $nm$ ). This shows that allowing more memory can lead to improved simulations even if the number of processors is held fixed, and also that the separation between PRIORITY( $\infty$ ) and COMMON( $\infty$ ) is **not**  $\Theta(\log n)$ .

#### REFERENCES

- [AHU] A.A. AHO, J.E. HOPCROFT, AND J.D. ULLMAN, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.
- [B] P. BEAME, *Limits on the power of concurrent-write parallel machines*, Proc. 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 169-176.
- [CDR] S.A. COOK, C. DWORK, AND R. REISCHUK, *Upper and lower time bounds for parallel random access machines without simultaneous writes*, this Journal, 15 (1986), pp. 87-97.
- [CSV] A.K. CHANDRA, L.J. STOCKMEYER, AND U. VISHKIN, *Complexity theory of unbounded fan-in parallelism*, Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science, 1982, pp. 1-13.
- [FMW] F.E. FICH, F. MEYER AUF DER HEIDE, AND A. WIGDERSON, *Lower bounds for parallel random access machines with unbounded shared memory*, in *Advances In Computing Research*, F. Preparata, ed., Jai Press, to appear.
- [FRW] F.E. FICH, P.L. RAGDE, AND A. WIGDERSON, *Relations between concurrent-write models of parallel computation*. Proc. 3rd Annual ACM Symposium on Principles of Distributed Computing, 1984, pp. 179-189.
- [FRW2] ———, *Simulations among concurrent-write PRAMs*, *Algorithmica*, 1987, to appear.
- [FW] S. FORTUNE AND J. WYLLIE, *Parallelism in random access machines*, Proc. 10th Annual ACM Symposium on Theory of Computing, 1978, pp. 114-118.
- [Ga] Z. GALIL, *Optimal parallel algorithms for string matching*, Proc. 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 240-248.
- [Go] L. GOLDSCHLAGER, *A unified approach to models of synchronous parallel machines*, *J. ACM*, 29 (1982), pp. 1073-1086.
- [Gr] A. GREENBERG, *Efficient algorithms for multiple access channels*, Ph.D Thesis, University of Washington, Seattle, WA, 1983.
- [KMR] R. KANNAN, G. MILLER, AND L. RUDOLPH, *A sublinear parallel algorithm for computing the greatest common divisor of two integers*, Proc. 25th Annual IEEE Symposium on Foundations of Computer Science, 1984, pp. 7-11.
- [Ku] L. KUCERA, *Parallel computation and conflicts in memory access*, *Inform. Process. Lett.*, 14 (1982), pp. 93-96.
- [LY1] M. LI AND Y. YESHA, *Separation results for ROM and nondeterministic models of parallel computation*, Tech. Rept. CISRC-TR-86-7, Ohio State University, 1985.
- [LY2] ———, *New lower bounds for parallel computation*, Proc. 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 177-187.

- [MW] F. MEYER AUF DER HEIDE AND A. WIGDERSON, *The complexity of parallel sorting*, Proc. 20th IEEE Symposium on Foundations of Computer Science, 1985, pp. 532-540.
- [P] W.J. PAUL, *Realizing Boolean functions on disjoint sets of variables*, Theoret. Comp. Sci., 2 (1976), pp. 383-396.
- [Ra] P. RAGDE, *Lower bounds for parallel computation*. Ph.D. Thesis, University of California at Berkeley, 1986.
- [Re] R. REISCHUK, *Simultaneous writes of parallel random access machines do not help to compute simple arithmetic functions*, manuscript, 1984.
- [RSSW] P. RAGDE, W. STEIGER, E. SZEMERÉDI, AND A. WIGDERSON, *The parallel complexity of the element distinctness function is  $\Omega(\sqrt{\log n})$* , SIAM J. Discrete Math., to appear.
- [S] M. SNIR, *On parallel searching*, Department of Computer Science, The Hebrew University of Jerusalem, Research Report 83-21 (June 1983).
- [SV] Y. SHILOACH AND U. VISHKIN, *Finding the maximum, merging and sorting on parallel models of computation*, J. Algorithms, 2 (1981), pp. 88-102.
- [TV] R.E. TARJAN AND U. VISHKIN, *Finding biconnected components and computing tree functions in logarithmic parallel time*, Proc. 25th Annual Symposium on Foundations of Computer Science, 1984, pp. 12-20.
- [U] D. ULIG, *On the synthesis of self-correcting schemes from functional elements with a small number of reliable elements*, Math. Notes. Acad. Sci. USSR, 15 (1974), pp. 558-562.
- [V] U. VISHKIN, *Implementation of simultaneous memory address access in models that forbid it*, Tech. Rept. 210, Israel Institute of Technology, July 1981; J. Algorithms, to appear.
- [VW] U. VISHKIN AND A. WIGDERSON, *Trade-offs between depth and width in parallel computation*, this Journal, 14 (1985), pp. 303-314.
- [Y] A. YAO, *Probabilistic computations: towards a unified measure of complexity*, Proc. 18th Annual Symposium on Foundations of Computer Science, 1977, pp. 222-227.

**ERRATUM:**  
**On Restricting the Size of Oracles**  
**Compared with Restricting Access to Oracles\***

TIMOTHY J. LONG†

The statement  $\text{NP}(\text{K} \oplus \text{S}) \cap \text{co-NP}(\text{K} \oplus \text{S}) \subseteq \text{NP}(\text{S})$  appearing on lines 4 and 18 of page 591<sup>1</sup> should be changed to  $\text{NP}(\text{S}) - (\text{NP}(\text{K} \oplus \text{S}) \cap \text{co-NP}(\text{K} \oplus \text{S})) \neq \emptyset$ .

**Acknowledgment.** The author would like to thank Shouwen Tang for bringing this error to his attention.

---

\* Received by the editors September 8, 1987; accepted for publication January 26, 1988.

† Department of Computer and Information Sciences, Ohio State University, Columbus, Ohio 43210.

<sup>1</sup> SIAM J. Comput., 14 (1985), pp. 585-597.

## EXISTENCE, UNIQUENESS, AND CONSTRUCTION OF REWRITE SYSTEMS\*

NACHUM DERSHOWITZ†, LEO MARCUS‡, AND ANDRZEJ TARLECKI§

**Abstract.** The construction of term-rewriting systems, specifically by the Knuth–Bendix completion procedure, is considered. We look for conditions that might ensure the existence of a finite canonical rewriting system for a given equational theory and that might guarantee that the completion procedure will find it. We define several notions of equivalence between rewriting systems in the ordinary and modulo case, and examine uniqueness of systems and the need for backtracking in implementing completion.

**Key words.** term-rewriting systems, termination, Knuth–Bendix completion procedure, equational theories, rewriting modulo a congruence

**AMS(MOS) subject classifications.** 68Q50, 68T15, 08B05, 03C05, 03B35

**1. Introduction.** Much is known about the theory of term-rewriting systems (see [15] and [10] for surveys). However, there have been many unanswered questions about the existence of rewrite systems and the applicability of the Knuth–Bendix completion procedure (KB) [14], [19] and its extensions [16], [21], [26], [3] for finding them. In certain cases KB generates a “canonical” term-rewriting system that decides validity in a given equational theory. In general, the procedure takes a finite set of equations and an ordering on terms, and either halts with success, aborts (fails), or loops infinitely. KB completion is being used in many program-specification and theorem-proving applications (see, for example, [8], [9], [12], [13], [15], [23], [18], and [6]).

Two basic questions arise: when does a given decidable equational theory have a decision procedure in the form of a canonical rewrite system, and when does the Knuth–Bendix procedure generate such a rewrite system for a given equational theory? As a partial answer, the following facts are discussed in this paper:<sup>1</sup>

1. There are decidable equational theories which require an expanded language in order to obtain a canonical rewrite system to decide them, and others which do not have canonical rewrite systems in any language (Examples 1 and 2).
2. For a given reduction ordering KB cannot terminate successfully with two different systems ([6], [24], Theorem 5).
3. Conditions are given which imply the uniqueness of a term-rewriting system modulo a congruence (§ 4). If any one of these conditions is relaxed, uniqueness is lost (Corollary 12, Examples 6–11).
4. For a given theory and ordering on terms, KB cannot both succeed and loop, depending on the choice of equation to orient into a rule ([8], Theorem 17).
5. Given a theory with an equivalent rewrite system,  $R$ , and the ordering induced by  $R$ , KB will not abort on the first step, but can abort on a later step (Theorem 2, Example 3).
6. KB can abort with different results (Example 12).

\* Received by the editors August 26, 1985; accepted for publication (in revised form) July 13, 1987. This research was partially supported by the Aerospace Corporation.

† Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801.

‡ Computer Science Laboratory, The Aerospace Corporation, P.O. Box 92957, Los Angeles, California 90009.

§ Institute of Computer Science, Polish Academy of Sciences, 00-901 Warsaw, Poland.

<sup>1</sup> The numbers refer to theorems and examples in the sections that follow.

7. KB can succeed or abort, depending on choice of representation for the theory (Example 4).
8. KB can succeed or abort, depending on the choice of equation to orient, thus backtracking is necessary (Example 14).
9. Any total monotonic well-ordering on terms allows KB to succeed (without backtracking) on a ground (variable-free) theory ([20], Theorem 18).
10. Success of KB cannot be guaranteed by adding function symbols to the language (Example 13).

These results have important implications for the implementation of the Knuth-Bendix procedure in general, and in the modulo case, for associative commutative theories.

This paper expands on [11], which was originally motivated by a seminar on term-rewriting systems given by the first author at The Aerospace Corporation in the summer of 1981.

**2. Definitions and notation.** If  $L$  is an alphabet (denumerable set of function symbols and constants, signature),  $T(L, X)$  denotes the (countably infinite) set of (first-order) terms constructed from symbols in  $L$  and variables from a (fixed denumerable) set  $X$ . It is well known that  $T(L, X)$  with the operations defined in the natural way constitutes a free  $L$ -algebra over  $X$ , which we denote by  $T(L, X)$  as well. Terms are usually denoted by small letters of the English alphabet:  $l, r, s, t$ , etc. A *term-rewriting system* (or simply *rewrite system*)  $R$  is a set of ordered pairs of terms called *rules*, and is written  $l \rightarrow r$ . (We do not insist a priori that all variables in  $r$  are also in  $l$ .)

If  $\sigma: X \rightarrow T(L, X)$  is a substitution of terms for variables, then  $l\sigma$  denotes the term  $l$  with each occurrence of a variable  $x$  that appears in  $l$  replaced by the term  $x\sigma$  to which  $x$  is mapped under  $\sigma$ . We use the notation  $s = c[u]$  to indicate that a term  $s$  contains a subterm  $u$  within context  $c$ ; then we refer to the term  $s$  with that occurrence of  $u$  replaced by  $v$  as  $c[v]$ . If a term  $s$  contains an instance  $l\sigma$  of the left-hand side of a rule  $l \rightarrow r \in R$ , then  $s = c[l\sigma]$  rewrites to  $t = c[r\sigma]$  and we write  $s \xrightarrow{1}{R} t$ . Clearly,  $\xrightarrow{1}{R}$  is closed with respect to substitution and subterm replacement, i.e.,  $s \xrightarrow{1}{R} t$  implies  $c[s\sigma] \xrightarrow{1}{R} c[t\sigma]$ , for any context  $c$  and substitution  $\sigma$ . A *derivation* in  $R$  is a sequence  $t_0 \xrightarrow{1}{R} t_1 \xrightarrow{1}{R} t_2 \xrightarrow{1}{R} \dots$ . The notation  $s \xrightarrow{*}{R} t$  indicates that there is a (perhaps empty) derivation in  $R$  from  $s$  to  $t$ , and  $s \xrightarrow{+}{R} t$  indicates that there is a nonempty derivation in  $R$  from  $s$  to  $t$ . The notation  $s \xleftrightarrow{*}{R} t$  indicates that there is a sequence of applications of  $\xrightarrow{1}{R}$  and  $\xleftarrow{1}{R}$  between  $s$  and  $t$ . (In other words,  $\xrightarrow{*}{R}$ ,  $\xleftarrow{*}{R}$  and  $\xleftrightarrow{*}{R}$  are the reflexive-transitive, transitive, and symmetric-reflexive-transitive closures, respectively, of  $\xrightarrow{1}{R}$ .)

A rewrite system  $R$  is (*finitely*) *terminating* if there is no infinite  $R$ -derivation. Note that a system containing a rule  $l \rightarrow r$  with a variable in  $r$  but not in  $l$  is nonterminating. A term  $s$  is in ( $R$ -) *normal form* if there is no  $t$  (other than perhaps  $s$ ) such that  $s \xrightarrow{1}{R} t$ . (The standard definition of normal form is slightly stronger in that it does not allow a normal form to reduce to itself.) We write  $s \xrightarrow{1}{R} t$  if  $s \xrightarrow{*}{R} t$  and  $t$  is in normal form;  $R(s)$  denotes the set of normal forms of  $s$ . Two terms  $r$  and  $s$  are *confluent* in  $R$  if there is a  $t$  such that  $r \xrightarrow{*}{R} t$  and  $s \xrightarrow{*}{R} t$ , written  $r \downarrow_R s$ . If  $R$  is finite and terminating, then the relation  $\downarrow_R$  is recursive. A rewrite system  $R$  is *confluent* if for every  $r$  and  $s$ ,  $r \xleftrightarrow{*}{R} s$  implies that  $r$  and  $s$  are confluent in  $R$ .

Let  $E$  be an equational theory, presented as a set of equations between terms in  $T(L, X)$  of the form  $s \equiv t$ . Thus,  $s \equiv t \in E$  means that the given equation actually appears in the set  $E$ , while  $E \vdash s \equiv t$  means that  $s \equiv t$  is valid in  $E$ , i.e., it follows from  $E$  by equational logic (substitution of equals for equals). Two theories are *equivalent*

if they give the same set of valid equations. A theory is *finitely based* if it is presentable as a finite set of equations. The alphabet  $L(E)$  (or  $L(R)$ ) is the set of function symbols actually appearing in  $E$  (or  $R$ ). A system  $R$  *recognizes* the equational theory  $E$  if  $E \vdash s \equiv t$  if and only if  $s \downarrow_R t$ ;  $R$  *recognizes  $E$  in an expanded language*  $L(R) \supseteq L(E)$  if for all  $s, t \in T(L(E), X)$ ,  $E \vdash s \equiv t$  if and only if  $s \downarrow_R t$ .

A system  $R$  is *canonical* if it is terminating and confluent. It is *canonical for a theory  $E$*  if it is terminating and recognizes  $E$ . Note that if  $R$  recognizes  $E$ , then it must be confluent. If  $R$  recognizes  $E$  and is finite and terminating, then  $R$  decides  $E$ .

Let  $\sim$  be an arbitrary congruence on  $T(L, X)$ , closed with respect to substitution. A set of rules  $R$  is a rewrite system *modulo* a congruence relation  $\sim$  if the left and right sides of the rules in  $R$  are  $\sim$ -congruence classes. Operationally, we let the left and right sides be terms, but allow a term  $u$  to reduce to  $v$  (mod  $\sim$ ) if  $u \sim u'[l\sigma]$  and  $u'[r\sigma] \sim v$  for some  $l \rightarrow r \in R$ .<sup>2</sup> (If  $l \sim l', r \sim r'$ , then  $l \rightarrow r$  and  $l' \rightarrow r'$  are considered to be the same rule.) Thus, for the modulo case,  $\frac{1}{R}$  is a binary relation on the quotient algebra  $T(L, X)/\sim$ , closed with respect to substitution and subterm replacement. If  $\sim$  is the identity relation, then  $R$  modulo  $\sim$  reduces to the case of an ordinary rewrite system. All the previous definitions, viz. termination, normal form, and confluence, generalize to modulo systems  $R$ .

A prime use for rewrite systems modulo a congruence is in rewrite systems for associative-commutative theories (see [26]). Notice that in the modulo case, if  $s \sim t$  and  $t \xrightarrow{*}_R u$ , then  $s \xrightarrow{*}_R u$ ; in particular, if  $s \sim t$  then  $s \xrightarrow{*}_R t$ .

**3. Equational theories and rewrite systems.** There are inherent limitations to rewriting. Not all classes of algebras can be described equationally (e.g., fields). Not all equational theories are finitely based (e.g., the equational theory of the algebra over  $T(\{\cdot, 0, 1, 2\}, X)$  with binary function  $\cdot$  satisfying  $0 \cdot x \equiv x \cdot 0 \equiv 0$ ,  $1 \cdot 1 \equiv 0$ ,  $1 \cdot 2 \equiv 1$ ,  $2 \cdot 1 \equiv 2 \equiv 2 \cdot 2$ : see [28]). Not all finitely based equational theories are decidable (e.g., some Thue systems; see [7]). Not every decidable equational theory is amenable to canonical (modulo identity) rewriting (e.g., the commutativity axiom). See [5], [17] and [25] for related theorems and examples pertaining to string rewriting. Work on related questions appears in [2].

*Example 1.* Let  $E = \{fg^i hx \equiv fg^j hx: 0 \leq i, j < \omega\}$  be a theory over  $T(\{f, g, h, a\}, \{x\})$ . Then there is no finite rewrite system recognizing  $E$  (since all rules must be of the form  $fg^m hx \rightarrow fg^n hx$ , all but finitely many  $fg^k hx$  would be irreducible), but there is a finite rewrite system that recognizes  $E$  in an expanded language:  $R = \{fgx \rightarrow fg'x, g'gx \rightarrow gg'x, g'hx \rightarrow hx\}$  in the language  $\{f, g, h, g', a\}$  ( $fg^i hx \xrightarrow{*}_R fhx$ ).

*Example 2.* Not all decidable equational theories have finite recognizing rewrite systems in extended languages. For example, let  $E$  be the theory of one commutative binary function symbol  $f$ . Assume  $R$  is system in an extended language that recognizes  $E$ . Let  $fx y$  and  $fy x$  reduce to some normal form  $N$ . Then a derivation of  $N$  from  $fy x$  can be obtained from the derivation of  $N$  from  $fx y$  by reversing the roles of  $x$  and  $y$ . Thus,  $N$  cannot contain the variables  $x$  or  $y$ . (Either  $x$  appears “before”  $y$  in  $N$  or it does not.) Therefore  $N$  is also derivable from  $fuv$ , for any variables (or terms)  $u, v$ . But this represents a new equality not valid in  $E$ , namely  $fx y = fuv$ , a contradiction.

This raises the open question: For which decidable equational theories are there extended canonical rewrite systems?

Of course, by encoding or ordering variables we may transform a decidable equational theory to a Turing machine, and that Turing machine to a rewrite system.

<sup>2</sup> A weaker system is obtained by just allowing a rule  $l \rightarrow r$  to be applicable to any term  $l' \sim l\sigma$ , yielding a term  $r' \sim r\sigma$ ; the congruence cannot be applied in the rest of  $u$ . See [26] and [16].



But that rewrite system does not serve as a canonical system for the original theory. Next we give some examples of the relation between an equational theory and a finite canonical rewrite system that recognizes it. First, a lemma in which  $E$  and  $R$  need not be related.

**LEMMA 1.** *Let  $R$  be any rewrite system and  $E$  any equational theory. If  $E \vdash u \equiv v$  for some term  $v$  in  $R$ -normal form and any other term  $u$ ,  $u \neq v$ , then there is an axiom  $s \equiv t \in E$  such that  $s$  or  $t$  is in  $R$ -normal form.*

*Proof.* Let  $u = w_0 \equiv w_1 \equiv \dots \equiv w_n = v$  ( $n \geq 1$ ) be an equational proof of  $u \equiv v$  in  $E$ . That is, for each  $i$  ( $1 \leq i \leq n$ ),  $w_{i-1} = c[s_i\sigma]$  and  $w_i = c[t_i\sigma]$ , for some context  $c$ , substitution  $\sigma$ , and axiom  $s_i \equiv t_i$  (or  $t_i \equiv s_i$ ) in  $E$ . Since  $v = w_n$  is in  $R$ -normal form,  $t_n$  must also be. Hence  $s_n \equiv t_n$  (or  $t_n \equiv s_n$ ) is an axiom of  $E$ , one side of which ( $t_n$ ) is in  $R$ -normal form.  $\square$

**THEOREM 2.** *If  $R$  is a canonical rewrite system for an equational theory  $E$ , and  $E$  is nontrivial, then there is  $s \equiv t \in E$  such that  $s \xrightarrow{+}_R t$  (or  $t \xrightarrow{+}_R s$ ).*

*Proof.*  $R \neq \emptyset$  since  $E$  is not trivial. Therefore, there is  $s' \rightarrow t' \in R$  such that  $t'$  is in  $R$ -normal form; otherwise  $R$  would not be terminating. Now,  $E \Vdash s' \equiv t'$ , and so by Lemma 1, there is  $t$  in  $R$ -normal form and  $s$  such that  $s \equiv t \in E$ . But then it follows that  $s \xrightarrow{+}_R t$ .  $\square$

The above theorem however does not generalize to more than one equation. Instead, we have that even if  $R$  is a finite canonical rewrite system for  $E' \cup R'$  (interpreting the rules in  $R'$  as equations), every term appearing in  $E'$  is in  $R'$ -normal form,  $E'$  is nontrivial, and  $R' \subseteq R$ , there still may be no  $s \equiv t \in E'$  such that  $s \xrightarrow{+}_R t$  or  $t \xrightarrow{+}_R s$ .

*Example 3.* Let  $R' = \{1x \rightarrow x, x1 \rightarrow x, xx^{-1} \rightarrow 1, x^{-1}x \rightarrow 1, x^{-1-1} \rightarrow x, x(x^{-1}y) \rightarrow y, x^{-1}(xy) \rightarrow y, (xy)^{-1} \rightarrow y^{-1}x^{-1}, (xy)z \rightarrow x(yz)\}$ , and  $R = R' \cup \{1^{-1} \rightarrow 1\}$ . Then  $R$  is a finite canonical rewrite system for groups (see [15]) and is equivalent to  $R' \cup \{1^{-1}x \equiv x1^{-1}\}$ . (It is sufficient to show that  $R' \cup \{1^{-1}x \equiv x1^{-1}\} \vdash 1^{-1} = 1$ .) The terms  $1^{-1}x$  and  $x1^{-1}$  are in  $R'$ -normal form, but  $1^{-1}x \not\xrightarrow{+}_R x1^{-1}$  and  $x1^{-1} \not\xrightarrow{+}_R 1^{-1}x$ . (Of course, both terms rewrite in  $R$  to  $x$ .)

Even if  $E = \{s_1 \equiv t_1, \dots, s_n \equiv t_n\}$  has a finite canonical rewrite system, and the equations in  $E$  are independent, there need not be a finite canonical  $S$  for  $E$  such that for every  $i$ ,  $s_i \rightarrow t_i \in S$  or  $t_i \rightarrow s_i \in S$ .

*Example 4.* Let  $E$  be the axiomatization  $\{x(yz) \equiv (xy)z, xx^{-1} \equiv 1, 1x \equiv x, x1 \equiv 1x\}$  of group theory. It can be shown that  $E$  is independent, but there can be no finite canonical rewrite system in which  $x1 \rightarrow 1x$  or vice versa.

Even if  $E$  has a finite canonical rewrite system,  $E \vdash s_i \equiv t_i$  for  $i = 1, \dots, n$ , and  $\{s_1 \rightarrow t_1, \dots, s_n \rightarrow t_n\}$  is terminating, there need not be a canonical  $R$  for  $E$  with  $s_i \rightarrow t_i \in R$  for all  $i$ .

*Example 5.* Let  $E = \{a \equiv b\}$ , in which case  $E \vdash fa \equiv fb, gb \equiv ga$ . The system  $R = \{fa \rightarrow fb, gb \rightarrow ga\}$  is terminating, but cannot be extended to include either  $a \rightarrow b$  or  $b \rightarrow a$  without engendering nontermination.

**LEMMA 3.** *Let  $S$  be a finite canonical rewrite system which recognizes an equational theory  $E$  and let  $R$  be any system with  $\xrightarrow{+}_R \subseteq \xrightarrow{+}_S$  that does not recognize  $E$ . Then there are terms  $s$  and  $t$  in  $R$ -normal form such that  $E \vdash s \equiv t$  and  $s \not\xrightarrow{+}_S t$ .*

*Proof.* If  $R$  is terminating (since  $S$  is) and does not recognize  $E$ , then there must exist two distinct  $R$ -normal forms,  $u$  and  $v$ , such that  $E \vdash u \equiv v$ . Let  $t$  be the common  $S$ -normal form of  $u$  and  $v$ . Then  $t$  is also in  $R$ -normal form. Since  $u \neq v$ , we have  $u \not\xrightarrow{+}_S t$  or  $v \not\xrightarrow{+}_S t$  (or both).  $\square$

**4. Uniqueness.** In this section we examine various criteria for similarity of rewrite systems with the goal of characterizing varieties of uniqueness.

First, let us give some informal examples which answer some of the obvious questions. Must two rewrite systems that decide the same theory be identical? No; consider  $R = \{fx \rightarrow a\}$ ,  $S = \{fy \rightarrow a\}$ . Must they be the same up to variable renaming? No; consider  $R = \{a \rightarrow b\}$ ,  $S = \{b \rightarrow a\}$ . What if  $R \cup S$  has no cycles? No:  $R = \{a \rightarrow b, b \rightarrow c\}$ ,  $S = \{a \rightarrow c, b \rightarrow c\}$ . What if the right-hand sides are irreducible? No:  $R = \{fx \rightarrow gx, fa \rightarrow c, ga \rightarrow c\}$ ,  $S = \{fx \rightarrow gx, ga \rightarrow c\}$ . Assume in addition that the left-hand sides reduce to only one term. Then we do get that  $R$  and  $S$  have the same “semantics,” i.e.,  $\frac{1}{R} = \frac{1}{S}$  (compare Theorem 10). But we still have the example of  $R = \{fa \rightarrow fb, a \rightarrow b\}$ ,  $S = \{a \rightarrow b\}$ . If no rule contains an instance of another, then  $R = S$  up to renaming of variables (see Theorem 5).

Let us illustrate additional problems we have to cope with in the case of modulo systems.

*Example 6.* The following two different modulo systems decide the same theory over  $T(\{f, g, a, b\}, \{x\})$  and satisfy the conditions mentioned above for ordinary (modulo identity) rewrite systems:  $R = \{a \rightarrow b\}$ ,  $S = \{ga \rightarrow gb\}$ , and  $\sim$  is presented by  $\{fgx \sim x\}$ .

*Example 7.* What if no left-hand side is congruent to its own subterm? Still the systems can be different: Take  $fg hx \sim fx$ ,  $R = \{fx \rightarrow a\}$ ,  $S = \{fgx \rightarrow a\}$ .

As we will see, if no nontrivial instance of a left-hand side is a proper subterm of a congruent term, then the systems are the same up to renaming and congruence (see Corollary 12 and Lemma 13).

Now more formally, given rewrite systems  $R$  and  $S$ , some notions of similarity are:

- (1) If  $R$  and  $S$  satisfy for all  $l \rightarrow r \in R$  there is a renaming of variables  $\theta$  such that  $l\theta \rightarrow r\theta \in S$ , we say that  $R$  is *syntactically contained in  $S$* . If  $S$  is also syntactically contained in  $R$ , then they are *isomorphic*, or *syntactically equivalent*.
- (2) If  $R$  and  $S$  satisfy for all  $l \rightarrow r \in R$ ,  $l \xrightarrow{\frac{1}{S}} r$ , we say that  $R$  is *semantically contained in  $S$* . If  $S$  is also semantically contained in  $R$ , we say they are *semantically equivalent*.
- (3) If  $R$  and  $S$  satisfy for all  $l \rightarrow r \in R$ ,  $l \xrightarrow{\frac{*}{S}} r$  (or equivalently: for all  $l$  and  $r$ , if  $l \xrightarrow{\frac{*}{R}} r$ , then  $l \xrightarrow{\frac{*}{S}} r$ ), we say that  $R$  is *derivationally contained in  $S$* . If also  $S$  is derivationally contained in  $R$ , we say they are *derivationally equivalent*.
- (4) If  $R$  and  $S$  satisfy for all terms  $s$  and  $t$ , if  $s \downarrow_R t$ , then  $s \downarrow_S t$ , we say that  $R$  is *operationally contained in  $S$* . If  $S$  is also operationally contained in  $R$ , then they are *operationally equivalent*.
- (5) If  $R$  and  $S$  satisfy for all terms  $s$  and  $t$ , if  $s \xrightarrow{\frac{\diamond}{R}} t$ , then  $s \xrightarrow{\frac{\diamond}{S}} t$ , we say that  $R$  is *deductively contained in  $S$* . If  $S$  is also deductively contained in  $R$ , then they are *deductively equivalent*.

These criteria can be compactly defined as:

- (1)  $R \subseteq S$  (up to renaming) syntactic containment,
- (2)  $\frac{1}{R} \subseteq \frac{1}{S}$  semantic containment,
- (3)  $\frac{*}{R} \subseteq \frac{*}{S}$  derivational containment,
- (4)  $\downarrow_R \subseteq \downarrow_S$  operational containment,
- (5)  $\frac{\diamond}{R} \subseteq \frac{\diamond}{S}$  deductive containment,
- (6)  $R = S$  isomorphism,
- (7)  $\frac{1}{R} = \frac{1}{S}$  semantic equivalence,

- (8)  $\xrightarrow{*}_R = \xrightarrow{*}_S$  derivational equivalence,
- (9)  $\downarrow_R = \downarrow_S$  operational equivalence,
- (10)  $\xleftrightarrow{*}_R = \xleftrightarrow{*}_S$  deductive equivalence.

These criteria all apply to rewriting modulo a congruence  $\sim$  as well (in which case if  $l\theta \sim l'$  and  $r\theta \sim r'$  for some variable renaming  $\theta$ , then  $l \rightarrow r$  and  $l' \rightarrow r'$  are considered the same).

LEMMA 4. *The above criteria satisfy: (1)  $\Rightarrow$  (2)  $\Rightarrow$  (3)  $\Rightarrow$  (4)  $\Rightarrow$  (5), and hence (6)  $\Rightarrow$  (7)  $\Rightarrow$  (8)  $\Rightarrow$  (9)  $\Rightarrow$  (10).*

*Proof.* All clear, except perhaps (4)  $\Rightarrow$  (5), for which we note that  $\xleftrightarrow{*}_R$  and  $\xleftrightarrow{*}_S$  are the transitive closures of  $\downarrow_R$  and  $\downarrow_S$ , respectively.  $\square$

The reverse implications (5)  $\Rightarrow$  (4)  $\cdots$  do not hold. The obvious question is: Under what (interesting) additional conditions do they hold?

For ordinary rewrite systems (mod identify) we say that  $R$  is *reduced* if all right-hand sides are in  $R$ -normal form and all left-hand sides are in normal form with respect to all other rules in  $R$ . Any ordinary canonical system can be reduced to a deductively equivalent one ([24]; see also [2]).

The following is known.

THEOREM 5 (due to M. Ballantyne; mentioned without proof in [6]; a proof is given in [24]). *If two reduced systems  $R$  and  $S$  both recognize the same equational theory  $E$ , and if for some strict partial order  $>$ ,  $\frac{1}{R}, \frac{1}{S} \subseteq >$ , then  $R$  and  $S$  are isomorphic.*

We include the proof here for motivation, even though this is a special case of Corollary 12.

*Proof.* Let  $R$  and  $S$  be two canonical rewrite systems that recognize  $E$ , and for which  $\frac{+}{R}, \frac{+}{S} \subseteq >$ . Assume  $l \rightarrow r \in R$ . Since  $E \vdash l \equiv r$  and  $S$  recognizes  $E$ , there is a term  $u$  such that  $l \xrightarrow{*}_S u$  and  $r \xrightarrow{*}_S u$ . Since  $E \vdash r \equiv u$ ,  $R$  recognizes  $E$ , and  $R$  is reduced,  $u \xrightarrow{!}_R r$ . Thus we have that  $r \cong u$  and  $u \cong r$  ( $\cong$  is the reflective closure of  $>$ ), and so  $r = u$ . It follows that  $l \xrightarrow{+}_S r$ .

Similarly,  $l \rightarrow r \in S$  implies  $l \xrightarrow{+}_R r$ , and thus  $s \xrightarrow{1}{S} t$  implies  $s \xrightarrow{+}_R t$ .

Assume now that  $l \rightarrow r \in R$  but  $l \rightarrow r \notin S$  (even up to a renaming of variables). If  $l \xrightarrow{1}{S} t \xrightarrow{+}_S r$  for some  $t$ , then  $l \xrightarrow{+}_R t \xrightarrow{+}_R r$ . But this contradicts the premise that no rule in  $R$  other than  $l \rightarrow r$  can reduce  $l$ . So it must be that  $l \xrightarrow{1}{S} r$  by a single application of a rule  $l' \rightarrow r' \in S$  such that  $l \neq l'$  (even up to a renaming of variables). But then, too,  $l' \xrightarrow{+}_R r'$  would contradict  $R$  being reduced.  $\square$

We can extend and split the definition of reduced for  $R \text{ mod } \sim$  by using the following definitions:

- (i)  $R$  is *rhs-reduced* if the right-hand side of each rule is in  $R$ -normal form mod  $\sim$ ;
- (ii)  $R$  is *lhs-reduced* if  $l \xrightarrow{1}{R} u \xrightarrow{*}_R r$  implies  $r \sim u$  for any left-hand side  $l$  of a rule  $l \rightarrow r \in R$ ;
- (iii)  $R$  is *reduced* if it is both rhs-reduced and lhs-reduced.

But the uniqueness theorem does not extend immediately.

Example 8. The following reduced, canonical systems  $R$  and  $S \text{ mod } \sim$  for the same theory over  $T(\{f, g, c, d\}, \emptyset)$  are not isomorphic: Let  $\sim$  be presented by  $fgc \sim c$ ,  $fgd \sim d$ , and take  $R = \{c \rightarrow d\}$ ,  $S = \{gc \rightarrow gd\}$ .

THEOREM 6. *Let  $R$  be a rewrite system mod  $\sim$  deductively contained in a confluent system  $S \text{ mod } \sim$ . Then  $R$  is operationally contained in  $S$ .*

*Proof.* Let  $s \downarrow_R t$ . Then  $s \xleftrightarrow{*}_S t$  by deductive containment and  $s \downarrow_S t$  by confluence of  $S$ .  $\square$

**COROLLARY 7.** For confluent systems  $R$  and  $S \text{ mod } \sim$ ,  $R$  and  $S$  are operationally equivalent if and only if  $l \downarrow_S r$  for all  $l \rightarrow r \in R$  and  $l \downarrow_R r$  for all  $l \rightarrow r \in S$ .

*Proof.* If  $l \rightarrow r \in R$  implies  $l \downarrow_S r$ , then  $s \xrightarrow{1}{R} t$  implies  $s \downarrow_S t$  and  $R$  is deductively contained in  $S$ .  $\square$

We say that two rewrite systems  $\text{mod } \sim$ ,  $R$  and  $S$ , are *noninverting* if for all terms  $s$  and  $t$ ,  $s \xrightarrow{*}{R} t \xrightarrow{*}{S} s$  implies  $s \sim t$ .

**LEMMA 8.** Let  $R$  and  $S$  be two operationally equivalent systems  $\text{mod } \sim$ . If  $R$  and  $S$  are noninverting, then  $\xrightarrow{1}{R} = \xrightarrow{1}{S}$ .

*Proof.* We show only one direction; the other follows from symmetry. Let  $s \xrightarrow{1}{R} t$ . By operational equivalence there is a  $u$  such that  $s \xrightarrow{*}{S} u$  and  $t \xrightarrow{*}{S} u$ . Again, by operational equivalence,  $t \xrightarrow{*}{R} v$ ,  $u \xrightarrow{*}{R} v$  for some  $v$ . Since  $t$  is an  $R$ -normal form, we have  $u \xrightarrow{*}{R} t$ . By the noninverting property, it must then be that  $u \sim t$ , and therefore  $s \xrightarrow{*}{S} t$ . Suppose  $t \xrightarrow{+}{S} w$  for some  $w$ . As before, we must have  $w \sim t$ . But then  $t$  is an  $S$ -normal form, since it only reduces to congruent terms.  $\square$

*Example 9.* The following pair,  $R$  and  $S$ , of operationally equivalent rewrite systems modulo the identity are not derivationally equivalent:  $R = \{a \rightarrow b, c \rightarrow b\}$ ,  $S = R \cup \{a \rightarrow c\}$ .

**THEOREM 9.** Let  $R$  and  $S$  be two rewrite systems  $\text{mod } \sim$  for which  $R$  is operationally contained in  $S$ ,  $R$  is rhs-reduced, and every  $R$ -normal form is an  $S$ -normal form. Then  $R$  is derivationally contained in  $S$ .

Note that if  $\xrightarrow{1}{R} \subseteq \xrightarrow{1}{S}$ , then trivially any  $R$ -normal form is an  $S$ -normal form.

*Proof.* Let  $l \rightarrow r \in R$ . Then  $l \downarrow_S r$ . But since  $R$  is rhs-reduced and all  $R$ -normal forms are  $S$ -normal forms, it follows that  $l \xrightarrow{*}{S} r$ .  $\square$

*Example 10.* The following pair,  $R$  and  $S$ , of derivationally equivalent systems  $\text{mod } \sim$  over  $T(\{f, g, a, b\}, \emptyset)$  are not semantically equivalent: Let  $\sim$  be presented by  $\{fga \sim a\}$ ,  $R = \{a \rightarrow b, fgb \rightarrow b\}$ ,  $S = \{ga \rightarrow gb, fgb \rightarrow b\}$ .

**THEOREM 10.** Let  $R$  and  $S$  be two reduced rewrite systems  $\text{mod } \sim$  which are derivationally equivalent. Then  $R$  and  $S$  are semantically equivalent.

*Proof.* We shall prove that  $l \rightarrow r \in R$  implies  $l \xrightarrow{1}{S} r$ . Let  $l \rightarrow r \in R$ . Since  $l \not\sim r$  and  $R$  is derivationally contained in  $S$ ,  $l \xrightarrow{+}{S} r$ . Now assume that  $l \xrightarrow{1}{S} u \xrightarrow{*}{S} r$ ,  $l \not\sim u$ . Since  $S$  is derivationally contained in  $R$ , there is some  $v$  such that  $l \xrightarrow{1}{R} v \xrightarrow{*}{R} u \xrightarrow{*}{R} r$ .  $R$  is lhs-reduced so  $v \sim r$ , and so  $r \xrightarrow{*}{R} u$ . Now using the fact that  $R$  is rhs-induced,  $r \sim u$  and thus  $l \xrightarrow{1}{S} r$ .  $\square$

Were we to weaken the definition of reduced to allow trivial rules  $l \rightarrow r$ , where  $l \sim r$ , then we would only get  $\xrightarrow{0,1}{R} = \xrightarrow{0,1}{S}$ , which is slightly weaker than semantic equivalence.

*Example 11.* The following pair,  $R$  and  $S$ , of systems  $\text{mod } \sim$  over  $T(\{f, g, c\}, \{x\})$  are semantically equivalent, but not isomorphic: Let  $\sim$  be presented by  $fx \sim fghx$ , and take  $R = \{fx \rightarrow c\}$ ,  $S = \{fgx \rightarrow c\}$ .

A rewrite system  $R$  is *minimal* if no left-hand side  $l$  of a rule in  $R$  contains a proper instance of a left-hand side  $l'$  of a (not necessarily different) rule in  $R$ , that is,  $l \not\sim c[l'\sigma]$  for any nonempty context  $c$  or substitution  $\sigma$  that is more than a renaming of variables in  $l'$ .

**THEOREM 11.** Suppose  $R$  and  $S$  are semantically equivalent rewrite systems  $\text{mod } \sim$ . If  $R$  and  $S$  are both minimal, then  $R$  and  $S$  are isomorphic.

*Proof.* Let  $l \rightarrow r \in R$ . Then  $l \sim u[l'\sigma] \xrightarrow{1}{S} u[r'\sigma] \sim r$  for some  $l' \rightarrow r' \in S$ . Similarly,  $l' \sim v[\bar{l}\mu] \xrightarrow{1}{R} v[\bar{r}\mu] \sim r'$  for some  $\bar{l} \rightarrow \bar{r} \in R$ . That is,  $l \sim u[v[\bar{l}\mu]\sigma]$  and  $r \sim u[v[\bar{r}\mu]\sigma]$ . By minimality,  $u$  and  $v$  are empty and  $\sigma$  and  $\mu$  are no more than renamings (on the

appropriate sets of variables,  $\mu$  on variables of  $\bar{l}$ ,  $\sigma$  on variables of  $\bar{l}\mu$ ). Hence,  $l \sim u[l'\sigma] = l'$  and  $r \sim u[r'\sigma] = r'$  as desired up to renaming.  $\square$

**COROLLARY 12.** *Let  $R$  and  $S$  be deductively equivalent confluent rewrite systems  $\text{mod } \sim$ . If  $R$  and  $S$  are noninverting and  $R$  and  $S$  are each reduced and minimal, then  $R$  and  $S$  are isomorphic.*

This extends a result of Lankford and Ballantyne [22] in which they assume that the  $\sim$ -congruence classes are finite. None of the conditions can be omitted.

**LEMMA 13.** *A rewrite system  $R \text{ mod } \sim$  is minimal if no left-hand side  $l$  is reducible by another rule, and if  $l \not\sim c[l\sigma]$  for any nonempty context  $c$  or substitution  $\sigma$  that is more than renaming.*

*Proof.* Straightforward from the definition of minimality.  $\square$

**LEMMA 14.** *A terminating rewrite system  $R \text{ mod } \sim$  is reduced if it is minimal and no rule applies to either the left- or right-hand side of another rule.*

*Proof.* Let  $l \rightarrow r \in R$ . We are given that no other rule applies to  $l$  and by minimality,  $\sim$  does not create any nontrivial way for  $l$  to apply to itself. Furthermore, the right-hand side  $r$  must be a normal form, since no other rule applies, and were  $l$  to apply, the system would not be terminating. Hence  $R$  is reduced,  $\square$

**COROLLARY 15.** *A rewrite system  $R \text{ mod } \sim$  is minimal if no left-hand side  $l$  is reducible by another rule and  $\sim$  generates only finite congruence classes.*

*Proof.* Were  $l \sim c[l\sigma]$  for nonempty context  $c$ , then there would be an infinite congruence class  $l \sim c[l\sigma] \sim c[c[l\sigma]\sigma] \sim \dots$ . Suppose, then, that  $l \sim l\sigma$ . If either  $l$  or  $l\sigma$  contains a variable  $x$  not in the other, then by substituting terms for  $x$  we would obtain an infinite set of congruent terms. So were  $\sigma$  not a renaming it would have to map some variable to a nonvariable, in which case  $l \sim l\sigma \sim l\sigma\sigma \sim \dots$  would be an infinite congruence class.  $\square$

It is however undecidable if  $\sim$ -congruence classes are finite, even if  $\sim$  is presented by a finite, canonical rewrite system. If, however,  $\sim$  is presented by a finite set of ground equations, then finiteness is decidable (see [27]).

**5. Completion.** In this section we consider the relationship between the Knuth-Bendix completion procedure [19] and the existence of a finite canonical rewrite system for an equational theory. The procedure has been extended to the modulo  $\sim$  case by [16], [21], [26], [3], and [4], although we shall not consider that more complicated procedure here.

Let  $l \rightarrow r$  and  $l' \rightarrow r'$  be two (not necessarily distinct) rules in  $R$  whose variables have been renamed, if necessary, so that they are distinct. Assume  $l'$  overlaps  $l$ , that is,  $l = c[v]$  for some context  $c$  and nonvariable subterm  $v$  such that  $v\sigma = l'\sigma$  for some (most general) substitution  $\sigma$  for the variables of  $l$  and  $l'$ . Then the overlapped term  $l\sigma (= c\sigma[l'\sigma])$  can be rewritten as either  $r\sigma$  or  $c\sigma[r'\sigma]$ . These two possibilities are called a *critical pair*.

The *Knuth-Bendix completion procedure* ( $\text{KB}(E, >)$  or just  $\text{KB}$ ) takes as input a finite set  $E$  of equations and (a program to compute) a well-founded ordering  $>$  that is closed with respect to substitution and subterm replacement, i.e., a well-founded partial order on  $T(L(E), X)$  such that  $c[s\sigma] > c[t\sigma]$  where  $s > t$ , for any context  $c$ , substitution  $\sigma$ , and terms  $s$  and  $t$ .

The procedure consists of the following steps:

Repeat as long as equations are left in  $E$ . If none remain, terminate successfully.

- (1) Remove an equation  $s \equiv t$  (or  $t \equiv s$ ) from  $E$  such that  $s > t$ . If none exists, terminate with failure (abort).
- (2) Add the rule  $s \rightarrow t$  to  $R$ .
- (3) Use  $R$  to reduce the right-hand sides of existing rules.

- (4) Add to  $E$  all critical pairs formed using the new rule.
- (5) Remove from  $R$  all other rules whose left-hand side contains an instance of  $s$ .
- (6) Use  $R$  to reduce both sides of equations in  $E$ . Remove any equation whose reduced sides are identical.

We use the notation  $R \cup E \xrightarrow{\text{KB}} R' \cup E'$  to indicate the effect of one iteration of the above procedure. For an abstract version see [4]. The procedure can be optimized in various ways that do not concern us here.

**THEOREM 16** [19], [14]. *If KB terminates successfully for input equations  $E$ , then it returns as output a finite canonical system  $R$  for  $E$ .*

The completion procedure may fail in one of two fashions: it may be unable to add any rule because  $s$  and  $t$  are incomparable under the given ordering  $>$  for all equations  $s \equiv t$  in  $E$ ; or it may go on generating an infinite number of new rules without ever finding a finite canonical system. In the former case, we will say that the procedure *aborts*, in the latter that it *loops*. Clearly,  $\text{KB}(E, >)$  only generates rules  $l \rightarrow r$  that are true in the theory  $E$ , i.e.,  $E \vdash l \equiv r$ , and for which  $l > r$ .

**THEOREM 17** ([8], based on [14]). *Let  $R$  be a reduced finite canonical rewrite system for a finite equational theory  $E$ , and let KB be implemented by a “fair” scheduler, i.e., no critical pair that persists is ignored indefinitely. Then  $\text{KB}(E, \frac{+}{R})$  will generate  $R$  if it does not abort.*

*Proof.* Assume KB does not abort. Then in the limit (i.e., considering only those rules which persist from some point on) a reduced canonical system  $S$  will be generated for  $E$  (see [14] and [3]). By Theorem 5,  $R = S$ . KB terminates once  $R$  is generated, since critical pairs must now reduce to triviality.  $\square$

Even if  $R$  is a reduced finite canonical rewrite system for  $E$ ,  $\text{KB}(E, \frac{+}{R})$  need not generate  $R$  since it might abort. By Theorem 2,  $\text{KB}(E, \frac{+}{R})$  will not abort on the first step.

*Example 12.* Despite the existence of  $R = \{m \rightarrow c, n \rightarrow c, fc \rightarrow c\}$ ,  $\text{KB}(\{fn \equiv c, fm \equiv m, m \equiv n\}, \frac{+}{R})$ , aborts on all paths (see also [1].)

*Example 13.* As pointed out in [19], varying the procedure by expanding the language (adding rules  $s \rightarrow h(\bar{x})$  and  $t \rightarrow h(\bar{x})$ , where  $h$  is a new function symbol and  $\bar{x}$  are the variables in an unorientable equation  $s \equiv t$ ) may sometimes circumvent the abort case of completion. However, the following example shows that such expansions may also cause KB to become nonterminating:  $\{fm \equiv m, fn \equiv c, m \equiv n\} \xrightarrow{\text{KB}} \{fh \equiv h, fh \equiv c, m \rightarrow h, n \rightarrow h\} \xrightarrow{\text{KB}} \{fh \rightarrow h, h \equiv c, m \rightarrow h, n \rightarrow h\} \xrightarrow{\text{KB}} \{fk \equiv k, c \rightarrow k, h \rightarrow k, m \rightarrow k, n \rightarrow k\} \xrightarrow{\text{KB}} \dots \xrightarrow{\text{KB}} \{fl \equiv l, k \rightarrow l, \dots\} \xrightarrow{\text{KB}} \dots$  (using the subterm ordering for  $>$ ).

We now show that backtracking is sometimes necessary, that is, there are  $E$  and  $>$  for which  $\text{KB}(E, >)$  can either abort or succeed depending on the sequence in which the equations are chosen for orienting.

*Example 14.* Given the theory

$$E = \{k \equiv m, k \equiv n, fm \equiv m, fk \equiv c\}$$

and the ordering

$$k > m, n,$$

$$m, n, fm, fn, fc > c,$$

$$fm > m,$$

$$fn > n,$$

$$\#(m, n, fc), \#(k, fm, fn, fc), \text{ etc. } (\# \text{ designates pairwise incomparability})$$

Knuth–Bendix can either succeed and find:

$$E \xrightarrow{\text{KB}} \{k \rightarrow m, m \equiv n, fm \equiv m, fm \equiv c\} \xrightarrow{\text{KB}} \{k \rightarrow m, m \equiv n, fm \rightarrow m, m \equiv c\} \\ \xrightarrow{\text{KB}} \{k \rightarrow c, n \rightarrow c, fc \rightarrow c, m \rightarrow c\}$$

or else it can fail (abort) along the path:

$$E \xrightarrow{\text{KB}} \{n \equiv m, k \rightarrow n, fm \equiv m, fn \equiv c\} \xrightarrow{\text{KB}} \cdots \xrightarrow{\text{KB}} \{n \equiv m, k \rightarrow n, fm \rightarrow m, fn \rightarrow c\}.$$

Hence, KB should backtrack to try alternative choices, even for ground systems. Note that if in the previous example  $n$  and  $k$  were incomparable or the given order were extended to include either  $m > n$  or  $n > m$ , then KB would succeed along every path.

In general, we have Theorem 18.

**THEOREM 18.** *If the set of terms  $T(L, X)$  is totally ordered by  $>$ , then  $\text{KB}(E, >)$  does not need to backtrack.*

*Proof.* Obviously, KB cannot abort, and we have already seen (Theorem 17) that KB cannot both succeed and loop.  $\square$

As pointed out in [20], there is always such a total ordering for ground systems. As pointed out in [15], there need not be such an ordering for nonground systems.

**6. Conclusion.** We have discussed the question of when a given decidable equational theory possesses a canonical rewrite system in the original or expanded language. We have pointed to some inherent limitations in the ability of the Knuth–Bendix completion procedure to discover appropriate rewrite systems, even if the procedure is extended to backtrack upon failure or to introduce new function symbols. These limitations can be partially circumvented by allowing more deduction in the procedure (see [20], [3], [4]). In practice completion is frequently quite effective. We have shown that barring failure (and under reasonable conditions) the procedure will find the same rewrite system regardless of the choices made.

One of the remaining open questions is: Suppose there exists some finite canonical system for an equational theory  $E$ . Must there exist an ordering  $>$  for which the completion algorithm, given  $E$  and  $>$ , has a successful outcome. Another area worth investigating is the extent to which systems that rewrite modulo a congruence (whose classes are all finite) are sure to exist for decidable  $E$ .

**Acknowledgments.** The authors gratefully acknowledge discussions with Dallas Lankford and Pierre Lescanne.

#### REFERENCES

- [1] J. AVENHAUS, *On the termination of the Knuth–Bendix completion algorithm*, 120/84, Universität Kaiserslautern, Kaiserslautern, West Germany, 1985.
- [2] ———, *On the descriptive power of term rewriting systems*, J. Symbolic Computation, 2 (1986), pp. 109–122.
- [3] L. BACHMAIR, *Proof methods for equational theories*, Ph.D. thesis, University of Illinois, Urbana, IL, 1986.
- [4] L. BACHMAIR, N. DERSHOWITZ, AND J. HSIANG, *Orderings for equational proofs*, Proc. Symposium on Logic in Computer Science, Cambridge, MA, June, 1986, pp. 346–357.
- [5] R. BOOK, *Thue systems as rewriting systems*, Proc. First International Conference on Rewriting Methods and Techniques, Dijon, France, 1985.
- [6] G. BUTLER AND D. LANKFORD, *Experiments with computer implementations of procedures which often derive decision algorithms for the word problem in abstract algebras*, MTP-7, Louisiana Tech. University, Ruston, LA, August, 1980.

- [7] M. DAVIS, *Computability and Unsolvability*, McGraw-Hill, New York, London, Toronto, 1958.
- [8] N. DERSHOWITZ, *Applications of the Knuth-Bendix completion procedure*, Proc. Seminaire d'Informatique Théoretique, Univ. Paris VII, Paris, 1983.
- [9] ———, *Computing with rewrite systems*, Inform. and Control, 65 (1985), pp. 122-157.
- [10] N. DERSHOWITZ AND J.-P. JOUANNAUD, *Rewrite systems*, in Handbook of Theoretical Computer Science, A. Meyer, M. Nivat, M. Paterson, and M. Perrin, eds., North-Holland, Amsterdam, to appear.
- [11] N. DERSHOWITZ AND L. MARCUS, *Existence and construction of rewrite systems*, ATR-82(8478)-3, The Aerospace Corporation, December, 1982.
- [12] F. FAGES, *Le système KB: manuel de référence: présentation et bibliographie, mise en oeuvre*, R. G. 10.84, Greco de Programmation, C.N.R.S., Bordeaux, France, 1984.
- [13] S. GERHART, D. MUSSER, D. THOMPSON, D. BAKER, R. BATES, R. ERICKSON, R. LONDON, D. TAYLOR, AND D. WILE, *An overview of AFFIRM. A specification and verification system*, in Information Processing, IFIP, North-Holland, Amsterdam, 1980.
- [14] G. HUET, *A complete proof of correctness of the Knuth-Bendix completion algorithm*, J. Comput. System Sci., 23 (1981), pp. 11-21.
- [15] G. HUET AND D. C. OPPEN, *Equations and rewrite rules: a survey*, in Formal Language Theory: Perspectives and Open Problems, R. Book, ed., Academic Press, New York, 1980, pp. 349-405.
- [16] J.-P. JOUANNAUD AND H. KIRCHNER, *Completion of a set of rules modulo a set of equations*, SIAM J. Comput., 15 (1986), pp. 1155-1194.
- [17] D. KAPUR AND P. NARENDRAN, *A finite Thue system with decidable word problem and without equivalent finite canonical system*, Theoret. Comput. Sci., 35 (1985), pp. 337-344.
- [18] D. KAPUR AND G. SIVAKUMAR, *Experiments with and architecture of RRL, a rewrite rule laboratory*, Proceedings of an NSF Workshop on the Rewrite Rule Laboratory, 1983. Available as Report 84GEN008, General Electric Research and Development, Schenectady, NY, April 1984.
- [19] D. KNUTH AND P. BENDIX, *Simple Word Problems in Universal Algebras*, in Computational Problems in Abstract Algebra, J. Leech, ed., Pergamon Press, Oxford, 1970.
- [20] D. LANKFORD, *Canonical inference*, ATP-32, University of Texas, Austin, TX, December, 1975.
- [21] D. LANKFORD AND M. BALLANTYNE, *Decision procedures for simple equational theories with permutative axioms: complete sets of permutative reductions*, ATP-37, Departments of Mathematics and Computer Sciences, University of Texas, Austin, TX, April, 1977.
- [22] ———, *On the uniqueness of term rewriting systems*, unpublished manuscript, 1983.
- [23] P. LESCANNE, *Computer experiments with the REVE term rewriting system generator*, Proc. Tenth ACM Symposium on Principles of Programming Languages, Austin, TX, 1983, pp. 99-108.
- [24] Y. METIVIER, *About the rewriting systems produced by the Knuth-Bendix completion algorithm*, Inform. Process. Lett., 16 (1983), pp. 31-34.
- [25] F. OTTO, *Finite complete rewriting systems for the Jantzen monoid and the Greendlinger group*, Theoret. Comput. Sci., 32 (1984), pp. 249-260.
- [26] G. E. PETERSON AND M. E. STICKEL, *Complete sets of reduction for some equational theories*, J. Assoc. Comput. Math., 28 (1981), pp. 233-264.
- [27] J.-C. RAOULT, *Finiteness results on rewriting systems*, RAIRO Theoretical Informatics, 15 (1981), pp. 373-391.
- [28] W. TAYLOR, *Equational logic*, in Universal Algebra by G. Grätzer, 2nd ed., Springer-Verlag, Berlin, New York, Heidelberg, 1979, pp. 378-400.



## A NEW LOWER BOUND FOR THE SET-PARTITIONING PROBLEM\*

JOHN WELLIAVEETIL JOHN†

**Abstract.** Let  $S$  be a set of  $n$  elements that allows all possible total ordering and on which a total order exists that is initially not known. The problem of determining the  $t$  largest elements of  $S$  using binary comparisons between elements of  $S$  is the set-partitioning problem. In this paper we establish a new lower bound on the number of comparisons required for the set-partitioning problem. For sufficiently large  $n$ , our lower bound improves on all previous lower bounds for this problem for almost all values of  $t$ . Our technique is based on estimating the number of leaves of a decision tree, say  $T$ , for the problem. Let  $P$  denote the set of total orderings allowed on  $S$ . Let  $P'_1, P'_2, \dots, P'_m$  be a partition of  $P$  such that the set of leaves of  $T$  corresponding to  $P'_i$  is disjoint from those corresponding to  $P'_j$  whenever  $i \neq j$ . We determine subsets  $P_1, P_2, \dots, P_r$  of  $P$  and an integer  $k$ , such that for each  $i, 1 \leq i \leq m, P'_i \cap P_j \neq \emptyset$  holds for at most  $k$  distinct  $P_j$ 's. For each subset  $P_j, 1 \leq j \leq r$ , we establish a lower bound, say  $t_j$ , on the number of leaves of a decision tree for the set-partitioning problem when  $S$  is restricted to a total order from  $P_j$ . Then  $T$  has at least  $\sum_{j=1}^r t_j/k$  leaves.

**Key words.** analysis of algorithm, decision tree, lower bounds, partitioning problem

**AMS(MOS) subject classifications.** 68A10, 68A20, 68E99

**1. Introduction.** Let  $S$  be the set of elements  $\{a_1, a_2, \dots, a_n\}$  that allows all possible total ordering and on which a total order exists that is initially not known. The problem of determining the set of  $t$  largest elements of  $S$  is the *set-partitioning problem*. We measure the complexity of this problem in terms of the number of binary comparisons required to determine the set of  $t$  largest elements of  $S$ . The minimax complexity of the set-partitioning problem, denoted by  $U_t(n)$ , is the number of comparisons which is minimum over all permissible algorithms and maximum over all inputs of  $n$  elements for each algorithm. Since the complexity of finding the set of  $t$  largest elements of  $S$  is the same as that of finding the set of  $t$  smallest elements of  $S$ , we restrict ourselves to  $t \leq \lfloor n/2 \rfloor$ .

A problem that is closely related to the set-partitioning problem is the *selection problem*, where we are required to determine the  $t$ th largest element of  $S$ . The minimax complexity of the selection problem is denoted by  $V_t(n)$ . Since after determining the  $t$ th largest element of  $S$ , we know the  $t$  largest elements of  $S$  [12, Problem 2, § 5.3.3], we conclude that  $V_t(n) \geq U_t(n)$ . Both these problems have been extensively studied [1]–[12], [14]–[18] and the advances made in establishing upper and lower bounds for  $V_t(n)$  are given in [8], while those for  $U_t(n)$  are given in [11]. A new lower bound for the selection problem is given in [8]. Here it is shown that

$$V_t(n) \geq n + \lg \binom{n}{t-1} + O\left(\left[\lg \binom{n}{t-1}\right]^{1/2}\right)$$

where  $\lg$  denotes logarithm to the base 2.

The best general lower bound for the set-partitioning problem is due to Kirkpatrick. From [11] we can establish that for all  $t \geq 2$

\* Received by the editors June 23, 1986; accepted for publication (in revised form) June 24, 1987.

† Department of Information Systems and Computer Science, National University of Singapore, Kent Ridge, Singapore 0511.

$$(1) \quad U_t(n) \cong \begin{cases} \left\lfloor \frac{3n+t}{2} \right\rfloor - 3, & 2t \leq n < 3t, \\ n+t-3 + \sum_{j=0}^{t-2} \left\lceil \lg \frac{n-t+1}{t+j} \right\rceil, & n \geq 3t. \end{cases}$$

For small values of  $t$ , the Hadian-Sobel algorithm [6] for determining  $V_t(n)$  may be modified to determine  $U_t(n)$ . This modification yields the upper bound

$$U_t(n) \leq n - t + (t - 1) \lceil \lg(n - t + 1) \rceil.$$

Several modifications of the Hadian-Sobel algorithm for  $V_t(n)$  have been proposed [8], [10], [16] and these in turn may be adapted to give improved upper bounds for  $U_t(n)$  for a wide range of values of  $n$  and  $t$ . For large  $t$ , the algorithm of Schönhage et al. [17] gives the bound

$$U_t(n) \leq V_t(n) \leq 3n + o(n).$$

In this paper we establish a new lower bound for the set-partitioning problem. In § 3 we establish that

$$U_t(n) \geq J_t(n) = n + h$$

where  $h = \lg \binom{n}{t} - 2x - \lg(t(n-t)+1)$  and  $x$  is the least positive integer greater than  $[\lg \binom{n}{t} - \lg(t(n-t)+1)]^{1/2}$ . We further show that  $J_t(n)$  is an improvement over (1) for all  $t > c \lg n$  and  $n > n_0$  for some  $c > 0$  and  $n_0$  a positive integer. We establish that the improvement over this range is greater than  $c't$  for some  $c' > 0$ . In particular, when  $t = \lfloor n/2 \rfloor$ ,  $J_t(n)$  gives an improvement of  $n/4 - O(\sqrt{n})$  comparisons over (1).

**2. Definitions.** Comparison-based algorithms may be represented by a binary tree. We call such a tree a *decision tree* for the algorithm. Let  $T$  be a decision tree for the set-partitioning problem. Each internal node of  $T$  is labelled by a pair of elements from  $S$ . Let  $u$  be an internal node of  $T$  carrying the label  $e : f$ , where  $e$  and  $f$  belong to  $S$ . Then  $u$  denotes the comparison between  $e$  and  $f$  and one of the branches leading from  $u$  represents the relation  $e > f$  and the other the relation  $e < f$ . The *depth* of a node  $v$  is the number of internal nodes that precede  $v$  on the path from the root to  $v$  and hence by our definition the root has depth 0. The *height* of  $T$  is the maximum of the depth of any leaf of  $T$ . The tree  $T$  is *optimum* if it has the least height among all decision trees representing valid algorithms for the problem. The height of an optimal tree is denoted by  $U_t(n)$ .

Let  $u$  be a node of  $T$ . Define  $P_u$  to be the set of relations corresponding to the branches on the path from the root to  $u$ . Thus a decision tree  $T$  for a problem is a binary tree such that if  $v$  is a leaf of  $T$ , then from  $P_v$  and the hypothesis on the allowed orders on  $S$  we can deduce a solution to the problem. Let  $H_u$  be the directed graph, whose vertices correspond to the elements of  $S$ , with the edge  $(e, f)$  in  $H_u$  if and only if the relation  $e > f$  belongs to  $P_u$ . The directed graph  $H_u$  is *weakly connected* if the corresponding undirected graph, obtained by ignoring the direction of the edges in  $H_u$ , is connected.

Let  $Q$  be a set of leaves of  $T$ . Then  $T'(Q)$  is the subtree of  $T$  such that a node in  $T$  belongs to  $T'(Q)$  if and only if it lies on a path from the root to a leaf in  $Q$ . Consider a node that has only one child. The outcome of the comparison corresponding to this node may be deduced from the comparisons already made by the algorithm

and the restriction that the order on  $S$  be consistent with the partial order associated with a leaf in  $Q$ . If  $u$  is a node with one child, say  $v$ , then we delete  $u$  from  $T'(Q)$  and replace it with  $v$ . Repeating this process as many times as possible results in the decision tree  $T(Q)$  that distinguishes between two partial orders on  $S$  corresponding to two distinct leaves in  $Q$ . We call  $T(Q)$  the *subtree of  $T$  defined by  $Q$* .

Let  $W$  be a set of relations consistent with the total ordering on  $S$  and let  $C$  be a subset of  $S$ . Then  $W|C$  is the subset of  $W$  restricted to the elements of  $C$ . An element  $e$  of  $C$  is maximal (minimal) in  $C$ , with respect to  $W|C$ , if and only if there is no element  $f$  in  $C$  such that  $f > e$  ( $f < e$ ) is in  $W|C$ . Since a total order is assumed on  $S$ , every subset of it may be totally ordered. Let  $C[k]$  denote the  $k$ th largest element of  $C$  with respect to the total order on  $S$ .

**3. A lower bound for the general set-partitioning problem.** In this section we present our construction that yields a new lower bound for the set-partitioning problem. We establish a lower bound on the number of leaves of  $T$ , from which a lower bound on the height of  $T$  is obtained. In order to simplify our calculation, we denote the height of  $T$  by  $n+h$ . Our task then is to establish a lower bound on  $h$ . We shall consider values of  $t$  greater than 1, which implies from (1) that  $h$  is greater than 0 for sufficiently large  $n$ . Let  $C$  be a  $t$ -subset of  $S$ . To illustrate our construction, assume without loss of generality that  $C$  is the set  $\{a_1, a_2, \dots, a_t\}$ . For  $1 \leq j \leq t$ , let  $A_j = C - \{a_j\}$  and let  $B_j = S - A_j$ . From the definition of  $T$  it follows that if  $v$  is a leaf of  $T$  then from  $P_v$  we can deduce the set of  $t$  largest elements of  $S$ . Let  $L(C)$  denote the set of leaves of  $T$  that correspond to  $C$  being the set of  $t$  largest elements of  $S$ . Now consider the set of leaves of  $T$  denoted by  $Q_j$ , where  $Q_j = \{v \mid v \in L(A_j \cup \{a\}) \wedge a \in B_j\}$ . The following lemma is useful in establishing a lower bound on  $|Q_j|$ .

LEMMA 3.1. *If  $A_j$  belongs to the set of  $t$  largest elements of  $S$ , then  $T(Q_j)$  is a decision tree that determines the largest element of  $B_j$ .*

*Proof.* It is sufficient to show that if  $v'$  is a leaf of  $T(Q_j)$  then we can deduce from  $P_{v'}$  the largest element of  $B_j$  assuming  $A_j$  belongs to the set of  $t$  largest elements of  $S$ . Note that for every leaf  $v'$  of  $T(Q_j)$  there is a corresponding leaf  $v$  in  $L(A_j \cup \{a'\})$  for some  $a'$  belonging to  $B_j$ . If  $u$  is an internal node of  $T$  on the path from the root to  $v$  then by our construction of  $T(Q_j)$ ,  $u$  is in  $T(Q_j)$  only if the outcome of the comparison corresponding to  $u$  cannot be deduced from  $P_u$  and the hypothesis that  $A_j$  belongs to the set of  $t$  largest elements of  $S$ . From  $P_v$  we can deduce that  $a'$  belongs to the set of  $t$  largest elements of  $S$ . Hence from  $P_{v'}$  we can deduce that  $a'$  is the largest element of  $B_j$  assuming that  $A_j$  belongs to the set of  $t$  largest elements of  $S$ .  $\square$

We next consider the set of permissible orders on  $S$  for which the elements of  $C$ , with the possible exception of one, belong to the set of  $t$  largest elements of  $S$ . More precisely we consider the orders on  $S$  for which  $C[t-1] > (S-C)$  [2]. Let  $Q = \bigcup_{1 \leq i \leq t} Q(i)$ .

LEMMA 3.2. *Under the hypothesis  $C[t-1] > (S-C)$ [2],  $T(Q)$  is a decision tree that determines the set of  $t$  largest elements of  $S$  as  $C - \{\min(C)\} \cup \{\max(\{\min(C)\} \cup S - C)\}$ .*

*Proof.* Let  $v''$  be a leaf in  $T(Q)$  and let  $v$  be the corresponding leaf in  $Q_j$  for some  $j$ ,  $1 \leq j \leq t$ . Let  $v'$  be the leaf in  $T(Q_j)$  corresponding to  $v$ . Our construction of  $T(Q)$  implies that  $P_{v''}$  is contained in  $P_{v'}$ . Note that  $v$  corresponds to  $A_j$  belonging to the set of  $t$  largest elements of  $S$ . This implies that  $P_v$  and hence  $P_{v''}$  do not contain the relation  $a < b$  for any  $a$  in  $A_j$  and  $b$  in  $S - C$ .

We first consider the case when  $v$  corresponds to  $a_j$  belonging to the set of  $t$  largest elements of  $S$ . This implies that  $\min(C) = \max(\min(C) \cup S - C)$  and from  $P_v$

we can deduce  $C$  as the set of  $t$  largest elements of  $S$ . Let  $u$  be an internal node of  $T$  lying on the path from the root to  $v$ . Since  $P_u$  does not contain the relation  $a < b$  for any  $a$  in  $C$  and  $b$  in  $S - C$ , the definition of  $Q$  implies that both the left subtree and the right subtree of  $u$  have leaves belonging to  $Q$ . Hence  $u$  belongs to  $T(Q)$ . This implies that  $P_{v'} = P_v$  and hence from  $P_{v'}$  we can deduce  $C$  as the set of  $t$  largest elements of  $S$ .

Now consider the case where  $v$  corresponds to  $a_j$  not belonging to the set of  $t$  largest elements of  $S$ . Thus  $P_v$  contains the relation  $a_j < b$  for some  $b$  in  $S - C$ . Let  $u$  be the internal node of minimum depth on the path from the root to  $v$  that corresponds to the comparison between  $a_j$  and  $b$  for some  $b$  in  $S - C$  such that  $a_j < b$  is in  $P_v$ . By our choice of  $u$ ,  $P_u$  does not contain the relation  $a < b$  for any  $a$  in  $C$  and  $b$  in  $S - C$ . Thus  $u$  belongs to  $T(Q)$  and hence  $a_j < b$  is in  $P_{v'}$ . Now  $a_j < b$  and  $C[t - 1] > (S - C)[2]$  implies that  $A_j$  belongs to the set of  $t$  largest elements of  $S$  and  $a_j = \min(C)$ . Hence the  $t$  largest elements of  $S$  are  $A_j \cup \{\max(B_j)\}$ . By Lemma 3.1 we can deduce  $\max(B_j)$  from  $P_{v'}$  and the hypothesis  $A_j$  belongs to the set of  $t$  largest elements of  $S$ . Since  $P_{v'}$  is a subset of  $P_{v''}$ , we can deduce  $\max(B_j)$  from  $P_{v''}$  and the hypothesis  $C[t - 1] > (S - C)[2]$ . Hence from  $P_{v''}$  we can deduce the  $t$  largest elements of  $S$  as  $A_j \cup \{\max(B_j)\} = C - \{\min(C)\} \cup \{\max(\min(C) \cup S - C)\}$  assuming  $C[t - 1] > (S - C)[2]$ .  $\square$

Next we turn to the task of establishing a lower bound on  $|Q|$ . This is realized by establishing a lower bound on the number of leaves of  $T(Q)$  a decision tree of height at most  $n + h$  that determines  $C - \{\min(C)\} \cup \{\max(\{\min(C)\} \cup S - C)\}$  as the set of  $t$  largest elements of  $S$  under the hypothesis  $C[t - 1] > (S - C)[2]$ .

Let  $u$  be a node of  $T(Q)$  such that  $H_u$  does not contain an edge directed from an element of  $S - C$  to an element of  $C$ . The next four lemmas establish certain properties of  $u$  that are used in proving Theorem 3.7. This theorem establishes a lower bound on the number of leaves of  $T(Q)$  and hence a lower bound on  $|Q|$ . We define a *straddle* as an edge in  $H_u$  from an element in  $C$  to an element in  $S - C$ . Let  $m_u$  be the set of minimal elements of  $C$  with respect to  $P_u|C$  and let  $M_u$  be the set of maximal elements of  $S - C$  with respect to  $P_u|S - C$ . With each node  $u$  associate a function  $E_u$  with domain as  $m_u$  and whose range is the power set of  $M_u$ . For each  $e$  in  $m_u$  define  $E_u(e) = \{f | f \in M_u \wedge (e, f) \in H_u\}$  and let  $e_u$  be an element in  $m_u$  such that  $|E_u(e_u)| \leq |E_u(e)|$  for all  $e$  belonging to  $m_u$ . Now let  $x$  be the least positive integer satisfying the inequality  $x^2 - 2x \geq h$ . Our assumption of  $h > 0$  implies that  $x$  is greater than 1. The following lemma establishes an upper bound on  $|E_u(e_u)|$ .

LEMMA 3.3. *If  $|m_u| = x$ , then  $|E_u(e_u)| \leq x$ .*

*Proof.* Since  $P_u$  does not contain the relation  $a < b$  for any  $a$  in  $C$  and  $b$  in  $S - C$ , the subtree rooted at  $u$  contains leaves from  $L(C)$ . Since  $e_u$  is a minimal element of  $P_u|C$  there exists a leaf, say  $v$ , among these leaves of  $L(C)$  for which  $e_u$  is a minimal element of  $C$  with respect to  $P_v|C$ . From  $P_v$  we can deduce that  $e_u = \max(\{e_u\} \cup S - C)$ . Hence the subgraph of  $H_v$  induced by  $\{e_u\} \cup S - C$  is weakly connected and must have at least  $n - t$  edges. The subgraph of  $H_u$  induced by  $C$  has at most  $x$  components and hence has at least  $t - x$  edges in it. If  $|E_u(e_u)| > x$ , then  $m_u - \{e_u\}$  must have more than  $x(x - 1)$  straddles between them. This implies that  $H_v$  has more than  $(n - t) + (t - x) + x(x - 1) = n + x^2 - 2x \geq n + h$  edges. Since each edge in  $H_v$  results from a comparison corresponding to a node in  $T(Q)$  we conclude that  $T(Q)$  has height greater than  $n + h$ , contradicting our assumption on the height of  $T(Q)$ . Hence  $|E_u(e_u)| \leq x$ .  $\square$

We next turn to the problem of estimating the number of leaves in the subtree rooted at node  $u$  of the decision tree. The following lemma helps in establishing a lower bound for this problem.

LEMMA 3.4. *The subtree rooted at  $u$  may be pruned such that each leaf of the pruned tree has depth at least*

$$|M_u \cup \{e_u\} - E_u(e_u)| - 1.$$

*Proof.* Let  $e_u$  correspond to  $a_k$ , where  $k \in \{1 \cdots t\}$ . The node  $u$  belongs to  $T(Q_k)$  which, by Lemma 3.1, determines the largest element of  $B_k = \{e_u\} \cup S - C$  assuming  $A_k = C - \{e_u\}$  belongs to the set of  $t$  largest elements of  $S$ . The maximal elements of  $\{e_u\} \cup S - C$  in  $P_u$  is the set  $M_u \cup \{e_u\} - E_u(e_u)$  and hence the subgraph of  $H_u$  induced by  $M_u \cup \{e_u\} - E_u(e_u)$  contains only singletons. Let  $T_u$  denote the subtree of  $T(Q_k)$  rooted at  $u$ . Let  $Q'$  be the set of leaves of  $T_u$  such that  $v$  belongs to  $Q'$  if and only if  $P_v$  does not contain the relation  $a < b$  for any  $a$  in  $A_k$  and  $b$  in  $B_k$ . Hence if  $v$  belongs to  $Q'$  then from  $P_v$  we can deduce the largest element of the set  $M_u \cup \{e_u\} - E_u(e_u)$  without assuming  $C - \{e_u\}$  belongs to the set of  $t$  largest elements of  $S$ . Hence the subgraph of  $H_v$  induced by  $M_u \cup \{e_u\} - E_u(e_u)$  is weakly connected and has at least  $|M_u \cup \{e_u\} - E_u(e_u)| - 1$  edges. Since each of these edges correspond to a comparison we conclude that each leaf of  $T_u(Q')$ , the subtree of  $T_u$  defined by the set of leaves  $Q'$ , has depth at least  $|M_u \cup \{e_u\} - E_u(e_u)| - 1$ . The lemma follows from the fact that  $T_u(Q')$  is a subtree of  $T_u$ .  $\square$

From the symmetry of the problem,  $T(Q)$  is a decision tree that identifies the set  $(S - C) - \{\max(S - C)\} \cup \{\min(\{\max(S - C)\} \cup C)\}$  as the  $n - t$  smallest elements of  $S$ . This allows us to state the following two lemmas that may be proved along the lines of Lemmas 3.3 and 3.4 and hence their proofs are omitted here. As before, with each node  $u$  of  $T(Q)$  we associate a function  $\hat{E}_u$  with domain as  $M_u$  and whose range is the power set of  $m_u$ . For each  $f$  in  $M_u$  define  $\hat{E}_u(f) = \{e \mid e \in m_u \wedge (e, f) \in H_u\}$  and let  $f_u$  be an element in  $M_u$  such that  $|\hat{E}_u(f_u)| \leq |\hat{E}_u(f)|$  for all  $f$  belonging to  $M_u$ .

LEMMA 3.5. *If  $|M_u| = x$ , then  $|\hat{E}_u(f_u)| \leq x$ .*

LEMMA 3.6. *The subtree rooted at  $u$  may be pruned to obtain a decision tree each leaf of which has depth at least*

$$|m_u \cup \{f_u\} - \hat{E}_u(f_u)| - 1.$$

We are now ready to establish a lower bound on the number of leaves in  $T(Q)$ .

THEOREM 3.7. *The decision tree  $T(Q)$  has at least  $2^{n-2x}$  leaves.*

*Proof.* If  $x \geq t$  then  $|E_{\text{root}}(e) = 0|$  for all  $e$  belonging to  $C$  and  $|M_{\text{root}}| = |S - C| = n - t$ . By Lemma 3.4,  $T(Q)$  may be pruned to obtain a decision tree each leaf of which has depth at least  $|S - C|$ . In this case  $T(Q)$  has at least  $2^{n-t}$  leaves which are greater than  $2^{n-2x}$ .

Now assume  $x < t$ . We will prune  $T(Q)$  so that every leaf of the resulting tree has depth at least  $n - 2x$ . We start with the root and use a depth-first strategy to examine the nodes of  $T(Q)$ . Let  $u$  be a node of  $T(Q)$  examined during the pruning strategy. Our pruning is based on the value of  $|m_u|$  and  $|M_u|$ . For nodes with both  $|m_u| > x$  and  $|M_u| > x$ , the strategy given by case (a) is followed while case (b) deals with a node  $u$  for which  $|m_u| = x$  and case (c) for a node with  $|M_u| = x$ . Now  $|M_{\text{root}}| = n - t$ ,  $|m_{\text{root}}| = t$  and  $t \leq \lfloor n/2 \rfloor$  implies that  $|M_{\text{root}}| \geq |m_{\text{root}}| = t > x$ . Hence case (a) of the pruning strategy is applicable to the root.

(a) If  $|m_u| > x$  and  $|M_u| > x$  then the pruning strategy retains a child  $u'$  of  $u$  if and only if  $P_{u'}$  does not contain the relation  $a < b$  for any  $a$  in  $C$  and  $b$  in  $S - C$ . Furthermore, the number of minimal elements in  $m_{u'}$  (number of maximal elements in  $M_{u'}$ ) is at most one less than in  $m_u$  ( $M_u$ ). More specifically, we distinguish the following two cases:

(a.1) Let  $u$  denote the comparison between  $e$  and  $f$  belonging to  $S$ . If  $e$  belongs to  $C$  and  $f$  belongs to  $S - C$  then delete the subtree of the branch correspond-

ing to the outcome  $e < f$  and retain  $u$  and the branch corresponding to the outcome  $e > f$ . Since  $u$  has only one child we do not regard  $u$  as an internal node of the pruned tree. Note that if  $u'$  is the child of  $u$  then  $m_u = m_{u'}$  and  $M_u = M_{u'}$ . We next apply the pruning procedure to  $u'$ .

(a.2) In all other cases retain the node  $u$ . The pruning procedure is applied to each of the children of  $u$ .

(b) If  $|m_u| = x$  then by Lemma 3.4,  $m_u$  has an element  $e_u$  with  $|E_u(e_u)| \leq x$ . Let  $y = |M_u|$ . At least  $t - x$  internal nodes corresponding to comparisons between elements in  $C$  and  $n - t - y$  nodes corresponding to comparisons between elements in  $S - C$  must have preceded  $u$  in the pruned tree and these nodes resulted from case (a.2) given above. Hence each of these nodes has two children. Lemma 3.4 allows us to prune the subtree rooted at  $u$  such that each leaf of the resulting tree, say  $T'_u$ , has depth at least  $|M_u \cup \{e_u\} - E_u(e_u)| - 1$  with respect to  $u$  as the root. Since the depth of  $u$  in  $T(Q)$  is at least  $t - x + n - t - y = n - x - y$ , the depth of each leaf of  $T'_u$  in  $T(Q)$  is at least  $n - x - y + |M_u \cup \{e_u\} - E_u(e_u)| - 1 \geq n - x - y + y - x = n - 2x$ .

(c) The case  $|M_u| = x$  is similar to case (b) given above. From Lemmas 3.5 and 3.6 we conclude that the subtree rooted at  $u$  may be pruned such that each leaf in the pruned subtree has depth at least  $n - 2x$ .

Note that case (a) will never encounter a leaf node. This follows from the observation that if  $u$  is a child of  $u''$  then the pruning procedure will retain  $u$  if and only if  $P_u$  does not contain the relation  $a < b$  for any  $a$  in  $C$  and  $b$  in  $S - C$ . Hence if  $u$  is a leaf then at least  $t - |m_u| + n - t - |M_u| + |m_u||M_u|$  comparisons must have preceded  $u$  since each element in  $m_u$  must have been compared with each element in  $M_u$ . If  $|m_u| > x$  and  $|M_u| > x$ , then since  $x > 1$ , the leaf  $u$  has depth at least  $n + |m_u||M_u| - |m_u| - |M_u| > n + x^2 - 2x \geq n + h$ , contradicting our assumption on the height of  $T$ . Hence the pruning strategy will eventually encounter an internal node for which either  $|m_u| = x$  or  $|M_u| = x$ .

Thus the pruning strategy produces a tree for which every leaf has depth at least  $n - 2x$ . Hence  $T(Q)$  has at least  $2^{n-2x}$  leaves.  $\square$

Having computed a lower bound on the number of leaves in  $Q$ , we get an estimate on the number of leaves of  $T$  from the number of distinct  $Q$ 's and the number of times a specific leaf occurs in the different  $Q$ 's. We can choose the  $t$ -subset  $C$  in  $\binom{n}{t}$  ways. With each  $C$  is associated a  $T(Q)$  with at least  $2^{n-2x}$  leaves. The total number of leaves accounted for in this way is at least  $\binom{n}{t} 2^{n-2x}$ . However we must compensate for a leaf being counted more than once. Observe that if  $C$  and  $C'$  are distinct  $t$ -subsets of  $S$  corresponding to the decision trees  $T(Q)$  and  $T(Q')$ , respectively, of Lemma 3.2, then  $Q$  and  $Q'$  will have a leaf in common if and only if  $C$  and  $C'$  have exactly  $t - 1$  elements in common. For a given  $C$  there are  $t(n - t)$  distinct  $t$ -subsets of  $S$ , each different from  $C$ , that satisfy this property. Hence a leaf is counted at most  $t(n - t) + 1$  times in the summation over all  $t$ -subsets of  $S$ . This implies that  $T$  has at least

$$\frac{\binom{n}{t} 2^{n-2x}}{t(n-t)+1}$$

leaves. Since the height of  $T$  is  $n + h$ , we conclude that

$$2^{n+h} \geq \frac{\binom{n}{t} 2^{n-2x}}{t(n-t)+1}.$$

To summarize, if  $J_t(n)$  denotes our lower bound, then we have shown that

$$\begin{aligned} U_t(n) &\geq J_t(n) = n + h \\ &= n + \lg \binom{n}{t} - 2x - \lg(t(n-t) + 1) \end{aligned}$$

where  $x$  is the least positive integer satisfying  $x^2 - 2x \geq h$ . This implies that  $x$  is the least positive integer greater than or equal to  $[\lg \binom{n}{t} - \lg(t(n-t) + 1)]^{1/2}$ . Thus

$$J_t(n) = n + h = n + \lg \binom{n}{t} + O\left([\lg \binom{n}{t}]^{1/2}\right)$$

for  $t = \Omega(\lg n)$ .

**4. Comparison with Kirkpatrick's bounds.** We next establish that  $J_t(n)$  improves on Kirkpatrick's lower bounds as given by (1) for  $t > c \lg n$  and  $n > n_0$  where  $n_0$  is an integer and  $c$  a positive constant. We further show that for  $t$  in this range the improvement is  $c't$  comparisons for some  $c' > 0$ . Using Stirling's approximation for  $n!$ , a routine calculation establishes that

$$\lg \binom{n}{t} = t(\lg n - \lg t) - (n-t)(\lg(n-t) - \lg n) + O(\log t).$$

Setting  $z = t/n$  we find that over the interval  $2t - 1 \leq n < 3t$ , the difference between  $J_t(n)$  and Kirkpatrick's bound as given by (1) is  $n(-z \lg z - (1-z) \lg(1-z) - (1+z)/2) + O(\sqrt{n})$ . The function  $-z \lg z - (1-z) \lg(1-z) - (1+z)/2$  is convex over the region  $\frac{1}{3} \leq z \leq \frac{1}{2}$  and its minimum value over this interval is 0.25. This implies that our lower bound improves on Kirkpatrick's bound by at least  $0.50t + O(\sqrt{t})$  for  $t$  and  $n$  satisfying  $2t - 1 \leq n < 3t$ . For  $n \geq 3t$  we consider the sum

$$(2) \quad \sum_{j=0}^{t-2} \left[ \lg \frac{n-t+1}{t+j} \right]$$

in (1). Let  $2^k$  be the largest power of 2 that divides  $n/t$ . Then  $n/t = 2^k(1+a)$ , where  $0 \leq a < 1$  and the summand in (2) is less than or equal to  $k+1$ . We note that the summand in (2) can take the value  $k+1$  only when  $j < ta$ . This in turn implies that (2) is less than  $ta(k+1) + k(t-2-ta+1) = t(k+a) - k < t(k+a)$ . Hence the difference between  $J_t(n)$  and Kirkpatrick's lower bound for  $n \geq 3t$  is greater than

$$(3) \quad (t \lg n - t \lg t - tk - ta) + ((n-t)(\lg n - \lg(n-t)) - t) + O\left([\lg \binom{n}{t}]^{1/2}\right).$$

Since  $n = 2^k t(1+a)$  the term  $(t \lg n - t \lg t - tk - ta)$  in (3) reduces to  $t(\lg(1+a) - a)$  which is nonnegative for  $0 \leq a < 1$ . Since  $\ln(1-t/n) = -t/n - t^2/(2n^2) - O(t^3/n^3)$  for  $t/n \leq \frac{1}{3}$ , the term  $((n-t)(\lg n - \lg(n-t)) - t)$  is greater than  $((n-t)/\ln 2) \times [t/n + t^2/(2n^2)] - t$  which is greater than  $c''t$  for  $t/n \leq \frac{1}{3}$ , where  $c''$  is a positive constant. Since  $\lg \binom{n}{t} = O(t \lg n)$  we conclude that (3) is greater than  $c't$  for  $t > c \lg n$  and  $n > n_0$  for some  $c' > 0$ ,  $c > 0$  and  $n_0$  a positive integer.

#### REFERENCES

- [1] S. W. BENT AND J. W. JOHN, *Finding the median requires 2n comparisons*, Proc. 17th Annual ACM Symposium on Theory of Computing, Providence, RI, 1985, pp. 213-216.
- [2] M. BLUM, R. FLOYD, V. PRATT, R. RIVEST, AND R. TARJAN, *Time bounds for selection*, J. Comput. System Sci., 7 (1973), pp. 448-461.

- [3] W. CUNTO AND J. I. MUNRO, *Average case selection*, Proc. 16th Annual ACM Symposium on Theory of Computing, Washington, D.C., 1984, pp. 369-375.
- [4] R. W. FLOYD AND R. L. RIVEST, *Expected time bounds for selection*, Comm. ACM, 18 (1975), pp. 165-172.
- [5] F. FUSSENEGGER AND H. N. GABOW, *A counting approach to lower bounds for selection problems*, J. Assoc. Comput. Mach., 26 (1979), pp. 227-238.
- [6] A. HADIAN AND M. SOBEL, *Selecting the  $i$ th largest using binary errorless comparisons*, Colloq. Math. Soc. János Bolyai, 4 (1969), pp. 585-599.
- [7] L. HYAFIL, *Bounds for selection*, SIAM J. Comput., 5 (1976), pp. 109-114.
- [8] J. W. JOHN, *The complexity of selection problems*, Ph.D. dissertation, Computer Sciences Dept., University of Wisconsin, Madison, WI, May 1985.
- [9] ———, *A new lower bound for the set partition problem*, Tech. Report TRB7/86, Department of Information Systems and Computer Science, National University of Singapore, Kent Ridge, Singapore, 1986.
- [10] D. G. KIRKPATRICK, *Topics in complexity of combinatorial algorithms*, Tech. Report 74, Dept. of Computer Science, University of Toronto, Toronto, Ontario, Canada, 1974.
- [11] ———, *A unified lower bound for selection and set partitioning problems*, J. Assoc. Comput. Mach., 28 (1981), pp. 150-165.
- [12] S. S. KISLITSYN, *On the selection of the  $k$ th element of an ordered set by pairwise comparisons*, Sibirsk. Mat. Zh., 5 (1964), pp. 557-564.
- [13] D. E. KNUTH, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
- [14] J. I. MUNRO AND P. V. POBLETE, *A lower bound for determining the median*, Res. Report CS-82-21, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1982.
- [15] V. R. PRATT AND F. F. YAO, *On lower bounds for computing the  $i$ th largest element*, Proc. 14th IEEE Symposium on Switching and Automata Theory, Iowa City, Iowa, 1973, pp. 70-81.
- [16] P. V. RAMANAN AND L. HYAFIL, *New algorithms for selection*, J. Algorithms, 5 (1984), pp. 557-578.
- [17] A. SCHÖNHAGE, M. PATERSON, AND N. PIPPENGER, *Finding the median*, J. Comput. System Sci., 13 (1976), pp. 184-199.
- [18] C. K. YAP, *New upper bounds for selection*, Comm. ACM, 19 (1976), pp. 501-508.



## ON THE EXPECTED SUBLINEARITY OF THE BOYER-MOORE ALGORITHM\*

R. SCHABACK†

*The author gratefully dedicates this paper to the memory  
of his academic teacher, Professor H. Werner.*

**Abstract.** This paper analyzes the expected performance of a simplified version  $BM'$  of the Boyer-Moore string-matching algorithm. A probabilistic automaton  $A$ , which models the expected behavior of  $BM'$ , is set up under the assumption that both text and pattern are generated by a source which emits independent and uncorrelated symbols with an arbitrary distribution of probabilities. Formal developments lead then to the conclusion that  $A$  takes expected sublinear time in a variety of situations. The sublinear behavior can be quantitatively predicted by simple formulae involving the pattern length  $m$  and the alphabet's probabilistic properties. Finally, empirical evidence is provided which is in satisfactory accordance with the theory.

**Key words.** string searching, pattern matching, average case analysis of algorithms

**AMS(MOS) subject classifications.** 68Q25, 68P10, 68T10

**1. The problem.** Let  $\mathcal{A}$  be a finite alphabet,  $|\mathcal{A}| = n$ , and suppose strings

$$T = t_1 \cdots t_N, \quad t_i \in \mathcal{A}, \quad 1 \leq i \leq N \quad (\text{the "text"}),$$

$$S = s_1 \cdots s_m, \quad s_i \in \mathcal{A}, \quad 1 \leq i \leq m \leq N \quad (\text{the "pattern"})$$

are given. To avoid formal difficulties in later discussions, we assume  $S$  to be expanded to the left by "jokers," i.e., characters that match any other character from  $\mathcal{A}$ .

To determine the leftmost occurrence of  $S$  as a substring of  $T$  means then:

Find the smallest  $j$ ,  $m \leq j \leq N$ , such that  $s_{m-i} = t_{j-i}$  for  $0 \leq i \leq m-1$ , or output that no such  $j$  exists.

To solve this problem, Boyer and Moore [2] defined an algorithm of the following form:

```
j := m; k := 0;
REPEAT
{At this stage, k characters match:  $t_{j-i} = s_{m-i}$ ,  $0 \leq i \leq k-1$ }
  IF  $t_{j-k} = s_{m-k}$ 
  THEN  $k := k+1$ 
  ELSE BEGIN
     $j := j + \text{MOVE}(m, k, t_{j-k})$ ;
     $k := 0$ 
  END
UNTIL ( $k = m$ ) OR ( $j > N$ );
IF  $k = m$  THEN output (Pattern is found at position j)
ELSE output (No occurrence of pattern in text);
```

Variations of this algorithm depend on the function

$\text{MOVE}(m, k: \text{INTEGER}; t: \text{CHARACTER}): \text{INTEGER}$

\* Received by the editors February 24, 1986; accepted for publication (in revised form) October 12, 1987.

† Institut für Numerische und Angewandte Mathematik der Universität Göttingen, Lotzestraße 16-18, D-3400-Göttingen, Federal Republic of Germany.

which determines how many positions  $S$  may be moved forward along  $T$  to the next position where  $S$  can occur as a substring of  $T$ . In case  $k = 0$ , i.e., if the text character  $t = t_j$  does not match the last pattern character  $s_m$ , the pattern can be moved along the text as long as  $t$  does not occur within the pattern:

$$(1) \quad \text{MOVE}(m, 0, t) := \min \{ p \mid 1 \leq p \leq m, s_{m-p} = t \}.$$

For  $k > 0$  there are  $k$  matching characters  $s_{m-k+1} \cdots s_m = t_{j-k+1} \cdots t_j$  and we know that  $s_{m-k} \neq t_{j-k}$ . The simplest way to shift the pattern is to ignore most of this information and to match  $t_{j-k+1} = s_{m-k+1}$  with the next possible occurrence in the pattern:

$$(2) \quad \text{MOVE}(m, k, \cdot) := \text{MOVE}(0, m - k + 1, s_{m-k+1}) \quad (1 \leq k < m).$$

We use the notation  $BM'$  for the Boyer-Moore algorithm using (1) and (2) and consider the  $j$ -increment given by the MOVE function as the *progress* of the algorithm.

The original Boyer-Moore algorithm,  $BM$ , tries to match the whole subpattern  $s_{m-k+1} \cdots s_m$  with its rightmost recurrence in the pattern; an additional condition makes sure that the unsuccessful subpattern  $s_{m-k} \cdots s_m$  does not occur again:

$$\text{MOVE}^*(m, k, \cdot) := \min \left\{ p \mid \begin{array}{l} 1 \leq p \leq m, s_{m-i} = s_{m-p-i}, 0 \leq i \leq k-1 \\ s_{m-k} \neq s_{m-p-k} \text{ if } p+k < m \end{array} \right\} \quad (1 \leq k < m).$$

These definitions of MOVE formally require the pattern  $s$  to be expanded to the left by  $\max(1, m-1)$  jokers.

Since more progress may be possible when matching the rightmost occurrence of  $t = t_{j-k}$  in the subpattern  $s_1 \cdots s_{m-k}$ ,  $BM$  finally takes the maximum of  $\text{MOVE}^*(m, k, \cdot)$  and  $\text{MOVE}(m-k, 0, t)$  to define  $\text{MOVE}(m, k, t)$ . The effective construction of lookup tables for the implementation of MOVE is treated in the cited literature (e.g., [6]).

The worst-case performance of  $BM$  increases linearly with  $N$ , measured by the number of pairwise character comparisons. Knuth, Morris, and Pratt [6] first proved the bound  $7N$  in case the pattern does not occur in the text. This bound has been improved to  $4N$  by Guibas and Odlyzko [5], while the lower bound  $N - m + 1$  for **any** algorithm was established by Rivest [7]. The case of  $r$  occurrences of the pattern in the text led to the bound  $7N + 8rm - 14r$  in [6] (see also [4]), and Galil [4] introduced a variation of  $BM$  to get the bound  $14N$  for **any**  $r$ . Finally, Apostolico and Giancarlo [1] proved the bound  $2N$  for their variation of  $BM$ .

In this paper we neglect the problem of multiple occurrences of the pattern in the text and do not incorporate the corresponding variations of [4] and [1]. These are extensions for handling multiple occurrences and could be added on, if necessary. Since the occurrence of a large random pattern in a large random text is a highly improbable event, a study of the expected behavior of  $BM$  for large patterns can neglect multiple occurrences altogether.

The expected behavior of  $BM$  was studied only for the very special case of equally probable characters. The sublinearity (i.e., the expected number of character comparisons is less than  $N$ ) was proved in the original paper [2] by Boyer and Moore, while Knuth, Morris, and Pratt [6] gave a variation with expected  $\mathcal{O}(N/(m \log m))$  character comparisons. This behavior is optimal due to results of Yao [8].

This paper tries to eliminate the assumption of equally probable characters and to bridge the gap between theory and applications. For patterns and texts from natural languages the observed average behavior of  $BM$  and  $BM'$  is clearly sublinear, and we give useful formulae that predict this behavior.

In addition, the  $\mathcal{O}(N/(m \log m))$  result of [6] is carried over to the general case of arbitrary character probabilities, while a blocked variation of the simplified algorithm  $BM'$  will need  $\mathcal{O}(N/(m \log^2 m))$  comparisons.

**2. Probabilistic assumptions.** In the sequel, we consider an alphabet  $\mathcal{A}$  with characters  $c$  having probabilities

$$(3) \quad 0 < p(c) < 1, \quad \sum_{c \in \mathcal{A}} p(c) = 1.$$

We use  $q$  to denote the probability  $\sum_{c \in \mathcal{A}} p^2(c)$  that two randomly chosen characters match.

We switch now from  $BM'$  to a **completely randomized** model algorithm by assuming that

(1) any reference to some character of the text or the pattern will produce a (possibly new) random character;

(2) the MOVE  $(m, k, t)$  function is replaced by its expected value  $M(m, k)$  for all characters  $t$  and patterns  $S \in \mathcal{A}^m$ .

The first assumption means that each reference to some  $t_{j-k}$  or  $s_{m-k}$  acts like a procedure call that generates a random character independent of  $S, T, k$ , and  $j$ .

This seems to be quite restrictive and unrealistic at first glance, but we shall see later that the progress of the algorithm is so large that multiple references to text characters are rare events under a variety of circumstances. Therefore, the randomized algorithm can be expected to perform on its random data sources in the same way as the deterministic version of  $BM'$  performs on an input that is randomly chosen before execution. Furthermore, later results will show that for alphabets with the probabilistic properties of natural languages, the algorithm  $BM'$  spends most of the time comparing the last pattern character  $s_m$  with text characters  $t_i$  for values of  $i$  that are far away from each other. Then the randomized algorithm will model the behavior of the deterministic algorithm quite well even for natural languages, since natural language characters sampled over large intervals can be considered as random characters with fixed probabilities.

Our randomized algorithm replaces the comparison of  $t_{j-k}$  with  $s_{m-k}$  with a random decision between two alternatives with probabilities  $q$  and  $1 - q$ , respectively. Moreover, the terminating conditions of  $BM'$  (including detection of the pattern and exhaustion of the text) are completely ignored in order to simplify the following discussion. Thus, we get the following algorithm A:

```

j := m; k := 0;
REPEAT
  With probability q : k := k + 1
  ELSE (with probability 1 - q)
    BEGIN
      j := j + M(m, k)
      k := 0
    END
UNTIL FALSE;

```

The variable  $k$  in the algorithm A denotes a "state" corresponding to the situation of  $k$  matches in  $BM$  and  $BM'$ . In this sense A is a probabilistic automaton with an unbounded number of states. Transitions from state  $k$  to state  $k + 1$  occur with the probability  $q$  of a match, yielding no progress in  $j$ . With the probability  $1 - q$  of a mismatch, transitions from state  $k$  to state zero with progress  $M(m, k)$  occur. Reaching state  $m$  corresponds to an occurrence of the pattern in the text; higher states of A are purely formal. Each REPEAT-cycle will be called a *step*, and since  $BM$  has one character comparison for each step, we have one unit of "cost" per step in A. Expected sublinearity means then that the expected progress per step is larger than one.

**3. Theoretical considerations.** The expected behavior of  $A$  is described by the following.

LEMMA 3.1. *The probability  $\mu_{rk}$  that  $A$  is in state  $k$  after  $r$  steps is*

$$\mu_{rk} = q^k(1-q) \quad \text{for all } r > k \geq 0.$$

*The expected progress counted in states up to  $m-1$  is*

$$(4) \quad E_m = (1-q)^2 \sum_{k=0}^{m-1} q^k M(m, k)$$

*after at least  $m$  steps.*

*Proof.* Algorithm  $A$  starts in state 0 with probability one. Then the probability  $\mu_{rk}$  of  $A$  being in state  $k$  after  $r$  steps is

$$\begin{aligned} \mu_{rk} &= q^k(1-q), & 0 \leq k < r, \\ \mu_{rr} &= q^r, \\ \mu_{rk} &= 0, & k > r, \end{aligned}$$

as is easily seen by induction. After  $k$  steps, transitions from state  $k$  to state 0 occur with probabilities  $(1-q)^2 q^k$ , and these lead to the progress  $M(m, k)$ . Summing up gives (4).  $\square$

The lemma implies the following:

- The transient start phase of algorithm  $A$  to reach a step-independent probability for states  $0 \cdots m-1$  is short ( $m$  steps).
- The finiteness of the number  $m+1$  of actual states of  $BM$  and  $BM'$  as opposite to the infinite number of states of  $A$ , does not matter much because choosing small values of  $q$  ensures that higher states of  $A$  are very improbable.

We now evaluate  $M(m, k)$ .

LEMMA 3.2. *If  $S$  is a random string of length  $m$  (with a joker  $s_0$  added) and  $c$  is a random character, the random function*

$$f_m(c, S) = \min \{k \mid 1 \leq k \leq m, s_{m-k} = c\}$$

*has the expected value*

$$(5) \quad F_m = |\mathcal{A}| - \sum_{c \in \mathcal{A}} (1-p(c))^m.$$

*Furthermore,*

$$(6) \quad \begin{aligned} M(m, 0) &= F_m, \\ M(m, k) &= F_{m-k+1} \quad (1 \leq k < m), \\ M(m, k) &= 0 \quad \text{otherwise.} \end{aligned}$$

*Proof.* Clearly,  $f_m$  has the expected value

$$F_m = \sum_{i=1}^{m-1} i \cdot \sum_{c \in \mathcal{A}} (1-p(c))^{i-1} \cdot p^2(c) + m \cdot \sum_{c \in \mathcal{A}} (1-p(c))^{m-1} p(c),$$

because the progress  $i$ ,  $1 \leq i < m$ , implies  $(i-1)$  mismatches and one match, while  $i = m$  implies  $m-1$  mismatches (note that we ignore  $s_m$ ). A little calculation gives (5), and (1), (2) imply (6).  $\square$

We can now write (4) as

$$(7) \quad E_m = (1 - q)^2 \left( F_m + \sum_{k=1}^{m-1} q^k F_{m-k+1} \right)$$

where the values  $F_k$  are available from (5).

For small values of  $m$  we can use (7) and (5) directly to estimate the efficiency of the algorithm  $A$ . Using the probability distributions of characters of natural languages, we can tabulate (7) and (5) via (3). For example, Table 1 exhibits the corresponding values for the distribution of the 26 characters of the English language (data from [3]). Since later examples will show that  $A$  closely resembles  $BM$  and  $BM'$  even for natural languages, the user can easily estimate the expectable progress of  $BM$  and  $BM'$  by looking at such a table. Average patterns  $S$  of length 20 will, for instance, be moved forward about 11 characters per single-character comparison, when searched for in average English texts.

TABLE 1  
 $F_m$  and  $E_m$  for the English alphabet,  $q = 0.0658$ ,  $n = |A| = 26$ .

$m$	$F_m$	$E_m$	$m$	$F_m$	$E_m$
1	1.0000	0.8728	21	12.1401	11.3402
2	1.9342	1.7992	22	12.4420	11.6224
3	2.8081	2.6193	23	12.7304	11.8918
4	3.6263	3.3842	24	13.0059	12.1493
5	4.3935	4.1012	25	13.2695	12.3956
6	5.1136	4.7741	26	13.5218	12.6314
7	5.7902	5.4065	27	13.7636	12.8573
8	6.4268	6.0013	28	13.9954	13.0739
9	7.0263	6.5616	29	14.2178	13.2817
10	7.5915	7.0898	30	14.4313	13.4812
11	8.1250	7.5883	31	14.6366	13.6730
12	8.6291	8.0594	32	14.8339	13.8574
13	9.1059	8.5049	33	15.0238	14.0349
14	9.5573	8.9267	34	15.2067	14.2057
15	9.9851	9.3265	35	15.3829	14.3704
16	10.3911	9.7058	36	15.5528	14.5292
17	10.7765	10.0661	37	15.7168	14.6823
18	11.1429	10.4084	38	15.8750	14.8302
19	11.4916	10.7342	39	16.0279	14.9730
20	11.8236	11.0445	40	16.1756	15.1111

We now prove some lower bounds of  $F_m$ . First we concentrate on the case of small patterns.

LEMMA 3.3. For  $m \leq 1/q$ ,

$$(8) \quad F_m \geq m - m^2 q / 2 \geq m / 2.$$

*Proof.* For any real number  $x \geq 0$  we have

$$1 - (1 - x)^k \geq 1 - e^{-kx} \geq kx - \frac{k^2}{2} x^2$$

and (8) follows from (5).  $\square$

The progress of  $A$  for small patterns from large alphabets with small values of  $q$  (this occurs for natural languages) can be predicted by a useful rule of thumb.

THEOREM 3.1. *The algorithm A has expected progress*

$$E_m \geq (1 - q)(1 - q^2)(m - m^2q/2) \geq (1 - q)(1 - q^2)m/2,$$

$$E_m \approx m/2 \quad \text{for small } q$$

per character comparison, provided that  $2 \leq m \leq 1/q$ .

*Proof.* Combine the two major terms of (7) with (8).  $\square$

In Table 1,  $1/q \approx 15.2$ , so Theorem 3.1 is applicable for pattern lengths up to 15, when the probability distribution of characters in English texts is assumed. Within this range, expected progress is at least about  $m/2$ . This observation for a series of practical cases was the starting point for our investigation.

We now treat the case of large  $m$  but still keep the alphabet fixed. Our main result in this direction is the following.

THEOREM 3.2. *A distribution of  $n$  character probabilities  $p_i$  leads to sublinearity of A for sufficiently large pattern lengths  $m$ , if  $q = \sum_{i=1}^n p_i^2$  satisfies*

$$n(1 - q) > 1.$$

This is the case, if

$$(9) \quad q < \hat{q}_n := 1 - \frac{1}{n}.$$

The proof will be a consequence of the lemma following below, if  $m$  is large enough. If we sort the characters of  $\mathcal{A}$  in the form

$$1 > p_1 \geq p_2 \geq \dots \geq p_n > 0, \quad n = |\mathcal{A}|,$$

then there is some  $\gamma$  satisfying

$$(10) \quad q < \gamma < 1, \quad 1 - p_n \leq \gamma,$$

and we get the following.

LEMMA 3.4.

$$(11) \quad E_m \geq n(1 - q) \left( 1 - q^m - \frac{\gamma^{m+1}}{\gamma - q} \right).$$

*Proof.* Equation (5) implies

$$(12) \quad F_m \geq n(1 - (1 - p_n)^m) \geq n(1 - \gamma^m).$$

Using this in (7) gives

$$(13) \quad \begin{aligned} E_m &\geq (1 - q)^2 \sum_{k=0}^{m-1} q^k F_{m-k} \\ &\geq (1 - q)^2 n \sum_{k=0}^{m-1} q^k (1 - \gamma^{m-k}) \\ &= (1 - q)^2 \left( n \frac{1 - q^m}{1 - q} - n \gamma^m \frac{1 - q^m / \gamma^m}{1 - q / \gamma} \right) \\ &\geq n(1 - q) \left( 1 - q^m - \frac{\gamma^{m+1}}{\gamma - q} \right). \end{aligned}$$

Expected sublinearity of  $A$  means that the expected progress  $E_m$  per character comparison is greater than one. Equation (11) shows that for large patterns the product  $n(1 - q)$  occurs as the maximal expected progress; this proves Theorem 3.2.  $\square$

*Remarks.* (1) The model and the algorithm are in state 0 with probability circa  $1 - q$ . Higher states  $k$  have probability  $q^k(1 - q)$  and are very improbable indeed for small values of  $q$ .

(2) In state 0 the algorithms  $BM$  and  $BM'$  coincide. If  $A$  models  $BM'$  in state 0, then it models  $BM$  in that state, too.

(3) In state 0 the last character  $s_m$  of  $S$  is responsible for the progress. In case of sublinearity this character is tested against different characters from  $T$  in the major part of the character comparisons. Then the probabilistic assumptions are not very restrictive; the model  $A$  will closely resemble  $BM'$  (and  $BM$ ) in state 0 (and in general, because other states are improbable). Furthermore, the behavior of the model  $A$  and the algorithm  $BM'$  then is independent of the probability of pairs of characters; the single-character probabilities are sufficient to describe the situation, even for natural language strings.

(4) A value of  $q \approx 1$  spoils the performance of  $A$  and the quality of  $A$  as a model of  $BM'$ , while a large alphabet size  $n = |\mathcal{A}|$  and a large pattern size  $m$  act favorably.

(5) Inequality (11) shows that the size of the alphabet times the probability of a mismatch is the limiting factor for the efficiency of  $A$  for large patterns. This indicates that further speedup requires large alphabets or blocking strategies that let  $\mathcal{A}$  increase with  $m$ .

(6) The case  $q \approx 1$  would imply that a single character must have a probability close to one. Since always  $q \geq 1/n$  in an  $n$ -character alphabet, the case of a binary alphabet cannot lead to an efficiency larger than one and attains efficiency one only if both characters have equal probabilities. In this case, blocking will improve the performance of the algorithm (see below).

(7) Uniformly distributed character probabilities lead to  $q = 1/n$  and the conditions  $m \leq n$  and  $n > 2$  in Theorems 3.1 and 3.2, respectively.

(8) For states  $k \geq 1$  the efficiency of  $BM$  will exceed that of  $BM'$  (and  $A$ ) locally, because it makes at least the progress of  $BM'$  after any single specific comparison. However,  $BM$  is not superior for every text and pattern, because it may run into unfavorable regions of the text which the simplified version may happen to avoid.

To get a further speedup of the pattern-matching process, large alphabets with small values of  $q$  are needed. Therefore, we consider a  $b$ -fold blocking of the alphabet  $\mathcal{A}$ ,  $|\mathcal{A}| = n$ ,  $1 \leq b < m$  and study first the blocked  $BM$  version proposed by Knuth, Morris, and Pratt ([6, p. 341]). Their result (and proof technique) can be generalized as follows.

**THEOREM 3.3.** *There is an algorithm for pattern matching that inspects  $\mathcal{O}(N(\log_{1/q} m/m))$  characters in a random text with arbitrarily distributed characters.*

*Proof.* We follow [6] to combine steps of  $BM$  with an arbitrary linear worst-case algorithm. Each iteration shifts the pattern at least  $m - b$  positions to the right and consists of the following elementary steps:

(1) The last  $b$  characters of the pattern are compared with a block  $B$  of  $b$  text characters.

(2) In case of match, proceed to (4).

In case of mismatch, a function similar to (1) can be used to decide whether  $B$  occurs in the pattern at all.

If this is not the case, the pattern can be moved  $m - b$  positions along the text and the next iteration can be started.

Otherwise proceed to (3).

(3) In this case two blocks of pattern and text match somewhere; we ignore the possible shift given by step (2) and proceed to (4).

- (4) Use a linear worst-case algorithm to move the pattern at least  $m - b$  positions to the right (including the determination of possible occurrences of the pattern in the text) and perform another iteration. This step ignores the fact that there is some match of certain blocks of the pattern and the text.

Note that (1) and (2) are the *BM* part of the algorithm; they are equivalent to a state zero step of *A* on the blocked alphabet. Steps (3) and (4) use no more than  $2m$  characters of the text and therefore require an amount  $c \cdot m$  of work.

For small  $b$  and large  $m$  there will be no overlap of text blocks sampled in step (1) of the iteration. We can therefore assume that these parts of text and pattern are stochastically independent.

**LEMMA 3.5.** *The probability to match a random block  $B$  of  $b$  characters with an arbitrary block of  $b$  characters in a random pattern of  $m \geq b$  characters does not exceed  $(m - b + 1)q^b$ .*

*Proof.* There are  $m - b + 1$  possible positions for  $B$  to occur as a block within the pattern. The probability of occurring at a fixed position is  $q^b$ . The product of these numbers is a crude upper bound for the situation in the assertion.  $\square$

The expected progress of the algorithm in each iteration cycle is at least  $m - b$ , and the expected number of character comparisons is at most

$$(m - b + 1) \cdot q^b \cdot cm + 1 \cdot b$$

where we simply took the upper bound 1 for the probability of a mismatch of a random block. Now we use

$$b := \lfloor 2 \log_{1/q} m \rfloor$$

to get  $q^b \approx m^{-2}$ , and the expected efficiency will be

$$\frac{m - b}{\text{const.} + b} = \mathcal{O}(m / \log_{1/q} m). \quad \square$$

We do not lose too much if we simply apply the following blocked version of *A*.

**THEOREM 3.4.** *If  $A$  works on  $b$ -fold blocks in  $b$  parallel versions with  $b \approx \log_{1/q} m$ , the expected number of single-character comparisons is  $\mathcal{O}(N / m \log_{1/q}^2 m)$ .*

*Proof.* We apply former results for the alphabet  $\mathcal{A}^b$  and first use (8) to get

$$F_r \geq r - \frac{r^2}{2} q^b$$

for  $r$  blocks of  $b$  characters. We ignore the higher-order terms in  $E_r$  and find the lower bound

$$E_r \geq (1 - q^b)^2 F_r \geq r(1 - q^b)^2 \left(1 - \frac{r}{2} q^b\right).$$

Now consider a string of length  $m < q^{-m}$  over  $A$  and a blocking factor  $b$  with

$$m < \frac{1}{q^b}, \quad \text{i.e., } b \geq \lceil \log_{1/q} m \rceil, \quad 1 \leq b \leq m.$$

For simplicity, we can then define  $r \geq 1$  by

$$rb \leq m < (r + 1)b$$



and consider  $r$  blocked steps in our efficiency measure, since the efficiency is monotonic with respect to pattern length. Then

$$E_r \geq \frac{1}{2} \left( \frac{m}{b} - 1 \right) \left( 1 - \frac{1}{m} \right)^2,$$

using

$$rq^b < \frac{r}{m} \leq \frac{1}{b} \leq 1,$$

is a lower bound of our efficiency measure progress/cost in both block-by-block or character-by-character units. Since we need  $b$  versions of the algorithm, one for each block alignment, the total efficiency  $E_{m,b}$  will be at least

$$(14) \quad E_{m,b} \geq \frac{m}{2} \frac{1}{b} \left( \frac{1}{b} - \frac{1}{m} \right) \left( 1 - \frac{1}{m} \right)^2.$$

For  $b^* = \lceil \log_{1/q} m \rceil$  we have  $b^* \leq 1 + \log_{1/q} m$  and get

$$\begin{aligned} E_{m,b^*} &\geq \frac{m}{2} \frac{1}{1 + \log_{1/q} m} \left( \frac{1}{1 + \log_{1/q} m} - \frac{1}{m} \right) \left( 1 - \frac{1}{m} \right)^2 \\ &= \mathcal{O} \left( \frac{m}{\log_{1/q}^2 m} \right) \quad \text{for } m \rightarrow \infty, \end{aligned}$$

without any restrictions on  $n$ ,  $m$ , and  $q$  except  $m < q^{-m}$ , which is always satisfied for large  $m$ .  $\square$

The blocking strategy depends on  $m$  and  $q$ ; for instance, equal probabilities for 0 and 1 in the binary alphabet give  $q = \frac{1}{2}$  and  $b^* = \lceil \log_2 m \rceil$ . We can use (14) for the usual blocking factors  $b = 4, 8, 16, 32$ , provided that  $m < q^{-b}$  holds.

**4. Empirical observations.** To check the validity of our model algorithm  $A$  we tested  $A$ ,  $BM$ , and  $BM'$  on a variety of inputs. For a fixed alphabet  $\mathcal{A}$  with a specified character distribution we generated large samples of random strings  $S$  and  $T$  for values of  $m$  between 2 and 40. For each  $m$  we plotted the expected efficiency (4) of  $A$  versus the means of the observed efficiencies of  $BM$  and  $BM'$  (see Figs. 1–3). For a binary alphabet (see Fig. 1)  $BM$  exceeds  $BM'$  and  $A$  in efficiency. This is due to the fact that the efficiency of  $BM$  may well exceed the value of  $(1 - q) \cdot |\mathcal{A}| = 0.49374 \cdot 2 = 0.98748$ , which essentially bounds the efficiency of its competitors. But our theoretical results indicate that blocking should be used to avoid  $m \gg |\mathcal{A}|$ , and therefore this example is of minor significance.

For larger alphabets ( $|\mathcal{A}| = 8$  in Fig. 2,  $|\mathcal{A}| = 26$  in Fig. 3) the efficiency as modeled by  $A$  does resemble the actual efficiency of both versions of the Boyer–Moore algorithm quite well (the vertical lines denote confidence intervals at the 1 percent error level), but there is a small systematic overestimation of the efficiency of the simplified version that may be credited to multiple evaluations.

For the comparison on natural language strings we used a text of 4785 ASCII characters from a LATEX source of part of a chapter of a course in computer science, written in German. We chose a random sample of patterns occurring in the text (Fig. 4) and in a different chapter of the same course (Fig. 5). Then we plotted the expected efficiency of  $A$  against the observed efficiencies of  $BM$  and  $BM'$  as before. The results indicate once again that  $A$  closely describes the behavior of  $BM$  and  $BM'$ . Of course the examples with occurring patterns (Fig. 4) show an overestimation of the efficiency of  $A$ , because unexpectedly high states occur.

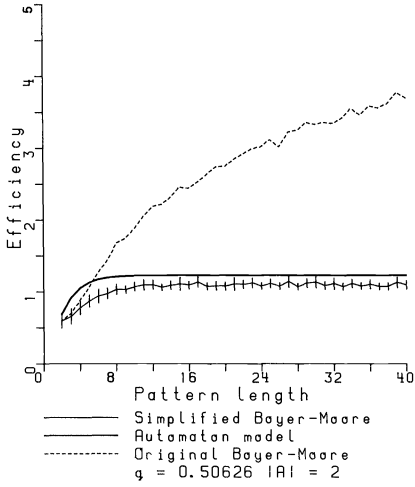


FIG. 1

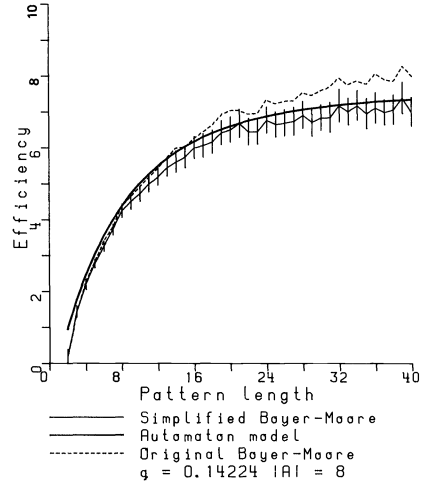


FIG. 2

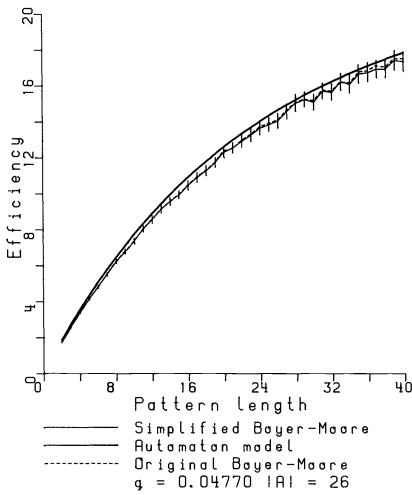


FIG. 3

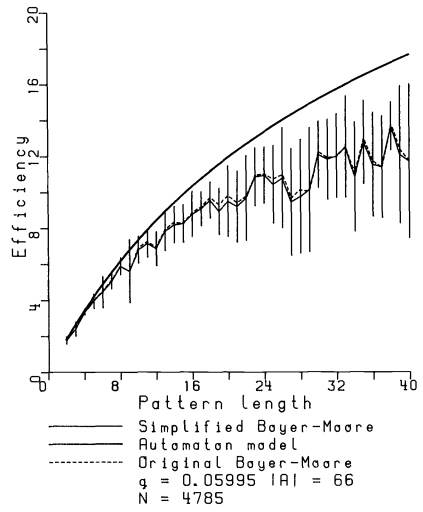


FIG. 4

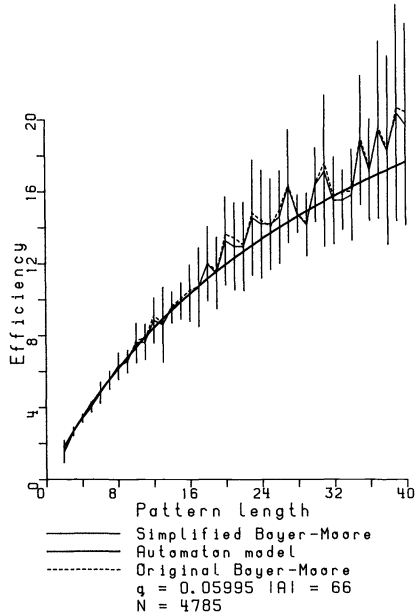


FIG. 5

**Acknowledgments.** The formulation of this paper was significantly improved by the constructive criticism of one of the referees.

## REFERENCES

- [1] A. APOSTOLICO AND R. GIANCARLO, *The Boyer-Moore-Galil string searching strategies revisited*, SIAM J. Comput., 15 (1986), pp. 98-105.
- [2] R. S. BOYER AND J. S. MOORE, *A fast string searching algorithm*, Comm. ACM, 20 (1977), pp. 323-350.
- [3] C. C. FOSTER, *Cryptanalysis for Microcomputers*, Hayden Book Company, 1982.
- [4] Z. GALIL, *On improving the worst case running time of the Boyer-Moore string searching algorithm*, Comm. ACM, 22 (1979), pp. 505-508.
- [5] L. J. GUIBAS AND A. M. ODLYZKO, *A new proof of the linearity of the Boyer-Moore string searching algorithm*, SIAM J. Comput., 9 (1980), pp. 672-682.
- [6] D. E. KNUTH, J. H. MORRIS, AND V. R. PRATT, *Fast pattern matching in strings*, SIAM J. Comput., 6 (1977), pp. 323-350.
- [7] R. L. RIVEST, *On the worst-case behavior of string-searching algorithms*, SIAM J. Comput., 6 (1977), pp. 669-674.
- [8] A. C.-C. YAO, *The complexity of pattern matching for a random string*, SIAM J. Comput., 8 (1979), pp. 368-387.

## LOCALITY, COMMUNICATION, AND INTERCONNECT LENGTH IN MULTICOMPUTERS\*

PAUL M. B. VITÁNYI†

**Abstract.** We derive a lower bound on the average interconnect (edge) length in  $d$ -dimensional embeddings of arbitrary graphs, expressed in terms of diameter and symmetry. It is optimal for all graph topologies we have examined, including complete graph, star, binary  $n$ -cube, cube-connected cycles, complete binary tree, and mesh with wraparound (e.g., torus, ring). The lower bound is technology independent, and shows that many interconnection topologies of today's multicomputers do not scale well in the physical world ( $d = 3$ ). The new proof technique is simple, geometrical, and works for wires with zero volume, e.g., for optical (fibre) or photonic (fibreless, laser) communication networks. Apparently, while getting rid of the "von Neumann" bottleneck in the shift from sequential to nonsequential computation, a new communication bottleneck arises because of the interplay between locality of computation, communication, and the number of dimensions of physical space. As a consequence, realistic models for nonsequential computation should charge extra for communication, in terms of time and space.

**Key words.** multicomputers, complexity of computation, locality, communication, wire length, general communication network, edge-symmetric graph, binary  $n$ -cube, cube-connected cycles, tree, Euclidean embedding, scalability, optical computing

**AMS(MOS) subject classifications.** 68C05, 68C25, 68A05, 68B20, 94C99

**1. The tyranny of physical space.** In many areas of the theory of parallel computation we meet graph-structured computational models. These models encourage the design of parallel algorithms where the cost of communication is largely ignored. Yet it is well known that the cost of computation—in both time and space—vanishes with respect to the cost of communication in parallel or distributed computing. As multiprocessor systems with really large numbers of processors start to be constructed, this effect becomes more and more apparent. Thinking Machines Corporation of Cambridge, Massachusetts, has just marketed the "Connection Machine," a massive multiprocessor parallel computer. The prototype contains microscopically fine-grained processor/memory cells, 65,536 of them, each with 4,096 bits of memory and a simple arithmetical unit. The communication network connecting the processors is packet-switched and based on the binary 16-cube. (A binary  $n$ -cube network consists of  $2^n$  nodes, each node identified by an  $n$ -bit name, and an edge between nodes which differ in a single bit.) This is implemented by packing a cluster of 16 processors and one router circuit on a single chip. The 4,096 routers (in *casu* chips) are connected by 24,576 bidirectional wires in the pattern of the binary 12-cube. The last chapter of [3], "New Computer Architectures and their Relationship to Physics or, Why Computer Science is No Good," expresses the dissatisfaction of the designers with traditional computer science, "which abstracts the wire away into a costless and volumeless idealized connection. [The] old models do not impose a locality of connection, even though the real world does. . . . In classical computation the wire is not even considered. In current engineering it may be the most important thing." Here we shall argue that,

---

\* Received by the editors February 13, 1987; accepted for publication (in revised form) September 16, 1987. This work was supported in part by the Office of Naval Research under contract N00014-85-K-0168, by the Office of Army Research under contract DAAG29-84-K-0058, by the National Science Foundation under grant DCR-83-02391, and by the Defense Advanced Research Projects Agency under contract N00014-83-K-0125. Preliminary results were reported in VLSI Algorithms and Architectures, Lecture Notes in Computer Science 227, Springer-Verlag, Berlin, New York, 1986.

† Massachusetts Institute of Technology, Laboratory for Computer Science, Cambridge, Massachusetts 02139 and Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam, the Netherlands.

while getting rid of the so-called “von Neumann” bottleneck,<sup>1</sup> in the shift from serial to nonserial computing, we run into a new *communication* bottleneck due to the three-dimensionality of physical space.

Models of parallel computation that allow processors to randomly access a large shared memory, such as PRAMs, or rapidly access a large number of processors, such as NC computations, can provide new insights in the inherent parallelizability of algorithms to solve certain problems.<sup>2</sup> For instance, in the form of distributing copies of the entire problem instance, or pieces of the problem instance, among an exponential number of processors in a linear number of steps (i.e., the number of steps in the longest causal chain is linear). Or, as in NC, among a polynomial number of processors in a polylogarithmic number of steps. This sometimes leads to the obscure thought that VLSI technology opens the way to implement tree machines which solve NP-complete problems in linear time. Now, the way a problem instance can be divided and partial answers put together may give genuine insight into its parallelizability. However, it *cannot* give a reduction from an asymptotic exponential time best algorithm in the sequential case to an asymptotic polynomial time algorithm in *any* parallel case. At least, if by “time” we mean time. This is a folklore fact dictated by the Laws of Nature. Namely, if the parallel algorithm uses  $2^n$  processing elements, regardless of whether the computational model assumes bounded fan-in and fan-out or not, it cannot run in time polynomial in  $n$ , because *physical space* has us in its tyranny. Namely, if we use  $2^n$  processing elements of, say, unit size each, then the tightest they can be packed is in a three-dimensional sphere of volume  $N = 2^n$ . Assuming that the units have no “funny” shapes, e.g., are spherical themselves, no unit in the enveloping sphere can be closer to all other units than a distance of radius  $R$  (Fig. 1),

$$(1.1) \quad R = \left( \frac{3N}{4\pi} \right)^{1/3}.$$

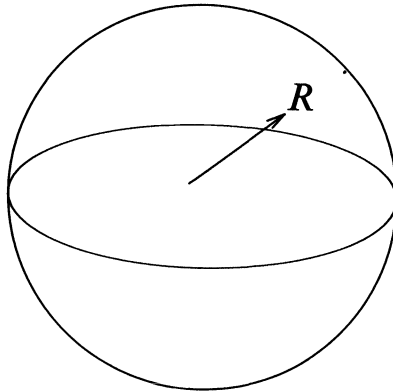


FIG. 1

<sup>1</sup> When the operations of a computation are executed serially in a single Central Processing Unit (CPU), each one entails a “fetch data from memory to CPU; execute operation in CPU; store data in memory” cycle. The cost of this cycle, and therefore of the total computation, is dominated by the cost of the memory accesses which are essentially operation-independent. This is called the “von Neumann” bottleneck, after the brilliant Hungarian mathematician John von Neumann.

<sup>2</sup> For example, in [10] it is demonstrated that any program that requires  $T$  steps on a CRCW PRAM with  $n$  processors and  $m$  shared variables ( $m$  polynomial in  $n$ ) can be simulated by a bounded degree network of  $n$  processors such as the Ultracomputer [7] that runs in deterministic “time”  $O(T(\log n)^2 \log \log n)$  steps.

Unless there is a major advance in physics, it is impossible to transport signals over  $2^{\alpha n}$  ( $\alpha > 0$ ) distance in polynomial  $p(n)$  time. In fact, the assumption of the bounded speed of light says that the lower time bound on *any* computation using  $2^n$  processing elements is  $\Omega(2^{n/3})$  outright. Or, for the case of NC computations which use  $n^\alpha$  processors,  $\alpha > 0$ , the lower bound on the computation time is  $\Omega(n^{\alpha/3})$ .<sup>3</sup> Science fiction buffs may want to keep open the option of embedding circuits in hyper dimensions. Counter to intuition, this does not help—at least, not all the way (see the Appendix). The situation is worse than it appears. At present, many popular multicomputer architectures are based on highly symmetric communication networks with small diameter. Like all networks with small diameter, such networks will suffer from the communication bottleneck above, i.e., they necessarily contain *some* long interconnects (embedded edges). However, the desirable fast permutation properties of symmetric networks do not come free, since they require that the average of *all* interconnects is long. (Note that “embedded edge,” “wire,” and “interconnect” are used synonymously.) This brings us to the main topic of this paper, the analysis of the *amount of wire required*. To prevent arguments that the results have little practical importance because they hold only asymptotically, or because processors are huge and wires thin, we calculate precisely without hidden constants<sup>4</sup> and assume that wires have length but no volume and can pass through everything. The key Theorem 2 in the next section gives a lower bound on the *average* edge length for *arbitrary* graphs that is arguably optimal.

Let us illustrate the novel approach with a popular architecture, say the *binary n-cube*. Recall that this is the network with  $N = 2^n$  nodes, each of which is identified by an  $n$ -bit name. There is a two-way communication link between two nodes if their identifiers differ by a single bit. The network is represented by an undirected graph  $C = (V, E)$ , with  $V$  the set of nodes and  $E \subseteq V \times V$  the set of edges, each edge corresponding with a communication link. There are  $n2^{n-1}$  edges in  $C$ . Let  $C$  be embedded in three-dimensional Euclidean space, each node as a sphere with unit volume. The distance between two nodes is the Euclidean distance between their centers. Let  $x$  be any node of  $C$ . There are at most  $2^n/8$  nodes within Euclidean distance  $R/2$  of  $x$ , with  $R$  as in (1.1). Then, there are  $\geq 7 \cdot 2^n/8$  nodes at Euclidean distance  $\geq R/2$  from  $x$  (Fig. 2). Construct a spanning tree  $T_x$  in  $C$  of depth  $n$  with node  $x$  as the root. Since the binary  $n$ -cube has diameter  $n$ , such a shallow tree exists. There are  $N$  nodes in  $T_x$ , and  $N - 1$  paths from root  $x$  to another node in  $T_x$ . Let  $P$  be such a path, and let  $|P|$  be the *number of edges* in  $P$ . Then  $|P| \leq n$ . Let  $length(P)$  denote the Euclidean length of the embedding of  $P$ . Since  $7/8$ th of all nodes are at Euclidean distance at least  $R/2$  of root  $x$ , the average of  $length(P)$  satisfies

$$(N - 1)^{-1} \sum_{P \in T_x} length(P) \geq \frac{7R}{16}.$$

The average Euclidean length of an embedded edge *in a path*  $P$  is bounded below as follows:

$$(1.2) \quad (N - 1)^{-1} \sum_{P \in T_x} \left( |P|^{-1} \sum_{e \in P} length(e) \right) \geq \frac{7R}{16n}.$$

<sup>3</sup> It is sometimes argued that this effect is significant for large values of  $n$  only, and therefore can safely be ignored. However, in the theory of computation many results are of asymptotic nature, i.e., they hold only for large values of  $n$ , so the effect is especially relevant there.

<sup>4</sup>  $\Omega$  is used sometimes to simplify notation. The constant of proportionality can be reconstructed easily in all cases, and is never very small.

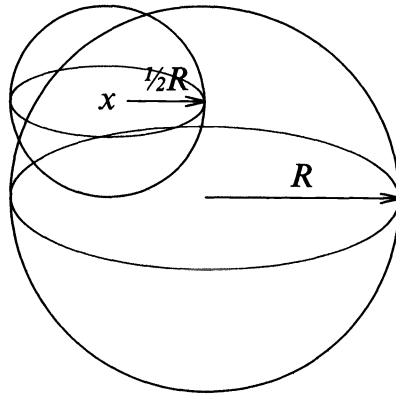


FIG. 2. At most 1/8th of all nodes in the large sphere are also contained in the small sphere centered on  $x$ .

This does *not* give a lower bound on the average Euclidean length of an edge, the average taken *over all edges* in  $T_x$ . To see this, note that if the edges incident with  $x$  have Euclidean length  $7R/16$ , then the average edge length *in each path* from the root  $x$  to a node in  $T_x$  is  $\geq 7R/16n$ , even if all edges not incident with  $x$  have length 0. However, the average edge length *in the tree* is dominated by the many short edges near the leaves, rather than the few long edges near the root. In contrast, in the case of the binary  $n$ -cube, because of its symmetry, if we squeeze a subset of nodes together to decrease local edge length, then other nodes are pushed farther apart increasing edge length again. We can make this intuition precise.

LEMMA 1. *The average Euclidean length of the edges in the three-space embedding of  $C$  is at least  $7R/(16n)$ .*

*Proof.* Denote a node  $a$  in  $C$  by an  $n$ -bit string  $a_1a_2 \cdots a_n$ , and an edge  $(a, b)$  between nodes  $a$  and  $b$  differing in the  $k$ th bit by

$$(a_1 \cdots a_{k-1}a_k a_{k+1} \cdots a_n, a_1 \cdots a_{k-1}(a_k \oplus 1)a_{k+1} \cdots a_n)$$

where  $\oplus$  denotes modulo 2 addition. Since  $C$  is an undirected graph, an edge  $e = (a, b)$  has two representations, namely  $(a, b)$  and  $(b, a)$ . Consider the set  $A$  of automorphisms  $\alpha_{v,j}$  of  $C$  consisting of

(1) modulo 2 addition of a binary  $n$ -vector  $v$  to the node representation, followed by

(2) a cyclic rotation over distance  $j$ .

Formally, let  $v = v_1v_2 \cdots v_n$ , with  $v_i = 0, 1$  ( $1 \leq i \leq n$ ), and let  $j$  be an integer  $1 \leq j \leq n$ . Then  $\alpha_{v,j}: V \rightarrow V$  is defined by

$$\alpha_{v,j}(a) = b_{j+1} \cdots b_n b_1 \cdots b_j$$

with  $b_i = a_i \oplus v_i$  for all  $i$ ,  $1 \leq i \leq n$ .

Consider the spanning trees  $\alpha(T_x)$  isomorphic to  $T_x$ ,  $\alpha \in A$ . The argument used to obtain (1.2) implies that for *each*  $\alpha$  in  $A$  separately, in each path  $\alpha(P)$  from root  $\alpha(x)$  to a node in  $\alpha(T_x)$ , the average of  $\text{length}(\alpha(e))$  over all edges  $\alpha(e)$  in  $\alpha(P)$  is at least  $7R/16n$ . Averaging (1.2) additionally over all  $\alpha$  in  $A$ , the same lower bound applies:

$$(1.3) \quad (N \log N)^{-1} \sum_{\alpha \in A} \left[ (N-1)^{-1} \sum_{P \in T_x} \left( |P|^{-1} \sum_{e \in P} \text{length}(\alpha(e)) \right) \right] \geq \frac{7R}{16n}.$$

Now fix a particular edge  $e$  in  $T_x$ . We sum  $length(\alpha(e))$  over all  $\alpha$  in  $A$ , and show that this sum equals twice the total edge length. Together with (1.3), this will yield the desired result. For each edge  $f$  in  $C$  there are  $\alpha_1, \alpha_2 \in A, \alpha_1 \neq \alpha_2$ , such that  $\alpha_1(e) = \alpha_2(e) = f$ , and for all  $\alpha \in A - \{\alpha_1, \alpha_2\}, \alpha(e) \neq f$ . (For  $e = (a, b)$  and  $f = (c, d)$  we have  $\alpha_1(a) = c, \alpha_1(b) = d$ , and  $\alpha_2(a) = d, \alpha_2(b) = c$ .) Therefore, for each  $e \in E$ ,

$$\sum_{\alpha \in A} length(\alpha(e)) = 2 \sum_{f \in E} length(f).$$

Then, for any path  $P$  in  $C$ ,

$$(1.4) \quad \sum_{e \in P} \sum_{\alpha \in A} length(\alpha(e)) = 2|P| \sum_{f \in E} length(f).$$

Rearranging the summation order of (1.3), and substituting (1.4), yields the lemma.  $\square$

**2. Interconnect length in Euclidean space.** Deriving the total required wire length for embeddings of networks in Euclidean space, I will not make any assumptions about the volume of a wire of unit length, or the way they are embedded in space. Compare this with previous VLSI-related arguments (see e.g., [9]) which are the only other ones on this issue known to me. It is consistent with our results that wires have zero volume, and that infinitely many wires pass through a unit area. Concretely, the problem is posed as follows. Let  $G = (V, E)$  be a finite undirected graph, without loops or multiple edges, embedded in Euclidean  $d$ -space. (For the physical space in which we put our computers,  $d = 3$ .) Let each embedded node have unit volume. For convenience of the argument, each node is embedded as a sphere, and is represented by the single point in the center. The distance between a pair of nodes is the Euclidean distance between the points representing them. The length of the embedding of an edge between two nodes is the distance between the nodes. How large does the average edge length need to be?

Theorem 2 expresses a lower bound on this quantity for any graph, in terms of certain symmetries and diameter. The new argument is based on graph automorphism, graph topology, and Euclidean metric. For each graph topology I have examined, the resulting lower bound turned out to be sharp. This includes the binary  $n$ -cube, cube-connected cycles (CCC), complete graph, star, complete binary tree, and meshes with wraparound such as ring and torus. It could be that the lower bound is optimal in general. All mentioned graphs, except the cube-connected cycles and tree, exhibit a type of symmetry called edge-symmetry. Because of the significance of this class of graphs, in Corollary 4 we set off a lower bound on the average interconnect length for edge-symmetric graphs in general.

**2.1. Lower bound based on symmetry and diameter.** What symmetry of a graph yields large edge length? Not that of the complete binary tree. There the diameter is small, yet the average Euclidean length of an embedded edge is  $O(1)$ . This is borne out by the familiar  $H$ -tree layout [9], where the average edge length is less than 3 or 4. The symmetry property we are after is "edge-symmetry." We recall the definitions from [2]. Let  $G = (V, E)$  be a simple undirected graph, and let  $\Gamma$  be the automorphism group of  $G$ . Two edges  $e_1 = (u_1, v_1)$  and  $e_2 = (u_2, v_2)$  of  $G$  are similar if there is an automorphism  $\gamma$  of  $G$  such that  $\gamma(\{u_1, v_1\}) = \{u_2, v_2\}$ . We consider only connected graphs. The relation "similar" is an equivalence relation, and partitions  $E$  into non-empty equivalence classes, called orbits,  $E_1, \dots, E_m$ . We say that  $\Gamma$  acts transitively on each  $E_i, i = 1, \dots, m$ . A graph is edge-symmetric if every pair of edges are similar ( $m = 1$ ). The following property of orbits is obvious.



*Property.* For each pair of edges  $e_1, e_2 \in E_i$ , the set  $\{\gamma \in \Gamma: \gamma(e_1) = e_2\}$  has  $|\Gamma|/|E_i|$  elements,  $i = 1, \dots, m$ . (Hint: Let  $0 \in E_i$  and  $\Gamma_0 = \{\gamma \in \Gamma: \gamma(0) = 0\}$ . For  $e, f \in E_i$ , define  $\gamma_{ef} \in \Gamma$  by  $\gamma_{ef}(e) = f$ . Fix  $e$  and  $f$  arbitrarily. Then  $\gamma \in \gamma_{e0}\Gamma_0\gamma_{0f}$  if and only if  $\gamma_{e0}^{-1}\gamma\gamma_{0f}^{-1} \in \Gamma_0$ .)

We need the following notions. Let  $D < \infty$  be the *diameter* of  $G$ . If  $x$  and  $y$  are nodes, then  $d(x, y)$  denotes the number of edges in a *shortest path* between them. For  $i = 1, \dots, m$ , define  $d_i(x, y)$  as follows. If  $(x, y)$  is an edge in  $E_i$  then  $d_i(x, y) = 1$ , and if  $(x, y)$  is an edge not in  $E_i$  then  $d_i(x, y) = 0$ . Let  $\Pi$  be the set of shortest paths between  $x$  and  $y$ . If  $x$  and  $y$  are not incident with the same edge, then  $d_i(x, y) = |\Pi|^{-1} \sum_{P \in \Pi} \sum_{e \in P} d_i(e)$ . Clearly,

$$(2.1) \quad d_1(x, y) + \dots + d_m(x, y) = d(x, y) \leq D.$$

Denote  $|V|$  by  $N$ . The *i*th orbit frequency is

$$\delta_i = N^{-2} \sum_{x, y \in V} \frac{d_i(x, y)}{d(x, y)},$$

$i = 1, \dots, m$ . Finally, define the *orbit skew coefficient* of  $G$  as  $M = \min \{|E_i|/|E|: 1 \leq i \leq m\}$ . Consider a  $d$ -space embedding of  $G$ , with embedded nodes, distance between nodes, and edge length as above. Let  $R$  be the *radius* of a  $d$ -space sphere with volume  $N$ , e.g., (1.1) for  $d = 3$ . We are now ready to state the main result. Just in case the reader does not notice, (i) is the most general form.

**THEOREM 2.** *Let graph  $G$  be embedded in  $d$ -space with the parameters above, and let  $C = (2^d - 1)/2^{d+1.5}$*

- (i) *Let  $l_i = |E_i|^{-1} \sum_{e \in E_i} l(e)$  be the average length of the edges in orbit  $E_i$ ,  $i = 1, \dots, m$ . Then,  $\sum_{1 \leq i \leq m} l_i \geq \sum_{1 \leq i \leq m} \delta_i l_i \geq CRD^{-1}$ .*
- (ii) *Let  $l = |E|^{-1} \sum_{e \in E} l(e)$  be the average length of an edge in  $E$ . Then,  $l \geq CRMD^{-1}$ .*

*Proof.* Without loss of generality, we give the proof for the physically relevant case  $d = 3$ . If  $x$  and  $y$  are nodes, let  $l(x, y)$  be the Euclidean *distance* between  $x$  and  $y$  in three-space. For  $i = 1, \dots, m$ , define  $l_i(x, y)$  as follows. If  $(x, y)$  is an edge in  $E_i$ , then  $l_i(x, y) = l(x, y)$ , and if  $(x, y)$  is an edge not in  $E_i$ , then  $l_i(x, y) = 0$ . If  $x$  and  $y$  are not incident with the same edge, then  $l_i(x, y) = |\Pi|^{-1} \sum_{P \in \Pi} \sum_{e \in P} l_i(e)$ , with  $\Pi$  as above. By the triangle inequality,

$$(2.2) \quad l(x, y) \leq l_1(x, y) + \dots + l_m(x, y).$$

Consider Fig. 2 again. Let  $x$  be any node of  $G$ . There are at most  $N/8$  nodes within distance  $R/2$  of  $x$ , with  $R$  given by (1.1). Therefore, there are  $\geq 7N/8$  nodes at distance  $\geq R/2$  from  $x$ , for  $N$  large enough. Thus, the sum of all  $l(x, y)$ , taken over all node pairs  $x, y$ , satisfies

$$(2.3) \quad \sum_{x, y \in V} l(x, y) \geq \frac{7RN^2}{16}.$$

Using (2.1) and (2.2), we obtain from (2.3),

$$(2.4) \quad \sum_{x, y \in V} \sum_{i=1}^m \frac{l_i(x, y)}{d(x, y)} \geq \sum_{x, y \in V} \frac{l(x, y)}{d(x, y)} \geq \frac{7RN^2}{16D}.$$

<sup>5</sup> This constant  $C$  can be improved. For  $d = 3$ ,  $C = 7/16$  is the value of  $c(1 - c^3)$  for  $c = 2^{-1}$ . This function reaches its optimum value  $(3/4)2^{-2/3}$  for  $c = 2^{-2/3}$ . By refining the argument we can improve the constant to  $\frac{3}{4}$ . Namely, to obtain (2.3), sum  $(c, c + dc]R\delta(x, y)$  with  $\delta(x, y) = 1$  if  $cR < l(x, y)$  and  $\delta(x, y) = 0$  otherwise, with  $c$  ranging from 0 to 1, for each pair of nodes  $x, y$ . This replaces  $C = 7/16$  in (2.3) by  $C = \int_0^1 (1 - c^3)dc = \frac{3}{4}$ . Similarly, in two dimensions we can improve  $C$  from  $3/8$  to  $2/3$ .

Now fix a particular edge  $e$  in some  $E_i$ . We average  $l(\gamma(e))$  over all  $\gamma$  in  $\Gamma$ . By the property above, there are precisely  $|\Gamma|/|E_i|$  distinct automorphisms in  $\Gamma$  that map edge  $e$  onto edge  $f$ , for each pair  $e, f \in E_i$ . Therefore, the sum of  $l(\gamma(e))$  over all  $\gamma$  in  $\Gamma$  equals precisely  $|\Gamma|/|E_i|$  times the sum of the lengths of all edges in  $E_i$ . Formally,

$$|\Gamma|^{-1} \sum_{\gamma \in \Gamma} l(\gamma(e)) = |E_i|^{-1} \sum_{f \in E_i} l(f) \quad \text{for each } e \in E_i, \quad i = 1, \dots, m,$$

and therefore, for all  $x, y \in V$ ,

$$(2.5) \quad |\Gamma|^{-1} \sum_{\gamma \in \Gamma} l_i(\gamma(x), \gamma(y)) = |E_i|^{-1} d_i(x, y) \sum_{f \in E_i} l(f) \quad \text{for } i = 1, \dots, m.$$

We now finish the argument. Averaging (2.4) additionally over all  $\gamma$  in  $\Gamma$ , leaves the lower bound invariant:

$$(2.6) \quad |\Gamma|^{-1} \sum_{\gamma \in \Gamma} \sum_{x, y \in V} \sum_{i=1}^m \frac{l_i(\gamma(x), \gamma(y))}{d(\gamma(x), \gamma(y))} \geq \frac{7RN^2}{16D}.$$

By rearranging the summation order in (2.6), and substitution of (2.5), we obtain

$$\sum_{i=1}^m \sum_{x, y \in V} \frac{d_i(x, y)}{d(x, y)} |E_i|^{-1} \sum_{e \in E_i} l(e) \geq \frac{7RN^2}{16D}.$$

That is,  $\sum_{1 \leq i \leq m} \delta_i l_i \geq 7R/(16D)$ . Since  $\delta_i \leq 1, i = 1, \dots, m$ , this proves (i). For the average edge length  $l$ , this yields  $l = \sum_{1 \leq i \leq m} (|E_i|/|E|) l_i \geq M \sum_{1 \leq i \leq m} l_i$ , which proves (ii).  $\square$

*Example 1. Binary  $n$ -cube.* Let  $\Gamma$  be an automorphism group of the binary  $n$ -cube, e.g.,  $A$  in the proof of Lemma 1. Let  $N = 2^n$ . The orbit of each edge under  $\Gamma$  is  $E$ . Substituting  $R, D, m = 1$ , and  $d = 3$  in Theorem 2(i) proves Lemma 1. Denote by  $L$  the total edge length  $\sum_{f \in E} l(f)$  in the three-space embedding of  $C$ . Then

$$(2.7) \quad L \geq \frac{7RN}{32}.$$

Recapitulating, the sum total of the lengths of the edges is  $\Omega(N^{4/3})$ , and the average length of an edge is  $\Omega(N^{1/3} \log^{-1} N)$ . (In two dimensions we obtain in a similar way  $\Omega(N^{3/2})$  and  $\Omega(N^{1/2} \log^{-1} N)$ , respectively.)

*Example 2. Cube-connected cycles.* The binary  $n$ -cube has the drawback of unbounded node degree. Therefore, in the fixed-degree version of it, each node is replaced by a cycle of  $n$  trivalent nodes [9]; whence the name *cube-connected cycles* or CCC. If  $N = n2^n$ , then the CCC version, say  $CCC = (V, E)$ , of the binary  $n$ -cube has  $N$  nodes,  $3N/2$  edges, and diameter  $D < 2.5n$ .

**COROLLARY 3.** *The average Euclidean length of edges in a three-space embedding of CCC is at least  $7R/(120n)$ .*

*Proof.* Denote a node  $a$  by an  $n$ -bit string with one marked bit,  $a = a_1 \dots a_{i-1} \bar{a}_i a_{i+1} \dots a_n$ . There is an edge  $(a, b)$  between nodes  $a = a_1 \dots a_{i-1} \bar{a}_i a_{i+1} \dots a_n$  and  $b = a_1 \dots a_{j-1} \bar{b}_j a_{j+1} \dots a_n$ , if either  $i \equiv j \pm 1 \pmod{n}$ ,  $a_i = b_i$  and  $a_j = b_j$  (edges in cycles), or  $i = j$  and  $a_i \neq b_i$  (edges between cycles). Consider the set  $A$  of automorphisms  $\alpha_{v,j}$ , with  $v = v_1 \dots v_n$  a binary  $n$ -vector and  $j$  an integer  $1 \leq j \leq n$ , such that

$$\alpha_{v,j}(a_1 \dots a_{i-1} \bar{a}_i a_{i+1} \dots a_n) = b_{j+1} \dots b_n b_1 \dots b_j,$$

with  $b_i = \overline{a_i \oplus v_i}$  and  $b_k = a_k \oplus v_k$  for  $k \neq i, 1 \leq k \leq n$ . Clearly,  $A$  is a subgroup of the automorphism group of CCC. The similarity relation induced by  $A$  partitions  $E$  in two orbits: the set of *cycle* edges and the set of *noncycle* edges. Since there are  $N/2$

noncycle edges,  $N$  cycle edges, and  $3N/2$  edges altogether, the orbit skew coefficient  $M$  is  $\frac{1}{3}$ . Substitution of  $R, D, M$ , and  $d = 3$  in Theorem 2(ii) yields the corollary.  $\square$

That is, the *total* edge length is  $\Omega(N^{4/3} \log^{-1} N)$  and the *average* edge length is  $\Omega(N^{1/3} \log^{-1} N)$ . (In two dimensions  $\Omega(N^{3/2} \log^{-1} N)$  and  $\Omega(N^{1/2} \log^{-1} N)$ , respectively.) Similar lower bounds are expected to hold for other fast permutation networks like the *butterfly*, *shuffle-exchange*, and *de Bruijn* graphs.

*Example 3.* Edge-symmetric graphs. Recall that a graph  $G = (V, E)$  is *edge-symmetric* if each edge is mapped to every other edge by an automorphism in  $\Gamma$ . We set off this case especially, since it covers an important class of graphs. (It includes the binary  $n$ -cube but excludes CCC.) Let  $|V| = N$  and  $D < \infty$  be the diameter of  $G$ . Substituting  $R, m = 1$ , and  $d = 3$  in Theorem 2(i) we obtain the following.

**COROLLARY 4.** *The average Euclidean length of edges in a three-space embedding of an edge-symmetric graph is at least  $7R/(16D)$ .*

For the complete graph  $K_N$ , this results in an average wire length of  $\geq 7R/16$ . That is, the average wire length is  $\Omega(N^{1/3})$ , and the total wire length is  $\Omega(N^{7/3})$ .

For the complete bigraph  $K_{1, N-1}$  (the star graph on  $N$  nodes) we obtain an average wire length of  $\geq 7R/32$ . That is, the average wire length is  $\Omega(N^{1/3})$ , and the total wire length is  $\Omega(N^{4/3})$ .

For an  $N$ -node  $\delta$ -dimensional mesh with wraparound (e.g., a ring for  $\delta = 1$ , and a torus for  $\delta = 2$ ; for a formal definition see Appendix), this results in an average wire length of  $\geq 7R/(8\delta N^{1/\delta})$ . That is, the average wire length is  $\Omega(\delta^{-1} N^{(\delta-3)/3\delta})$ , and the total wire length is  $\Omega(N^{(4\delta-3)/3\delta})$ .

To give some indication of the scope of Corollary 4, we note that every edge-symmetric graph with no isolated nodes is node-symmetric or bipartite, by a theorem attributed to Elayne Dauber [2], and that every Cayley graph is symmetric [1]. (A graph is symmetric if it is both node-symmetric and edge-symmetric. A graph is node-symmetric if for each pair of nodes there is an automorphism that maps one to the other.)

*Example 4.* Complete binary tree. The complete binary tree  $T_n$  on  $N - 1$  nodes ( $N = 2^n$ ) has  $n - 1$  orbits  $E_1, \dots, E_{n-1}$ . Here  $E_i$  is the set of edges at level  $i$  of the tree, with  $E_1$  is the set of edges incident with the leaves, and  $E_{n-1}$  is the set of edges incident with the root. Let  $l_i$  and  $l$  be as in Theorem 2 with  $m = n - 1$ . Then  $|E_i| = 2^{n-i}$ ,  $i = 1, \dots, n - 1$ , the orbit skew coefficient  $M = 2/(2^n - 2)$ , and we conclude from Theorem 2(ii) that  $l$  is  $\Omega(N^{-2/3} \log^{-1} N)$  for  $d = 3$ . This is consistent with the known fact  $l$  is  $O(1)$ . However, we obtain significantly stronger bounds using the more general part (i) of Theorem 2. In fact, we can show that one-space embeddings of complete binary trees with  $o(\log N)$  average edge length are impossible.<sup>6</sup>

**COROLLARY 5.** *The average Euclidean length of edges in a  $d$ -space embedding of a complete binary tree is  $\Omega(1)$  for  $d = 2, 3$ , and  $\Omega(\log N)$  for  $d = 1$ .*

*Proof.* Consider  $d$ -space embeddings of  $T_n$ ,  $d \in \{1, 2, 3\}$  and  $n > 1$ . By Theorem 2,

$$(2.8) \quad \sum_{i=1}^{n-1} \delta_i l_i \geq CRD^{-1}.$$

**CLAIM.**  $\delta_i \leq (n - 1)^{-1}$ , for  $i = 1, \dots, n - 1$ .

*Proof of claim.* The proof is by induction on  $n$ . Denote by  $\delta_i^{(n)}$  the  $i$ th orbit frequency of  $T_n$ , the complete binary tree with  $2^n - 1$  nodes. Note that  $T_{n+1}$  consists of two copies of  $T_n$ , with the roots attached to a root node that is in neither

<sup>6</sup> Using  $\sum_{i=1}^{n-1} l_i \geq CRD^{-1}$  instead of (2.8), also yields  $l$  is  $\Omega(1)$  for  $d = 2, 3$ , but only  $l$  is  $\Omega(\log \log N)$  for  $d = 1$ .

of them. Set  $\delta_i^{(j)} = 0$  for  $i \geq j$ . For  $n = 2$  the claim holds trivially. Assume the claim holds for  $n \geq 2$ . Then we prove it holds for  $n + 1$ , as follows. We obtain  $(2^{n+1} - 1)^2 \delta_i^{(n+1)}$  by dividing  $\sum_{x,y \in V} d_i(x,y)/d(x,y)$  in two parts: with both nodes  $x, y$  in the same  $T_n$  subtree and with nodes  $x, y$  in different subtrees. The first subsum equals  $2(2^n - 1)^2 \delta_i^{(n)}$ . To obtain the second subsum, we sum  $d_i(x,y)/d(x,y)$  with  $x$  and  $y$  ranging over the consecutive levels of different  $T_n$ -subtrees (so the shortest path between  $x$  and  $y$  contains the root of  $T_{n+1}$ ). This yields the following recurrences, for each  $i = 1, \dots, n$  (with  $\delta_n^{(n)} = 0$ ):

$$(2^{n+1} - 1)^2 \delta_i^{(n+1)} = 2(2^n - 1)^2 \delta_i^{(n)} + \sum_{j=0}^n 2^j \sum_{k=n-i+1}^n \frac{2^{k-1}}{k+j}.$$

Evaluating the double sum for  $i = n$ , and substituting  $\delta_i^{(n)} \leq (n - 1)^{-1}$ , we find after due computation,  $\delta_i^{(n+1)} \leq n^{-1}$ .  $\square$

Substitution of  $|E_i|, |E|, m$  in the expression for  $l$  in Theorem 2 gives

$$(2.9) \quad l = (2^n - 2)^{-1} \sum_{i=1}^{n-1} 2^{n-i} l_i.$$

Substitute in (2.8) the values of  $C, R$  (depending on  $d$ ) and  $D = 2(n - 1)$ . Next, substitute  $(n - 1)^{-1}$  for  $\delta_i$  and multiply both sides with  $n - 1$ . Use the resulting expression to substitute in (2.9), after rearranging the summation, as follows:

$$l = (2^n - 2)^{-1} \left( \sum_{j=1}^{n-1} 2^{j-1} \sum_{i=1}^{n-j} l_i + \sum_{i=1}^{n-1} l_i \right) \in \Omega \left( 2^{-(1-1/d)n} \sum_{j=1}^{n-1} 2^{(1-1/d)j} \right).$$

Therefore, for  $d = 2, 3$ , we obtain  $l$  is  $\Omega(1)$ . However, for  $d = 1, l$  is  $\Omega(\log N)$ .  $\square$

**2.2. Optimality conjecture.** There is evidence that the lower bound of Theorem 2 is optimal. Namely, it is within a constant multiplicative factor of an upper bound for several example graphs of various diameters. Consider only three-dimensional Euclidean embeddings, and recall the assumption that wires have length but no volume, and can pass through nodes. For the *complete graph*  $K_N$  with diameter 1, the lower bound on the average wire length is  $7R/16$ , while  $2R$  is a trivial upper bound. For the *star graph* on  $N$  nodes the bounds are  $7R/32$  and  $2R$ , respectively. The upper bound on the total wire length to embed the *binary  $n$ -cube* requires more work. Let  $N = 2^n$ .

The construction is straightforward. For convenience we assume now that each node is embedded as a three-space cube of volume 1. Recursively, embed the binary  $n$ -cube in a cube of three-dimensional Euclidean space with *sides* of length  $S_n$ . Use eight copies of binary  $(n - 3)$ -cubes embedded in Euclidean  $S_{n-3} \times S_{n-3} \times S_{n-3}$  cubes, with  $S_{n-3} = S_n/2$ . Place the eight small cubes into the large cube by fitting each small cube into an octant of the large cube. First connect the copies pairwise along the first coordinate to form four binary  $(n - 2)$  cubes. Connect these four pairwise along the second coordinate to form two binary  $(n - 1)$  cubes, which in turn are connected along the third coordinate into one binary  $n$ -cube. This requires no more than  $4 \cdot 2^{n-3}$  wires of length at most  $\sqrt{3/2} \cdot S_n$ , another  $2 \cdot 2^{n-2}$  wires of length at most  $3S_n/2$  and  $2^{n-1}$  wires of length at most  $\sqrt{3} \cdot S_n$ . Assume  $S_1 = 1$  and  $n - 1$  is a multiple of 3. Since  $S_n = 2S_{n-3}$ , we have  $S_n = 2^{(n-1)/3}$ . The *total* wire length  $L(n)$  required to embed the binary  $n$ -cube is

$$L(n) \leq 2^{n-1}(\sqrt{3/2} + 3/2 + \sqrt{3})S_n + 8L(n - 3) \\ \leq \sum_{i=1}^{(n-1)/3} 2^{4i} \cdot 2^{n-1-3i}(\sqrt{3/2} + 3/2 + \sqrt{3}).$$

Substitute  $i = -j + (n - 1)/3$  and round off the bracketed sum to 5 to obtain

$$L(n) < 5 \cdot 2^{4(n-1)/3} \sum_{j=0}^{(n-4)/3} 2^{-j}.$$

Summing the infinite series  $\sum_{j=0}^{\infty} 2^{-j}$  yields an upper bound  $L(n) < 4N^{4/3}$ . Together with Lemma 1, the optimum of the average interconnect length for the binary  $n$ -cube is in between  $7R/16n$  and  $8N^{1/3}/n$ .

For the *cube-connected cycles* with  $N = n2^n$  nodes, we derive an upper bound by the same argument. Squeeze the  $n$  nodes of each cycle in a three-space cube of volume  $n$  in the obvious way. This takes, say, about  $L_1 < n2^n$  total interconnect length for the cycle edges. Recall that each such cycle corresponds to a particular node of the binary  $n$ -cube above. Apply the same construction as for the binary  $n$ -cube with  $S_1 = n^{1/3}$ . Then obtain  $L_2 < 4 \cdot 2^{4n/3} n^{1/3}$  total interconnect length for the edges between cycles. Together with Corollary 3, we obtain that the optimum of the average interconnect length for the cube-connected cycles is in between  $7R/120n$  and  $8N^{1/3}/(3n) + 2/3$ . For  $\delta$ -dimensional meshes with wraparound, with  $\delta = 1, 2, 3$  and diameter  $2^{-1}\delta N^{1/\delta}$ , a lower bound of  $\Omega(1)$  follows from Corollary 4, and the upper bound is  $O(1)$  by the obvious embedding. Note that  $\delta = 1$  is the ring and  $\delta = 2$  is the torus. For the *complete binary tree*, for  $d = 2, 3$ , the  $H$ -tree construction gives an average edge length  $O(1)$  [9], matching the  $\Omega(1)$  lower bound. In the one-dimensional case, the obvious embedding gives  $O(\log N)$  average edge length, matching the lower bound  $\Omega(\log N)$  of Corollary 5.

**2.3. Robustness.** Theorem 2 is robust in the sense that if  $G' = (V', E')$  is a subgraph of  $G = (V, E)$ , and the theorem holds for either one of them, then a related lower bound holds for the other. Essentially, this results from the relation between the orbit frequencies of  $G, G'$ . Let us look at some examples, with  $d = 3$ .

Let a graph  $G$  have the binary  $n$ -cube  $C$  as a subgraph and  $N = 2^n$ . Let  $G$  have  $N' \cong 8N$  nodes and at most  $N' \log N'$  edges. The lower bound on the total wire length  $L(G)$  of a three-space embedding of  $G$  follows trivially from  $L(G) \geq L(C)$ , with  $L(C) \geq 7RN/32$  the total wire length of the binary  $n$ -cube. Therefore, expressing the lower bounds in  $N'$  and radius  $R'$  of a sphere with volume  $N'$  yields  $L(G) \geq 7R'N'/512$ , and the average edge length of  $G$  is at least  $7R'/(512 \log N')$ .

Let the binary  $n$ -cube  $C$  have a subgraph  $G$  with  $n2^{n-1} - 2^{n-5}$  edges. The lower bound on the total wire length  $L(G)$  of a three-space embedding of  $G$  follows from the observation that each deleted edge of  $C$  has length at most twice the diameter  $R$  of (1.1). That is,  $L(G) \geq L(C) - 2^{n-4}R$  with  $L(C)$  as above. Note that  $G$  has  $N' \cong 2^n - (2^{n-6}/n)$  nodes. Therefore, expressing the lower bounds in  $N'$  and radius  $R'$  of a sphere with volume  $N'$  yields  $L(G) \geq 5RN/32 \geq 5R'N'/32$ , and the average edge length of  $G$  is at least  $5R/16n \sim 5R'/(16 \log N')$ .

**3. Interconnect length and volume.** An effect that becomes increasingly important at the present time is that most space in the device executing the computation is taken up by the wires. Under very conservative estimates that the unit length of a wire has a volume which is a constant fraction of that of a component it connects, we can see above that in three-dimensional layouts for binary  $n$ -cubes, the volume of the  $N = 2^n$  components performing the actual computation operations is an asymptotic fastly vanishing fraction of the volume of the wires needed for communication:

$$\frac{\text{volume computing components}}{\text{volume communication wires}} \in o(N^{-1/3})$$

If we charge a constant fraction of the unit volume for a unit wire length and add the volume of the wires to the volume of the nodes, then the volume necessary to embed the binary  $n$ -cube is  $\Omega(N^{4/3})$ . However, this lower bound ignores the fact that the added volume of the wires pushes the nodes further apart, thus necessitating longer wires again. How far does this go? A rigorous analysis is complicated and is not important here. The following intuitive argument indicates well enough what we can expect. Denote the volume taken by the nodes as  $V_n$  and the volume taken by the wires as  $V_w$ . The total volume taken by the embedding of the cube is  $V_t = V_n + V_w$ . The total wire length required to lay out a binary  $n$ -cube as a function of the volume taken by the embedding is, substituting  $V_t = 4\pi R^3/3$  in (2.7),

$$L(V_t) \cong \frac{7N}{32} \left( \frac{3V_t}{4\pi} \right)^{1/3}.$$

Since  $\lim_{n \rightarrow \infty} V_n/V_w \rightarrow 0$ , assuming unit wire length of unit volume, we set  $L(V_t) \sim V_t$ . This results in a better estimate of  $\Omega(N^{3/2})$  for the volume needed to embed the binary  $n$ -cube. When we want to investigate an upper bound to embed the binary  $n$ -cube under the current assumption, we have a problem with the unbounded degree of unit volume nodes. There is no room for the wires to come together at a node. For comparison, therefore, consider the fixed-degree version of the binary  $n$ -cube, the CCC (see above), with  $N = n2^n$  trivalent nodes and  $3N/2$  edges. The same argument yields  $\Omega(N^{3/2} \log^{-3/2} N)$  for the volume required to embed CCC with unit volume per unit length wire. It is known, that every small degree  $N$ -vertex graph, e.g., CCC, can be laid out in a three-dimensional grid with volume  $O(N^{3/2})$  using a unit volume per unit wire length assumption [5]. This neatly matches the lower bound.

Because of current limitations to layered VLSI technology, previous investigations have focused on embeddings of graphs in two-space (with unit length wires of unit volume). We observe that the above analysis for two dimensions leads to  $\Omega(N^2)$  and  $\Omega(N^2 \log^{-2} N)$  volumes for the binary  $n$ -cube and the cube-connected cycles, respectively. These lower bounds have been obtained before, using *bisection-width* arguments and are known to be optimal [9]. It can be even worse, namely, in [6], [12] it is shown that we cannot always assume that a unit length of wire has  $O(1)$  volume (for instance, if we want to drive the signals to very high speed on chip).

**4. Conclusion.** In contrast to other investigations, my goal here was to derive *hard* lower bounds on the total wire length *independent* of the ratio between the volume of a unit length wire and the volume of a processing element. Clearly this is desirable, since this ratio changes with different technologies and granularity of computing components. The arguments we have developed are purely geometrical, apply to any graph, and give optimal lower bounds in all cases we have examined.

Such technology-independent, but huge, lower bounds are a theoretical prelude to many *wiring problems* currently starting to plague computer designers and chip designers alike. Formerly, a wire had magical properties of transmitting data "instantly" from one place to another (or better, to many other places). A wire did not take room, did not dissipate heat, and did not cost anything—at least, not enough to worry about. This was the situation when the number of wires was low, somewhere in the hundreds. Current designs use many millions of wires (on chip), or possibly billions of wires (on wafers). In a computation of parallel nature, most of the time seems to be spent on communication—transporting signals over wires. Thus, thinking that the von Neumann bottleneck has been conquered by nonsequential computation, we are unaware that a non von Neumann communication bottleneck looms large. The following innominate

quote covers this matter admirably:

Without me they fly they think;  
But when they fly I am the wings.

It is clear that these communication mishaps will influence the architecture and the algorithms to be designed for the massive multiprocessors of the future, just like existing algorithms influenced (or were inspired by) the novel architectures of today. What is needed, therefore, are *realistic* formal models for nonsequential computation. In particular, we need to formulate the appropriate cost measures for multicomputer computations. Such costs must account for the communication overhead in (physical) time due to the computer aggregates used in the computation and the overhead in space due to the topology of those aggregates. That is beyond the scope of this paper.

Mesh-connected architectures may be the ultimate solution for interconnecting the extremely large (in numbers) computer complexes of the future. Mesh architectures have desirable properties of *scalability*, *modular extensibility*, and *uniformity*, when embedded in physical space. These notions are generally used in a very loose fashion, and with a great deal of intuition, so I do not try to define them here. Circuits with lower bound  $f(N)$ ,  $f(N) \rightarrow \infty$  for  $N \rightarrow \infty$ , on the average interconnect length do not scale well. ( $N$  is the number of nodes.) Namely, composing a larger such circuit from smaller ones, the average wire length needs to increase. Thus, embeddings of such circuits are not uniformly modular extensible. This positive dependency of the interconnect length on the number of nodes to be connected we call nonscalability.

*Nonscalability.* No edge-symmetric graph on  $N$  nodes with a diameter  $o(N^{1/3})$  is scalable (i.e., uniformly modular extensible) when embedded in physical space.

Tomorrow, optical communication will be used in multicomputers, either wireless by means of lasers/infrared light or by using virtually unlimited bandwidth optical fiber or integrated waveguides [8]. In the current jargon: we can obtain three-dimensional mesh interconnect structures by stacking wafer circuit boards and providing optical interconnections vertically between wafers over the entire wafer in addition to planar connections. This may use hybrid mounting of optical components, combined with integrated optical waveguides and lenses on a large area silicon wafer-scale integrated (WSI) electronic circuit combining electronic and photonic functions [4]. However, it is unlikely that any clever scheme or technology will free us from practical communication problems forever. Even though Nature is not malicious, she is subtle.

**Appendix A.** What happens with embeddings in higher-dimensional spaces? Lest the reader conclude that I indulge in the same avoidance of reality that I decry in others, I have delegated this digression to the Appendix. These mathematical curiosities have no more bearing on realistic formal models for multicomputers than space warps have on the theory of propulsion of space vehicles.

**A.1. Communication and interconnect length in higher dimensions.** Assume that a node (processor) has unit volume, say spherical, in any number  $d$  of dimensions we care to consider. This is in order to obtain comparable reasoning to the physical relevant case of three dimensions. Our intuition about higher-dimensional Euclidean geometry turns out to be quite unreliable. The Euclidean volume  $V_d$  of a  $d$ -dimensional sphere of radius  $R_d$  is

$$V_d = \frac{(R_d)^d \pi^{d/2}}{\Gamma(1 + d/2)},$$

with  $\Gamma$  the gamma function providing a natural generalization of the factorial function. With radius 1 this gives, for dimensions  $d = 1, 2, \dots$ , the volumes 2, 3.14, 4.18, 4.93, 5.26, 4.72, 4.06,  $\dots$ . The volume of the unit radius sphere comes to a maximum for  $d = 5$  and falls off rather rapidly toward zero as  $d$  approaches infinity. On the other hand,  $d$  can be chosen to minimize the radius of a  $d$ -dimensional sphere of volume  $N$ . However, even with the optimal  $d$  (a function of  $N$ ) the radius is  $\Omega(\log^{1/2} N)$ . Namely, setting  $V_d = N$  and  $d = 2k$ , we have

$$N = \left(\frac{\pi^k}{k!}\right) (R_{2k})^{2k} = \frac{(\pi(R_{2k})^2)^k}{k!}.$$

By Stirling's approximation,

$$R_{2k} \sim \sqrt{\frac{k}{e\pi}} (N\sqrt{2\pi k})^{1/k}.$$

Observe that the lower bound in Theorem 2(i) is therefore  $\Omega(N^{1/d} \cdot d^{1/2} D^{-1})$ . Differentiating, we find that  $R_{2k}$  reaches its *minimum*  $R_{2k}^{\min}$  for

$$k \sim \log N,$$

where  $\log$  denotes the natural logarithm. Therefore, with  $N^{1/\log N} = e$ , and  $(2\pi k)^{1/k} \downarrow 1$  for  $k \rightarrow \infty$ , we obtain

$$R_{2k}^{\min} \sim \sqrt{\frac{\log N}{\pi}}.$$

We may think that it is the unfortunate accident of having a physical space of only three dimensions that makes it hard to embed edge-symmetric graphs with small diameter. However, this is not the case. By this analysis and Theorem 2, to embed edge-symmetric graphs of diameter  $o(\log^{1/2} N)$  requires the average length of an embedded edge to rise unbounded with  $N$ , independent of the number of dimensions. As another curiosity, the average edge length of the complete binary tree in  $d > 1$  dimensions is not  $O(1)$ , but turns out to be  $\Omega(d^{1/2})$ . That is, in higher dimensions the  $H$ -tree construction increasingly loses efficiency.

**A.2. Meshes in higher dimensions.** Let  $N = n^\delta$ ,  $n$  a positive integer. Define a  $\delta$ -dimensional *mesh with wraparound* as a set of nodes  $(i_1, \dots, i_\delta)$ ,  $i_j = 0, \dots, N^{1/\delta} - 1$  ( $1 \leq j \leq \delta$ ). Node  $(i_1, \dots, i_\delta)$  is connected by an edge with node  $(j_1, \dots, j_\delta)$ , if they are equal in all coordinates except one where they differ by 1 mod  $N^{1/\delta}$ .

Again assume that a node (processor) has unit volume in any number  $d$  of dimensions we care to consider. For  $d$ -dimensional embeddings of  $N$ -node,  $\delta$ -dimensional meshes with wraparound we have an average interconnect length  $\geq (2^d - 1)R_d / (2^d \delta N^{1/\delta})$ . This lower bound is a small positive constant for  $d \geq \delta$  and  $d$  is small (this is necessary because of the curious behavior of the ratio between volume and radius in higher dimensions). Since the lower bound can be matched by an upper bound, such meshes are feasible architectures for large  $N$ . However, since the average Euclidean interconnect length exceeds

$$\delta^{-1} N^{(\delta-d)/d\delta} \sqrt{\frac{d}{2\pi e}},$$



it rises unbounded with  $N$  for  $\delta > d$ . (It also rises unbounded with  $d$  for fixed  $N$  and  $\delta$ .)

**Acknowledgments.** Remarks by Andries Brouwer, Evangelos Kranakis, F. Tom Leighton, Lambert Meertens, Yoram Moses, and the referees were helpful.

#### REFERENCES

- [1] N. L. BIGGS AND A. T. WHITE, *Permutation Groups and Combinatorial Structures*, Cambridge University Press, Cambridge, 1979.
- [2] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.
- [3] W. D. HILLIS, *The Connection Machine*, MIT Press, Cambridge, MA, 1985.
- [4] L. A. HORNAK, S. K. TEWKSBURY, M. HATAMIAN, A. LIGTENBERG, B. SUGLA, AND P. FRANZON, *Through-wafer optical interconnects for multi-wafer wafer-scale integrated architectures*, Proc. SPIE Conference, San Diego, August, 1986.
- [5] F. T. LEIGHTON AND A. ROSENBERG, *Three-dimensional circuit layouts*, Tech. Report TM-262, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, June, 1984.
- [6] C. MEAD AND M. REM, *Minimum propagation delays in VLSI*, IEEE J. on Solid State Circuits, 17 (1982), pp. 773-775. (Correction: IEEE J. Solid State Circuits, 19 (1984), p. 162.)
- [7] J. T. SCHWARTZ, *Ultracomputers*, ACM Trans. Programming Languages and Systems, 2 (1980), pp. 484-521.
- [8] S. K. TEWKSBURY, L. A. HORNAK, AND A. LIGTENBERG, *The impact of component interconnects on future large scale systems*, Proc. IEEE, to appear.
- [9] J. D. ULLMAN, *Computational Aspects of VLSI*, Computer Science Press, Rockville, MD, 1984.
- [10] E. UPFAL AND A. WIGDERSON, *How to share memory in a distributed system*, J. Assoc. Comput. Mach., 34 (1987), pp. 116-127.
- [11] P. M. B. VITÁNYI, *Non-sequential computation and laws of nature*, in VLSI Algorithms and Architectures, Lecture Notes in Computer Science 227, Springer-Verlag, New York, Berlin, 1986, pp. 108-120.
- [12] ———, *Area penalty for sublinear signal propagation delay on chip*, Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 197-207.

## ISOMORPHISM TESTING OF UNARY ALGEBRAS\*

LUDEK KUČERA† AND VĚRA TRNKOVÁ‡

**Abstract.** Two problems are said to be polynomially equivalent if each is polynomially reducible to the other. A problem is said to be isomorphism-complete if it is polynomially equivalent to the graph isomorphism problem. We prove that the isomorphism problem in any variety of unary algebras is either isomorphism-complete, or solvable in polynomial time. We formulate a condition  $C$  under which the isomorphism problem in a variety  $V$  of unary algebras can be solved in polynomial time. We present an algorithm that can be applied to any pair of unary algebras with the same number of operations and decides whether the algebras are isomorphic or not in  $O(n^3)$  time, provided one of them belongs to a variety satisfying  $C$ . The validity of the last condition is tested by the algorithm in linear time.

**Key words.** isomorphism testing, isomorphism-complete, unary algebras

**AMS(MOS) subject classifications.** 68C25, 08A60

**1. Introduction.** The problem of the complexity of isomorphism testing of different mathematical structures has recently received much attention [17] because it is one of the most important problems in the class NP which is not known to be either NP-complete, or to belong to the class  $NP \cap \text{coNP}$  or even to the class P of polynomial-time computable problems. Originally the problem was to decide the question about the existence of an isomorphism of two graphs. It is known [7] that the isomorphism testing of any finite algebraic or relational structure can be reduced to the graph isomorphism in the sense of either Cook [5] or Karp [10]. Based on this, it was shown that the general graph isomorphism problem is equivalent (in the sense of mutual polynomial-time reducibility) to the isomorphism problem for many other structures, e.g., bipartite graphs, semigroups, automata etc. (see [4]). The problem with the computational complexity equivalent to the complexity of the graph isomorphism problem is usually called *isomorphism-complete*.

As in the theory of NP-completeness, we want to know under what constraints different problems remain isomorphism complete. This approach has already been used in the case of graphs, and the isomorphism completeness of a number of special classes of graphs has been proved. On the other hand, there exist highly nontrivial polynomial-time algorithms solving the isomorphism problem in several interesting classes of graphs (e.g., planar graphs [9], graphs of bounded genus [16], graphs of bounded degree [14] or graphs of bounded eigenvalue multiplicity [3]).

A disadvantage of graphs is that there is no natural and general procedure designed to generate special classes of them in a unified and organic way. From this point of view, there is a more suitable situation in universal algebra. Starting from the class of all universal algebras of a given type (which is known to be isomorphism-complete whenever the type consists of at least two unary operations or at least one  $n$ -ary operation,  $n \geq 2$  [7]), we can specialize the class by imposing equalities, i.e., creating varieties, to obtain a wide scale of various naturally defined classes of algebraic structures. Unlike the case of graphs, we know algebraic varieties in which the isomorphism problem could be simpler than any that is isomorphism-complete, and yet not computable in a polynomially bounded time. For example, the best-known algorithm to solve the group isomorphism problem runs in  $n^{O(\log n)}$  time [15], while all known

\* Received by the editors May 31, 1983; accepted for publication (in revised form) August 28, 1987.

† Department of Applied Mathematics, Charles University, Prague, Czechoslovakia.

‡ Mathematical Institute, Charles University, Prague, Czechoslovakia.

algorithms for the general graph isomorphism problem need time  $\Omega(\exp(\sqrt{cn \log n}))$  [2]. However, all such varieties involve at least binary operations. In the present paper we will show that no matter how rich and wide the scale of varieties of universal algebras is, there are just two degrees of complexity of isomorphism testing in unary algebras, namely isomorphism-completeness and polynomial-time computability (which, of course, can coincide if, e.g.,  $P = NP$ ).

We present the following results:

(i) In any variety of unary algebras, the isomorphism problem is either isomorphism complete or can be solved in polynomially bounded time;

(ii) A condition  $C$  on varieties of unary algebras is formulated such that if a variety  $V$  satisfies  $C$  then the isomorphism problem in  $V$  is polynomial-time solvable, else the problem is isomorphism-complete;

(iii) An algorithm is given that tests in linear time for two arbitrary unary algebras with the same number of operations whether at least one of them belongs to a variety satisfying  $C$ . If so, then it decides in  $O(n^3)$  time the existence or nonexistence of an isomorphism of the given algebras.

**2. Varieties of unary algebras.** A *unary algebra* with  $k$  operations is a tuple  $(X, f_1, \dots, f_k)$ , where  $X$  is a set and each  $f_i$  is a mapping of  $X$  into itself. The operations of a unary algebra  $A = (X, f_1, \dots, f_k)$ , being mappings of  $X$  into itself, can be composed and any word  $w = f_{i_1} \dots f_{i_m}$  is a unary operation again (the empty word is the identity map).

We denote by  $\text{ALG}(k)$  the class of all unary algebras with  $k$  operations. A *variety* in  $\text{ALG}(k)$  is a subclass  $V$  of  $\text{ALG}(k)$  which can be determined by a system of equations. Let us discuss this in more detail.

Denote by  $f^*$  the set of all words over the set  $\{f_1, \dots, f_k\}$  of operation names. Then  $V \subset \text{ALG}(k)$  is a variety if and only if there are sets  $P, \bar{P}$ , and collections  $\{(w_p, w'_p) \mid p \in P\}$ ,  $\{(w_p, w'_p) \mid p \in \bar{P}\}$  of pairs of elements of  $f^*$  such that  $V$  is precisely the class of all algebras  $(X, f_1, \dots, f_k)$  which fulfill the equations (1) and (2) below:

$$(1) \quad w_p(x) = w'_p(x) \quad \text{for all } x \in X, \quad p \in P,$$

$$(2) \quad w_p(x) = w'_p(y) \quad \text{for all } x, y \in X, \quad p \in \bar{P}$$

(where we denote by the same letter the operation name and the actual operation on an algebra).

If  $V$  can be presented by equations of the type (1) only (i.e.,  $\bar{P}$  is empty), it is called a *regular variety*.

Let us present another description of varieties in  $\text{ALG}(k)$ . Let  $M$  be a monoid (i.e., binary associative operation  $\circ$  on  $M$  is given with a unit  $1 \in M$ , i.e.,  $1 \circ m = m \circ 1 = m$  for all  $m \in M$ ). Let  $G_1, \dots, G_k$  be a  $k$ -tuple of generators of  $M$ . Then  $(M, \{G_1, \dots, G_k\})$  determines the following two varieties  $V_1(M, \{G_1, \dots, G_k\})$  and  $V_0(M, \{G_1, \dots, G_k\})$  in  $\text{ALG}(k)$ .

Define a monoid homomorphism  $H: f^* \rightarrow M$  by  $H(f_i) = G_i$  for  $i = 1, \dots, k$ .

$V_1(M, \{G_1, \dots, G_k\})$  is a regular variety defined by the following system of equations:

$$(3) \quad w(x) = w'(x) \quad \text{whenever } H(w) = H(w').$$

$V_0(M, \{G_1, \dots, G_k\})$  is a variety given by all the equations of the system (3) together with all equations of the following form:

$$(4) \quad w(x) = w(y) \quad \text{whenever } H(w) \text{ is a left zero of } M \\ \text{(i.e., } H(w) \circ m = H(w) \text{ for all } m \in M).$$

Now, let a variety  $V$  in  $\text{ALG}(k)$  be given. Denote by  $M(V)$  the monoid  $f^*/\equiv$ , where  $\equiv$  is the following congruence:

$$(5) \quad w \equiv w' \quad \text{if and only if } w(x) = w'(x) \text{ for each algebra } (X, f_1, \dots, f_k) \text{ in } V \text{ and each } x \in X.$$

Denote by  $H : f^* \rightarrow f^*/\equiv = M(V)$  the corresponding monoid-homomorphism. Put  $G_i = H(f_i)$ ,  $i = 1, \dots, k$  and  $M = M(V)$ . Then, clearly,

$$V \subset V_1(M, \{G_1, \dots, G_k\}).$$

We show that

$$\begin{aligned} &\text{either } V = V_1(M, \{G_1, \dots, G_k\}), \\ &\text{or } V = V_0(M, \{G_1, \dots, G_k\}). \end{aligned}$$

First suppose that  $V$  is a regular variety given by equations  $w_p(x) = w'_p(x)$  for  $p \in P$ . Then the above equivalence (5) on  $f^*$  is the smallest congruence on  $f^*$  for which

$$w_p \equiv w'_p \quad \text{for all } p \in P.$$

This implies that  $V = V_1(M, \{G_1, \dots, G_k\})$  (with  $M = f^*/\equiv$  and  $G_i = H(f_i)$ , of course). Next, if  $V$  is not regular, it is given by equations (1) plus some equations (2)  $w_p(x) = w'_p(y)$   $p \in \bar{P}$ . However, the monoid  $M(V) = f^*/\equiv$  is defined by (5) as in the previous case. We show that  $V = V_0(M, \{G_1, \dots, G_k\})$  with  $M$  and  $G_1, \dots, G_k$  as above.

Replace any equation

$$w_p(x) = w'_p(y)$$

in the defining system by the following two equations:

$$w_p(x) = w'_p(x) \quad \text{and} \quad w_p(x) = w_p(y).$$

The first can be added to the system (1) and this does not influence the definition of  $\equiv$  in (5), so we can suppose that the system (2) is formed by equations of the form  $w_p(x) = w_p(y)$ . For any such  $w_p$

$$w_p(x) = w_p(v(x)) \quad \text{for all } x \in X \text{ and all } v \in f^*,$$

so that  $w_p \equiv w_p \circ v$  for each  $v \in f^*$ . Consequently,  $H(w_p)$  is a left zero of the monoid  $M(V) = f^*/\equiv$ . Hence

$$V_0(M, \{G_1, \dots, G_k\}) \subset V.$$

Conversely, let  $A = (X, f_1, \dots, f_k)$  be an algebra in  $V$  and let  $H(w)$  be a left zero of  $M(V) = f^*/\equiv$ . Then  $w \equiv w \circ v$  for all  $v \in f^*$ ; hence  $w(x) = w(v(x))$  for all  $x \in X$ . Since at least one equation of the form

$$w_p(x) = w_p(y)$$

is valid in  $A$  (for all  $x, y \in X$ ), we have

$$w(x) = w(w_p(x)) = w(w_p(y)) = w(y)$$

so that  $w(x) = w(y)$  is valid in the algebra  $A$ , and hence,  $A \in V_0(M, \{G_1, \dots, G_k\})$ . Thus  $V = V_0(M, \{G_1, \dots, G_k\})$ . We conclude that any variety  $V$  in  $\text{ALG}(k)$  determines uniquely a monoid  $M = M(V)$  and a set  $\{G_1, \dots, G_k\}$  of generators of  $M$ . Moreover,

$$\begin{aligned} &\text{either } V = V_1(M, \{G_1, \dots, G_k\}) \quad (\text{if } V \text{ is regular}), \\ &\text{or } V = V_0(M, \{G_1, \dots, G_k\}). \end{aligned}$$

Let us mention explicitly that if  $M$  has no left zeros, then  $V_1(M, \{G_1, \dots, G_k\}) = V_0(M, \{G_1, \dots, G_k\})$ . If  $M$  has at least one left zero, the two varieties are distinct. In the latter case,  $V_0(M, \{G_1, \dots, G_k\})$  is precisely the class of all connected algebras in  $V_1(M, \{G_1, \dots, G_k\})$ . Let us recall that an algebra  $(X, f_1, \dots, f_k)$  is called *connected* if, for every  $x, y \in X$ , there are elements  $x = x_0, x_1, \dots, x_s = y$  in  $X$  and words  $w_j, v_j$  in  $f^*$ ,  $j = 1, \dots, s$ , such that

$$(6) \quad w_j(x_{j-1}) = v_j(x_j) \quad \text{for all } j = 1, \dots, s.$$

Indeed, if  $A = (X, f_1, \dots, f_k)$  is in  $V_0(M, \{G_1, \dots, G_k\})$  and  $H(w) \in M$  is a left zero of  $M$ , then

$$w(x) = w(y) \quad \text{in } A \text{ for all } x, y \in X,$$

and hence  $A$  is connected. Conversely, let  $A$  be a connected algebra in  $V_1(M, \{G_1, \dots, G_k\})$ . If  $H(w) \in M$  is a left zero, then  $w(x) = w(v(x))$  for all  $x \in X$  and all  $v \in f^*$ . Since  $A$  is connected, we can find  $x = x_0, \dots, x_s = y$  and  $w_j, v_j$ , which fulfill (6) for all  $x, y \in X$ . Then

$$\begin{aligned} w(x) &= w(w_1(x_0)) = w(v_1(x_1)) = \dots = w(w_s(x_{s-1})) \\ &= w(v_s(x_s)) = w(y), \end{aligned}$$

which implies  $A \in V_0(M, \{G_1, \dots, G_k\})$ .

**3. Isomorphism-complete varieties in  $\text{ALG}(k)$ .** Given a monoid  $M$ , its left ideal is a subset  $A \subset M$  such that

$$m \in M \quad \text{and} \quad a \in A \quad \text{implies} \quad m \circ a \in A.$$

Following [18], we call a monoid *linear* if and only if the set of its left ideals is linearly ordered by inclusion, i.e., if  $A$  and  $A'$  are left ideals of  $M$ , then

$$\text{either } A \subset A' \quad \text{or} \quad A' \subset A.$$

Otherwise,  $M$  is called *nonlinear*. Clearly,  $M$  is nonlinear if and only if there exist  $m_0, m_1 \in M$  such that both  $m_0 \notin M \circ m_1$  and  $m_1 \notin M \circ m_0$ . In fact, if such  $m_0, m_1$  exist, then  $A = M \circ m_0$  and  $A' = M \circ m_1$  are incompatible left ideals. Conversely, if  $A, A'$  are incompatible left ideals, then any  $m_0 \in A - A'$  and  $m_1 \in A' - A$  have the above property.

*Examples.* (a) Let  $V$  be the variety in  $\text{ALG}(2)$  given by the following equations:

$$f_1(x) = f_1 f_1(x), \quad f_2 f_1(x) = f_1 f_2(y), \quad f_2(x) = f_2 f_2(x).$$

Then  $M = M(V) = \{1, G_1, G_2, G_1 \circ G_2\}$  is the monoid with two generators  $G_1, G_2$  and the defining relations

$$G_1 = G_1 \circ G_1, \quad G_2 \circ G_1 = G_1 \circ G_2, \quad G_2 = G_2 \circ G_2.$$

The element  $G_1 \circ G_2$  is the unique left zero of  $M$ . This monoid is not linear because

$$G_1 \notin M \circ G_2 \quad \text{and} \quad G_2 \notin M \circ G_1.$$

(b) Let  $V$  be the variety in  $\text{ALG}(2)$  given by the following equations:

$$f_1(x) = f_1 f_1(x) = f_2 f_1(x), \quad f_2(x) = f_2 f_2(x) = f_1 f_2(x).$$

Then  $M = M(V) = \{1, G_1, G_2\}$  is the monoid with two generators  $G_1, G_2$  and the defining relations

$$G_1 = G_1 \circ G_1 = G_2 \circ G_1, \quad G_2 = G_2 \circ G_2 = G_1 \circ G_2.$$

This monoid has no left zero. It is also nonlinear because  $G_1 \notin M \circ G_2$  and  $G_2 \notin M \circ G_1$ .

(c) On the other hand, if  $V$  is the variety in  $\text{ALG}(2)$  given by the following equations:

$$f_1(x) = f_1 f_1(x) = f_2 f_1(y), \quad f_2(x) = f_2 f_2(x) = f_1 f_2(y),$$

then the following equations can be derived:

$$f_1(x) = f_2 f_1(x) = f_2 f_1(y) = f_1(y),$$

$$f_2(x) = f_1 f_2(x) = f_1 f_2(y) = f_2(y).$$

Hence, (with the choice  $y = f_2(x)$ )

$$f_1(x) = f_1 f_2(x) = f_2(x),$$

and  $M(V) = \{1, G_1\}$  (with  $G_1 \circ G_1 = G_1, G_2 = G_1$ ) is linear.

(d) The monoid  $M = M(V)$  of the variety  $V$  in  $\text{ALG}(2)$  given by the following equation:

$$f_1 f_2(x) = f_1(x),$$

though infinite, is also linear. Indeed,  $M = \{G_2^k \circ G_1^n \mid n, k = 0, 1, 2, \dots\}$ ; if  $m_0 = G_2^{k_0} \circ G_1^{n_0}, m_1 = G_2^{k_1} \circ G_1^{n_1}$ , then  $G_2^{k_0} \circ G_1^{n_0-n_1} \circ m_1 = m_0$  for  $n_0 > n_1$  and  $G_2^{k_0-k_1} \circ m_1 = m_0$  for  $k_0 > k_1, n_0 = n_1$ .

The following theorem is proved implicitly in [18].

**SICHLER'S THEOREM.** *Let  $V$  be an arbitrary variety in  $\text{ALG}(k)$ . If its monoid  $M(V)$  is nonlinear and finite, then  $V$  is isomorphism-complete.*

Let us present an outline of the proof. Let  $G = (X, E)$  be an undirected finite graph, i.e., the vertex set  $X$  is finite and  $E$  is a set of two-element subsets of  $X$ . Call a graph  $(X, E)$ -complete if  $E$  is the set of all two-element subsets of  $X$ . The degree  $\text{deg}(v)$  of a vertex  $v$  of  $(X, E)$  is the number of edges (i.e., elements of  $E$ ) containing  $v$ . Let  $\text{GRA}$  denote the class of all finite undirected incomplete connected graphs with all vertices of degree greater than two. It is easy to see that the isomorphism problem in  $\text{GRA}$  is isomorphism-complete. The proof of the isomorphism completeness of any variety  $V \subset \text{ALG}(k)$  with a finite nonlinear monoid  $M(V)$  proceeds as follows: we construct, for each  $G \in \text{GRA}$ , an algebra  $\phi(G) \in V$  such that

- (1) the size of  $\phi(G)$  is bounded by a polynomial in the size of  $G$ ,
- (2) graphs  $G, G' \in \text{GRA}$  are isomorphic if and only if  $\phi(G)$  and  $\phi(G')$  are isomorphic.

We sketch the construction presented in [18]. First, a graph  $G = (X, E) \in \text{GRA}$  is represented as  $(R_G, p_0, p_1)$ , where  $R_G \subset X \times E$  consists of all pairs  $(v, e)$  with  $v \in e$ ; for  $r = (v, e) \in R_G$ ,

$$p_0(r) = v \quad \text{and} \quad p_1(r) = e.$$

It is easy to verify that if  $G, G' \in \text{GRA}$  are represented as  $(R_G, p_0, p_1)$  and  $(R_{G'}, p'_0, p'_1)$ , then each isomorphism  $\varphi: G \rightarrow G'$  determines a bijection  $f: R_G \rightarrow R_{G'}$  such that, for  $i = 0, 1$ ,

$$(7) \quad p_i(r) = p_i(s) \Rightarrow p'_i(f(r)) = p'_i(f(s))$$

and vice versa: each bijection  $f: R_G \rightarrow R_{G'}$ , which fulfills (7) determines an isomorphism  $\varphi: G \rightarrow G'$ . (Given  $\varphi, f$  is defined by  $f((v, \{v, v'\})) = (\varphi(v), \{\varphi(v), \varphi(v')\})$ .)

Now, let  $V$  be a variety in  $\text{ALG}(k)$  with nonlinear  $M(V) = M$ . There are  $m_0, m_1 \in M$  such that  $m_0 \notin M \circ m_1$  and  $m_1 \notin M \circ m_0$ . The idea of the construction of the algebra  $\phi(G) \in V$  is to use the operations  $m_0, m_1$  in the role of the above maps  $p_0$  and  $p_1$  on  $R_G$ . Hence, given a graph  $G = (X, E) \in \text{GRA}$ , let  $F(R_G)$  be the algebra generated

freely in  $V$  by the set  $R_G$  and let  $\theta_G$  denote the least congruence on  $F(R_G)$  containing all the pairs:

- (i)  $(m_0(r), m_0(s))$  for  $r, s \in R_G$  with  $p_0(r) = p_0(s)$ ;
- (ii)  $(m_1(r), m_1(s))$  for  $r, s \in R_G$  with  $p_1(r) = p_1(s)$ ; and, for technical reasons, also pairs
- (iii)  $(r, b(r))$  for  $r \in R_G, b \in B$ , where  $B \subset M$  is the set of all invertible elements  $b$  with  $m_i \circ b, m_i \circ b^{-1} \in M \circ m_i, i = 0, 1$ .

Then we put

$$\phi(G) = F(R_G) / \theta_G.$$

Let us mention explicitly that if  $M$  is finite, the size of  $F(R_G)$  (hence of  $\phi(G)$ ) is bounded by a polynomial depending on the size of  $G$ . Indeed, for a regular variety  $V$ , the elements of  $F(R_G)$  are in one-to-one correspondence with the elements of  $M \times R$  (under the bijection  $(m, r) \rightarrow m(r)$ ) and if  $V = V_0(M, \{G_1, \dots, G_k\})$ , we identify every  $m(r)$  with every  $m(s), r, s \in R_G$ , whenever  $m$  is a left zero of  $M$ .

The algebras  $\phi(G) \in V$  have the following property: for every  $G, G' \in \text{GRA}, G$  and  $G'$  are isomorphic if and only if  $\phi(G)$  and  $\phi(G')$  are isomorphic.

The proof of the implication

$$G \cong G' \Rightarrow \phi(G) \cong \phi(G')$$

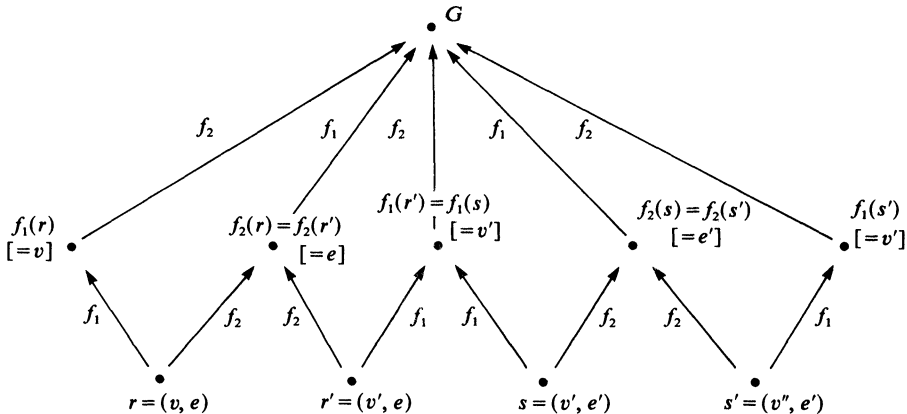
is quite easy. If  $G$  and  $G'$  are isomorphic, then there is a bijection  $f: R_G \rightarrow R_{G'}$ , which fulfills (7). This implies that its free extension  $F(f): F(R_G) \rightarrow F(R_{G'})$  preserves the congruences  $\theta_G, \theta_{G'}$  and the same is true for  $F(f^{-1}): F(R_{G'}) \rightarrow F(R_G)$ , so that  $F(f)$  determines an isomorphism of  $\phi(G)$  onto  $\phi(G')$ .

The proof of the converse, i.e.,

$$\phi(G) \cong \phi(G') \Rightarrow G \cong G'$$

is more delicate and we illustrate the idea in the above examples (a), (b) of varieties in  $\text{ALG}(2)$ .

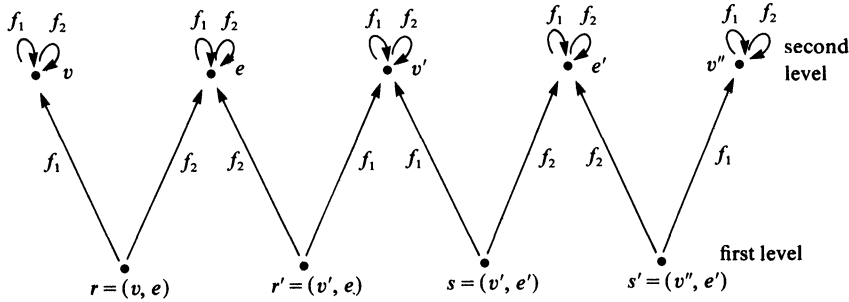
(a) Since  $M = \{1, G_1, G_2, G_1 \circ G_2\}$ , no element distinct from 1 is invertible so that the condition (iii) in the definition of  $\theta_G$  is trivial. Since  $G_1 \circ G_2$  is a left zero of  $M$ ,  $F(R_G) = \{r, f_1(r), f_2(r) \mid r \in R_G\} \cup \{\sigma_G\}$ , where  $\sigma_G = f_1 f_2(r) = f_2 f_1(r)$  for all  $r \in R_G$ . Hence, for each pair of edges with a common vertex in  $G$ , say  $e = \{v, v'\}, e' = \{v', v''\}$ , we have elements  $r = \{v, e\}, r' = \{v', e\}, s = \{v', e'\}, s' = \{v'', e'\}$  in  $R_G$ ; hence elements  $r, r', s, s', f_i(r), f_i(r'), f_i(s), f_i(s') (i = 1, 2)$  and  $\sigma_G$  in  $F(R_G)$ . Let us take  $f_1$  as  $m_0$  and  $f_2$  as  $m_1$ . Then the congruence  $\theta_G$  gives the following identifications:



If  $h: \phi(G) \rightarrow \phi(G')$  is an isomorphism, then necessarily  $h(\sigma_G) = \sigma_{G'}$  because  $\sigma_G$  is the unique point of  $\phi(G)$  with  $f_1(\sigma_G) = f_2(\sigma_G) = \sigma_G$  and similarly  $\sigma_{G'}$  in  $\phi(G')$ . Then  $h$

has to map any point  $x \in \phi(G)$  with  $f_2(x) = \sigma_G \neq x$  onto a point  $x' \in \phi(G')$  with  $f_2(x') = \sigma_{G'} \neq x'$ . This defines a bijection of the vertex set of  $G$  onto the vertex set of  $G'$ . It can be easily seen from the above diagram that this bijection is an isomorphism of  $G$  onto  $G'$ .

(b) Since  $M = \{1, G_1, G_2\}$ , the condition (iii) in the definition of  $\theta_G$  is trivial again;  $F(R_G)$  consists of  $\{r, f_1(r), f_2(r) \mid r \in R_G\}$ . Hence, for every pair of edges  $e = \{v, v'\}$ ,  $e' = \{v', v''\}$  with a common vertex we have  $r = \{v, e\}$ ,  $r' = \{v', e\}$ ,  $s = \{v', e'\}$ ,  $s' = \{v'', e'\}$  in  $R_G$  as before; and  $r, r', s, s', f_i(v), f_i(r'), f_i(s), f_i(s'), i = 1, 2$  in  $F(R_G)$ . The congruence  $\theta_G$  identifies them as follows:



If  $h : \phi(G) \rightarrow \phi(G')$  is an isomorphism, it maps any  $x \in \phi(G)$  with  $f_1(x) = f_2(x) = x$  to a point  $x' \in \phi(G')$  with the analogous property; hence  $h$  preserves the “levels.” Since  $h$  has also to preserve the following property:

$$\text{There is } y \neq x \text{ with } f_1(y) = x,$$

it defines a bijection of the vertex set of  $G$  onto the vertex set of  $G'$ . This bijection is an isomorphism of  $G$  onto  $G'$  because the following property:

$$\text{There is } y \neq x \text{ with } f_2(y) = x$$

has also to be preserved by  $h$ .

If the monoid  $M$  is more complicated, the construction is less lucid, but the above two “levels” appear in  $\phi(G)$  always and their points cannot be glued together by  $\theta_G$ —this is caused by the nonlinearity of  $M$ . The assumptions about graphs in GRA make it possible “to recognize these two levels in  $\phi(G)$ ” in the general case. The general proof is rather technical and difficult, and the reader is kindly referred to [18]. An outline of the proof of Sichler’s Theorem has been presented to emphasize some properties of the construction (above all the finiteness), which were not explicitly stated in the original paper.

**4. Algorithms for isomorphism testing of unary algebras.** Before we describe our algorithm, we introduce some notation, and prove some lemmas.

Given an algebra  $A = (X, f_1, \dots, f_k)$ , we denote by  $n$  the number of elements of  $X$  and by  $M$  the smallest set of mappings of  $X$  into itself containing the identity mapping of  $X$ , and mappings  $f_1, \dots, f_k$ , which is closed under the composition. Given  $x \in X$ , define  $\text{tr}(x) = \{f(x) \mid f \in M\}$  and  $\text{TR}(x) = \text{tr}(x) \cup \{y \mid f(y) = x \text{ for some } f \in M\}$ .

Let us write  $x \rightarrow y$  if  $f(x) = y$  for some  $f \in M$ , and  $x \sim y$  if both  $x \rightarrow y$  and  $y \rightarrow x$ . Notice that  $\rightarrow$  is a reflexive and transitive relation, and  $\sim$  is an equivalence.

Denote by  $\bar{X} = X/\sim$  the set of all equivalence classes of  $\sim$ . Then  $x, y \in a \in \bar{X}$  implies  $\text{tr}(x) = \text{tr}(y)$  and  $\text{TR}(x) = \text{TR}(y)$ . Sometimes, we write  $\text{tr}(a)$  instead of  $\text{tr}(x)$ , and similarly for  $\text{TR}(a)$ . Given  $a, b \in \bar{X}$ , let us write  $a \Rightarrow b$  if  $x \rightarrow y$  for some  $x \in a, y \in b$ ,  $a \Rightarrow b$  if  $a \Rightarrow b, a \neq b$ , and  $a \Rightarrow c \Rightarrow b$  for no  $c$  different from  $a, b$ .

LEMMA.  $\Rightarrow$  is the smallest transitive and reflexive relation on  $\bar{X}$  containing  $\Rightarrow$ .



We say that an algebra  $A$  is *linear* if for every  $a \in \bar{X}$  there exists at most one  $b \in \bar{X}$  such that  $a \Rightarrow b$ . The linearity of unary algebras is related to the previously defined concept of linearity of monoids as follows.

LEMMA. *Let  $V$  be a variety in  $\text{ALG}(k)$ . If the monoid  $M(V)$  is linear, then any algebra  $A \in V$  is linear.*

*Proof.* Let  $A = (X, f_1, \dots, f_k)$  be an element of  $V$  and  $a, b_1, b_2$  be elements of  $\bar{X}$  such that  $a \Rightarrow b_i, i = 1, 2$ . There are  $x \in a, y_1 \in b_1, y_2 \in b_2$  such that  $x \rightarrow y_1, x \rightarrow y_2$  and therefore  $y_1 = w_1(x), y_2 = w_2(x)$  for some  $w_1, w_2 \in f^*$ . Let  $m_1, m_2$ , respectively be elements of  $M(V)$ , determined by  $w_1, w_2$ , respectively, and denote  $I_i = M(V) \circ m_i$  for  $i = 1, 2$ .  $I_1$  and  $I_2$  are left ideals of  $M(V)$  and therefore if  $M(V)$  is linear, then either  $I_1 \subset I_2$  or  $I_2 \subset I_1$ . Suppose, without loss of generality, that  $I_1 \subset I_2$ . Since  $m_1 \in I_1$ ,  $m_1$  is equal to  $m \circ m_2$  for some  $m \in V$  and therefore

$$y_1 = w_1(x) = w(w_2(x)) = w(y_2) \quad \text{for some } w \in f^*.$$

In this case we have  $y_2 \rightarrow y_1, a \Rightarrow b_2 \Rightarrow b_1$  and hence  $b_2 = b_1$ , because  $a \Rightarrow b_1$  and  $a \neq b_2$ .  $\square$

Our algorithm is based on the solution of the following two problems:

PROBLEM 1. Given an algebra  $A$ , decide whether  $A$  is linear.

PROBLEM 2. Given two linear algebras  $A, B$ , decide whether  $A$  and  $B$  are isomorphic.

If two algebras  $A, B$  are given, our algorithm first uses the procedure solving Problem 1 and applies it to both  $A$  and  $B$ . There are three possibilities:

(1) Neither  $A$  nor  $B$  is linear. In this case the algorithm reports that it is not able to solve the problem, and it stops.

(2) Just one of the algebras  $A, B$  is linear. In this case  $A$  and  $B$  are not isomorphic.

(3) Both  $A$  and  $B$  are linear, and the second procedure is applied to solve the isomorphic problem.

In the next section, we shall show that the algorithm has the properties announced in the Introduction. Now, let us turn to Problem 1. Given  $x, y \in X, a, b \in \bar{X}$ , we shall write  $x \succ y$  if  $y = f_i(x)$  for some  $i = 1, \dots, k$ , and  $a \succ b$  if  $a \neq b$  and  $x \succ y$  for some  $x \in a, y \in b$ .

LEMMA.  $\Rightarrow$  is the smallest reflexive and transitive relation containing  $\succ$ .

LEMMA. If  $a \Rightarrow b$ , then  $a \succ b$ .

*Proof.* Consider  $a, b \in \bar{X}$ . If  $a \Rightarrow b$ , then  $a \neq b$  and  $a \Rightarrow b$ . Since  $\Rightarrow$  is the reflexive and transitive closure of  $\succ$ , there is a sequence  $a = a_0 \succ a_1 \succ \dots \succ a_m = b$  such that  $m \geq 1$  and  $a_i \neq a_j$  for  $i \neq j$ .

If  $m > 1$  then  $a \Rightarrow a_1 \Rightarrow b, a \neq a_1 \neq b$ , which contradicts the definition of  $\Rightarrow$ .  $\square$

LEMMA.  $\succ$  is an acyclic relation.

*Proof.* If  $a_0 \succ \dots \succ a_m \succ a_0$ , then there are  $x_i, y_i \in a_i$  such that  $x_{i-1} \rightarrow y_i$  for  $i = 1, \dots, m$  and  $x_m \rightarrow y_0$ . Since  $x_i \sim y_i$  for all  $i, x_0 \rightarrow y_1 \rightarrow x_1 \rightarrow \dots \rightarrow y_m \rightarrow x_m \rightarrow y_0 \rightarrow x_0$  and therefore  $x_0 \sim x_1 \sim \dots \sim x_m$ , which is a contradiction because it implies  $a_0 = \dots = a_m$ .  $\square$

LEMMA. *Let  $A$  be a linear algebra and suppose that the set  $\bar{X}$  is topologically ordered with respect to  $\succ$ , i.e., elements of  $\bar{X}$  are denoted by  $a_1, \dots, a_m$  in such a way that  $a_i \succ a_j$  implies  $i < j$ . Then  $a \Rightarrow b$  if and only if*

$$b = a_I \quad \text{where } I = \min \{i \mid a \succ a_i\}.$$

*Proof.* Denote  $Y = \{i \mid a \succ a_i\}$ . The set  $Y$  is nonempty if and only if there exists  $c \in \bar{X}$  such that  $a \Rightarrow c$ , because  $a \Rightarrow c$  implies  $a \succ c$ , and the relations  $\Rightarrow$  and  $\succ$  have the same reflexive and transitive closure (equal to  $\Rightarrow$ ). Since  $A$  is linear, if there is  $b$

such that  $a \Rightarrow b$ , then  $b$  is unique. Hence, it is sufficient to prove that if  $Y$  is nonempty and  $I = \min(Y)$ , then  $a \Rightarrow a_I$ .

Since  $a \succcurlyeq a_I$ , we have  $a \Rightarrow a_I$  and  $a \neq a_I$ . If there is  $c$  such that  $a \Rightarrow c \Rightarrow a_I$  and  $a \neq c \neq a_I$ , then there are  $j_1, \dots, j_p$  such that  $a = a_{j_0} \succcurlyeq a_{j_1} \succcurlyeq \dots \succcurlyeq a_{j_p} = c$ , which implies  $j_1 \leq j_p < I$  and we obtain a contradiction with the definition of  $I$ .  $\square$

We are now ready to formulate the following algorithm.

ALGORITHM FOR PROBLEM 1.

1. Determine the relation  $\succcurlyeq$ .
2. Determine the equivalence classes of  $\sim$  by finding strong components of the relation  $\succcurlyeq$ .
3. Determine the relation  $\succcurlyeq$ .
4. Find a topological ordering of  $X$  with respect to the relation  $\succcurlyeq$  (the ordering exists because the relation is acyclic).
5. For any  $a \in \bar{X}$ , find the minimum of the set  $\{b \mid a \succcurlyeq b\}$  with respect to the topological ordering constructed in 4, and denote it by  $m_a$  ( $m_a$  is undefined if  $a \succcurlyeq b$  for no  $b \in \bar{X}$ ). Denote by  $\Rightarrow'$  the relation consisting of all arrows  $a \Rightarrow' m_a$ . (Note that if  $A$  is linear, then  $\Rightarrow'$  is equal to  $\Rightarrow$ .)
6. The relation  $\Rightarrow'$  is a forest whose connected components are trees directed to their roots. Using the depth-first search (see, e.g., [19]), determine whether the following is true:

for every edge  $a \Rightarrow b$ , the element  $b$  belongs to the path from  $a$  to the root of the component of the relation  $\Rightarrow'$  containing  $a$ .

7. Use the fact that the condition given in 6 is valid if and only if the investigated algebra is linear.

LEMMA. *The running time of the algorithm for Problem 1 is  $O(nk)$ .*

*Proof.* The number of edges of the relation  $\succcurlyeq$  is at most  $nk$ . The relation  $\succcurlyeq$  cannot have more edges than the relation  $\succcurlyeq$ . The possibility to perform the computation in the time  $O(nk)$  is obvious for parts 1, 3, 5 of the algorithm and follows from [19] or [11], respectively, for parts 2, 4, respectively. The relation  $\Rightarrow'$  has at most  $n$  edges and therefore time  $O(n)$  is sufficient to search it. At any moment of the search we know which vertices belong to the path from the current vertex to the root of the component, and therefore the testing of the condition given in 6 needs  $O(nk)$  time.  $\square$

As a by-product of the computation, we have constructed the relation  $\Rightarrow$ , provided  $A$  is linear. The relation  $\Rightarrow$  will be used in further computation.

Our solution of Problem 2 is based on a canonical encoding of the sets  $\text{tr}(x)$ . By a *canonical encoding* we mean a mapping  $C$  from the class of all pairs  $(A, x)$ , where  $A \in \text{ALG}(k)$  and  $x$  is an element of  $A$ , into the class of all finite sequences of natural numbers such that  $C(A, x) = C(B, y)$  if and only if there exists a bijection  $F: \text{tr}_A(x) \rightarrow \text{tr}_B(y)$  with the following property:

$$F(x) = y, \quad F(f_i(z)) = g_i(F(z)) \quad \text{for each } z \in \text{tr}_A(x) \text{ and } i = 1, \dots, k,$$

where  $f_1, \dots, g_1, \dots$ , respectively, denote the operation of  $A, B$ , respectively.

We shall deal with the following canonical encoding: Let  $\triangleleft$  be a linear ordering of  $f^*$  defined as follows:  $w_1 \triangleleft w_2$  if either the length of  $w_1$  is less than the length of  $w_2$ , or the length of  $w_1$  and  $w_2$  are equal and  $w_1$  is lexicographically smaller than  $w_2$  (we suppose that  $f_1 \triangleleft f_2 \triangleleft \dots \triangleleft f_k$ ).

In other words,  $1 \triangleleft f_1 \triangleleft \dots \triangleleft f_k \triangleleft f_1 f_2 \triangleleft \dots \triangleleft f_1 f_k \triangleleft f_2 f_1 \triangleleft \dots \triangleleft f_2 f_2 \triangleleft \dots \triangleleft f_k f_k \triangleleft f_1 f_1 f_1 \triangleleft \dots$ .

$C(A, x)$  is determined as follows.

*Canonical encoding of the sets  $\text{tr}(x)$ .*

- (1) Given  $y \in \text{tr}(x)$ , denote by  $w(y)$  the smallest element  $w$  of  $f^*$  with respect to  $\triangleleft$  such that  $y = w(x)$ .
- (2) Order elements of  $\text{tr}(x)$  as  $y_1, \dots, y_m$  in such a way that  $w(y_1) \triangleleft w(y_2) \triangleleft \dots \triangleleft w(y_m)$ .
- (3) The encoding  $C(A, x)$  is the unique sequence

$$c_{11}, \dots, c_{1,k}, \dots, c_{n,1}, \dots, c_{m,k}$$

such that  $f_j(y_i) = y_{c_{i,j}}$  for  $i = 1, \dots, m, j = 1, \dots, k$ .

It is easy to see that  $C$  is really a canonical encoding because the sequence  $C(A, x)$  is, in fact, a description of the restrictions of the operations  $f_1, \dots, f_k$  of the algebra  $A$  to the set  $\text{tr}(x)$ , rewritten with respect to the isomorphism-invariant ordering  $y_1, \dots, y_m$  of  $\text{tr}(x)$ . Moreover, this encoding can be easily computed because the ordering of  $\text{tr}(x)$  can be found by the next algorithm (where  $Q$  is a first-in-first-out queue and  $M$  is a subset of  $\text{tr}(x)$ ).

*begin*

$Q :=$  the queue containing  $x$ ;  $M := \{x\}$ ;  $i := 1$ ;  $y_1 := x$ ;

*while*  $Q$  is nonempty *do*

*begin* remove the first element of  $Q$  from  $Q$  and denote it by  $z$ ;

*for*  $j := 1$  *to*  $k$  *do* *if*  $f_j(z) \notin M$  *then*

*begin*  $i := i + 1$ ;  $y_i := f_j(z)$ ;  $M := M \cup \{f_j(z)\}$ ;

*insert*  $f_j(z)$  at the end of the queue  $Q$  *end*

*end*

*end*;

□

We conclude the following lemma.

LEMMA. *There exists a canonical encoding  $C$  such that  $C(A, x)$  is a sequence of at most  $nk$  natural numbers not greater than  $n$  and  $C(A, x)$  can be computed in time  $O(nk)$ , where  $n$  ( $k$ , respectively) is the number of elements (operations, respectively) of the algebra  $A$ .*

ALGORITHM FOR PROBLEM 2. We can suppose without loss of generality that the given algebras are connected. The first part of the algorithm consists of an application of the above encoding procedure to all elements of both algebras, sorting the elements lexicographically according the values  $C(A, x)$  or  $C(B, x)$ , and partitioning them in such a way that two elements are in the same class if and only if their encodings are equal. The encoding of all elements can be clearly computed in  $O(n^2k)$  time. The same time bound applies to the sorting and the partitioning if  $nk$ -pass radix sort is used (see [12]).

As a result of the first part of the algorithm, we relabel all elements of both algebras in such a way that  $x$  and  $y$  obtain the same label if and only if their encodings are equal, i.e., there is an isomorphism of  $\text{tr}(x)$  onto  $\text{tr}(y)$  which maps  $x$  onto  $y$ . We denote the label of  $x$  by  $\lambda(x)$ .

We can suppose that during the test of linearity of algebras we choose  $x_{a,b} \in a$  and  $F_{a,b} \in \{f_1, \dots, f_k\}$  for any couple of elements  $a, b$  of  $\bar{X}$  such that  $a \Rightarrow b$  in such a way that  $F_{a,b}(x_{a,b}) \in b$ .

The second part of the algorithm is based on the recursive function  $\psi(x, a, b)$ , where  $x \in a \in \bar{X}$ ,  $b \in \bar{Y}$ , which returns the set of all  $y \in b$  such that there exists an isomorphism  $\phi: \text{Tr}(x) \rightarrow \text{TR}(y)$  for which  $\phi(x) = y$ . It is evident that the answer to Problem 2 can be obtained by calling  $\psi(x, a_0, b_0)$ , where  $a_0, b_0$ , respectively, are roots of the relation of the tested algebras, and  $x$  is an arbitrary element of  $a_0$ . (Since

we have supposed that  $A, B$  are connected linear algebras, the roots  $a_0, b_0$  are uniquely determined.)

The function  $\psi$  is given by the next algorithm:

0. function  $\psi(x, a, b)$ ;
1. begin
2. find all  $a_1, \dots, a_r \in \bar{X}$  such that  $a_i \Rightarrow a$  and denote  $x_{a_i, a}$  by  $x_i$ ;
3. find all  $b_1, \dots, b_s \in \bar{Y}$  such that  $b_i \Rightarrow b$ ;
4. if  $r \neq s$  then return the empty set else go to 5;
5.  $Z :=$  the set of all  $y \in b$  such that  $\lambda(x) = \lambda(y)$ ;
6. if  $r = 0$  or  $Z = \emptyset$  then return the set  $Z$ ;
7. for  $i := 1$  to  $r$  do for  $j := 1$  to  $r$  do  $Z_{i,j} := \psi(x_i, a_i, b_j)$ ;
8. for  $i := 1$  to  $r$  do
9.     begin find natural numbers  $j_0, \dots, j_p$  such that  $F_{a_i, a} = f_{j_0}$  and  $f_{j_p} \circ \dots \circ f_{j_1}(x_i) = x$ ;
10.     for all  $z \in b_1 \cup \dots \cup b_r$  do  $G_i(z) := g_{j_p} \circ \dots \circ g_{j_0}(z)$
11.     end;
12. for all  $y \in Z$  do
13.     begin
14.     for  $i := 1$  to  $r$  do for  $j := 1$  to  $r$  do  
             $W_{i,j} :=$  the set of all  $z \in Z_{i,j}$  such that  $G_i(z) = y$ ;
15.     if there exists no permutation  $\pi$  of  $\{1, \dots, r\}$  such that  
             $W_{m, \pi(m)} \neq \emptyset$  for every  $m = 1, \dots, r$   
            then delete  $y$  from  $Z$
16.     end
17. end;

First, we have to prove the correctness of  $\psi$ . We do it by induction on the length  $H$  of the longest sequence of the form  $c_0 \Rightarrow c_1 \Rightarrow \dots \Rightarrow c_H = a, c_0, \dots, c_H \in \bar{X}$ .

Let us suppose that  $y \in \psi(x, a, b)$ ; hence  $\lambda(x) = \lambda(y)$ . As we have already shown, there exists an isomorphism  $\phi_0: \text{tr}(x) \rightarrow \text{tr}(y)$  such that  $\phi_0(x) = y$ . If  $H = 0$ , then  $r = 0$ ,  $\text{tr}(x) = \text{TR}(x)$  and  $\text{tr}(y) = \text{TR}(y)$ , and the correctness of the algorithm is proved. Let us suppose that  $H > 0$ , i.e.,  $r > 0$ , and  $\psi$  is correct for  $H - 1$ . We know that there exists a permutation  $\pi$  of  $\{1, \dots, r\}$  and elements  $y_i \in \psi(x_i, a_i, b_{\pi(i)})$  such that  $G_i(y_i) = y$ . By the induction hypothesis, there exists isomorphisms  $\phi_i: \text{TR}(x_i) \rightarrow \text{TR}(y_i)$  such that  $\phi_i(x_i) = y_i$ . If  $j_0, \dots, j_p$  are numbers obtained in line 9 for a given value  $i$ , then we know that  $\phi_i(x) = \phi_i(f_{j_p} \circ \dots \circ f_{j_0}(x_i)) = g_{j_p} \circ \dots \circ g_{j_0}(\phi_i(x_i)) = G_i(y_i) = y = \phi_0(x)$  and therefore,  $\phi_i$  and  $\phi_0$  coincide on  $\text{tr}(x)$ .

Put

$$\phi(z) = \begin{cases} \phi_i(z) & \text{if } z \in \text{TR}(x_i), \\ \phi_0(z) & \text{if } z \in \text{tr}(x). \end{cases}$$

This formula gives an isomorphism of  $\text{TR}(x)$  onto  $\text{TR}(y)$  such that  $\phi(x) = y$ , provided  $\phi$  is defined correctly. Thus, it is sufficient to prove that

$$\begin{aligned} \text{tr}(x) \subset \text{TR}(x_i), \text{TR}(x) = \text{TR}(x_1) \cup \dots \cup \text{TR}(x_r) \quad \text{and} \\ \text{TR}(x_i) \cap \text{TR}(x_j) = \text{tr}(x) \quad \text{for } i \neq j. \end{aligned}$$

The proof follows immediately from the fact that, given  $c \in \bar{X}$ ,  $\text{tr}(c)$  is a union of  $c$  with all successors of  $c$  with respect to the relation  $\Rightarrow$ ,  $\text{TR}(c)$  is a union of  $c$  with all

successors and predecessors of  $c$  with respect to  $\Rightarrow$ , and  $a_1, \dots, a_r$  form the set of all immediate predecessors of  $a$  with respect to  $\Rightarrow$ .

Conversely, let us suppose that there exists an isomorphism  $\phi : \text{TR}(a) \rightarrow \text{TR}(b)$  such that  $\phi(x) \in b$ . We want to prove that  $\phi(x)$  is an element of  $\psi(x, a, b)$ . In line 5,  $\phi(x)$  is inserted into  $Z$ , and therefore, if  $H = 0$ , then  $\phi(x) \in Z = \psi(x, a, b)$ . Suppose that  $H > 0$ . It is evident that, in this case,  $r = s > 0$  in lines 2, 3 and according to the induction hypothesis, we know that there exists a permutation  $\pi$  of  $\{1, \dots, r\}$  such that  $W_{m, \pi(m)} \neq \emptyset$  for every  $m = 1, \dots, r$  when lines 12–17 are performed for  $y = \phi(x)$ , namely, the permutation determined by  $\phi(x_m) \in b_{\pi(m)}$ . It follows that  $\phi(x)$  is not deleted from  $Z$  and is an element of the resulting  $\psi(x, a, b)$ .

Given  $x \in a \in X$ , let us denote the number of elements of the set  $\{f^{-1}(x) \mid f \in M\} = (\text{TR}(x) - \text{tr}(x)) \cup a$  by  $n_x$ . Our last task is to prove that there exists a constant  $c_0$  such that  $\psi(x, a, b)$  can be computed in at most  $c_0 n_x^3$  steps, provided  $\Rightarrow$  and  $\lambda$  are given. Again, we proceed by the induction of the number  $H$ .

It is easy to show that  $O(n_x)$  steps are sufficient to perform the lines 1–6. Thus, our proposition is true for  $H = 0$ . Let us denote the number of elements of  $a$  by  $n_0$  and use  $n_i$  instead of  $n_{x_i}$ . If our time bound is true for  $H - 1$ , then the line 7 requires at most

$$\begin{aligned} c_0 \sum_{i=1}^r \sum_{j=1}^r n_i^3 &\leq c_0 \sum_{i=1}^r n_i \sum_{j=1}^r n_j^2 \leq c_0 (n_1 + \dots + n_r) \cdot (n_1 + \dots + n_r)^2 \\ &\leq c_0 n_x^3 \end{aligned}$$

steps if we ignore terms of smaller order.

To perform line 9, it is sufficient to find a path from  $F_{a_i, a}(x_i)$  to  $x$  in the set  $a$  with respect to the relation  $\rightarrow$  defined in the first part of the section. Thus,  $O(n_0)$  steps are sufficient (see e.g., [1]) and  $p = O(n_0)$ . It follows that the total time spent on line is  $O(n_0 r) = O(n_0 (n_1 + \dots + n_r))$ .

An efficient search for a permutation  $\pi$  in line 15 can be based on the next lemma.

LEMMA. *If  $W_{u,w}, W_{v,w}, W_{v,z}$  are nonempty for some  $u, v, w, z$ , then the set  $W_{u,z}$  is also nonempty.*

*Proof.* If  $W_{u,w}$  is nonempty, then there exists an isomorphism  $\phi_{u,v} : \text{TR}(u) \rightarrow \text{TR}(w)$  such that  $G_u(\phi_{u,v}(x_u)) = y$ , which implies  $y = G_u \phi_{u,v}(x_u) = \phi_{u,w} f_{j_p} \dots f_{j_0}(x_u) = \phi_{u,w}(x)$ . Similarly, we can suppose that there are isomorphisms  $\phi_{v,w} : \text{TR}(v) \rightarrow \text{TR}(w)$  and  $\phi_{v,z} : \text{TR}(v) \rightarrow \text{TR}(z)$  such that  $\phi_{v,w}(x) = \phi_{v,z}(x) = y$ .

Now, the mapping  $\phi_{v,z} \phi_{v,w}^{-1} \phi_{u,w} : \text{TR}(u) \rightarrow \text{TR}(z)$  is also an isomorphism which maps  $x$  into  $y$ . Since  $\phi_{u,z}(x_u) \in Z_{u,z} = \psi(x_u, a_u, b_z)$  and  $G_i(\phi_{u,z}(x_u)) = y$ , it is easy to see that  $W_{u,z} \neq \emptyset$ .  $\square$

It follows from the above lemma that a permutation  $\pi$ , if it exists, can be found by putting

$$\pi(m) = \text{an arbitrary element } j \text{ of the set } \{1, \dots, r\} - \{\pi(1), \dots, \pi(m-1)\} \text{ such that } W_{m,j} \neq \emptyset,$$

for  $m = 1, \dots, r$ . It follows that  $O(r^2)$  steps are sufficient to decide the existence of  $\pi$ .

It is easy to see that lines 1–6 and 8–17 require at most  $c n_0 (n_1 + \dots + n_r)^2$  steps for some constant  $c$ . If  $c_0$  has been chosen so that  $c_0 \geq c$ , then  $\psi$  can be computed in at most

$$c n_0 (n_1 + \dots + n_r)^2 + c_0 (n_1 + \dots + n_r)^3 \leq c_0 (n_0 + \dots + n_r)^3 = c_0 n_x^3$$

steps.

Combining the computation of  $\lambda$  and  $\psi$ , we obtain the following theorem.

**THEOREM.** *The existence of an isomorphism of two linear unary algebras with  $n$  elements and  $k$  operations can be tested in time  $O(n^2k + n^3)$ .*

If we work in a fixed variety of algebras then  $k$  is fixed, and we obtain the  $O(n^3)$  time bound.

**5. Main results.** The next two theorems are the main results of the present paper.

**THEOREM 1.** *There exists an algorithm  $\mathcal{A}$  such that, given two unary algebras  $A$  and  $B$  with  $n$  elements and  $k$  operations, either  $\mathcal{A}$  fails after  $O(nk)$  steps, this occurs only if each variety  $V$  containing either  $A$  or  $B$  is isomorphism-complete, or  $\mathcal{A}$  decides the existence of an isomorphism of  $A$  and  $B$  after  $O(n^2(n+k))$  steps.*

**THEOREM 2.** *Let  $V$  be a variety of unary algebras. If  $M(V)$  has a finite nonlinear factor-monoid then the isomorphism problem in  $V$  is isomorphism-complete, else we can solve the isomorphism problem in  $V$  in a polynomially bounded time (hence the condition  $C$ , mentioned in the abstract, is that every finite factor-monoid of  $M(V)$  is linear).*

*Proof of Theorems.* Let  $A = (X, f_1, \dots, f_k)$  be a unary algebra and let  $M_A$  be the monoid of maps  $X \rightarrow X$  generated by the operations  $f_1, \dots, f_k$ . If  $A$  is not linear (in the sense of § 4), then  $M_A$  is also not linear (in the sense of § 3). Now, we show that the algorithm described in § 4 satisfies Theorem 1. Indeed, given two unary algebras  $A$  and  $B$  with  $n$  elements and  $k$  operations, it decides after  $O(nk)$  steps whether they are linear or not. If both  $A$  and  $B$  are nonlinear, then both varieties  $V_A$  and  $V_B$  in  $\text{ALG}(k)$ , generated by  $A$  and by  $B$  have nonlinear finite monoids  $M(V_A)$  and  $M(V_B)$ , because  $M(V_A)$  is isomorphic to  $M_A$ , and analogously for  $B$  (see [6]); hence  $V_A$  and  $V_B$  are isomorphism-complete in view of the Sichler Theorem. Any variety with  $A \in V$  contains  $V_A$ , and hence, it is also isomorphism-complete, and analogously for  $B$ . Thus, if  $V$  is a variety such that either  $A \in V$  or  $B \in V$ , then  $V$  is isomorphism-complete.

If  $M(V)$  has a finite nonlinear factor-monoid  $M'$ , then  $V(M') \subset V$  and  $V(M')$  is isomorphism complete, by the Sichler Theorem. It follows that  $V$  is also isomorphism complete. If  $M(V)$  has no finite nonlinear factor-monoid, then any finite algebra  $A \in V$  must be linear, because the monoid  $M_A$ , being a finite factor-monoid of  $M(V)$ , is linear. Thus, the algorithm described in § 4 solves the isomorphism problem in  $V$  in the polynomially bounded time.  $\square$

**Acknowledgment.** We are indebted to J. Sichler for fruitful correspondence and for turning our attention to the nonlinear monoids and to paper [18].

#### REFERENCES

- [1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [2] L. BABAI, W. M. KANTOR, AND E. M. LUKS, *Computational complexity and the classification of finite simple groups*, Proc. 24th Annual IEEE Symposium on Foundations of Computer Science, 1983, pp. 162-171.
- [3] L. BABAI, D. YU GRIGORYEV, AND D. M. MOUNT, *Isomorphism of graph with bounded eigenvalue multiplicity*, Proc. 14th ACM Symposium on Theory of Computing, 1982.
- [4] K. S. BOOTH AND C. J. COLBOURN, *Problems polynomially equivalent to graph isomorphism*, Tech. Report CS-77-04, University of Waterloo, Waterloo, Ontario, Canada, June 1979.
- [5] S. A. COOK, *The complexity of theorem proving procedures*, Proc. 3rd Annual ACM Symposium on Theory of Computing, 1971, pp. 151-158.
- [6] G. GRÄTZER, *Universal Algebra*, Princeton University Press, Princeton, NJ; 1968, second edition, D. Van Nostrand, Springer-Verlag, Berlin, New York, Heidelberg, 1979.
- [7] Z. HEDRLÍN AND A. PULTR, *On full embeddings of categories of algebras*, Illinois J. Math., 10 (1966), pp. 392-405.
- [8] P. HELL, *Full embeddings into some categories of graphs*, Algebra Universalis, 2 (1972), pp. 129-141.

- [9] J. E. HOPCROFT AND J. WONG, *Linear time algorithm for isomorphism of planar graphs*, Proc. 6th Annual ACM Symposium on Theory of Computing, 1974, pp. 172–184.
- [10] R. M. KARP, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972.
- [11] D. E. KNUTH, *The Art of Computer Programming, Vol. 1, Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.
- [12] ———, *The Art of Computer Programming, Vol. 3, Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
- [13] L. KUČERA AND V. TRNKOVÁ, *Isomorphism completeness for some algebraic structures*, in Fundamentals of Computation Theory, F. Gécseg, ed., Lecture Notes in Computer Science 117, Springer-Verlag, Berlin, New York, Heidelberg, 1981.
- [14] E. M. LUKS, *Isomorphism of graphs of bounded valence can be tested in polynomial time*, Proc. 21st Annual IEEE Symposium on Foundations of Computer Science, 1980, pp. 42–49.
- [15] G. L. MILLER, *On the  $n^{\log n}$  isomorphism technique*, Proc. 10th Annual ACM Symposium on Theory of Computing, 1978, pp. 51–58.
- [16] ———, *Isomorphism testing for graph of bounded genus*, Proc. 12th Annual ACM Symposium on Theory of Computing, 1980, pp. 225–235.
- [17] R. C. READ AND D. G. CORNEIL, *The graph isomorphism disease*, J. Graph Theory, 1 (1977), pp. 339–363.
- [18] J. SICHLER, *Group-universal unary varieties*, Algebra Universalis, 11 (1980), pp. 12–21.
- [19] R. E. TARJAN, *Depth first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

## EFFICIENT PARALLEL EVALUATION OF STRAIGHT-LINE CODE AND ARITHMETIC CIRCUITS\*

GARY L. MILLER<sup>†</sup>, VIJAYA RAMACHANDRAN<sup>‡</sup>, AND ERICH KALTOFENS<sup>§</sup>

**Abstract.** A new parallel algorithm is given to evaluate a straight-line program. The algorithm evaluates a program over a commutative semi-ring  $R$  of degree  $d$  and size  $n$  in time  $O((\log n)(\log nd))$  using  $M(n)$  processors, where  $M(n)$  is the number of processors required for multiplying  $n \times n$  matrices over the semi-ring  $R$  in  $O(\log n)$  time.

**Key words.** parallel computation, straight-line code, arithmetic circuits

**AMS(MOS) subject classifications.** 68Q40, 68Q35, 68Q25

**1. Introduction.** In this paper, we consider the problem of dynamic evaluation of a straight-line program in parallel. This is a generalization of the result of Valiant, Skyum, Berkowitz, and Rackoff [VSBR]. They consider the problem of taking a straight-line program and transforming it into a program of "shallow" depth. Their transformation is performed by a sequential polynomial time algorithm. We show how to construct this "shallow" program with at most the same size and the same time bounds on-line, no preprocessing, as their off-line algorithm.

We consider two basically equivalent models of evaluation over a semi-ring: straight-line programs and arithmetic circuits. In the introduction we will restrict our discussion to the former model while most of the rest of the paper will deal with the latter model. A *straight-line program* over a commutative semi-ring  $R = (R, +, \times, 0, 1)$  is a sequence of assignment statements of the form  $a \leftarrow b + c$  or  $a \leftarrow b \times c$ , where  $b$  and  $c$  are either elements of  $R$  or previously assigned variables. We will assume that the semi-ring operations can be performed in unit time. Let  $M(n)$  denote the number of processors required to multiply two  $n \times n$  matrices in  $\log n$  time over the semi-ring  $R$  [AHU], [CWb].

A special case of a straight-line program is a Boolean circuit. Ladner has shown that the Boolean circuit evaluation problem is  $P$ -complete [Lad]. It is therefore believed that this evaluation problem is not in  $NC$  [Coo]. In this paper, we show that circuits of degree  $d$  and size  $n$  (we define the degree of a circuit in Definition 2.3) can be evaluated in time  $O(\log n(\log nd))$  using  $M(n)$  processors. The crucial difference between this result and the result in Valiant, Skyum, Berkowitz, and Rackoff [VSBR] is that our algorithm need not know the degree of the circuit in advance. As a nontrivial application of our procedure we can also compute the degree of a circuit in the above time and processor bounds. This follows because the operations of maximum and sum form a commutative semi-ring over the nonnegative integers. We know of no other

---

\* Received by the editors April 22, 1987; accepted for publication (in revised form) August 19, 1987. A preliminary version of this paper "Efficient Parallel Evaluation of Straight-Line Code," appeared in Lecture Notes in Computer Science, Vol. 227, pp. 236-245, 1986, Springer-Verlag.

<sup>†</sup> Mathematical Sciences Research Institute and Department of Computer Science, University of Southern California, Los Angeles, California 90089-0782. The research of this author was supported in part by National Science Foundation grant DCS-8514961.

<sup>‡</sup> Mathematical Sciences Research Institute and Coordinated Science Laboratory, University of Illinois, Urbana, Illinois 61801-3082. The research of this author was supported by National Science Foundation grant ECS-8404866, Semiconductor Research Corporation grant RSCH 84-06-049-6, and by an IBM Faculty Development Award.

<sup>§</sup> Mathematical Sciences Research Institute and Computer Science Department, Rensselaer Polytechnic Institute, Troy, New York 12181.



parallel algorithm for computing the degree that satisfies the above time and processor bounds.

**2. Preliminaries.** We view a straight-line program as a special case of a more general object, an arithmetic circuit. Our results are more easily applied to arithmetic circuits:

DEFINITION 2.1. An *arithmetic circuit* is an edge-weighted directed acyclic graph (DAG) (where the weights on the edges are from the semi-ring  $R$ ) satisfying the following conditions:

- Each node is labeled as one of three types: a leaf, a multiplication node, or an addition node.
- Leaves are assigned a value in  $R$ , denoted  $\text{value}(v)$  for a leaf  $v$ .
- The indegree of a leaf node is zero, a multiplication node is two, and an addition node is nonzero.
- All edges are directed away from leaves.
- There are no edges from multiplication nodes to multiplication nodes.

Note that any circuit can be modified to satisfy the last condition by simply adding a dummy addition node of indegree and outdegree 1 in the middle of each edge that connects two multiplication nodes. We say an edge is a *plus-plus* edge if it connects two addition nodes. The *size* of an arithmetic circuit  $U$  is the number of nodes in  $U$ . The *subcircuit evaluating  $v$* , denoted by  $U_v$ , is the subcircuit induced by all nodes that are contained on some path to  $v$ . A node  $w$  is a *child* of  $v$  if there exists an edge from  $w$  to  $v$ . A node of outdegree 0 is called an *output node*.

DEFINITION 2.2. We define the *value* of each node  $v$  in an arithmetic circuit  $U_v$  by induction on the size of  $U_v$ . The value for a leaf is given by the definition of an arithmetic circuit. If the node  $v$  is an additional node with children  $v_1, \dots, v_k$  then the value of  $v$  is defined by:

$$\text{value}(v) = \sum_{i=1}^k \text{value}(v_i) \cdot U(v_i, v),$$

where  $U(v_i, v)$  is the weight on the edge from  $v_i$  to  $v$ . If, on the hand,  $v$  is a multiplication node with children  $v_1$  and  $v_2$ , then

$$\text{value}(v) = \text{value}(v_1) \cdot \text{value}(v_2) \cdot U(v_1, v) \cdot U(v_2, v).$$

We will restrict our attention to circuits where any edge entering a multiplication node has weight 1. All the algorithms in this paper preserve this restriction. Thus, the value of the multiplication node  $v$  is  $\text{value}(v_1) \cdot \text{value}(v_2)$ . The *value* of a circuit is a vector of all its node values.

Given a straight-line program, we obtain its arithmetic circuit by constructing a node for each statement and for each input variable, and an edge from node  $i$  to node  $j$  if  $j$  is a statement that uses the variable evaluated at statement  $i$ . All edge weights are set to 1, and nodes corresponding to input variables are given values assigned to the corresponding variables.

DEFINITION 2.3. The (*algebraic*) *degree* of a node in an arithmetic circuit is defined inductively: a leaf has degree 1, an addition node has degree equal to the maximum degree of its children, and a multiplication node has degree equal to the sum of the degree of its children. The *degree* of an arithmetic circuit is the maximum over the degree of its nodes.

**3. The algorithm.** In this section, we describe our algorithm for arithmetic circuit evaluation. The value of the circuit will be obtained by repeated application of a procedure called *Phase*. This procedure takes as input an arithmetic circuit and returns

a new circuit with the same nodes such that every node will have the same value as before. Repeated application of *Phase* will eventually return with the value of the circuit.

In a natural way an arithmetic circuit can be viewed as an upper-triangular matrix  $U$  with zero diagonal, where the entry  $U_{ij}$  is the weight on the edge from node  $v_i$  to node  $v_j$  if the edge exists; it is zero otherwise. We need three submatrices derived from  $U$ :

$$U(+, +)_{ij} = \begin{cases} U_{ij} & \text{if } v_i \text{ and } v_j \text{ are addition nodes} \\ 0 & \text{otherwise,} \end{cases}$$

$$U(X, +)_{ij} = \begin{cases} U_{ij} & \text{if } v_j \text{ an addition node} \\ 0 & \text{otherwise,} \end{cases}$$

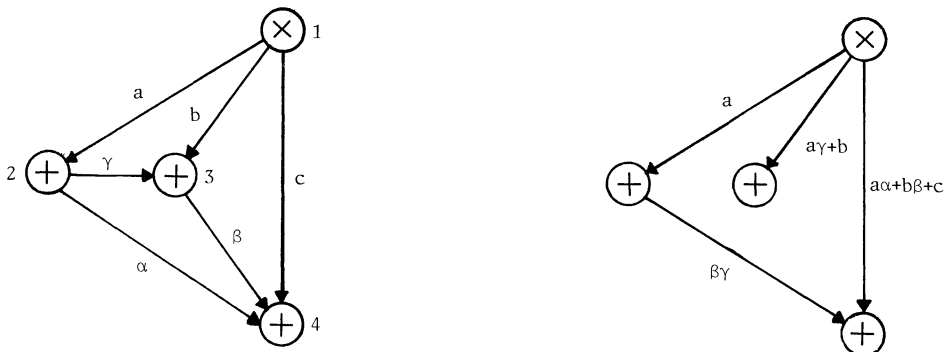
$$U(X, X)_{ij} = \begin{cases} U_{ij} & \text{if } v_i \text{ or } v_j \text{ is not an addition node} \\ 0 & \text{otherwise.} \end{cases}$$

The matrix  $U(+, +)$  corresponds to the subcircuit containing only plus-plus edges, while  $U(X, +)$  corresponds to the subcircuit containing any edge terminating at an addition mode. While the matrix  $U(X, X)$  corresponds to the subcircuit containing those only edges such that at least one end node is not an addition node. Thus,  $U(+, +) + U(X, X) = U$ . We can now define the procedure Matrix Multiply (MM). The procedure uses one matrix multiplication and one matrix addition over the semi-ring  $R$ . Thus, it can be performed in  $O(\log n)$  time using  $O(n^{2.49})$  processors for many semi-rings. In Fig. 1, we give an example of procedure MM.

**Procedure MM( $U$ )**

$$U \leftarrow U(X, +) \cdot U(+, +) + U(X, X)$$

We need two more procedures called Plus Evaluate ( $Eval_+$ , see Fig. 2), and Multiplication Evaluate or Shunt ( $Eval_\times$ , see Fig. 3). The first of these procedures simply evaluates an addition node if all its children have been evaluated. The first part of the second procedure evaluates a multiplication node if both its children have been evaluated. The new idea is the second part of the procedure which we call *Shunt*. Here we do partial evaluation of a multiplication node when only one of its arguments



$$\begin{pmatrix} 0 & a & a\gamma + b & a\alpha + b\beta + c \\ 0 & 0 & 0 & \beta\gamma \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & a & b & c \\ 0 & 0 & \gamma & \alpha \\ 0 & 0 & 0 & \beta \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \gamma & \alpha \\ 0 & 0 & 0 & \beta \\ 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & a & b & c \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

FIG. 1. An arithmetic circuit before and after an application of procedure MM.

has been evaluated. Figure 4 shows the effect of applying  $Eval_x$  to a circuit. Leaves are denoted by square boxes and nonleaves by circles. The value of each leaf is written in its box and the weight of an edge is written alongside it. The left circuit is before  $Eval_x$  and the right is after  $Eval_x$ . Zero weight edges have been removed.

The procedures  $Eval_+$ ,  $Eval_x$ , and MM can all be performed on a PRAM in  $O(\log n)$  time. The processor count for MM is the number of processors required for matrix multiplication for the particular semi-ring of the circuit. Procedures  $Eval_+$  and  $Eval_x$  need only  $O(n^2)$  processors. To see that  $Eval_x$  can be performed with  $O(n^2)$  processors, note that the number of terms  $F_{lji}$  in line (\*) is at most the number of edges. Thus, we simply sort these terms on their key  $(l, i)$  using say a randomized parallel bucket sort [Rei] or a deterministic comparison-based sorting algorithm [Col], [AKS] and then sum the terms using parallel list-ranking [MR], [Vis], [CV], [AM].

It is interesting to point out a strong analogy between the procedures Rake and Compress used to evaluate expression trees, see [MR], and our new procedures. One can view  $Eval_+$  and  $Eval_x$  as removing the leaves of an arithmetic circuit, i.e., Rake; while Matrix Multiplication, MM, “compresses” addition chains, a natural generalization of Compress [MR]. In fact, the  $Eval_x$  is a combination of a Rake and a Compress step since it removes leaves in the first part and does a partial compress in the second part.

Another analogy can be made between Top-Down algorithms and Bottom-Up ones. Brent gave a Top-Down parallel algorithm for expression evaluation [Bre], while Miller and Reif gave a Bottom-Up parallel algorithm for the problem [MR]. On the other hand, Valiant, Skyum, Berkowitz, and Rackoff gave a Top-Down parallel algorithm for arithmetic circuit evaluation [VSBR]; in this paper, we give a Bottom-Up parallel algorithm for this problem.

**Procedure  $Eval_+(U)$**

```

for all addition nodes  $v_j$  whose children are leaves do
  value  $(v_j) \leftarrow \sum_{i=1}^n \text{value}(v_i) \cdot U_{ij}$ 
  set  $v_j$  to a leaf
   $U_{ij} \leftarrow 0$  for  $i \in \{1, \dots, n\}$ 
od

```

FIG. 2. The procedure plus evaluation.

**Procedure  $Eval_x(U)$**

```

for all multiplication nodes  $v_j$  with children  $v_k$  and  $v_l$ , both of which are leaves,
do
  value  $(v_j) \leftarrow \text{value}(v_k) \cdot \text{value}(v_l)$ 
  Set  $v_j$  to a leaf
   $U_{kj} \leftarrow 0$  and  $U_{lj} \leftarrow 0$ 
od
for all  $U_{ji}$  where  $v_j$  is a multiplication node with children  $v_k$  and  $v_l$ 
  and  $v_k$  is a leaf and  $v_l$  is not do
   $F_{lji} \leftarrow \text{value}(v_k) \cdot U_{ji}$ 
od
for all pairs  $(l, i)$  do
   $W_{li} \leftarrow \sum_j F_{lji}$ 
   $U_{li} \leftarrow U_{li} + W_{li}$ 
   $U_{ji} \leftarrow 0$ 
od

```

(\*)

FIG. 3. The procedure multiplication evaluation or shunt.

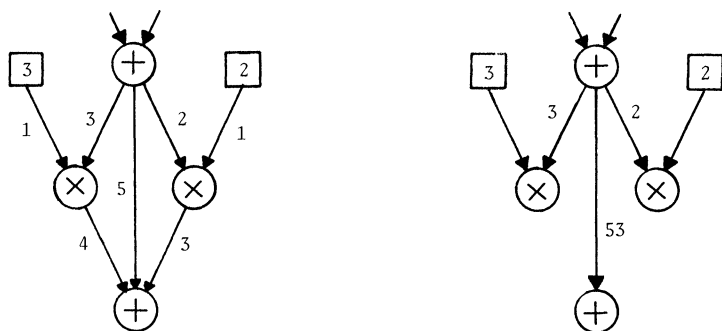


FIG. 4. An arithmetic circuit before and after an application of procedure  $Eval_x$ .

We combine these three procedures,  $MM$ ,  $Eval_+$ , and  $Eval_x$ , into a single procedure *Phase* that we will repeatedly apply until the value of the arithmetic circuit is returned:

```

Procedure Phase ( $U$ )
  do
     $U \leftarrow MM(U)$ 
     $U \leftarrow Eval_+(U)$ 
     $U \leftarrow Eval_x(U)$ 
  od
    
```

To show that *Phase* is correct (sound) it will suffice to prove the following lemma.

LEMMA 3.1. *The procedures  $MM$ ,  $Eval_+$ , and  $Eval_x$  applied to an arithmetic circuit return new circuits with the same value.*

The proof of the lemma follows, by a straightforward proof by induction on the size of  $U$ , using the associative, commutative, and distributive properties of  $R$ .

In Fig. 5, we show the effect of applying the different procedures to a circuit. We represent leaves by square boxes and addition or multiplication nodes by circles. All isolated nodes have been deleted and edge weights have been ignored. We start with the circuit (a) and apply procedure  $MM$  obtaining circuit (b), to which circuit (b) we apply procedure  $Eval_+$  obtaining circuit (c), to which we then apply  $Eval_x$  obtaining circuit (d).

**4. The height of an arithmetic circuit.** In this section, we define the height of a node. This notion is the main tool we shall use to analyse the procedure *Phase*. In Theorem 4.2, we will prove an upper bound on the height in terms of the size and the degree of a circuit. We will show in the next section that every application of *Phase* reduces the height of the circuit by a factor of approximately one half. The above two facts prove the main theorem of this paper.

DEFINITION 4.1. The *height* of a node is defined inductively:

- (1) A leaf has *height* 1.
- (2) A multiplication node has height equal to the sum of the heights of its children.
- (3) If  $v$  is an addition node then the height of  $v$  equals  $\max(a + 1/2, m)$ , where  $a$  equals the maximum height of any child of  $v$  which is an addition node, and  $m$  equals the maximum of the heights of the children which are either a leaf or a multiplication node.

The *height* of a circuit  $U$  is the maximum height of any node in  $U$ .

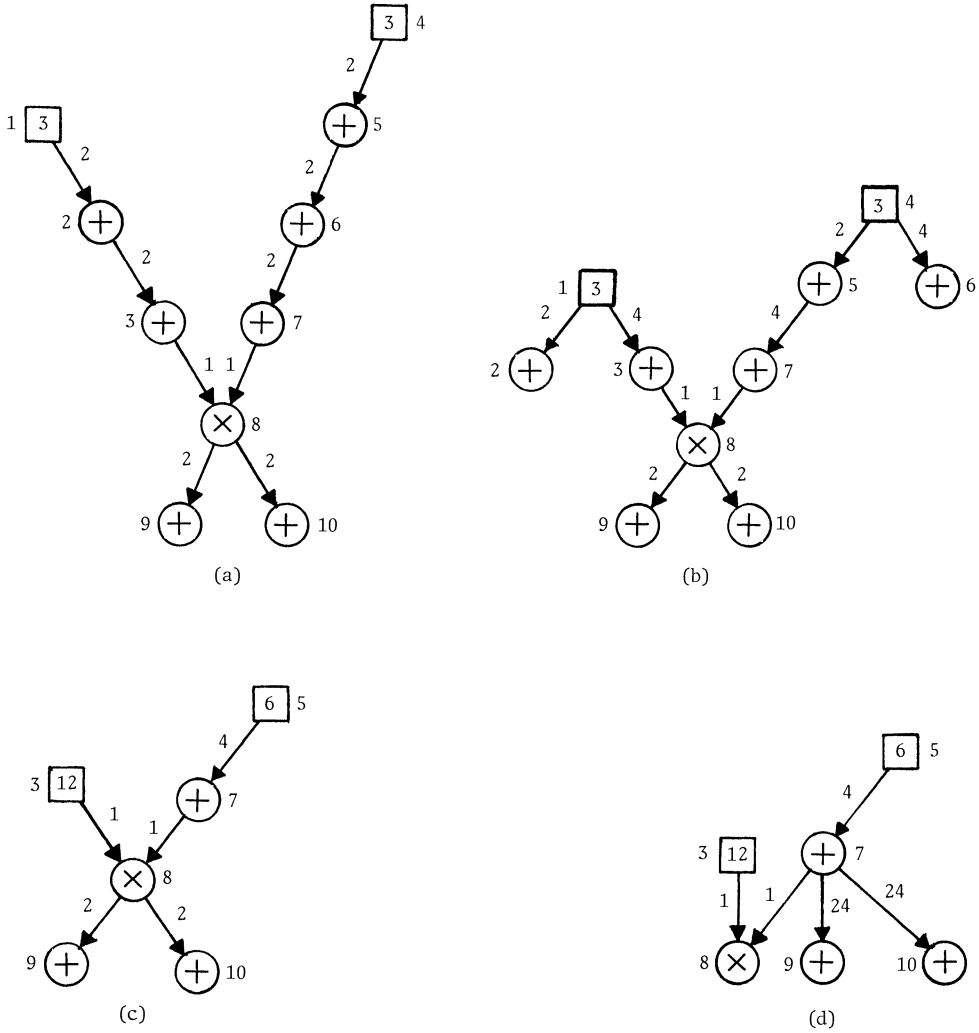


FIG. 5. An arithmetic circuit after successive application of the procedures: MM, Eval<sub>+</sub>, and Eval<sub>×</sub>.

We say a child  $w$  of an addition node  $v$  is *dominant* if either  $w$  is a multiplication node and  $h(v) = h(w)$  or it is an addition node and  $h(v) = h(w) + \frac{1}{2}$ , i.e., the height of  $w$  determines the height of  $v$ . We can now prove the upper bound on the height of a circuit.

**THEOREM 4.2.** *If  $U$  is an arithmetic circuit of degree  $d$  and  $e$  is the number of plus-plus edges, then the height of  $U \leq \frac{1}{2}e \cdot d + d$ .*

*Proof.* The proof is by induction on the number of nodes  $n$  in the subcircuit  $U_v$ . We start with subcircuits of size one, leaves. The height of a leaf is one which is clearly less than or equal to  $e + 1$ . Suppose the theorem is true for subcircuits of size  $\leq n$ . We show the theorem holds for circuits of size  $n + 1$ . Let  $U_v$  be a subcircuit with  $n + 1$  nodes. Let  $v_1, \dots, v_k$  be the children of  $v$  having degrees  $d_1, \dots, d_k$  and heights  $h_1, \dots, h_k$ , respectively. The subcircuits evaluating  $v_1, \dots, v_k$  are of size  $\leq n$ . Therefore, by induction  $h_i \leq \frac{1}{2}e'd_i + d_i$ , for  $1 \leq i \leq k$ , where  $e'$  is the number of plus-plus edges in

$U_v$ . There are two cases:  $v$  is either an addition node or a multiplication node. We treat the two cases separately.

First, suppose that  $v$  is a multiplication node. The degree  $d$  of  $v$  equals  $d_1 + \dots + d_k$  and the height, by induction, is  $\leq \sum_{i=1}^k \frac{1}{2}e'd_i + d_i$ , which is equal to  $\frac{1}{2}e'd + d$ . Thus, the theorem holds in this case, since  $e' \leq e$ . Second, suppose that  $v$  is an addition node. Again, there are two cases: either a dominant child is an addition node or it is a multiplication node. The most interesting case is the first case. Suppose that  $v_1$  is a dominant addition node, i.e.,  $h_1 \geq h_i, 1 \leq i \leq k$ . Here the degree  $d$  of  $v$  will be greater than or equal to  $d_1$ , while the height  $h = h_1 + \frac{1}{2} \leq \frac{1}{2}e'd_1 + d_1 + \frac{1}{2} \leq \frac{1}{2}e'd + d + \frac{1}{2}$ . Since we have at least one new plus-plus edge we know that  $e' \leq e - 1$ . Thus,  $h \leq \frac{1}{2}(e - 1)d + d + \frac{1}{2} = \frac{1}{2}ed - \frac{1}{2}d + d + \frac{1}{2}$ . Using the fact that  $d \geq 1$  we get the desired estimate,  $h \leq \frac{1}{2}ed + e$ .  $\square$

**5. Analysis of the algorithm.** In this section we use the height of a circuit to analyse the number of applications of *Phase* needed to evaluate a circuit of height  $h$ . We start by stating and proving the main technical lemma from which the main theorem will follow. Recall that all procedures defined so far take circuits to circuits. They modify the edge structure but map nodes to nodes in a one-to-one way. Thus, we may view the procedures as maps of circuits to circuits which are themselves surjective on nodes. Throughout this section let  $U$  be a circuit and  $U'$  its image under the transformation *Phase*. Similarly, if  $v$  is a node of  $U$  then its image under *Phase* will be denoted by  $v'$ .

LEMMA 5.1. *If  $U$  and  $U'$  are arithmetic circuits as above and  $v'$  is a node of  $U'$  which is not a leaf and not an output node, then the height of  $v$  is at least twice the height of  $v'$ .*

*Proof.* Let  $v'$  be a node of  $U'$  which is neither a leaf nor an output node. The proof will be by induction on the size of the subcircuit  $U'_v$ . We begin with the case when all the children of  $v'$  are leaves. There are two subcases: either  $v'$  is an addition node or it is a multiplication node. First, suppose that  $v'$  is an addition node. We must show that the height of  $v$  is at least 2, where  $v$  is the preimage of  $v'$ . Suppose by way of a contradiction that the height of  $v$  is  $< 2$ . Now,  $v$  cannot be of height 1 because a height 1 node must either be a leaf or all its children are leaves. Thus, one application of  $Eval_+$  will transform  $v$  into a leaf, a contradiction. If, on the other hand, the height is  $3/2$  then all the dominant children of  $v$  are addition nodes whose children are leaves. Thus, after MM and  $Eval_+$  the node  $v$  will be a leaf, and hence  $v'$  will be a leaf. This proves the case when  $v'$  is an addition node of height 1.

We next consider the more interesting case when  $v'$  is a multiplication node with both its children leaves. It will suffice to show that both children of  $v$  have height at least 2. Suppose that one child  $w$  has height less than 2. In this case, after MM and  $Eval_+$  the node  $w$  will be a leaf. Thus, after  $Eval_\times$  the vertex  $v$  will be either a leaf or an output node, depending on whether the other child of  $v$  is a leaf or not after  $Eval_+$ , a contradiction. This proves the initial cases of the induction.

The inductive case for multiplication nodes is rather straightforward. The only difficulty arises when one of the two children of  $v'$  is a leaf. We handle this by noting that in the last paragraph we actually proved something slightly stronger. Namely, if  $v'$  is a multiplication node which is not an output node and  $w'$  is a child of  $v'$  which is a leaf then the height of  $w$  is at least 2. Thus, induction for the multiplication nodes follows. We have only to prove the induction for addition nodes.

Suppose that  $v'$  is an addition node. Let  $w'$  be a dominant child of  $v'$ . If  $w'$  is a multiplication node the theorem follows easily. Thus, we may assume that  $w'$  is an addition node. It will suffice to prove the following claim.

CLAIM. The height of  $w$  is  $\leq$  the height of  $v$  minus 1, i.e.,  $h(w) \leq h(v) - 1$ .

*Proof of claim.* Note that both  $v$  and  $w$  are addition nodes. If there is a path in  $U$  from  $w$  to  $v$  containing two or more edges, then the claim follows by the definition of height. Thus, the only path from  $w$  to  $v$  is a singleton edge. But this is a contradiction, since procedure  $MM$  will then remove this edge and the procedures  $Eval_+$  and  $Eval_\times$  cannot replace it since there are now no paths from  $w$  to  $v$ . This proves the claim and the theorem.  $\square$

By Lemma 5.1, after  $\lceil \log_2 h \rceil$  applications of  $Phase$  to a circuit of height  $h$  the resulting circuit will contain only leaves and output nodes. Thus, in one more application of  $Phase$  (only  $Eval_+$  and  $Eval_\times$  are needed) all nodes will be leaves; the circuit has been evaluated. With a slightly more careful analysis the number of applications can be bounded by  $\lceil \log_2 h \rceil + 1$ . We state this fact as a theorem.

**THEOREM 5.2.** *If  $U$  is an arithmetic circuit with height  $h$ , then after  $\lceil \log_2 h \rceil + 1$  applications of  $Phase$ , all nodes of  $U$  are evaluated.*

The upper bounds given in Theorem 5.2 are optimal for our procedure  $Phase$ . In Fig. 6 we exhibit a circuit  $C_k$ , for  $k \geq 2$ , of height  $2^k - \frac{1}{2}$  which requires  $2^k$  applications of  $Phase$ . It is not hard to see that  $C_2$  requires 2 applications of  $Phase$ ; and the subcircuit evaluating  $v$  contained in  $Phase(C_{k+1})$  equals  $C_k$ , for  $k \geq 2$ .

We can now prove the main theorem of the paper.

**THEOREM 5.3.** *If  $U$  is an arithmetic circuit of degree  $d$  and size  $n$  then the value can be computed in parallel in time  $O((\log n)(\log nd))$  using at most  $M(n)$  processors.*

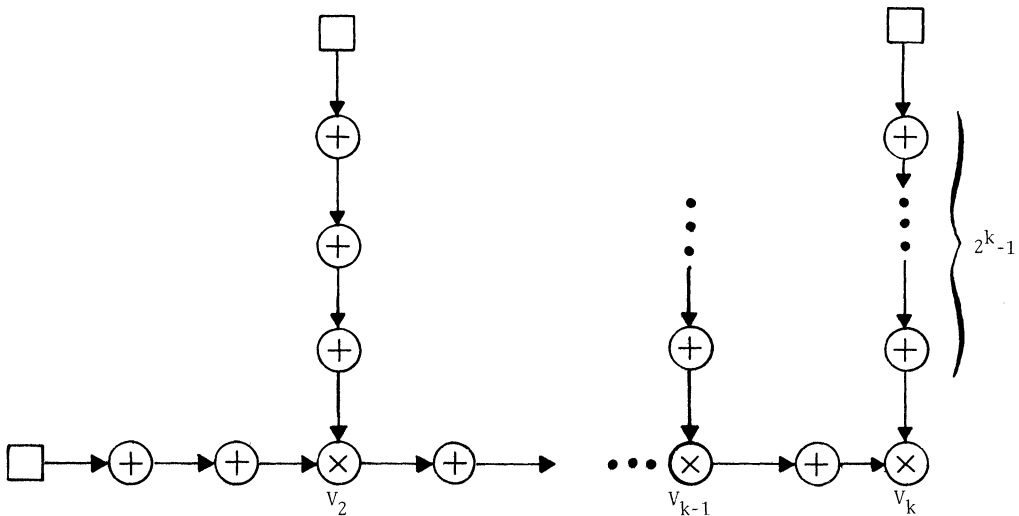


FIG. 6. The arithmetic circuit  $C_k$ ; a worst-case example for  $Phase$ .

*Proof.* By Theorem 5.2, procedure  $Phase$  need only be applied  $\lceil \log h \rceil + 1$  times, where  $h$  is the height of  $U$ . By Theorem 4.2,  $h = O(e \cdot d)$ . Thus,  $Phase$  is applied  $O(\log nd)$  times. Now, each application of  $Phase$  requires only  $\log n$  parallel time. The processor-expensive step is the matrix multiplication in  $MM$ , which can be performed using  $O(M(n))$  processors.  $\square$

We give a few simple corollaries to Theorem 5.3. We say a function  $g(n)$  is pseudopolynomial in  $n$  if  $g(n) = O(n^{\log^k n})$  for some constant  $k$ . That is  $\log(g(n)) = O((\log n)^{k+1})$ .

**COROLLARY 5.4.** *To determine if a straight-line program has pseudopolynomial degree is in  $NC$  for each constant  $k$ .*

COROLLARY 5.5. *The value of a straight-line program of pseudopolynomial degree can be computed in NC for each constant  $k$  where the input values are integers and operations are addition and multiplication.*

To see the last corollary we observe that the output of a straight-line program of pseudopolynomial degree has polynomial size in binary in terms of the size of the program.

**6. Open questions.** We know of no similar results for noncommutative rings. We note that for arithmetic circuits over the ring of  $n \times n$  matrices one can expand the matrix operations into the underlying commutative ring operations and apply the methods of this paper.

Extension of this work to rings with division would also be interesting.

Several new related results have occurred since the original writing of this paper. Matrix multiplication can now be performed using  $O(n^{2.376})$  processors, [CWa]. The ideas in this paper have been extended to more complex domains, [MT]. Finally, an analysis of the main theorem has been found that does not use the height metric, [May].

## REFERENCES

- [AHU] A. AHO, J. HOPCROFT, AND J. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [AKS] M. AJTAI, J. KOMLOS, AND E. SZEMEREDI, *An  $O(n \log n)$  sorting network*, Proc. 15th Annual Symposium on the Theory of Computing, ACM, Boston, April 1983, pp. 1-9.
- [AM] R. ANDERSON AND G. L. MILLER, *Optimal parallel algorithm for list ranking*, Proc. 16th Annual International Conference on Parallel Processing, submitted.
- [Bre] R. P. BRENT, *The parallel evaluation of general arithmetic expressions*, J. Assoc. Comput. Mach. 21 (1974), 201-208.
- [Col] R. COLE, *Parallel merge sort*, FOC27, IEEE, Toronto, October 1987, pp. 511-516.
- [Coo] S. A. COOK, *Towards a complexity theory of synchronous parallel computation*, L'Enseignement Mathématique XXVII (1981), pp. 99-124.
- [CV] R. COLE AND U. VISHKIN, *Deterministic coin tossing with applications to optimal list ranking*, Inform. and Control, 70 (1986), pp. 32-53.
- [CWa] D. COPPERSMITH AND S. WINOGRAD, *Matrix multiplication via arithmetic progressions*, Proc. 19th Annual ACM Symposium on Theory of Computing, ACM, New York, May 1987, pp. 1-6.
- [CWb] D. COPPERSMITH AND S. WINOGRAD, *On the asymptotic complexity of matrix multiplication*, SIAM J. Comput., 11 (1982), pp. 472-492.
- [Lad] R. E. LADNER, *The circuit value problem is log space complete for P*, SIGACT News, 7 (1975), pp. 18-20.
- [May] E. W. MAYR, *The Dynamic Tree Expression Problem*, Tech. Report STAN-CS-87-1156, Stanford University, Department of Computer Science, May 1987.
- [MR] G. L. MILLER AND J. H. REIF, *Parallel tree contraction and its applications*, Proc. 26th Symposium on Foundations of Computer Science, IEEE, Portland, OR, 1985, pp. 478-489.
- [MRK] G. L. MILLER, V. RAMACHANDRAN, AND E. KALTOFEN, *Efficient Parallel Evaluation of Straight-Line Code*, pp. 236-245, Lecture Notes in Computer Science, 227, Springer-Verlag, Berlin, New York, 1986.
- [MT] G. L. MILLER AND S.-H. TENG, *Dynamic parallel complexity of computational circuits*, Proc. 19th Annual ACM Symposium on Theory of Computing, ACM, New York, May 1987, pp. 254-264.
- [Rei] J. H. REIF, *An optimal parallel algorithm for integer sorting*, Proc. 26th Annual Symposium on Foundations of Computer Science, IEEE, Portland, OR, October 1985, pp. 496-504.
- [Vis] U. VISHKIN, *Randomized speed-ups in parallel computation*, Proc. 16th Annual ACM Symposium on Theory of Computing, ACM, Washington D.C., April 1984, pp. 230-239.
- [VS] L. G. VALIANT AND S. SKYUM, *Fast Parallel Computation of Polynomials Using Few Processors*, pp. 132-139, Lecture Notes in Computer Science, 118, Springer-Verlag, Berlin, New York, 1981.
- [VSBR] L. G. VALIANT, S. SKYUM, S. BERKOWITZ, AND C. RACKOFF, *Fast parallel computation of polynomials using few processors*, SIAM J. Comput., 12 (1983), 641-644.



## THE COMPLEXITY OF NEAR-OPTIMAL PROGRAMMABLE LOGIC ARRAY FOLDING\*

S. S. RAVI† AND ERROL L. LLOYD‡

**Abstract.** The problem of optimally folding a Programmable Logic Array (PLA) is known to be NP-complete. Motivated by the practical importance of this problem, we address the question of obtaining good, though not necessarily optimal, foldings. Two sets of results are presented. First, we show that three natural variants of the folding problem are equivalent with respect to approximation, in the sense that either they are all efficiently approximable or none of them is efficiently approximable. Next, we show for one of the variants (optimal bipartite folding) that if there is a polynomial time approximation algorithm (heuristic) which, for every PLA, produces a folding that is within a fixed factor of an optimal folding, then for any constant  $\epsilon > 0$ , there is a heuristic which, for every PLA, produces a folding that is within a factor of  $(1 + \epsilon)$  of the optimal folding. This result strongly suggests that the optimal folding problem is not efficiently approximable for arbitrary PLAs. In a companion paper, we have presented efficient heuristics for certain restricted classes of PLAs.

**Key words.** NP-complete, approximation, heuristics, PLA, folding, VLSI

**AMS(MOS) subject classification.** 68Q25

**1. Introduction.** Programmable Logic Arrays (PLAs) are used extensively in VLSI systems to implement combinational logic functions [MC80]. The regularity in the structure of PLAs facilitates the automatic generation and compaction of their layouts. A PLA consists of two matrices of circuit elements, called the AND and OR planes. The inputs to the PLA are combined in the AND plane to produce the necessary product terms. These product terms are then combined in the OR plane to produce the required outputs. For details regarding the implementation of a PLA, we refer the reader to [MC80]. For our purposes, a schematic diagram (Fig. 1) is adequate. In Fig. 1, the inputs and the outputs are along the columns and the product terms are along the rows. The **area** of a PLA is proportional to the product of the number of rows and columns. The logic function implemented by a PLA is determined by the **personalized** row-column intersections (these have been indicated by dark circles in Fig. 1). The other row-column intersections have no electrical significance.

In practice, it is often the case that only 10 percent of the intersections are personalized [EL84], [HNS82]. For such sparse PLAs, if the structure shown in Fig. 1 is implemented directly, a considerable amount of chip area will be wasted. To reduce the area, a technique called **folding** is commonly used [W79], [HNS82], [EL84]. This technique combines two electrical lines (rows or columns) and makes them share the same physical line. The two electrical lines are, however, disconnected by placing a *cut*. We obtain column folding or row folding depending upon whether we combine columns or rows. Due to the symmetry in the structure of a PLA, algorithms used to perform column folding can be directly used to perform row folding.<sup>1</sup> In view of this observation, we present our results in terms of column folding.

---

\* Received by the editors April 29, 1985; accepted for publication (in revised form) September 1, 1987. This research was supported in part by the National Science Foundation under grants MCS-8103713 and DCI-8603318.

† Department of Computer Science, State University of New York, Albany, New York 12222.

‡ Department of Computer Science, University of Pittsburgh, Pittsburgh, Pennsylvania 15260.

<sup>1</sup> It is possible to perform both row and column folding on the same PLA [HNS82].

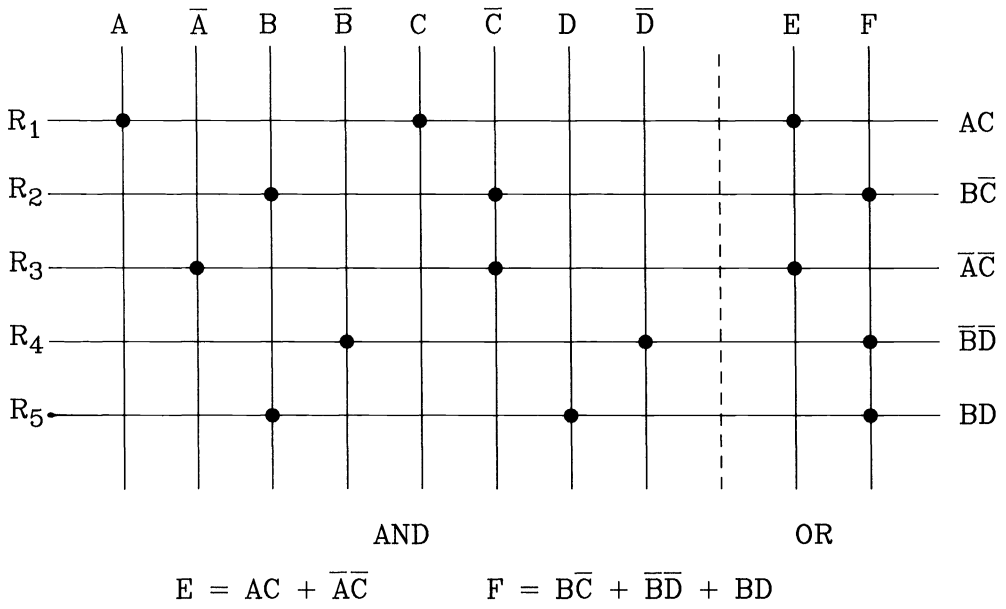


FIG. 1. Schematic diagram of a PLA.

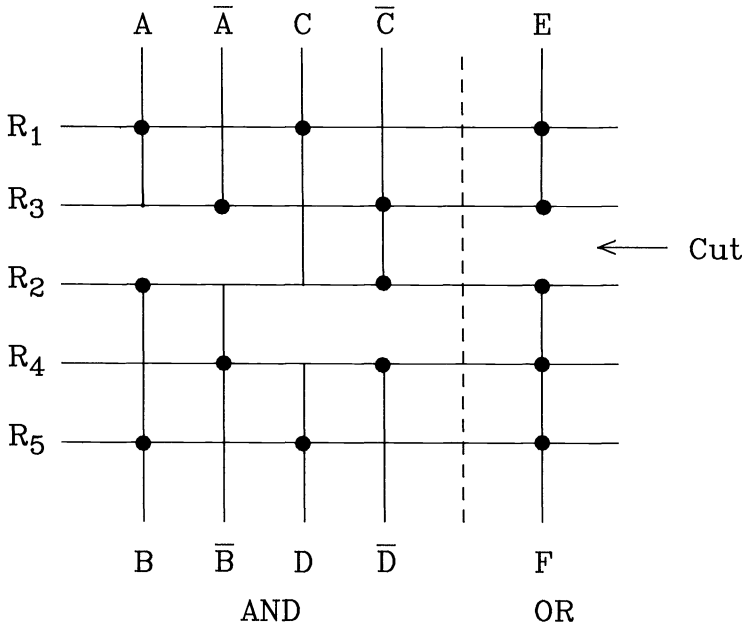


FIG. 2. A column-folded PLA.

A column-folded version of the PLA of Fig. 1 is shown in Fig. 2. Notice that when a pair of columns is folded, one of the columns is introduced from the top and the other from the bottom. By folding pairs of columns, the number of columns (and hence also the area) of a PLA is reduced. Thus, the larger the number of columns folded, the smaller the area of the PLA. However, two constraints restrict the number of column pairs that can be folded. First, a pair of columns can be folded only if they are not both personalized along the same row. Second, since folding permutes the rows (see Fig. 2), we can physically implement the folding of a given set of column pairs only if there is some permutation of the rows that allows all of the pairs in the set to be folded. For a detailed discussion of these constraints, the reader is referred to [HNS82] and [R84]. We say that a set of folding pairs is **implementable** if and only if it satisfies both of the above constraints. The **optimal folding problem** is to find a largest implementable set of folding pairs. This problem has been shown to be NP-complete in [LV82]. Only exponential time algorithms are currently known for NP-complete problems and it is widely believed that polynomial time algorithms do not exist for such problems [GJ79]. However, exponential time algorithms may not be suitable for the optimal folding problem since PLAs with a few hundred lines arise in practice [HNS82], [EL84]. This motivates the study of heuristics for the optimal folding problem. In this paper, we present results which suggest strongly that the optimal folding problem is not efficiently approximable for arbitrary PLAs. In a companion paper [RL84b], we have presented efficient heuristics for certain restricted classes of PLAs.

The remainder of this paper has been organized as follows. Section 2 provides a brief description of the graph theoretic model (due to Hachtel et al. [HNS82]) for the optimal folding problem. Section 3 discusses a restricted form of folding, namely **bipartite folding** (proposed by Egan and Liu [EL84]). In that section, it is shown that the optimal bipartite folding problem is just as hard to approximate as the unrestricted folding problem. It is also shown that if there is a polynomial time approximation algorithm for the optimal bipartite folding problem that approaches the optimal value to within *any* constant factor  $K$  for an arbitrary PLA, then for every  $\epsilon > 0$ , there is a heuristic which approaches the optimal value to within  $(1 + \epsilon)$ . Section 4 discusses a constrained form of bipartite folding. In that section, it is shown that the constrained bipartite folding problem is just as hard to approximate as the unrestricted folding and the bipartite folding problems. Section 5 offers conclusions and suggestions for further research.

**2. Graph theoretic model for folding.** The formulation of the PLA folding problem in graph theoretic terms is due to Hachtel, Newton, and Sangiovanni-Vincentelli [HNS82]. In this formulation, each PLA is represented by an undirected graph<sup>2</sup>  $G(V, E)$ , called its **intersection graph**. Each node in this graph corresponds to a column of the PLA. There is an edge between nodes  $i$  and  $j$  if columns  $i$  and  $j$  of the PLA are both personalized along the same row (i.e., columns  $i$  and  $j$  *cannot* be folded with each other). A pair of distinct columns  $p, q$  which can be folded with each other forms an **unordered folding pair**. An **ordered folding pair**  $(p, q)$  specifies that the columns  $p$  and  $q$  are to be folded in such a way that  $p$  is above  $q$ . A set  $S$  of ordered folding pairs which has the property that each line of the PLA appears in at most one element of each pair in  $S$ , forms an **ordered folding set**. Throughout this paper, we use the term “folding set” to mean an ordered folding set. Each ordered pair in  $S$  is represented

<sup>2</sup> We do not allow “self-loops” in undirected graphs.

in the intersection graph by a directed edge. If columns  $p$  and  $q$  have been folded with column  $p$  above column  $q$ , we add the directed edge from  $p$  to  $q$ . Thus, we obtain a mixed graph  $M(V, E, A)$ , where  $A$  is the set of directed edges. Since each column is folded at most once, notice that the edges in  $A$  form a *matching* (i.e., no two edges in  $A$  are incident on the same node).

The advantage of the mixed graph representation is that it is possible to provide a graph theoretic characterization of the implementability of a folding set [HNS82]. In order to state that result, we need to introduce the notion of an **alternating cycle** in a mixed graph. Informally, an alternating cycle is a cycle in which directed and undirected edges are traversed alternately. Formally, an alternating cycle  $C$  in the mixed graph  $M(V, E, A)$  is a sequence of edges  $\langle (v_1, v_2), \{v_2, v_3\}, (v_3, v_4), \dots, \{v_{2k-2}, v_{2k-1}\}, (v_{2k-1}, v_{2k}), \{v_{2k}, v_1\} \rangle$  such that each directed edge in  $C$  is in  $A$  and each undirected edge in  $C$  is in  $E$ . With this definition, we can state the theorem characterizing the implementability of a folding set.

**THEOREM 1.** *Given an intersection graph  $G(V, E)$  and an ordered folding set  $S$ , the set  $S$  is implementable if and only if the mixed graph  $M(V, E, A)$  obtained from  $G$  and  $S$  does not contain an alternating cycle.*

For a proof of this theorem, we refer the reader to [HNS82]. We observe that the above result makes it possible for us to study the PLA folding problem as a problem on mixed graphs.

As pointed out in § 1, the optimal folding problem has been shown to be NP-complete in [LV82]. In that paper, the authors have also shown that the problem remains NP-complete even when only one of the planes of the PLA is to be folded. In view of that result, we assume throughout this paper that there is only one plane to be folded. Thus, we use the words “PLA” and “one plane of a PLA” interchangeably. In this paper, our goal is to study the complexity of obtaining “near-optimal” folding sets. We now provide a formal definition of a near-optimal folding set. We assume that an instance of a folding problem is specified by an undirected graph  $G(V, E)$ , with  $|V| = n$ . The term **heuristic** is used to refer to any *polynomial time* algorithm that produces implementable folding sets. Let  $\text{OPT}(G)$  and  $H(G)$  denote, respectively, the number of folding pairs in an optimal folding set and in a folding set produced by a heuristic  $H$ . The **folding ratio** of the heuristic  $H$  is the maximum value of the ratio  $\text{OPT}(G)/H(G)$ , where the maximization is carried out over all intersection graphs with at most  $n$  nodes. Thus, the folding ratio of a heuristic is a worst-case measure of its performance. The smaller the folding ratio, the better the performance of the heuristic. In general, the folding ratio of a heuristic is a function of  $n$ . However, if the ratio grows with  $n$ , we can make the performance of the heuristic arbitrarily poor by choosing a sufficiently large  $n$ . Therefore, we will investigate whether there is a heuristic whose folding ratio is a **constant  $K$**  independent of  $n$ . Such a heuristic will be referred to as a **good heuristic**.

We should point out that there are numerous NP-complete problems for which constant worst-case ratios are obtainable in polynomial time. Examples of such problems include vertex cover, bin packing, and the traveling salesman problem with triangle inequality [GJ79]. On the other hand, for some problems such as the traveling salesman problem without triangle inequality [GJ79] and the augmented set basis problem [GM82], it has been shown that constant worst-case ratios cannot be obtained in polynomial time if the complexity classes **P** and **NP** are different.

Approximation algorithms for PLA folding have also been considered in [LV82]. They proved the following theorem which shows a relationship between the bandwidth minimization problem [GJ79] and the optimal folding problem.

**THEOREM 2.** *Let  $G$  be an intersection graph with  $n$  nodes. Then  $G$  has an ordered and implementable folding set of  $k$  pairs if and only if the bandwidth of  $G$  is strictly less than  $n - k$ .*

The authors of [LV82] point out that the above result makes it possible for us to obtain a heuristic for the optimal folding problem from a heuristic for the bandwidth problem. In fact, such a folding heuristic was implemented by Elaine Eschen (Master's Thesis, University of California, Berkeley) and was found to compare favorably with industrial codes.<sup>3</sup> From a worst-case standpoint, however, it is not clear from Theorem 2 whether a good heuristic for the bandwidth problem can be used directly to obtain good folding sets. Elaborating on this question, we assume for a moment that there is a heuristic  $B$  for the bandwidth problem, which is guaranteed to approach the optimal bandwidth to within a factor of (say) 2. Consider an undirected graph  $G$  with  $n$  nodes, whose bandwidth is  $n/2$  (assume also that  $n$  is even). By Theorem 2, the size of an optimal folding set for the corresponding PLA is as large as  $n/2 - 1$ . When we run the heuristic  $B$  on  $G$ , it is possible that  $B$  might report the bandwidth of  $G$  as  $n - 1$ . Thus according to Theorem 2, we may not be able to select any folding pair. Hence, the existence of a good heuristic for the bandwidth problem does not necessarily imply the existence of a good heuristic for the folding problem. Analogously, the results presented in this paper do not have any obvious impact on the approximability of the bandwidth problem.

In the sequel, we address the near-optimal folding problem and two of its variants. In this context, a problem is **efficiently approximable** if there is a good heuristic for it. Two problems are **equivalent with respect to approximation** if the existence of a good heuristic for one implies the existence of a good heuristic for the other and vice versa. This notion allows us to build a class of problems which has the property that either all of the problems in the class are efficiently approximable or none of them is efficiently approximable.

Throughout this paper we use undirected graphs to represent PLAs. Our choice is justified by the following observation.

**OBSERVATION 1.** Every undirected graph  $G(V, E)$  is the intersection graph of some PLA.

*Proof.* Let  $|V| = n$  and  $|E| = m$ . Assume without the loss of generality that  $V = \{1, 2, 3, \dots, n\}$  and that the nodes  $1, 2, 3, \dots, t$  of  $G$  are **isolated nodes**. (If  $G$  does not have isolated nodes, then  $t = 0$ .) We will construct a PLA  $P$  with  $m + t$  rows (numbered 1 through  $m + t$ ) and  $n$  columns (numbered 1 through  $n$ ) such that  $G$  is the intersection graph of  $P$ . The first  $t$  rows of the PLA are personalized to correspond to the isolated nodes. This is done by personalizing only the intersection of row  $i$  with column  $i$ , for  $i = 1, 2, \dots, t$ . Each of the remaining  $m$  rows of the PLA will represent one edge of  $G$ . We arbitrarily label the edges of  $G$  with the numbers 1 through  $m$ . If the edge numbered  $k$  is incident on nodes  $i$  and  $j$ , we personalize the intersections of columns  $i$  and  $j$  with row  $k$ . It is easy to verify that  $G$  is the intersection graph of the resulting PLA.  $\square$

In view of Theorem 1 and Observation 1, it is possible to study the complexity of the near-optimal folding problem in the graph theoretic domain. The subsequent sections of this paper describe how certain restricted forms of folding can be characterized in terms of properties of undirected graphs and how the near-optimal versions of all of these problems are related.

<sup>3</sup> We thank one of the referees for providing this information.

**3. Bipartite folding.** Thus far, we have discussed a general (**unrestricted**) form of folding. In this section, we examine a restricted form of folding, called **bipartite folding**. We prove that the bipartite folding problem can also be formulated as a graph problem. Moreover, the graph problem does not involve directed edges and so the condition for a bipartite folding set to be implementable is much simpler than the corresponding condition for an unrestricted folding set. Using this restricted form of folding, we prove results which strongly suggest that good heuristics do not exist for the optimal unrestricted and bipartite folding problems unless  $P = NP$ .

**3.1. Characteristics of bipartite folding.** Bipartite folding was introduced by Egan and Liu [EL84]. The unrestricted form of folding places the cuts (separating the two electrical lines) at various heights. In a bipartite folding, all of the cuts are required to be **at the same height**. Thus, the folding shown in Fig. 2 is *not* a bipartite folding. A bipartite-folded version of the PLA of Fig. 1 appears in Fig. 3. A bipartite folding set  $S$  is **implementable** if and only if there is some arrangement of the product terms that satisfies the constraints induced by each folding pair in  $S$  and allows all of the cuts to be placed at the same height. Bipartite folding derives its name from the fact that it partitions the set of product terms into only *two* sets, those above the cut level and those below.

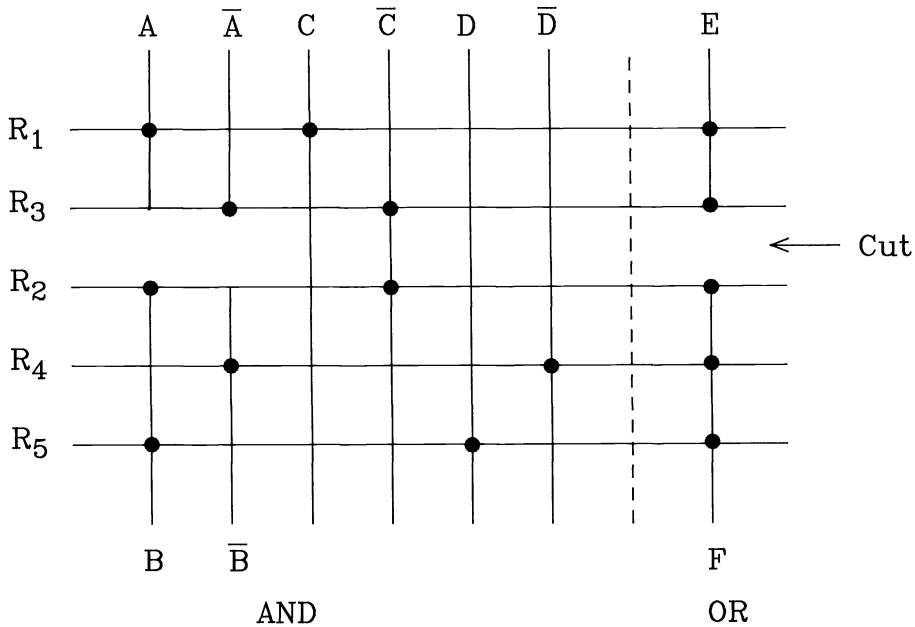


FIG. 3. A bipartite-folded PLA.

Bipartite folding is attractive in practice due to the ease with which a row folding can be carried out after a bipartite column folding. This row folding is easier because after a bipartite column folding, we can treat the two halves of the PLA as two separate PLAs and carry out row folding. Again, we can use the same algorithm to perform both row and column folding. Further, the following result from [EL84] points out that we can effectively bound the loss due to this restricted form of folding.

**THEOREM 3.** *Let  $OPT$  and  $OPTB$  denote, respectively, the sizes of an optimal unrestricted folding set and an optimal bipartite folding set for a PLA. Then  $OPTB \cong \lceil OPT/2 \rceil$ .*

Egan and Liu found experimentally that an optimal bipartite folding set is usually much larger than the lower bound given by the above theorem. They also showed that the optimal bipartite folding problem is NP-complete. Thus, the restriction to bipartite folding does not convert the NP-complete optimal folding problem into an easy (polynomial time solvable) problem. More importantly, it presents an interesting approximation problem, namely the problem of obtaining near-optimal bipartite folding sets. Theorem 3 indicates that the approximation problems for unrestricted folding and bipartite folding may actually be related to each other. We will examine the nature of this relationship in the subsequent sections. To begin, we show how the bipartite folding problem may be expressed as a graph problem.

**3.2. Graph theoretic characterization.** In § 2, we saw that the optimal folding problem may be phrased as a problem on mixed graphs. In this section, we show that the optimal bipartite folding problem can also be phrased as a graph theoretic problem. This formulation is useful in studying the problem of obtaining near-optimal bipartite folding sets.

**OBSERVATION 2.** Let  $G(V, E)$  be the intersection graph of a PLA and let  $S = \{(p_i, q_i) : 1 \leq i \leq r\}$  be an ordered and implementable folding set for  $G$ . Then,  $S$  is an implementable bipartite folding set if and only if for all  $i$  and  $j$ ,  $1 \leq i, j \leq r$ , the edge  $\{p_i, q_j\}$  is *not* in  $E$ .

*Sketch of proof.*

**IF.** Suppose that for all  $i, j$ ,  $1 \leq i, j \leq r$ , the edge  $\{p_i, q_j\}$  is not in  $E$ . It follows that the set  $T_P$  of rows personalized along any of the columns  $p_1, p_2, \dots, p_r$  and the set  $T_Q$  of rows personalized along any of the columns  $q_1, q_2, \dots, q_r$  must be *disjoint*. Therefore, we can obtain a bipartite folding by placing all of the rows in  $T_P$  (in any order) followed by all of the rows in  $T_Q$ . The cuts are made between the last row in  $T_P$  and the first row in  $T_Q$ . The lines  $p_1, p_2, \dots, p_r$  are run from the top and the lines  $q_1, q_2, \dots, q_r$  are run from the bottom.

**ONLY IF.** Suppose  $S$  is an implementable bipartite folding set, yet the edge  $\{p_i, q_j\}$  is in  $E$  for some  $i$  and  $j$ . Clearly  $i \neq j$ , because  $(p_i, q_i)$  is a folding pair. Since the edge  $\{p_i, q_j\}$  is in  $E$ , there is some row  $t_k$  which is personalized along both  $p_i$  and  $q_j$ . However, since  $p_i$  is run from the top and  $q_j$  is run from the bottom, it is easy to see that the position of  $t_k$  with respect to the cut level cannot be specified unambiguously. This contradicts the implementability of  $S$ .  $\square$

The above observation leads to the following definition of implementable bipartite folding sets.

**DEFINITION.** Let  $G(V, E)$  be the intersection graph of a PLA. Suppose  $P$  and  $Q$  are subsets of  $V$ . The **bipartite folding formed by  $P$  and  $Q$**  (denoted by  $[P, Q]$ ) is **implementable** if and only if the following conditions hold:

- (i) The sets  $P$  and  $Q$  are *disjoint*.
- (ii) For every node  $p \in P$  and  $q \in Q$ , the edge  $\{p, q\}$  is not in  $E$ .

The **size** of a bipartite folding  $[P, Q]$  is defined as

$$\text{SIZE}([P, Q]) = \text{Min} \{|P|, |Q|\},$$

where  $|P|$  and  $|Q|$  denote the cardinalities of sets  $P$  and  $Q$ , respectively. For convenience, we allow the cardinalities of the sets  $P$  and  $Q$  to be different. However, the size of an implementable bipartite folding set is the smaller of  $|P|$  and  $|Q|$ . Notice that every implementable bipartite folding set is (trivially) also an implementable unrestricted folding set.

The optimal bipartite folding problem requires that we find an implementable bipartite folding set of largest size. As indicated in § 3.1, this problem is NP-complete.

Thus, it is interesting to consider heuristics for this problem. The performance measure is again the folding ratio, except that we define the folding ratio with respect to the size of an optimal bipartite folding set. In particular, we are interested in examining whether there is a good heuristic for the bipartite folding problem.

In [EL84], Egan and Liu have discussed an exponential time (backtracking) algorithm that produces optimal bipartite folding sets. However, they report that for a PLA with about 100 lines, the folding program took about 23 minutes of CPU time. Such a program may not be desirable in an interactive chip design environment. For these applications, a fast heuristic which produces near-optimal bipartite folding sets would be more suitable. No results on heuristics for bipartite folding are currently available so it is not clear whether we need to resort to exponential time algorithms. Finally, in addition to its practical importance, the approximation problem for bipartite folding has an impact on the approximation problem for unrestricted folding as discussed in the next section.

**3.3. Near-optimal bipartite folding.** In this section, we prove that the optimal unrestricted folding problem and the optimal bipartite folding problem are equivalent with respect to approximation. This is accomplished by showing how a good heuristic for one problem may be converted into a good heuristic for the other problem. Before we can do this, however, we need to discuss an efficient method to convert an unrestricted folding set into a bipartite folding set without losing too many folding pairs.

**3.3.1. Obtaining bipartite folding sets from unrestricted folding sets.** Given an implementable unrestricted folding set  $S$  of size  $r$ , we can find an implementable bipartite folding set of size at least  $\lceil r/2 \rceil$  in the following manner:

(1) Find an implementation of the given folding set  $S$ . This step requires us to find an arrangement of the rows that satisfies all of the constraints induced by the folding pairs. This can be done in polynomial time as discussed in [HNS82].

(2) Arrange the folding pairs in  $S$  such that the heights of the cuts are in nonincreasing order. Note that this does not require any rearrangement of the rows.

(3) Form a bipartite folding set  $[P, Q]$  with the first  $\lceil r/2 \rceil$  upper lines as the set  $P$  and the last  $\lceil r/2 \rceil$  lower lines as the set  $Q$ .

The correctness of this procedure is obvious and from the above comments it is seen that the procedure can be implemented in polynomial time. Fig. 4 illustrates this conversion procedure. Fig. 4(a) shows a set of folding pairs arranged according to the heights of their cuts and Fig. 4(b) shows the resulting bipartite folding set.

A graph theoretic method to carry out the above conversion appears in [RL84a].

**3.3.2. Relationship between near-optimal unrestricted folding and near-optimal bipartite folding.** We are now ready to relate the near-optimal unrestricted folding and the near-optimal bipartite folding problems. In this section, we prove that these two problems are equivalent with respect to approximation. The next section examines the complexity of near-optimal bipartite folding.

**THEOREM 4.** *The optimal unrestricted folding problem and the optimal bipartite folding problem are equivalent with respect to approximation.*

*Proof.* The proof is in two parts. In Part 1, we assume that there is a good heuristic for the optimal bipartite folding problem and prove that there is a good heuristic for the unrestricted folding problem. In Part 2, we assume that the unrestricted folding problem has a good heuristic and show how we can obtain a good heuristic for the bipartite folding problem.

Part 1. Suppose that there is a good heuristic  $H_B$  for the optimal bipartite folding problem. We claim that  $H_B$  is also a good heuristic for the optimal unrestricted folding



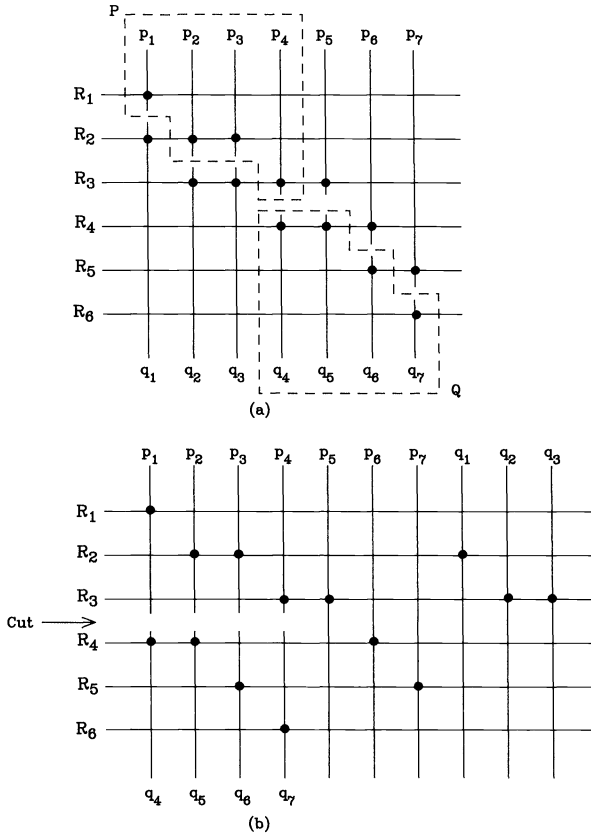


FIG. 4. Obtaining a bipartite folding set from an unrestricted folding set.

problem. To see this, let  $R$  denote the folding ratio of  $H_B$ . (Note that  $R$  is defined with respect to an optimal bipartite folding set.) Let  $G$  be an arbitrary intersection graph. We use  $OPT(G)$  and  $OPTB(G)$  to denote the sizes of an optimal unrestricted folding set and an optimal bipartite folding set for  $G$ , respectively. Suppose that  $H_B$  produces a bipartite folding set  $[P, Q]$  of size  $r$  for  $G$ . Since the folding ratio of  $H_B$  is  $R$ , we must have

$$(1) \quad \frac{OPTB(G)}{r} \leq R.$$

We can produce an “unrestricted” folding set of size  $r$  from  $[P, Q]$  by arbitrarily pairing up a node in  $P$  with a node in  $Q$ . Therefore, the folding ratio  $R'$  of  $H_B$  with respect to an optimal unrestricted folding set would be given by

$$R' = \frac{OPT(G)}{r}.$$

However, by Theorem 3,  $OPTB(G) \geq \lceil OPT(G)/2 \rceil$  or  $OPT(G) \leq 2 OPTB(G)$ . Hence,

$$(2) \quad R' \leq 2 \frac{OPTB(G)}{r}.$$

From inequalities (1) and (2) we see that  $R' \leq 2R$ . Hence, the folding ratio for  $H_B$ , when used for the unrestricted folding problem, is bounded by another constant, namely  $2R$ .

Part 2. Suppose  $H_U$  is a good heuristic for the unrestricted folding problem, with a folding ratio of  $R$ . Consider the following heuristic  $H_B$  for the bipartite folding problem:

(1) Run  $H_U$  on the given intersection graph  $G$ . Let  $S = \{(p_i, q_i) : 1 \leq i \leq r\}$  be the implementable folding set of size  $r$  produced by  $H_U$ .

(2) Produce a bipartite folding set  $[P, Q]$  of size at least  $\lceil r/2 \rceil$  from  $S$ , using the conversion procedure discussed in § 3.3.1.

Since conversion procedure runs in polynomial time,  $H_B$  is a polynomial time heuristic. By proceeding in a manner analogous to Part 1, it is easy to show that  $H_B$  has a folding ratio of at most  $2R$ . Thus,  $H_B$  is a good heuristic for the bipartite folding problem. This completes the proof of Part 2 and also that of Theorem 4.  $\square$

**3.3.3. On the complexity of near-optimal bipartite folding.** Theorem 4 shows that obtaining a good heuristic for the unrestricted folding problem is at least as hard as obtaining a good heuristic for the bipartite folding problem. We now prove an important result which provides substantial evidence for us to conclude that the bipartite folding problem (and hence also the unrestricted folding problem) is hard to approximate. The essence of this result is the following: if there is a heuristic  $H$  for the optimal bipartite folding problem with a folding ratio of  $K$  (a constant), however large  $K$  may be, we can find a heuristic  $H'$  whose folding ratio is arbitrarily close to 1. In this section, we present a proof of this result and then examine its significance. We begin with a lemma which forms the basis for the above result.

LEMMA 1. *Suppose  $H_B$  is a polynomial time heuristic for the optimal bipartite folding problem, with a constant folding ratio  $K > 1$ . Then there is a polynomial time heuristic  $H'_B$  for the optimal bipartite folding problem, for which the folding ratio is at most  $\sqrt{K}$ .*

*Proof.* Let  $G(V, E)$  be an arbitrary intersection graph. Given the heuristic  $H_B$ , the steps of heuristic  $H'_B$  are as follows:

(1) Transform the graph  $G(V, E)$  into another graph  $G'(V', E')$ . This transformation is such that if an optimal bipartite folding set for  $G$  is of size  $b$ , the size of an optimal bipartite folding set for  $G'$  is at least  $b^2$ .

(2) Execute  $H_B$  on  $G'$ . Let  $[P', Q']$  be the bipartite folding set produced, with  $\text{SIZE}([P', Q']) = r$ . Since  $H_B$  has a folding ratio of  $K$ , notice that  $r \geq b^2/K$ .

(3) Transform the bipartite folding set  $[P', Q']$  of size  $r$  for  $G'$  into a bipartite folding set  $[P, Q]$  for  $G$  with  $\text{SIZE}([P, Q]) \geq \lceil \sqrt{r} \rceil$ . The heuristic  $H'_B$  outputs  $[P, Q]$ . Suppose we assume, for a moment, that the transformations in steps (1) and (3) above can be carried out in polynomial time. Then  $H'_B$  would be a polynomial time heuristic. Further, the size of the folding set  $[P, Q]$  produced by  $H'_B$  satisfies the inequality

$$\text{SIZE}([P, Q]) \geq \left\lceil \left( \frac{b^2}{K} \right)^{1/2} \right\rceil \geq \frac{b}{\sqrt{K}}.$$

Thus, the folding ratio for  $H'_B$  is at most  $b/(b/\sqrt{K}) = \sqrt{K}$ . Therefore, we have only to prove that the transformations in steps (1) and (3) can be carried out in polynomial time to complete the proof of this lemma. Let us denote these transformations by T1 and T2, respectively. We now present the details of T1 and T2.

TRANSFORMATION T1. Let  $V = \{1, 2, 3, \dots, n\}$ . The graph  $G'(V', E')$  is obtained as follows:

$$V' = \{\langle i, j \rangle : 1 \leq i \leq n, 1 \leq j \leq n\},$$

$$E' = \{ \{ \langle i, j \rangle, \langle k, l \rangle \} : \{ i, j \} \cap \{ k, l \} \neq \emptyset \text{ or } \{ i, k \} \in E \text{ or } \{ i, l \} \in E \text{ or } \{ j, k \} \in E \text{ or } \{ j, l \} \in E \}.$$

Since  $G'$  has only  $n^2$  nodes, and hence only  $O(n^4)$  edges, it is clear that the above transformation can be carried out in polynomial time. We now prove that the graph  $G'$  has the required property.

CLAIM 1. Let  $[P, Q]$  be an implementable bipartite folding set of size  $b$  for  $G$ .<sup>4</sup> Suppose  $P = \{ p_1, p_2, \dots, p_b \}$  and  $Q = \{ q_1, q_2, \dots, q_b \}$ . Define sets  $P'$  and  $Q'$  by

$$P' = \{ \langle p_i, p_j \rangle : 1 \leq i \leq b, 1 \leq j \leq b \}, \quad Q' = \{ \langle q_i, q_j \rangle : 1 \leq i \leq b, 1 \leq j \leq b \}.$$

Then,  $[P', Q']$  is an implementable bipartite folding set of size  $b^2$  for  $G'$ .

*Proof.* Since  $P'$  and  $Q'$  contain  $b^2$  nodes each, we need only prove that  $[P', Q']$  is an implementable bipartite folding set for  $G'$ . We begin by noting that since  $P$  and  $Q$  are disjoint,  $P'$  and  $Q'$  are also disjoint. Thus if  $[P', Q']$  is not implementable, by Observation 2, we must have  $\langle p_u, p_v \rangle$  in  $P'$  and  $\langle q_x, q_y \rangle$  in  $Q'$  (for some  $u, v, x, y$ ) such that the edge  $e = \{ \langle p_u, p_v \rangle, \langle q_x, q_y \rangle \}$  is in  $E'$ . However, by our construction of  $G'$ , it follows that at least one of the edges  $\{ p_u, q_x \}$ ,  $\{ p_u, q_y \}$ ,  $\{ p_v, q_x \}$ , and  $\{ p_v, q_y \}$  must be in  $E$  and that contradicts the implementability of  $[P, Q]$ . Therefore,  $[P', Q']$  must be implementable. This completes the proof of Claim 1.

TRANSFORMATION T2. Transformation T2 produces a bipartite folding set of an appropriate size for  $G$ , starting from a bipartite folding set for  $G'$ . We present the details of transformation T2 in the proof of the following claim.

CLAIM 2. Let  $[P', Q']$  be an implementable bipartite folding set of size  $r$  for  $G'$ . Then, we can find, in polynomial time, an implementable bipartite folding set of size at least  $\lceil \sqrt{r} \rceil$  for  $G$ .

*Proof.* Suppose that  $P' = \{ \langle p_i, q_i \rangle : 1 \leq i \leq r \}$  and  $Q' = \{ \langle x_i, y_i \rangle : 1 \leq i \leq r \}$ . Define

$$P = \{ p_i : \langle p_i, q_i \rangle \in P' \text{ or } \langle q_i, p_i \rangle \in P' \text{ for some } i, 1 \leq i \leq r \},$$

$$Q = \{ x_i : \langle x_i, y_i \rangle \in Q' \text{ or } \langle y_i, x_i \rangle \in Q' \text{ for some } i, 1 \leq i \leq R \}.$$

It is easy to see that  $|P| \geq \lceil \sqrt{r} \rceil$  and  $|Q| \geq \lceil \sqrt{r} \rceil$ . Further, it is straightforward to verify that  $[P, Q]$  is an implementable bipartite folding set for  $G$ . This completes the proof of Claim 2 as well as that of Lemma 1.  $\square$

THEOREM 5. Suppose  $H$  is a polynomial time heuristic for the optimal bipartite folding problem, with a folding ratio of  $K$  (a constant). Then for every  $\varepsilon > 0$ , there is a polynomial time heuristic  $H_\varepsilon$  with a folding ratio of at most  $1 + \varepsilon$ .

*Proof.* The idea is to apply Lemma 1 an appropriate number of times. In particular, given an  $\varepsilon > 0$ , choose the smallest integer  $t$  which satisfies the inequality

$$(3) \quad 2^t \geq \frac{\log_2 K}{\log_2 (1 + \varepsilon)}.$$

Since  $K$  and  $\varepsilon$  are constants,  $t$  is also a constant. For convenience, we let  $x = 2^t$ . From inequality (3) it is easy to see that

$$(4) \quad K^{1/x} \leq (1 + \varepsilon).$$

---

<sup>4</sup> If either  $P$  or  $Q$  contains more than  $b$  nodes, we arbitrarily discard the extra nodes. Hence from now on, we will assume that unless otherwise specified, both of the sets in a bipartite folding contain the same number of nodes.

Let  $G(V, E)$  be an arbitrary intersection graph with an optimal bipartite folding set of size  $b$ . We transform  $G$  into a graph  $G'$  by successively applying transformation T1,  $t$  times. Since  $t$  is a constant and T1 is a polynomial time transformation, the construction of  $G'$  can be done in polynomial time. Let  $\text{OPTB}(G')$  denote the size of an optimal bipartite folding set for  $G'$ . Since every application of T1 at least squares the size of an optimal folding set (Claim 1 in Lemma 1), it follows that

$$(5) \quad \text{OPTB}(G') \geq b^{2^t} = b^x.$$

We use  $G'$  as the input to  $H$ . Since  $H$  has a folding ratio of  $K$ ,  $H$  must obtain a bipartite folding set of size at least  $b^x/K$ .

From this folding set, however, we can obtain a folding set for  $G$  by applying transformation T2, also  $t$  times. Clearly, this can also be done in polynomial time. From Claim 2 of Lemma 1, it is easy to see that the resulting bipartite folding set has a size  $s$  given by

$$(6) \quad s \geq \frac{b}{(K^{1/x})}.$$

We let  $H_\epsilon$  output this folding set. Hence the folding ratio of  $H_\epsilon$  is at most  $b/s \leq K^{1/x} \leq (1 + \epsilon)$  from inequality (4).

Therefore, assuming that  $H$  is given, we have found a heuristic  $H_\epsilon$  which has a folding ratio of at most  $(1 + \epsilon)$ . This completes the proof of Theorem 5.  $\square$

We believe that Theorem 5 is an important contribution of this paper. It suggests strongly that any search for a good heuristic for bipartite folding is unlikely to succeed. Theorem 5 in conjunction with Theorem 4 indicates that a similar search for the unrestricted folding problem is also likely to be futile. We now expand on these observations.

Since the optimal bipartite folding problem is not a number problem, the NP-completeness result of [EL84] implies that the problem is NP-complete in the *strong sense* [GJ79]. Further, since the size of an optimal bipartite folding set is at most  $n/2$ , it follows from a result of Garey and Johnson [GJ79, pp. 140-141] that there is no *fully* polynomial time approximation scheme (i.e., an approximation algorithm whose running time is a polynomial function of both  $n$  and  $1/\epsilon$ ) for this problem. This does not, however, rule out the possibility of a polynomial time approximation scheme (i.e., an approximation algorithm whose running time is a polynomial in  $n$ , but the exponent is a function of  $1/\epsilon$ ). Theorem 5 indicates that if there is a heuristic with a constant folding ratio  $K$ , irrespective of how large  $K$  is, we have a polynomial time approximation scheme for the bipartite folding problem. No such approximation scheme is currently known for any problem which is not a number problem. Worse yet, Theorem 5 also shows that if we can prove that approaching the optimal value to within a factor of  $1 + \epsilon$  is hard, however small  $\epsilon$  may be, achieving *any* constant factor is hard. A result similar to Theorem 5 is known for the maximum clique problem. In view of that result, it is widely believed that the clique problem does not have a good heuristic [GJ79]. The same appears to be true for the bipartite folding problem.

**3.4. Constrained bipartite folding.** What makes the bipartite folding problem difficult? It is tempting to conclude that the difficulty lies in deciding the position of a column with respect to the cut level (i.e., whether a column is to be placed above or below the cut level). We now consider a variation of the bipartite folding problem in which the position of each column is known a priori. We are simply required to

find the largest implementable bipartite folding set. We call this the **constrained bipartite folding problem**. Our motivation in studying this problem was to investigate whether this constrained version of the bipartite folding problem is easier than the unconstrained version. The results presented in this section provide a negative answer to this question.

The graph model for the constrained bipartite folding problem is the following: We are given a *bipartite intersection graph*  $G(V1, V2, E)$ , where  $V1$  corresponds to the set of columns to be placed on one side of (say, above) the cut and  $V2$  corresponds to the set of columns to be placed on the other side of the cut. Given such a graph, the optimum constrained bipartite folding problem requires us to find node sets  $P$  and  $Q$  such that

- (a)  $P \subseteq V1$  and  $Q \subseteq V2$ ,
- (b) for every  $p \in P$  and  $q \in Q$ , the edge  $\{p, q\}$  is not in  $E$ , and
- (c)  $\text{Min}\{|P|, |Q|\}$  is maximized.

The above problem is known as the **maximum balanced independent set problem**, and (as can be expected) is NP-complete [GJ79]. In view of this result, it is interesting to investigate whether there are good heuristics for this problem. However, our next theorem shows that the constrained bipartite folding problem is no easier to approximate than the (unconstrained) bipartite folding problem.

**THEOREM 6.** *The constrained bipartite folding problem and the (unconstrained) bipartite folding problem are equivalent with respect to approximation.*

*Proof.* The proof is again in two parts.

Part 1. Suppose there is a good heuristic  $H_C$  for the constrained bipartite folding problem.

Let  $R$  be the folding ratio for  $H_C$ . We now show that we can devise a heuristic  $H_B$  (which also has a folding ratio of  $R$ ) for the unconstrained bipartite folding problem. Let  $G(V, E)$  be an arbitrary intersection graph. The heuristic  $H_B$  consists of two steps:

(i) Transform the graph  $G(V, E)$  into a bipartite graph  $G'(V1, V2, E')$  as follows. Assume that  $V = \{1, 2, 3, \dots, n\}$ . Let  $V1 = \{v_{11}, v_{12}, \dots, v_{1n}\}$  and  $V2 = \{v_{21}, v_{22}, \dots, v_{2n}\}$ . For each  $i, 1 \leq i \leq n$ , we add the edge  $\{v_{1i}, v_{2i}\}$  to  $E'$ . Further, for every edge  $\{i, j\}$  in  $E$ , we add the edges  $\{v_{1i}, v_{2j}\}$  and  $\{v_{1j}, v_{2i}\}$  to  $E'$ .

(ii) Execute  $H_C$  on  $G'$ . From the folding set produced by  $H_C$ , construct a folding set for  $G$ , as described in the following claim.

**CLAIM 1.** Suppose  $[P', Q']$  is any implementable bipartite folding set for  $G'$ , with  $P' = \{v_{1i_1}, v_{1i_2}, \dots, v_{1i_m}\}$  and  $Q' = \{v_{2j_1}, v_{2j_2}, \dots, v_{2j_m}\}$ . Define  $P = \{i_1, i_2, \dots, i_m\}$  and  $Q = \{j_1, j_2, \dots, j_m\}$ . Then  $[P, Q]$  is an implementable bipartite folding set for  $G$ .

*Proof of Claim 1.* First, we note that the sets  $P$  and  $Q$  are disjoint. This is because if  $i_x = j_y$ , for some  $x$  and  $y$ , the construction would have added the edge  $\{v_{1i_x}, v_{2j_y}\}$  to  $E'$ . This would contradict the implementability of  $[P', Q']$ .

Also, if for some  $i_x$  in  $P$  and some  $j_y$  in  $Q$  the edge  $\{i_x, j_y\}$  is in  $E$ , then in the course of constructing  $G'$ , we would have added the edge  $\{v_{1i_x}, v_{2j_y}\}$  to  $E'$ , and this again contradicts the implementability of  $[P', Q']$ . This completes the proof of Claim 1.

Observe that the size of the folding set produced by  $H_B$  is the same as that produced by  $H_C$ . In order to prove that the folding ratio of  $H_B$  is the same as that of  $H_C$ , we need the following result.

**CLAIM 2.** The graph  $G'$  has an implementable bipartite folding set of size  $m$  if and only if  $G$  has an implementable bipartite folding set of size  $m$ .

*Proof of Claim 2.* The "only if" part is precisely Claim 1. So, we prove only the "if" part. Suppose  $[P, Q]$  is an implementable bipartite folding set of size  $m$  for  $G$ .

Let  $P = \{i_1, i_2, \dots, i_m\}$  and  $Q = \{j_1, j_2, \dots, j_m\}$ . Define,

$$P' = \{v_{1i_1}, v_{1i_2}, \dots, v_{1i_m}\}, Q' = \{v_{2j_1}, v_{2j_2}, \dots, v_{2j_m}\}.$$

Notice that  $P' \subseteq V1$  and  $Q' \subseteq V2$ . It is easy to verify that  $[P', Q']$  is an implementable bipartite folding set for  $G'$ . This completes the proof of Claim 2.

From Claim 2 it follows immediately that the sizes of the *optimal* bipartite folding sets for  $G$  and  $G'$  are equal. Claim 1 shows that the size of the bipartite folding set for  $G$  produced by  $H_B$  is exactly equal to the size of the bipartite folding set for  $G'$  produced by  $H_C$ . Hence the folding ratio for  $H_B$  is equal to that of  $H_C$ . Thus,  $H_B$  is also a good heuristic.

Part 2. Suppose there is a good heuristic  $H_B$  for the unconstrained bipartite folding problem. We now describe how to obtain a heuristic  $H_C$  for the constrained bipartite folding problem. Further,  $H_C$  will have the same folding ratio as  $H_B$ .

Given  $G(V1, V2, E)$ , we construct  $G'(V', E')$  as follows. Let  $V' = V1 \cup V2$ . The nodes in  $V1$  are connected together to form a clique, as are the nodes in  $V2$ . The edge set  $E'$  consists of all the edges in  $E$  and the clique edges just added. It is straightforward to verify (using Observation 2) that any nonempty implementable bipartite folding set  $[P, Q]$  for  $G'$  is such that  $P \subseteq V1$  and  $Q \subseteq V2$ . Since the set of edges between  $V1$  and  $V2$  is the same in  $G$  and  $G'$ , it follows that *any* implementable bipartite folding set for  $G'$  is also an implementable bipartite folding set for  $G$  and vice versa. In particular, *optimal* bipartite folding sets for  $G$  and  $G'$  are identical.

The heuristic  $H_C$  constructs  $G'$  and runs  $H_B$  on  $G'$ . Then  $H_C$  outputs the bipartite folding set produced by  $H_B$ . It is immediate that the folding ratios for  $H_C$  and  $H_B$  are equal. Hence  $H_C$  is a good heuristic for the constrained bipartite folding problem. This completes the proof of Theorem 6.  $\square$

The following corollary is a direct consequence of Theorems 4 and 6.

**COROLLARY 1.** *The unrestricted folding problem, the unconstrained bipartite folding problem and the constrained bipartite folding problem are all equivalent with respect to approximation.*

It also follows from the proof of Theorem 6 that a result identical to Theorem 5 holds for the constrained bipartite folding problem. That is, if there is a heuristic for the constrained bipartite folding problem with a constant folding ratio, then there is a heuristic with a folding ratio arbitrarily close to 1.

**4. Conclusions.** In this paper, we addressed the problem of obtaining near-optimal folding sets. With respect to this problem, we showed that three versions of the optimal folding problems are equivalent. We also presented results which strongly suggest that there are no good heuristics for the optimal folding problem.

The main problem left open by our work is to actually prove that the approximation problem is NP-hard. Theorem 5 suggests an approach towards solving this problem: For some  $\epsilon > 0$  (probably very small), show that the problem of obtaining a folding ratio of  $(1 + \epsilon)$  is NP-hard. A proof of this result might suggest a way to attack the similar question for the clique problem. It will also be interesting to examine whether the clique problem and the bipartite folding problem are equivalent with respect to approximation. (The currently known reduction from the clique problem to the constrained bipartite folding problem [J83] does not preserve approximations.) As far as we have been able to ascertain, the bipartite folding problem is only the second "bonafide" member (the first being the clique or the independent set problem) of the class of nonnumber problems for which a result similar to Theorem 5 has been proven. It will be interesting to investigate whether there are other problems in this class.

**Acknowledgment.** We thank the referees for providing several valuable suggestions. The construction used in the proof of Lemma 1, which is substantially simpler than our original construction, was provided by one of the referees. This referee also provided the conversion algorithm of § 3.3.1.

## REFERENCES

- [EL84] J. R. EGAN AND C. L. LIU, *Optimal bipartite folding and partitioning of a PLA*, IEEE Trans. CAD Integrated Circuits Systems, 3 (1984), pp. 191-199.
- [GJ79] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
- [GM82] V. GLIGOR AND D. MAIER, *Finding augmented set bases*, SIAM J. Comput., 11 (1982), pp. 602-609.
- [HNS82] G. D. HACHTEL, A. R. NEWTON, AND A. L. SANGIOVANNI-VINCENTELLI, *An algorithm for optimal PLA folding*, IEEE Trans. CAD Integrated Circuits Systems, 1 (1982), pp. 63-77.
- [J83] D. S. JOHNSON, personal communication, November 1983.
- [LV82] M. LUBY, V. VAZIRANI, U. VAZIRANI, AND A. L. SANGIOVANNI-VINCENTELLI, *Some theoretical results on PLA folding*, in Proc. IEEE International Conference on Circuits and Computers, 1982, pp. 165-170.
- [MC80] C. MEAD AND L. CONWAY, *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA, 1980.
- [R84] S. S. RAVI, *Heuristics for PLA folding: an analytical approach*, Ph.D. dissertation, Tech. Report 84-1, Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, July 1984.
- [RL84a] S. S. RAVI AND E. L. LLOYD, *The complexity of near-optimal PLA folding*, Tech. Report 84-18, Department of Computer Science, State University of New York, Albany, NY, December 1984.
- [RL84b] ———, *Worst case bounds for PLA folding heuristics*, Tech. Report 84-19, Department of Computer Science, State University of New York, Albany, NY, December 1984.
- [W79] R. A. WOOD, *A high density programmable logic array chip*, IEEE Trans. on Comput., 28 (1979), pp. 602-608.

## PARALLEL ALGORITHMS FOR TERM MATCHING\*

CYNTHIA DWORK<sup>†</sup>, PARIS C. KANELLAKIS<sup>‡</sup>, AND LARRY STOCKMEYER<sup>†</sup>

**Abstract.** We present a randomized parallel algorithm for term matching. Let  $n$  be the number of nodes of the directed acyclic graphs (dags) representing the terms to be matched. Then our algorithm uses  $O(\log^2 n)$  parallel time and  $M(n)$  processors, where  $M(n)$  is the complexity of  $n \times n$  matrix multiplication. The randomized algorithm is of the Las Vegas type, that is, the answer is always correct, although with small probability the algorithm might fail to produce an answer. The number of processors is a significant improvement over previously known bounds. Under various syntactic restrictions on the form of the input dags, only  $O(n^2)$  processors are required in order to achieve deterministic  $O(\log^2 n)$  parallel time. Furthermore, we reduce directed graph reachability to term matching using constant parallel time and  $O(n^2)$  processors. This is evidence that no deterministic algorithm can significantly beat the processor bound of our randomized algorithm. We also improve the P-completeness result of Dwork, Kanellakis, and Mitchell on the unification problem, showing that unification is P-complete even if both input terms are linear, i.e., no variable appears more than once in each term.

**Key words.** unification, term matching, parallel algorithms, logic programming

**AMS(MOS) subject classification.** 68Q

**1. Introduction.** Unification of terms is an important step in resolution theorem proving [14], with applications to a variety of symbolic computation problems. In particular, unification is used in PROLOG interpreters [2], [7], type inference algorithms [10] and term-rewriting systems [6]. Informally, two symbolic terms  $s$  and  $t$  are unifiable if there exists a substitution of additional terms for variables in  $s$  and  $t$  such that under the substitution the two terms are syntactically identical. For example, the terms  $f(x, x)$  and  $f(g(y), g(g(z)))$  are unified by substituting  $g(z)$  for  $y$  and  $g(g(z))$  for  $x$ .

Unification was defined in 1964 by Robinson in his seminal paper "A Machine Oriented Logic Based on the Resolution Principle" [14]. Robinson's unification algorithm required time exponential in the size of the terms. The following years saw a sequence of improved unification algorithms, culminating in 1976 with the linear time algorithm of Paterson and Wegman [12]. A general interest in parallel computing, together with specific interest in parallelizing PROLOG, led Dwork, Kanellakis, and Mitchell to search for a fast (time polynomial in  $\log n$ ) processor efficient (polynomially many processors) parallel unification algorithm [4]. Their results were negative: they proved that unification is complete for polynomial time, even if the input terms are represented as trees. A similar result was independently derived by Yasuura [18]. (The result in [18] is slightly weaker because it proves completeness for a more restricted class of inputs.) Thus, the existence of a fast, efficient parallel algorithm is *popularly unlikely*, in that it would contradict the popularly believed complexity theoretic conjecture that P, the class of problems solvable sequentially in polynomial time, is not

---

\* Received by the editors July 28, 1986; accepted for publication (in revised form) June 24, 1987. This is a revised and expanded version of the paper "Parallel Algorithms for Term Matching," appearing in the Proceedings of the 8th International Conference on Automated Deduction, July 1986, Oxford, United Kingdom, Lecture Notes in Computer Science, Vol. 230, pp. 416-430, © 1986 by Springer-Verlag.

<sup>†</sup> IBM Almaden Research Center, Department K53/802, San Jose, California 95120.

<sup>‡</sup> Computer Science Department, Brown University, Providence, Rhode Island 02912 and Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The research of this author was supported in part by an IBM Faculty Development Award and in part by National Science Foundation grant MCS-8210830.



contained in NC, the class of problems solvable in polylogarithmic time using polynomially many processors. However, Dwork et al. [4] found that term matching, a special case of unification in which one of the terms contains no variables, is in NC. Term  $s$  matches term  $t$  if there exists a substitution  $\sigma$  mapping variables in  $s$  to terms, such that  $\sigma(s)$ , the term obtained by replacing each occurrence of each variable  $x$  in  $s$  by  $\sigma(x)$ , is syntactically equal to  $t$ . Dwork et al. [4] obtained a matching algorithm requiring  $O(\log^2 n)$  time and about  $O(n^5)$  processors. Motivated by [4] some researchers interested in parallelizing PROLOG examined extant PROLOG programs to see whether in practice unification can be replaced by term matching. Preliminary results show that “often” the full power of general unification is not needed, and that term matching indeed suffices [9].

Let  $\text{NC}^k(f(n))$  be the class of problems solvable in time  $O(\log^k n)$  using  $f(n)$  processors on inputs of size  $n$ . Similarly, let  $\text{RNC}^k(f(n))$  be the class of problems solvable by a randomized algorithm in time  $O(\log^k n)$  using  $f(n)$  processors on inputs of size  $n$ . In defining these classes, our model of computation is the Concurrent-Read Exclusive-Write Parallel RAM (PRAM) [5], with word size  $O(\log n)$  on inputs of size  $n$ .

The algorithm of [4] shows that term matching is in  $\text{NC}^2(M(n^2))$ , where  $M(m)$  is the number of arithmetic operations required for  $m \times m$  matrix multiplication. Coppersmith and Winograd [3] show that  $M(m) = O(m^{2.5})$ , so the algorithm of [4] uses about  $n^5$  processors. The current paper provides substantially improved upper bounds on processors for the term-matching problem at no asymptotic cost in running time. However, the new algorithm is randomized, in that the individual processors make random choices (flip coins). It is a Las Vegas algorithm: an answer is always correct, but there is some small probability (taken over the set of coin tosses) that an execution of the algorithm will fail to produce an answer. In that case the algorithm can be run again. Our approach will be to show how to test two terms for syntactic equivalence in  $\text{RNC}^2(M(n))$ , where  $n$  is the total number of nodes in the dag representations of the two terms. This is the only randomized portion of the algorithm. We then show that term matching reduces to equivalence testing, and the reduction can be performed in  $\text{NC}^2(n^2)$ . Since  $M(n) \cong n^2$  the principal result for term matching then follows as an easy corollary.

The remainder of the paper is organized as follows. Section 2 contains results on testing two terms for syntactic equivalence. In addition, this section contains some “evidence” that  $M(n)$  is a lower bound on the number of processors needed for NC solutions to both term matching and equivalence, by showing that the directed acyclic graph reachability problem reduces to testing for equivalence by an  $\text{NC}^0(n^2)$  reduction. Section 3.1 describes the reduction from term matching to equivalence testing. For certain special cases we can improve on the time and/or processor bounds obtained in the general case when both terms are represented by arbitrary dags. These results are described in § 3.2. Finally, § 4 strengthens the known P-completeness results for unification, showing that unification is P-complete even if both terms are linear (each variable appears at most once in each term) and are represented by trees, but where there can be sharing of variables. (In contrast, if there is no sharing of variables and one of the terms is linear, then the problem can be solved in NC as we show in § 3.3.) The proof of P-completeness of unification of linear terms is quite different from and more intricate than that of [4], and in a sense provides a strongest possible P-completeness result for a restricted form of unification.

**2. Testing for equivalence of terms.** A *term* is defined recursively as follows. A variable symbol is a term; if  $f$  is a  $k$ -ary function symbol,  $k \geq 0$  (0-ary function symbols correspond to constants), and  $t_1, \dots, t_k$  are terms, then  $f(t_1, \dots, t_k)$  is a term.

A *dag* is a directed acyclic graph. If  $G$  is a dag and  $u$  is a node of  $G$ , then the subdag *rooted at  $u$*  is the subdag consisting of all nodes and edges reachable from  $u$ . A term  $t$  can be represented by a labeled dag in a very natural way. If  $t$  is a constant or a variable, then the representation is just a single node labeled  $t$ . When  $t = f(t_1, \dots, t_k)$ ,  $k \geq 1$ ,  $t$  can be represented by a dag consisting of a node labeled  $f$  with  $k$  outedges, labeled, respectively, 1 through  $k$ , such that the head of the edge labeled  $i$  is the root of a subdag representing  $t_i$ , for each  $i = 1, \dots, k$ . Figure 1 shows some examples of labeled dags and their corresponding terms. Note that if the term contains a repeated subexpression then its corresponding dag is not unique; for example, a term  $g(t, t)$  may be represented by using a single dag for both occurrences of  $t$  or by using a separate dag for each occurrence. A node  $u$  of a dag is a *root* if there are no edges directed into  $u$ , and  $u$  is a *leaf* if there are no edges directed out of  $u$  (each leaf must be labeled by either a constant or a variable). A term is *linear* if no variable appears more than once in the term. If the term  $t$  is linear and if  $G$  is any dag representation of  $t$ , then for each variable  $x$  occurring in  $t$  there is exactly one path in  $G$  from the root to a node labeled  $x$ . Let  $u$  and  $v$  be nodes of outdegree  $k$ , and let  $u_i$  (respectively,  $v_i$ ) denote the head of the edge from  $u$  (respectively,  $v$ ) with label  $i$ ,  $i = 1, \dots, k$ . Then we say  $u_i$  ( $v_i$ ) is the  *$i$ th child of  $u$*  (respectively, of  $v$ ), and we say  $u_i$  and  $v_i$  are *corresponding children* of  $u$  and  $v$ .

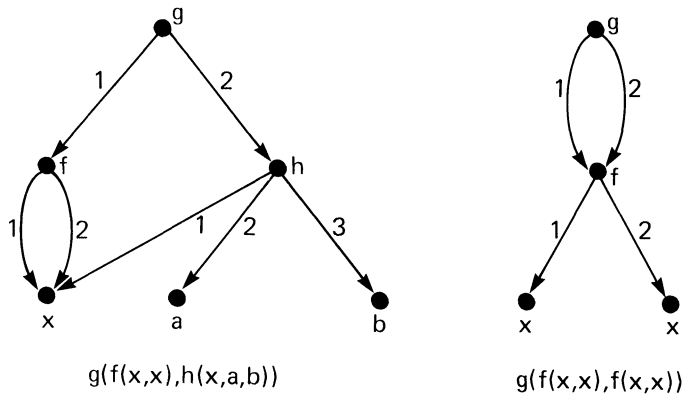


FIG. 1. Some examples of term dags.

Two rooted dags  $G$  and  $H$  are said to be *equivalent*, written  $G \equiv H$ , if they represent the same term. An instance of the equivalence problem is a triple  $(D, r_1, r_2)$ , where  $D$  is a labeled dag with two roots  $r_1$  and  $r_2$ . The two terms to be tested for equivalence are the terms represented by the subdags rooted at the two roots. ( $D$  could consist of two connected components, one with root  $r_1$  and the other with root  $r_2$ . In general, we allow the dags rooted at  $r_1$  and  $r_2$  to share nodes since our equivalence algorithm handles this smoothly.) The time and processor complexities of our algorithms are expressed as functions of  $n$ , the size of an instance, defined to be the maximum of the number of nodes of  $D$  and the maximum outdegree of any node of  $D$ . Since in most applications the number of nodes will dominate the maximum outdegree, there is no harm in viewing  $n$  as the number of nodes. For cases where the maximum outdegree dominates, it is easy to express the complexities of our algorithms as functions of two separate parameters, the number of nodes and the maximum outdegree.

All logarithms in this paper are to the base 2.

**2.1. A Las Vegas algorithm for testing equivalence of terms.** Let  $M(n)$  be an upper bound of the form  $cn^\omega$ , for constants  $c > 0$  and  $\omega > 2$ , on the number of arithmetic

operations in a straight-line algorithm which computes  $n \times n$  matrix multiplication where the algorithm contains no divisions and where all scalars (i.e., numerical constants) are integers. The Coppersmith–Winograd algorithm [3] is of this form, so  $M(n) = O(n^{2.5})$ .

Consider an instance  $(D', r_1, r_2)$  of the equivalence problem and let  $n$  be its size. Let  $G'$  and  $H'$  be the dags rooted at  $r_1$  and  $r_2$ , respectively. Since two terms are equal if and only if they are equal when we substitute constants for variables, we may assume that  $D'$  contains no variables. This will simplify matters in this section since we will want to use “variable” to refer to polynomial variables rather than term variables. Briefly, our approach is to represent  $G'$  and  $H'$  by multivariate polynomials  $P_{G'}$  and  $P_{H'}$  in such a way that  $G'$  and  $H'$  are equivalent if and only if the corresponding polynomials are equivalent. We then use a randomized algorithm of Schwartz [15] to check the equivalence of the polynomials. Since Schwartz’s result deals with sequential algorithms, we have some additional work to prove that the algorithm can be modified to run in  $\text{RNC}^2(M(n))$ .

In defining the polynomials, it is useful to first perform some modifications to the dag as follows. Given a labeled dag  $D'$ , we first add a new node  $z$  to  $D'$ , and add edges from every node in  $D'$  to the new node  $z$ , making  $z$  the unique leaf of the resulting dag, call it  $D$ . Each new edge  $(v, z)$  is  $D$  labeled with the outdegree of  $v$  in  $D$ . This process of adding a new leaf, connecting it to all nodes in the original dag, and labeling the new edges, is called *preparing* the dag, and  $D$  is said to be *prepared*. We view  $D$  as a term dag with the single constant symbol  $z$ ; the arity of each function symbol labeling a node of  $D$  (except the new node) is one more than the arity of the function symbol labeling the corresponding node of  $D'$ . If  $G$  and  $H$  are the prepared dags rooted at  $r_1$  and  $r_2$ , respectively, in  $D$  then it is obvious that  $G$  and  $H$  are equivalent if and only if  $G'$  and  $H'$  are equivalent. Therefore, we shall work with prepared dags in the remainder of this section.

Given a labeled prepared dag  $G$ , we define the corresponding polynomial  $P_G$  as follows. Our intention is to assign variables to edges; in § 2.1, “variable” means a variable of some polynomial  $P_G$ . Each path from the root to the leaf will then yield a monomial defined as the product of the variables assigned to the edges along the path. The final polynomial will then be the sum over all paths  $p$  from root to leaf of the monomial corresponding to  $p$ . We will show that two dags are equivalent if and only if their corresponding polynomials are equivalent. Reducing equivalence testing for term dags, which is in NC, to testing equivalence of polynomials, which is not even known to be in P, is not obviously progress. However, we will then apply the algorithm of Schwartz, modified to run in RNC, testing equivalence of the polynomials at a randomly selected point. A negative answer to this test is a proof of inequivalence. As we will see, a positive answer can sometimes be turned into a proof of equivalence. Sometimes a positive result will be inconclusive, in which case the algorithm can be run again.

We now describe the selection of the variables. Consider an arbitrary path from the root to the leaf. In selecting the variables corresponding to the edges of the path, care must be taken that the ordering information is not lost. In other words, given the monomial corresponding to the path we must be able to reconstruct the path. To this end, for each node  $v$  we compute the number of paths from  $v$  to  $z$  in  $G$ . Note that since  $G$  is prepared, if  $v$  is a proper ancestor of  $u$  then the number of paths from  $v$  to  $z$  exceeds the number from  $u$  to  $z$ ; this fact will play an important role in the proof of Lemma 1.

**VARIABLE NAMING RULE.** Let  $v$  be a node of  $G$  labeled with the  $k$ -ary function

symbol  $f$ , and let  $m$  be the number of paths from  $v$  to the leaf  $z$ . Then for each  $j \in \{1, \dots, k\}$ ,  $f_j^m$  is the variable assigned to the edge from  $v$  labeled  $j$ .

For each path  $p$  directed from the root of  $G$  to  $z$ , let the monomial  $\mu(p)$  be the product of all the variables corresponding to the edges of  $p$ . Then  $P_G = \sum_p \mu(p)$ .

LEMMA 1. *Two prepared term dags  $G$  and  $H$  are equivalent if and only if  $P_G$  and  $P_H$  are equivalent polynomials.*

*Proof.* We begin with the if direction. Let the *degree* of a polynomial denote the maximum number of variables in any monomial of the polynomial. Then  $P_G \equiv P_H$  implies that the two polynomials are of the same degree. The proof proceeds by induction on  $k$ , the degree of the polynomials.

$k = 1$ . In this case the two polynomials each consist of a single monomial (this is because of the way we defined a prepared dag). Thus, for some unary function symbols  $g$  and  $h$  we have  $P_G = g_1^1$  and  $P_H = h_1^1$ . Clearly, these polynomials can be equivalent if and only if  $g_1^1 = h_1^1$ , so the symbols  $g$  and  $h$  must be identical.

$k > 1$ . We assume the result inductively for polynomials of degree less than  $k$  and prove it for  $k$ . Let  $g$  (respectively,  $h$ ) be the label of the root of  $G$  (respectively,  $H$ ) and let  $i$  (respectively,  $j$ ) be the number of paths from the root of  $G$  (respectively,  $H$ ) to the leaf  $z$ . Write  $P_G = \sum_{x=1}^a g_x^i t_x$  and  $P_H = \sum_{x=1}^b h_x^j s_x$ , where the superscripts in the  $t_x$  and  $s_x$  are less than  $i$  and  $j$ , respectively. Then  $i = j$ , as otherwise the maximum superscripts of the two sets of variables differ. Thus, none of the  $t_x$  contain any variable name of the form  $h_y^i$  and similarly, no  $s_x$  contains any  $g_y^i$ . Because the only function symbol superscripted with  $i$  in  $P_H$  is  $h$ , it follows that  $g = h$ , whence  $a = b$ . Rewriting  $P_H$  with  $g$  replacing  $h$  yields  $P_H = \sum_{x=1}^a g_x^i s_x$ . For any  $x \in \{1, \dots, a\}$ , setting  $g_y^i$  to 0 for all  $y \neq x$  yields  $g_x^i t_x = g_x^i s_x$ , whence  $t_x = s_x$ . Since the degree of  $t_x$  is strictly less than  $k$ , we see by the inductive hypothesis that the term represented by  $t_x$  is equivalent to the term represented by  $s_x$ . Let  $r_1$  and  $r_2$  be the roots of  $G$  and  $H$ , respectively. Since  $g = h$  both roots are labeled with the same function symbol. Further,  $t_x = s_x$  for all  $x$ , so the corresponding arguments of  $g$  are the same in the two dags. Thus, the two dags are equivalent.

We now prove that if the two prepared dags are equivalent then the corresponding polynomials are equivalent. Because the height of a dag corresponds to the maximum depth of nesting of parentheses in the term it represents, if  $G \equiv H$  then the two dags are of the same height. The proof proceeds by induction on  $k$ , the height of the dags. We strengthen the induction hypothesis, showing that if two dags are equivalent then the number of paths from the root to the leaf is the same in the two dags.

$k = 1$ . All prepared dags of height 1 contain a unique edge, from root to leaf, so the corresponding terms are just 1-ary function symbols. Thus,  $G \equiv H$  implies the two dags are identical, whence they give rise to the same polynomial.

$k > 1$ . We assume the result inductively for dags of height less than  $k$  and prove it for  $k$ . Let  $r_1$  and  $r_2$  be the roots of  $G$  and  $H$ , respectively. If  $G \equiv H$  then the two roots are labeled with the same function symbol, say  $g$ , whence both roots have the same outdegree, say  $a$ . Further, because the dags are equivalent, for each  $i \in \{1, \dots, a\}$ , the subdag rooted at the  $i$ th child of  $r_1$  is equivalent to the subdag rooted at the  $i$ th child of  $r_2$ . Since these subdags are of height strictly less than  $k$  we see by the inductive hypothesis the polynomials corresponding to the subdags rooted at the  $i$ th children of the roots are equivalent, and the number of paths from the  $i$ th child of  $r_1$  to the leaf is equal to the number of paths from the  $i$ th child of  $r_2$  to the leaf, for all  $i$ . Thus, the total number of root-leaf paths is the same in the two dags. Let  $m$  denote this number. Let  $P_i$  denote the polynomial corresponding to the subdag rooted at the  $i$ th child of either root. Then  $P_G = g_1^m P_1 + \dots + g_a^m P_a$ , and this is precisely  $P_H$ .  $\square$

*Remarks.* (1) If the variables were not superscripted with the path numbers Lemma 1 would not hold. This is because the variables  $f_1$  and  $f_2$  commute, i.e.,  $f_1 f_2 = f_2 f_1$ , but the directed paths labeled  $f_1 f_2$  and  $f_2 f_1$  are not the same in a dag.

(2) Two very natural approaches to handling the commutativity problem do *not* work. Affixing  $c \times c$  matrices to the edges, for some constant  $c$ , cannot work. This follows from a theorem of Amitsur and Levitzki [1] which states that there are polynomials  $Q_1(x_1, \dots, x_{2c})$  and  $Q_2(x_1, \dots, x_{2c})$  with zero-one coefficients, involving noncommuting variables  $x_1, \dots, x_{2c}$ , such that  $Q_1$  and  $Q_2$  are not equivalent in general, but  $Q_1$  and  $Q_2$  are equivalent over the ring of  $c \times c$  matrices. A second approach, computing for each node  $v$  the maximum distance from  $v$  to  $z$  (instead of the number of paths) requires a special kind of matrix multiplication, in which the inner operation is  $+$  and the outer operation is  $\max$ . It is not known how to compute the “product” of two  $n \times n$  matrices in  $M(n)$  steps using this definition of multiplication.

In order to use Lemma 1 to test for equivalence of term dags, we must address several issues.

The number of paths from a node to the leaf may be as large as  $n^{n-1}$ , even though the outdegree of each node is bounded by  $n$ . Since the word size of our parallel random access machine (PRAM) is at most logarithmic in  $n$  we cannot actually compute these numbers. Similarly, we cannot evaluate the polynomials  $P_G$  and  $P_H$ , as for many possible choices of values for the variables the values of the polynomials will be too large to handle. We use modular arithmetic to handle these problems, performing all arithmetic modulo random primes, which in turn raises the question of how to obtain a random prime.

Testing equivalence of multivariate polynomials is not even known to be solvable in polynomial time, let alone in NC, even when we can actually evaluate the polynomials, much less when we cannot. For this we resort to a modification of Schwartz’s randomized algorithm.

The Schwartz algorithm is Monte Carlo, in that an answer of “inequivalent” is always correct, but an answer of “equivalent” is correct only with high probability. Thus, the final issue is that of deriving a proof of equivalence when Schwartz’s algorithm tells us two polynomials are equivalent. If the polynomials are arbitrary this problem is open. Our polynomials are not arbitrary; they are constructed from term dags. This can sometimes be exploited to obtain a proof of equivalence.

LEMMA 2. *For each constant  $k$ , there is an  $\text{RNC}^2(\log^3 n)$  algorithm that, on input  $n$ , with probability at least  $1 - n^{-k}$  produces a random prime  $q \in I = \{2, \dots, n^k\}$ .*

*Proof.* Our approach is to choose  $r$  random numbers,  $q_1, \dots, q_r \in I$  where  $r = d \log^2 n$  for an appropriate constant  $d$ . Each  $q_i$  is tested for primality  $(k+2) \log n$  times in parallel using Rabin’s randomized test [13]. Of those  $q_i$  having passed all tests one is selected at random. Rabin’s algorithm runs in time  $O(\log^2 n)$  on inputs from  $I$ . When given a prime, the algorithm always answers “prime”; when given a composite, it answers “composite” with probability at least  $\frac{1}{2}$ .

There are two ways in which the algorithm could fail to produce a prime. It may be that none of the  $q_i$  are prime. The Prime Number Theorem implies that for some constant  $c$ , independent of  $n$ , the probability that all  $r$  are composite is at most  $(1 - c/\log n)^r$ , which is at most  $n^{-k}/2$  if  $r = d \log^2 n$ , for some constant  $d$  independent of  $n$ . If at least one  $q_i$  is actually prime, it is still possible that a composite will be chosen. However, since Rabin’s algorithm errs with probability at most  $\frac{1}{2}$ , the probability that a composite passes all  $(k+2) \log n$  tests is at most  $n^{-k-2}$ . Thus, the probability that even one composite passes all tests is at most  $nn^{-k-2}$ , which is less than  $n^{-k}/2$  for sufficiently large  $n$ .  $\square$

The first source of error in our randomized protocol for testing equivalence of term dags is in the selection of  $q$ , because our algorithm may fail to produce a prime. However, even if a prime  $q$  is produced, performing arithmetic modulo  $q$  introduces a second source of error, as distinct numbers  $r$  and  $s$  may be congruent mod  $q$ . Thus, when computing the superscripts for the variables mod  $q$ , two variables assigned to edges from nodes with differing numbers of paths to the leaf may receive the same superscript. We must therefore choose  $q$  from a range sufficiently large as to make this event unlikely.

DEFINITION. Integer  $m$  is *bad* for the pair of integers  $(r, s)$  if  $r \neq s$  but  $r$  and  $s$  are congruent mod  $m$ .

LEMMA 3. For every pair of distinct numbers  $r, s \leq n^n$  there are at most  $n \log n$  bad primes.

*Proof.* Let  $\{p_1, \dots, p_x\}$  be the set of primes bad for  $(r, s)$ . Then the product  $\pi = p_1 p_2 \dots p_x$  is also bad for  $(r, s)$ . Since  $r, s \leq n^n$  it must be that  $\pi \leq n^n$  (otherwise the two numbers could not be congruent mod  $\pi$ ). On the other hand, since each  $p_i \geq 2$  we have  $2^x \leq \pi$ . Thus,  $2^x \leq \pi \leq n^n$ , whence  $x \leq n \log n$ , as was to be shown.  $\square$

Let us say a prime  $q$  is *bad* for a prepared dag if there exist two nodes  $u$  and  $v$  in the dag such that  $r$  and  $s$  are the number of paths to the leaf from  $u$  and  $v$ , respectively,  $r \neq s$ , and  $q$  is bad for  $r, s$ .

COROLLARY 4. Let  $D$  be a prepared dag with  $n$  nodes in which each node has outdegree at most  $n$ . Then a random prime  $q$  drawn from  $[2, n^k]$  is bad for  $D$  with probability at most  $O(n^{3-k} \log^2 n)$  for any fixed  $k \geq 3$ .

*Proof.* There are at most  $n^2$  pairs of nodes, and for each pair there are at most  $n \log n$  bad primes, so there are at most  $n^3 \log n$  primes bad for  $D$ . By the Prime Number Theorem there are  $\Omega(n^k / (k \log n))$  primes in  $[2, n^k]$ . Thus, if a prime is selected uniformly at random from this range, then the probability that the chosen prime is bad for  $D$  is at most

$$O\left(n^2 \log n \frac{k \log n}{n^k}\right) = O(n^{3-k} \log^2 n). \quad \square$$

Let  $Q$  be a polynomial in  $t$  variables with integer coefficients. An *assignment* for  $Q$  is a  $(t+1)$ -tuple of integers  $A = (i_1, \dots, i_t, p)$ . We define

$$Q(A) = Q(i_1, \dots, i_t) \pmod{p}.$$

An assignment  $A$  is a *modular zero* of  $Q$  if  $Q(A) = 0$ . We let  $\max_v(Q, m)$  denote the maximum value attained by  $Q$  over the rectangle in which the absolute value of each variable is bounded by  $m$ .

In order to bound the probability of error in testing the polynomials for equality, we appeal to the following theorem due to Schwartz [15].

THEOREM (Schwartz). Let  $2m + 1 \geq c \cdot \deg(Q)$ , let  $I$  be the set of integers of absolute value  $\leq m$ , let  $J$  be a set of primes, and suppose that the product of the  $c^{-1}|J| + 1$  smallest primes in  $J$  exceeds  $\max_v(Q, m)$ . Then if  $Q$  is not identically equal to zero, the number of elements of  $I^t \times J$  which are modular zeros of  $Q$  is at most  $2c^{-1}|I|^t|J|$ .

The  $Q$  we will be considering is  $P_G - P_H$ , which is of degree at most  $n$  (the number of nodes in the union of the two dags), and contains  $t \leq n^2$  variables. Suppose we wish to bound the probability of choosing a modular zero by  $n^{-k}$ . We apply Schwartz's Theorem as follows. Let  $b \geq k + 1$  and  $m = n^b$ . Let  $J$  be the set of primes in  $[2, n^{2b}]$ , and  $I = [-n^b, n^b]$ , where  $J$  is the set from which we select our random prime  $p$  and  $I$  is the interval from which each of our  $t$  variables is chosen. Each term in  $Q$  is the

product of at most  $n$  variables, and  $Q$  contains at most  $n^n$  terms, so

$$\max_v (Q, n^b) \leq n^{bn}.$$

To satisfy the conditions of the theorem we take  $c = (2n^b + 1)/n$ . Estimating the number of primes in  $J$  to be  $dn^{2b}/2b \log n$ , for some constant  $d$ , the condition that the product of the  $c^{-1}|J| + 1$  smallest primes exceeds  $\max_v (Q, m)$  is clearly satisfied if

$$\frac{dn^{2b+1}}{(2n^b + 1)2b \log n} \geq n(b+1) \log n$$

(because each prime is no smaller than 2). This reduces to

$$dn^{2b} \geq 2b(b+1)(2n^b + 1)(\log n)^2,$$

which is clearly true for all sufficiently large  $b$ , assuming  $n \geq 2$ . We therefore have the following corollary.

**COROLLARY 5.** *Let  $Q$  be a polynomial of degree  $n \geq 2$  such that  $Q \neq 0$ . For any fixed  $k$  there is a constant  $b$ , depending only on  $k$ , such that by choosing a random assignment  $A$  for  $Q$  by choosing a random prime in the interval  $[2, n^{2b}]$  and randomly selecting values for the variables in  $Q$  from  $I = [-n^b, n^b]$ ,  $\Pr[Q(A) = 0] \leq n^{-k}$ .  $\square$*

When our polynomial  $Q$  is formed from two term dags it is obvious how we obtain a proof of inequality ( $Q(A) \neq 0$ ), while if  $Q(A) = 0$  we may have hit a modular zero at  $A$ . However, in some cases we can actually obtain a proof of equivalence. In part this is due to the fact that when we evaluate the polynomials corresponding to the two roots of our dags, we simultaneously evaluate all  $n$  polynomials corresponding to the subdags rooted at each of the  $n$  nodes, just as when testing directed graph reachability by computing the transitive closure of the adjacency matrix we obtain reachability information for all pairs of points.

Let  $D$  be a prepared dag with two roots, and let  $G$  and  $H$  be the subdags rooted at these roots. Choose variables for the edges according to the Variable Naming Rule. For each node  $v$ , let  $P_v$  be the polynomial induced by the subdag rooted at  $v$ . By convention,  $P_z$  is the identically zero polynomial where  $z$  is the unique leaf added when the dag was prepared. Running our parallelized version of Schwartz's algorithm with a particular assignment  $A$  to the variables induces an equivalence relation on the nodes of  $D$ , where distinct nodes  $x$  and  $y$  are in the same class if and only if the algorithm tells us  $P_x(A) = P_y(A)$ . Let us denote this relation by  $x \equiv_A y$ . For each pair of  $\equiv_A$  nodes we check two things. First, we verify that both nodes are labeled with the same function symbol. Second, we check that the corresponding children of each pair of  $\equiv_A$  nodes are also  $\equiv_A$ . We claim that if both these tests are passed for all pairs of  $\equiv_A$  nodes then the subdags rooted at  $x$  and  $y$  are equivalent.

**LEMMA 6.** *Let  $D$  and  $A$  be as above. Let  $x$  and  $y$  be nodes of  $D$  satisfying  $x \equiv_A y$ . If for all  $u, v$  such that  $u \equiv_A v$  and  $u$  and  $v$  both belong to the union of the subdags induced by  $x$  and  $y$ , it is the case that the labels of  $u$  and  $v$  agree and each pair of corresponding children of  $u$  and  $v$  are  $\equiv_A$ , then the subdags rooted at  $x$  and  $y$  are equivalent.*

*Proof.* The proof is by induction on  $k$ , the length of the longest path from  $x$  to the leaf  $z$ .

$k = 0$ . In this case  $x = z$ , which has the label  $z$ . If the conditions of the lemma are satisfied then  $y$  has label  $z$ , so  $y = z$  as well.

$k > 0$ . Assume the result inductively for  $k - 1$ , and assume the conditions of the lemma hold. Then the labels of  $x$  and  $y$  agree. Further, corresponding children of  $x$  and  $y$  are  $\equiv_A$ . Thus, by induction, the subdags rooted at their corresponding children are equivalent. It follows immediately that the subdags rooted at  $x$  and  $y$  are equivalent.  $\square$

Lemma 6 says it is sometimes possible to prove equivalence of polynomials generated from term dags. In particular, this can be done when for every pair of  $\equiv_A$  nodes, the subdags rooted at the two nodes are actually equivalent. It is possible that two nodes labeled with different function symbols are “equivalent” under  $\equiv_A$ . In this case we can prove nothing about the ancestors of these nodes and we must run the algorithm again.

**THEOREM 7.** *The problem of testing equivalence of two term dags can be solved by a Las Vegas algorithm in  $\text{RNC}^2(M(n))$ . For any fixed  $k$  the probability that no answer is produced can be bounded above by  $n^{-k}$ .*

*Proof.* The algorithm is as follows. The algorithm involves an integer parameter  $k'$  which is chosen depending on  $k$ . Let  $(D', r_1, r_2)$  be an instance of the equivalence problem, consisting of a dag  $D'$  and two roots.

1. Prepare  $D'$  by adding a new leaf  $z$ , adding an edge from each node to  $z$ , and labeling the added edges. Let  $D$  denote the resulting dag.
2. Choose a random prime  $q \in [2, n^{k'}]$ .
3. For each node  $v$  compute the number (mod  $q$ ) of paths from  $v$  to  $z$ .
4. For each edge  $e$  compute the name of the associated variable according to the Variable Naming Rule.
5. Sort the variable names and remove duplicates from the sorted list.
6. Choose an assignment  $A$  by choosing random values from the range  $[-n^{k'}, n^{k'}]$  for the variables in the list produced at the previous step and by choosing a random prime  $p \in [2, n^{2k'}]$ .
7. For each node  $v$  in  $D$  evaluate  $P_v$ , the polynomial corresponding to the subdag rooted at  $v$ , at the point  $A$  chosen in step 6. If  $r_1 \not\equiv_A r_2$  then output “inequivalent” and halt.
8. If  $r_1 \equiv_A r_2$  then try to prove equivalence by the method of Lemma 6. If successful, output “equivalent,” else output “?”.

We now describe steps 3, 7, and 8 in more detail.

For simplicity, let  $n$  denote the number of nodes in the prepared dag  $D$  (the original dag plus the new leaf). To perform step 3, we define an  $n \times n$  matrix  $E$  whose  $ij$  entry is the number of edges from  $i$  to  $j$  (mod  $q$ ). In other words,  $E_{ij}$  is the number (mod  $q$ ) of paths of length 1 from  $i$  to  $j$  (diagonal entries are 0). In general,  $(E^k)_{ij}$  is the number (mod  $q$ ) of paths of length  $k$  from  $i$  to  $j$ . Since we are interested in all paths of all lengths from each node to  $z$ , we compute the sum of all powers of  $E$  from  $E^0$  to  $E^{m-1}$  where  $n \leq m < 2n$  and  $m$  is a power of 2. To do this we use the well-known identity

$$I + E + E^2 + E^3 + \dots + E^{m-1} = (I + E)(I + E^2)(I + E^4) \dots (I + E^{m/2}).$$

Multiplication of  $m \times m$  matrices can be computed in  $\text{NC}^1(M(n))$  (see, e.g., Pan and Reif [11, Appendix A]). Therefore, the powers of  $E$  needed to compute the rhs can be computed in  $\text{NC}^2(M(n))$  by repeated squaring. All arithmetic is done mod  $q$ . Since the rhs contains at most  $\log n$  terms the product can also be computed in  $\text{NC}^2(M(n))$  once the powers of  $E$  are computed. Let  $E^* = \sum_{i=0}^{m-1} E^i$  (mod  $q$ ). Then for each node  $v$ ,  $E_{vz}^*$  is the number (mod  $q$ ) of paths from  $v$  to  $z$ .

The evaluations of the polynomials  $P_v$  proceed in a similar fashion. Thus, to perform step 7 we define a matrix  $B$  whose  $ij$  entry is the sum of the values chosen for the variables assigned to the edges from  $i$  to  $j$ . Viewed differently,  $B_{ij}$  is the value of the polynomial corresponding to the subdag containing paths of length 1 from  $i$  to  $j$ . In general,  $(B^k)_{ij}$  is the value of the polynomial which is the sum of the monomials



corresponding to all paths of length  $k$  from  $i$  to  $j$ . As in the case of the computation of path numbers, we are interested in  $B^* = I + B + B^2 + B^3 + \dots + B^{m-1}$ . This is computed as described above, all arithmetic being performed mod  $p$ . Then  $B_{uz}^* = P_v(A)$ . For any two nodes  $x$  and  $y$  we have

$$x \equiv_A y \Leftrightarrow B_{xz}^* = B_{yz}^*.$$

To perform step 8, the verification of equivalence, we proceed as follows. For all nodes  $x$  and  $y$  such that  $x \equiv_A y$  we check that the labels of  $x$  and  $y$  are equal (if they are not then we can prove nothing, and the algorithm outputs “?”). Given that they are equal, we wish to check that corresponding children of  $\equiv_A$  nodes are themselves  $\equiv_A$ . Let  $e = (u, v)$  be an edge with label  $i$ . Corresponding to  $e$  there is a triple  $(B_{uz}^*, i, B_{vz}^*) = (P_u(A), i, P_v(A))$ . The set of triples corresponding to all the edges of the prepared dag are sorted lexicographically. We then examine the sorted list for a pair of adjacent triples whose first and second components match but whose third components differ. If no such pair exists the algorithm outputs “equivalent,” else it outputs “?”.

This completes the description of the algorithm. It remains to prove correctness when the random choices are good and to analyze the probability of making a bad random choice. Let  $G$  and  $H$  be the subdags rooted at  $r_1$  and  $r_2$ , respectively.

By Lemma 1,  $G \equiv H$  implies  $P_G \equiv P_H$ , and in the absence of a bad random choice in step 2,  $G \not\equiv H$  implies  $P_G \not\equiv P_H$ . If  $P_G \not\equiv P_H$ , then in the absence of a bad choice for the assignment  $A$  at step 6,  $r_1 \not\equiv_A r_2$ , so we have a proof that  $G \not\equiv H$ . If  $P_G \equiv P_H$ , then, in the absence of a bad choice for  $A$ , for all pairs of nodes  $u, v$  such that  $u \equiv_A v$  it is the case that the subdags rooted at  $u$  and  $v$  represent the same terms, in which case we have a proof of equivalence by Lemma 6.

This proves correctness in the absence of bad choices. We now bound the probability of making a bad choice.

As shown in Lemma 2 and Corollary 4, we can bound the probability of a bad choice in step 2 by  $n^{-k}$  for any fixed  $k$ . We now examine the probability of obtaining a “?” output given that no bad choice was made in step 2.

A “?” will be produced if for some pair of nodes  $u$  and  $v$ , the subdags rooted at the nodes are inequivalent but  $u \equiv_A v$ . By Lemma 2 and Corollaries 4 and 5 the probability of this occurring for a particular pair of nodes can be bounded above by  $n^{-k}$  for any fixed  $k$ . As there are only  $n^2$  pairs of nodes, the probability of this event occurring for any pair of nodes can be similarly bounded.

This completes the proof of Theorem 7.  $\square$

**2.2. Directed reachability reduces to equivalence testing.** The *dag reachability* problem is: Given a dag  $D$  and two distinguished nodes  $s$  and  $t$  of  $D$ , does there exist a path from  $s$  to  $t$ ? The size of the instance is the number of nodes in  $D$ . While directed reachability is known to be complete for  $\text{NSPACE}(\log n)$  with respect to logspace reductions, and therefore to be in NC, little is known about the number of processors needed to solve this problem in polylog time. In fact, all known NC or RNC algorithms compute the transitive closure of the adjacency matrix for  $D$  by repeated squaring and therefore use  $M(n)$  processors (to within logarithmic factors). Thus, while reducing directed reachability to term equivalence does not yield a lower bound on the number of processors needed to test for equivalence in NC, it does provide some “evidence” that  $M(n)$ , the processor bound obtained in Theorem 7, cannot be significantly improved.

**THEOREM 8.** *The directed acyclic graph reachability problem is  $\text{NC}^0(n^2)$  reducible to the equivalence testing problem.*

*Proof.* Given a dag  $D_1$  with distinguished nodes  $s_1$  and  $t_1$  we will construct a pair of term dags  $E_1$  and  $E_2$  which will be equivalent if and only if there is no path from  $s_1$  to  $t_1$  in  $D_1$ . The construction is illustrated in Fig. 2.

Without loss of generality we assume  $D_1$  has a unique root (if this is not the case then create a new root with edges to each of the original roots). We begin by turning  $D_1$  into a term dag. For each  $k$ ,  $0 \leq k \leq n$ , and for each node  $v$  of  $D_1$  with outdegree  $k$ , label  $v$  with the function symbol  $f^k$ . Label the outedges from 1 to  $k$  in arbitrary order.

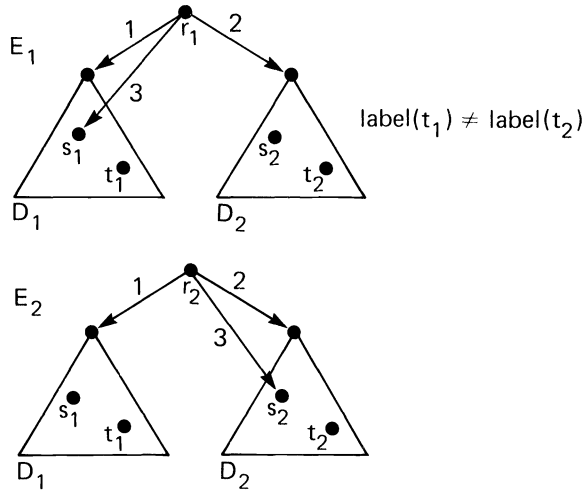


FIG. 2. The dags constructed in the proof of Theorem 8.

Create a copy  $D_2$  of  $D_1$ , identical to  $D_1$  but with  $t_2$ , the  $D_2$  copy of  $t_1$ , labeled with a new function symbol  $g$ . Let  $s_2$  be the  $D_2$  copy of  $s_1$ . There is a path from  $s_1$  to  $t_1$  if and only if  $s_1$  and  $s_2$  are the roots of inequivalent subdags. However, we are not done, since the  $s$  nodes are internal, and we have been assuming that an equivalence algorithm takes as input two roots and determines whether the dags rooted at these roots represent equivalent terms.

We next create a new dag  $E_1$  composed of  $D_1$  and  $D_2$  by creating a new root  $r_1$  and making the roots of  $D_1$  and  $D_2$  first and second children, respectively, of  $r_1$ . Let  $s_1$  be the third child of  $r_1$ . The edges from  $r_1$  to its children are labeled accordingly, and  $r_1$  is labeled with function symbol  $f^3$ .

Finally, we create a copy  $E_2$  of  $E_1$ , identical to  $E_1$  but with the 3-edge of  $r_2$ , the root of  $E_2$ , pointing to the  $E_2$  copy of  $s_2$  (instead of to the copy of  $s_1$ ). The subdags rooted at the first and second children of the two roots are identical by construction. Thus, the two dags are equivalent if and only if the subdags rooted at the third children are equivalent. As observed above, this holds if and only if there is no path from  $s_1$  to  $t_1$  in  $D_1$ .

If  $D_1$  contains  $n$  nodes then the four copies can be constructed in  $O(1)$  time using  $n^2$  processors (one per edge). The two new roots and six additional edges are added in constant time as well.  $\square$

**3. Reducing term matching to equivalence testing.** In § 2.1, we showed that two term dags can be tested for equivalence in  $\text{RNC}^2(M(n))$ . We now show that these bounds apply to the term-matching problem as well. Recall that term  $s$  matches term  $t$  if and only if there exists a  $\sigma$  mapping variables to terms such that  $\sigma(s) = t$ . Note

that matching is not a symmetric relation. An instance of the term-matching problem is a pair of disjoint labeled term dags representing the terms  $s$  and  $t$  to be matched. The size of the instance is the number of nodes in the union of the two dags. Although our matching algorithm requires the two dags to be disjoint, nondisjoint instances can be transformed to disjoint instances by performing a reachability computation from the two roots; as in the proof of Theorem 7, this can be done in  $\text{NC}^2(M(n))$ . In particular, disjointness is not needed in Corollary 10. Both term matching and equivalence testing are special cases of the unification problem. In the term-matching problem one of the terms (the second one) is considered *constant*:  $\sigma$  is not applied to the second term. In the equivalence problem, both terms are considered constant.

In § 3.1, we describe our results for general dags. Special cases are discussed in § 3.2.

**3.1. Term matching on general dags.** In this section we will prove the following theorem.

**THEOREM 9.** *Term matching reduces to testing for equivalence in  $\text{NC}^2(n^2)$ .*

From Theorems 7 and 9 we obtain our main result.

**COROLLARY 10.** *Term matching can be solved by a Las Vegas algorithm in  $\text{RNC}^2(M(n))$ .*

As mentioned in the Introduction we also have some specialized algorithms for term matching which depend on the form of the constant term. However, the general outline of the algorithm is the same in all cases. For ease of exposition we break the algorithm into four main steps. Since the first term, henceforth  $s$ , will always be represented by an arbitrary dag  $G$ , we may assume without loss of generality that each variable of  $s$  is the label of exactly one node of  $G$  (multiple copies of  $x$  can be merged into one copy).

Steps 1–4 below outline all our matching algorithms. Let  $G$  and  $H$  denote the dags representing  $s$  and  $t$ , respectively.

**Match ( $G, H$ ).**

1. A processor is assigned to each node of  $G$ . A spanning tree  $T$  of  $G$  is formed by having the processor assigned to each node  $v$  of  $G$  (except the root) arbitrarily choose one of the edges directed into  $v$ .

For each node  $v$  in  $G$ , there is a unique sequence of spanning tree edges from the root of  $G$  to  $v$ . We use  $\rho(v)$  to denote the sequence of edge labels along this path. An *embedding* is a mapping  $d$  from nodes of  $G$  to nodes of  $H$  such that  $d$  maps the root of  $T$  to the root of  $H$  and if  $v$  and  $u$  satisfy  $\rho(v) = \rho(u) \cdot i$  then there is an edge labeled  $i$  from  $d(u)$  to  $d(v)$ . In other words,  $d(u)$  is that node in  $H$  reached by following the edge sequence  $\rho(u)$  from the root of  $H$ . If  $T$  cannot be embedded in  $H$  then there is a sequence of edge labels present in  $G$  but absent from  $H$ ; hence  $G$  does not match  $H$ .

2. Embed  $T$  in  $H$ , if possible, and let  $d$  be the resulting embedding. Check that for all nodes  $u$  of  $T$ ,  $u$  and  $d(u)$  are labeled by the same function symbol. If there is no embedding or if the check fails, then output “no match.”
3. For each node  $v$  of  $G$  labeled by a variable, say  $x$ , let  $\sigma(x)$  be the term rooted at  $d(v)$  in  $H$ . Apply the substitution  $\sigma$  to  $G$  by replacing all edges of the form  $(u, v)$  by  $(u, d(v))$ . Let  $C = \sigma(G)$  denote the dag obtained from  $G$  by performing the substitution.
4. Test dags  $C$  and  $H$  for equivalence.  $G$  matches  $H$  if and only if  $C \equiv H$ .

Note that the mapping  $\sigma$  can be described by specifying, for each variable  $x$  labeling a node  $v$  of  $G$ , the pair  $(x, d(v))$ .

To prove correctness of the algorithm we need the following lemma.

LEMMA 11. *Let  $G$ ,  $H$ , and  $C$  be as in Algorithm Match. Then  $G$  matches  $H$  if and only if  $C \equiv H$ .*

*Proof.* Clearly, if  $C \equiv H$  then  $G$  matches  $H$ , as then  $C \equiv \sigma(G) \equiv H$ , where  $\sigma$  is the substitution described in step 3.

To prove the opposite direction we note that if  $G$  matches  $H$ , then the substitution  $\sigma$  found by the algorithm is the only one possible. Suppose otherwise that there was some other  $\sigma'$  such that  $\sigma'(G) \equiv H$ . Let  $x$  be any variable of  $G$  and let  $v$  be the node labeled by  $x$ . Since the roots of  $\sigma'(G)$  and  $H$  are equivalent, the two nodes reached by following the path  $\rho(v)$  from both roots must be equivalent. These nodes are  $v$  in  $G$  and  $d(v)$  in  $H$ , where  $d$  is the embedding found by the algorithm. Therefore,  $\sigma'(x) = \sigma(x)$ .  $\square$

To prove Theorem 9 we need only prove we can perform the embedding (step 2) in  $NC^2(n^2)$ . To do this, we will need a few lemmas.

LEMMA 12. *Given a directed path  $p$  with  $n$  nodes whose nodes are labeled with function symbols and whose edges are labeled with integers, and given a term dag  $H$  with  $k$  nodes ( $k \geq n$ ), there is an algorithm which checks whether  $p$  can be embedded in  $H$  (and produces an embedding if there is one) in  $NC^1(kn)$ .*

*Proof.* We form the product graph  $Z = p \times H$  whose nodes are all pairs of nodes  $(u, v)$  where  $u \in p$  and  $v \in H$ . There is an edge labeled  $i$  from  $(u, v)$  to  $(u', v')$  in  $Z$  if and only if there are edges  $(u, u')$  in  $p$  and  $(v, v')$  in  $H$  both labeled  $i$ . Let  $p_1$  and  $p_n$  be the initial and final nodes of  $p$ , respectively, and let  $r$  be the root of  $H$ . Then  $p$  can be embedded in  $H$  if and only if there is a path in  $Z$  from  $(p_1, r)$  to  $(p_n, w)$  for some node  $w$  of  $H$ .

Note that since  $p$  is a path, each node in the product graph  $Z$  has outdegree at most 1. We can therefore apply the standard technique of "pointer chasing." We assign a processor to each node of  $Z$ . These processors initialize two arrays; a bit array  $E$  which indicates whether a node is known to be reachable from  $(p_1, r)$  (should one exist), and a successor array  $S$ .

1. Initialization:

If  $a = (p_1, r)$  then  $E(a) := 1$ , else  $E(a) := 0$ .

If there is an edge of  $Z$  from  $a$  to  $b$ , then  $S(a) := b$ .

For all nodes  $w$  in  $H$ ,  $S((p_n, w)) := (p_n, w)$ .

2. Repeat  $\lceil \log n \rceil$  times:

If  $E(a) = 1$  then  $E(S(a)) := 1$ ;

$S(a) := S(S(a))$ .

3. Test for embedding:

If  $S((p_1, r)) = 0$  then output "No embedding" and halt.

4. Find embedding given that one exists:

If  $E((u, v)) = 1$  then  $v$  is the embedding of  $u$ .

The number of nodes  $a$  for which  $E(a) = 1$  doubles at each step until all nodes reachable from  $(p_1, r)$  are found. Thus, after execution of the algorithm  $E(a) = 1$  if and only if  $a$  is reachable from  $(p_1, r)$ . Moreover, after execution of the algorithm  $S((p_1, r)) \neq 0$  if and only if a node of the form  $(p_n, w)$  is reachable from  $(p_1, r)$ .  $\square$

LEMMA 13 (Tarjan and Vishkin [16]). *Let  $T$  be a tree of size  $m$ . There are  $NC^1(m)$  algorithms which produce (a) the depth-first numbering of  $T$  and, (b) for each node  $u$  of  $T$ , the number of nodes in the subtree rooted at  $u$ .*

The following lemma shows that the embedding of a tree in a dag can be accomplished in  $NC^2(n^2)$ , completing the proof of Theorem 9.

LEMMA 14. *The problem of embedding a tree  $T$  having  $m$  nodes in a dag  $H$  having  $k$  nodes is in  $\text{NC}^2(km)$ .*

*Proof.* The embedding is done by a recursive divide-and-conquer approach.

1. For each node  $v \in T$  compute the size of the subtree rooted at  $v$ . This can be done in  $\text{NC}^1(m)$  by Lemma 13(b). Let the *weight* of a directed edge  $(u, v)$  be the size of the subtree rooted at  $v$ . Consider any path originating at the root of  $T$  defined by selecting the heaviest edge out of each node to be in the path (ties are broken arbitrarily). Let us call such a path a *heavy path*.
2. Embed the heavy path. This can be done in  $\text{NC}^1(km)$ , by Lemma 12.
3. For each node  $v$  in the heavy path embedded in step 2, embed all the children of  $v$ . This can be accomplished in  $\text{NC}^0(km)$ .
4. For each node  $u$  embedded in step 3, embed the subtree rooted at  $u$  in the subdag rooted at  $d(u)$  where  $d$  is the embedding constructed in step 3. These embeddings are performed recursively in parallel.

Although the number of nodes not embedded by the end of step 3 may be quite large, each remaining *subtree* contains at most  $m/2$  nodes. Letting  $\text{Time}(m)$  be the parallel time to embed a tree of size  $m$ , this yields the recurrence relation

$$\text{Time}(m) = c \log m + \text{Time}(m/2),$$

which has solution  $O(\log^2 m)$ . Since the subtrees embedded in step 4 are pairwise disjoint, it is easy to see that  $km$  processors suffice to do all these embeddings in parallel.  $\square$

**3.2. Special cases of term matching.** In this section, we describe algorithms for checking equivalence of constant dags  $C$  and  $H$  (as obtained in step 3 of Algorithm Match) when  $H$  is of a special form. When used in step 4 of Algorithm Match these yield improved results for term matching. We assume as in § 3.1 that the two input dags are disjoint.

Let  $u$  and  $v$  be nodes of a term dag. We say  $u \equiv v$  if the term represented by the subdag rooted at  $u$  equals the term represented by the subdag rooted at  $v$ . A dag is *compact* if  $u \equiv v \Rightarrow u = v$ , for all nodes  $u$  and  $v$ .

The proof of the following lemma is straightforward.

LEMMA 15. *Let  $G$  be an arbitrary dag with spanning tree  $T$ . Let  $H$  be a second dag and let  $d$  be an embedding of  $T$  in  $H$ . Then  $G$  matches  $H$  if and only if the following two conditions are satisfied.*

(a) *For each node  $v$  of  $G$  not labeled with a variable name, the label of  $v$  equals the label of  $d(v)$ .*

(b) *For each edge  $e = (u, v)$  in  $G$  but not in the spanning tree  $T$ ,  $d(u)_i \equiv d(v)$ , where  $i$  is the label of  $e$  and  $d(u)_i$  denotes the  $i$ th child of  $d(u)$  in  $H$ .  $\square$*

Lemma 15 implies term matching is particularly easy when  $H$  is a compact dag, for in that case checking conditions of the form  $d(u)_i \equiv d(v)$  reduces to checking  $d(u)_i = d(v)$ .

COROLLARY 16. *The problem of determining whether an arbitrary dag matches a compact dag is in  $\text{NC}^2(n^2)$ .  $\square$*

A slightly harder case is when  $H$  is a tree. However, since the tree representation of a term is unique, testing equivalence of trees (as required by Condition (b) of Lemma 15) reduces to checking that the trees are identical, and while this is more difficult than checking equality of nodes we can solve it efficiently using some of the results of Tarjan and Vishkin stated in Lemma 13. In effect, we will change  $H$  into a compact dag by determining, for all nodes  $u$  and  $v$  of  $H$ , whether  $u \equiv v$ .

The following uses the depth-first numbering produced by Lemma 13(a). This numbering respects the edge labels, in the sense that for each node  $v$  the numbers in the subtree rooted at the  $i$ th child of  $v$  are smaller than those assigned to nodes in the subtree rooted at the  $(i + 1)$ st child.

LEMMA 17. *Checking equivalence of trees of size  $n$  is in  $NC^1(n)$ .*

*Proof.* We will show that the depth-first numbers and node labels of the nodes in a tree completely specify the tree. The lemma then follows by Lemma 13(a).

Given a tree  $T$  of size  $n$  we show by induction on  $k$  that the subtree consisting of the first  $k$  nodes in the depth-first numbering of  $T$  can be recreated from the depth-first numbers and labels of these nodes. In the following, let “node  $i$ ” denote the node in  $T$  with depth-first number  $i$ .

The basis  $k = 1$  is trivial, since the root is always labeled 1.

$k > 1$ . We assume the result for subtrees of size  $k - 1$  and prove it for  $k$ . Given the subtree  $S$  consisting of the first  $k - 1$  nodes of  $T$ , we find the largest  $i \leq k$  such that  $m$ , the number of children of node  $i$  in  $S$ , is less than the arity of the label of node  $i$ . Then node  $k$  is the  $(m + 1)$ st child of node  $i$ . Clearly,  $k$  cannot be the child of any node with depth-first number greater than  $i$ , since all of these have all their children in  $S$ . On the other hand, node  $k$  cannot be the child of any node with depth-first number less than  $i$ . To see this, assume for the sake of contradiction that node  $k$  is the child of a node  $u$ , where  $u < i$ . Let  $w$  be the least common ancestor of  $u$  and  $i$ . Let  $j$  be the label of the edge from  $w$  on the path to  $i$ . By the inductive hypothesis,  $w$  was correctly given its first  $j$  children in the construction of  $S$ . In particular, the  $j$ th child of  $w$  is added only if the subtrees rooted at the first  $j - 1$  children are complete. Thus, if  $u$  does not have all its children in  $S$  it must be that  $u = w$ . In this case, the first available slot for another child of  $u$  is the  $(j + 1)$ st. But all descendants of the  $j$ th child of  $u$  must have numbers smaller than that of the  $(j + 1)$ st, so if  $i$  does not have all its children in  $S$ ,  $k$  cannot be a child of  $u$ .  $\square$

Our approach to matching a general dag with a tree is to find all equivalent nodes, thereby effectively turning the tree into a compact dag. We first eliminate many pairs of nodes from consideration since they are obviously not equivalent, and then apply Lemma 17 in parallel to the remaining pairs.

The size of a node  $v$ , denoted  $size(v)$ , is the number of nodes in the subtree rooted at  $v$ . Clearly, nodes  $u$  and  $v$  can be equivalent only if  $size(u) = size(v)$ , but the converse is false. However, this observation allows us to bound the total cost of checking the subtree equivalences to yield the following.

LEMMA 18. *The problem of determining all equivalent nodes of a tree is in  $NC^1(n^2)$ , where  $n$  is the size of the tree.*

*Proof.* The general idea is to apply Lemma 17 in parallel to all pairs of nodes  $u, v$  for which  $size(u) = size(v)$ . However, we must be careful, as a brute force analysis leads to a processor bound of  $O(n^3)$ . For each  $i, 1 \leq i \leq n$ , let  $n_i$  denote the number of nodes of size  $i$ . By Lemma 17, the number of processors sufficient to check equivalence of all pairs  $u, v$  of nodes of size  $i$  is  $in_i^2$ . Thus, the total number of processors needed to find all equivalent nodes is  $\sum_i in_i^2$ .

Now,  $\sum_i n_i \leq n$ , and for each  $i, in_i \leq n$ . Multiplying each side of the last inequality by  $n_i$  yields  $in_i^2 \leq nn_i$ . Putting all this together we obtain

$$\sum_i in_i^2 \leq \sum_i nn_i \leq n \sum_i n_i \leq n^2. \quad \square$$

Applying the algorithm of Lemma 18 and then using the approach of Lemma 15 proves that testing whether an arbitrary dag matches a tree is in  $NC^2(n^2)$ . In fact we can do better in terms of time.

**THEOREM 19.** *Term matching for a general dag  $G$  and a constant tree  $H$  can be done in  $\text{NC}^1(n^2)$ .*

*Proof.* We only have to argue for  $O(\log n)$  instead of  $O(\log^2 n)$  parallel time.

In the case that  $H$  is a tree, the only part of the algorithm which uses time  $\log^2 n$  is the embedding of the spanning tree  $T$  in the tree  $H$ . Since  $T$  and  $H$  are both trees, the embedding can be done in time  $O(\log n)$  as follows.

Construct the product graph  $Z$  whose nodes are all pairs  $(u, v)$  such that  $u$  is a node of  $T$  and  $v$  is a node of  $H$ . There is an edge labeled  $i$  directed from  $(u, v)$  to  $(u', v')$  if there is an edge labeled  $i$  from  $u$  to  $u'$  in  $T$  and an edge labeled  $i$  from  $v$  to  $v'$  in  $H$ . Since  $T$  and  $H$  are both rooted trees, it is easy to see that  $Z$  is a forest of rooted trees. Letting  $r_1$  and  $r_2$  be the roots of  $T$  and  $H$ , respectively, we want to find all nodes of  $Z$  which are reachable from  $(r_1, r_2)$  since the embedding maps  $u$  to  $v$  if and only if  $(u, v)$  is reachable from  $(r_1, r_2)$ . Since  $Z$  contains at most  $m = n^2$  nodes this can be done in  $\text{NC}^1(n^2)$  as follows.

We first transform  $Z$  from a forest of trees to a single tree by creating a new root and creating an edge from the new root to each of the roots in  $Z$ . For simplicity, let  $Z$  denote the resulting tree. We then compute a depth-first numbering of  $Z$  and  $\text{size}(a)$  for each node  $a$  in  $Z$ . By Lemma 13 these tasks can be accomplished in  $\text{NC}^1(m)$ . Let  $a$  be a node of  $Z$  and let  $k$  be the depth-first number of  $a$ . Then the descendants of  $a$  are those nodes with depth-first numbers  $k, \dots, k + \text{size}(a) - 1$ .  $\square$

Verma, Krishnaprasad, and Ramakrishnan [17] have recently shown that term matching of two trees is in  $\text{NC}^2(n)$  (whereas  $\text{NC}^1(n^2)$  is a corollary of Theorem 19).

**3.3. Another special case of unification in NC.** In this section, we examine unification of two terms in the special case that the terms share no variables and at least one of the terms is linear. In performing unification on term dags we need a way of representing the result. In general, two terms are unifiable if and only if a certain type of equivalence relation can be constructed on the nodes of the labeled dag representing these terms. Given this relation, we can define the *reduced* graph, obtained by coalescing all equivalent nodes into a single node. We can extract a unifier  $\sigma$  from the reduced graph by taking  $\sigma(x)$  to be the term in the reduced graph that is represented by the node formed from the equivalence class of  $x$ .

A relation  $R$  on the nodes of a term dag is a *correspondence* relation if for all pairs of nodes  $u, v$  in the dag

$$uRv \Rightarrow u_iRv_i,$$

where  $u_i$  and  $v_i$  are corresponding children, respectively, of  $u$  and  $v$ . A correspondence relation that is also an equivalence relation will be called a *c-e relation*.

A relation  $R$  on the nodes of a term dag is *homogeneous* if for all pairs of nodes  $u, v$  which are labeled by function symbols we have

$$uRv \Rightarrow \text{label}(u) = \text{label}(v).$$

An equivalence relation  $R$  on the nodes of a dag is *acyclic* if the  $R$ -equivalence classes are partially ordered by the arcs of the dag. Paterson and Wegman [12] have shown that if  $u$  and  $v$  are nodes of a labeled dag  $G$  then the terms represented by the subdags with roots  $u$  and  $v$ , respectively, are unifiable if and only if there exists an acyclic homogeneous c-e relation  $R$  on the nodes of the dags satisfying  $uRv$ . Since we are considering the special case where one of the terms is linear and the two terms do not have any variables in common, it is easy to see that cyclic c-e relations cannot arise, so we do not mention the acyclicity condition further.

A substitution  $\sigma$  is *more general* than a substitution  $\tau$  if there exists a substitution  $\rho$  with  $\tau = \rho \circ \sigma$ . If  $R$  is the minimal c-e relation with  $uRv$  then the unifying substitution obtained from the reduced graph defined by  $R$  as described above is the most general unifier. The size of the most general unifier is the number of nodes in the reduced graph obtained by coalescing nodes equivalent under  $R$ , where  $R$  is minimal.

LEMMA 20. *Unification of two linear terms represented by trees with no shared variables can be solved in  $NC^1(n^2)$ . Moreover, the most general unifier is linear and has size at most the sum of the sizes of the original trees.*

*Proof.* Let  $T_1$  and  $T_2$  be trees representing linear terms with no shared variables, and  $n$  be the size of the instance. As in the proof of Theorem 19, we construct the product graph  $Z$  whose nodes are all pairs of the form  $(u, v)$ , where  $u$  is a node of  $T_1$  and  $v$  is a node of  $T_2$ . There is an edge labeled  $i$  directed from  $(u, v)$  to  $(u', v')$  in  $Z$  if there are edges labeled  $i$  from  $u$  to  $u'$  in  $T_1$  and from  $v$  to  $v'$  in  $T_2$ . As in the proof of Theorem 19, letting  $r_1$  and  $r_2$  denote the roots of the two trees, respectively, we find all nodes in the product graph reachable from  $(r_1, r_2)$ . We do not repeat the details here. Let  $R$  be the relation on nodes of  $T_1 \cup T_2$  defined by:  $uRv$  if and only if  $(u, v)$  is reachable in  $Z$  from  $(r_1, r_2)$ . For all pairs of internal nodes  $u, v$  we see that  $uRv \Rightarrow u_i R v_i$ , where  $u_i$  and  $v_i$  are corresponding children of  $u$  and  $v$ , respectively. It is also clear that because each node in the union of the original trees is related to at most one other node,  $R$  is an equivalence relation (trivially). We note that  $R$  is the minimal c-e relation in which the two roots are related. Once we have constructed  $Z$  and determined  $R$ , we can easily check  $R$  for homogeneity. If so, then as shown in [12], the two trees are unifiable. We construct the reduced graph and, from it, the most general unifier, as described in the beginning of this section.

Because  $Z$  contains at most  $n^2$  nodes, this can all be done in  $NC^1(n^2)$ . Further, because the reduced graph contains no more nodes than the union of  $T_1$  and  $T_2$ , the most general unifier has size at most the sum of the sizes of the two trees. Finally, the most general unifier is linear since it is obtained by a substitution which maps each variable to a linear term such that different variables are mapped to linear terms involving disjoint sets of variables.  $\square$

Using Lemma 20 we can now prove the main result of this section.

THEOREM 21. *Let  $T$  be a tree representing a linear term and  $H$  be an arbitrary rooted dag sharing no variables with  $T$ . Then the unification problem for  $T$  and  $H$  can be solved in  $NC^2(n^2)$ .*

*Sketch of Proof.* Without loss of generality we assume that for each variable  $x$  there is at most one node of  $H$  labeled with  $x$ . Let  $r_1$  and  $r_2$  be the roots of  $T$  and  $H$ , respectively. Again we are searching for the minimal c-e relation  $R$  on the nodes of  $T \cup H$  such that  $r_1 R r_2$ . We first modify the recursive embedding technique of Lemma 14 to handle the embedding of a tree in a dag when the dag is not necessarily constant, as in the present case. Thus, there could be some nodes of  $T$  that cannot be embedded in  $H$  because some path in  $H$  ends with a node labeled by a variable while the corresponding path in  $T$  continues. As in the case of Lemma 14, the modified algorithm is in  $NC^2(n^2)$ . We define  $R$  to be the reflexive transitive closure of  $R'$ , where  $uR'v$  if  $u$  is mapped to  $v$  by the embedding. Check  $R$  for homogeneity.

If  $uRv$  where  $u$  is a node of  $T$  and  $v$  is a node of  $H$ , and  $u$  is labeled by a variable  $x$ , then we define  $\sigma(x)$  to be the term represented by the subdag rooted at  $v$ . Because there is a unique path to a node labeled  $x$  in  $T$  there is nothing to check. More interesting is the case when there exist several nodes  $u_1, \dots, u_k$  in  $T$  and a node  $v$  in  $H$  labeled with a variable  $y$  such that  $u_i R v$ ,  $i = 1, \dots, k$ . This can happen if there are  $k$  paths to  $v$  in  $H$ . In this case we must unify all  $k$  trees rooted at the  $u_i$ . However,



because  $T$  is linear these subtrees share no variables, so we can apply Lemma 20. In order to perform all the unifications quickly in parallel, we split the subtrees into at most  $k/2$  pairs, and unify all pairs in parallel. By Lemma 20 the size of the most general unifier for each pair is no larger than the sum of the sizes of the original trees. We can therefore apply the lemma recursively on the  $\leq \lceil k/2 \rceil$  remaining trees. Because  $k \leq n$  and each application of Lemma 20 can be performed in  $\text{NC}^1(n^2)$  the entire procedure is in  $\text{NC}^2(n^2)$ .  $\square$

*Remark.* By modifications to the proofs of Lemma 20 and Theorem 21, it is easy to see that unification of a linear term with an arbitrary term can be done in NC even if both terms are represented by general dags. An outline of the algorithm follows. Let  $G$  and  $H$  be the given dags with roots  $r_1$  and  $r_2$ , respectively, where  $G$  represents a linear term and  $H$  represents an arbitrary term. Form the product graph  $Z$  as in the proof of Lemma 20, and solve a reachability problem (in  $\text{NC}^2(M(n^2))$ ) to find all nodes of  $Z$  reachable from  $(r_1, r_2)$ . Let  $R$  be the reflexive transitive closure of the relation  $R'$  defined by  $uR'v$  if and only if  $(u, v)$  is reachable from  $(r_1, r_2)$ . Proceeding as in the proof of Theorem 21, the only difference is the case where there are several nodes  $u_1, \dots, u_k$  in  $G$  and a node  $v$  in  $H$  labeled with a variable such that  $u_i R v$  for all  $i$ . Now the  $u_i$  are roots of subdags which represent linear terms; as before, these subdags share no variables. By again solving a reachability problem on a product graph, the unification problem for two linear terms which do not share variables and which are represented by general dags can be solved in NC. As before, the most general unifier is linear and its size is at most the size of the union of the two dags (nodes which appear in both dags are counted only once in the union).

**4. Unification of linear terms is complete for P.** Recall that a term is linear if no variable appears more than once in the term.

**THEOREM 22.** *Unification is P-complete even if both terms are linear, are represented by trees, and have all function symbols with arity  $\leq 2$ .*

*Proof.* The proof is by a reduction from the circuit value problem (CVP) which was proved P-complete by Ladner [8]. Because of the nature of our reduction, it is useful to require that instances of CVP be in a particular form described next. An instance of CVP is a dag whose nodes are of four types. An *input node* has no edges directed in and one edge directed out; this edge is called an *input edge*. An *output node* has one edge directed in and no edges directed out; each dag has exactly one output node and the edge directed into this node is called the *output edge*. A *NAND node* has two edges directed in and one edge directed out. A *fan-out node* has one edge directed in and any nonzero number of edges directed out. In addition, each input edge is labeled with a Boolean value, either 0 (false) or 1 (true). Given the assignments of Boolean values to the input edges, Boolean values are associated with all the other edges in the obvious way: the value of the edge directed out of a NAND-node is the Boolean NAND of the values of the two edges directed in; the value of all edges directed out of a fan-out node is the same as the value of the edge directed in. The problem CVP is to recognize the set of instances such that the output edge has value 1. (Although Ladner's proof of the P-completeness of CVP uses Boolean functions other than NAND, any such function can be built from a small number of NANDs and Boolean constants, so CVP as defined above is P-complete.)

Given an instance  $G$  of CVP, we transform it to a pair of linear tree terms,  $T_1$  and  $T_2$ , with roots  $r_1$  and  $r_2$ , respectively. For simplicity, we let the two trees have function symbols with arities greater than 2. The trees can then be further transformed by replacing each subterm  $f^{(k)}(t_1, t_2, \dots, t_k)$  involving a  $k$ -ary function symbol by

the term  $f(t_1, f(t_2, \dots, f(t_{k-1}, t_k)) \dots)$  involving the 2-ary function symbol  $f$ . In addition to the two roots,  $T_1$  has nodes  $A_e$  and  $B_e$  and  $T_2$  has nodes  $C_e$  and  $D_e$  for each edge  $e$  of  $G$ . Unifications among these four nodes encode the Boolean value of  $e$  as follows (in this proof we indicate a unification between two nodes by writing  $\sim$  between them): if  $e$  has value 0, then  $A_e \sim D_e$  and  $B_e \sim C_e$ ; if  $e$  has value 1, then  $A_e \sim C_e$  and  $B_e \sim D_e$ . The appropriate unifications of nodes corresponding to input edges are forced by making these nodes be corresponding children of the roots. For example, if  $e$  is the  $i$ th input edge and if  $e$  is assigned value 1, then there is an edge labeled  $2i-1$  from  $r_1$  to  $A_e$ , an edge labeled  $2i-1$  from  $r_2$  to  $C_e$ , an edge labeled  $2i$  from  $r_1$  to  $B_e$ , and an edge labeled  $2i$  from  $r_2$  to  $D_e$ .

For each fan-out node and NAND node of  $G$ , edges and nodes are added to the trees to force the unifications encoding Boolean values to be propagated correctly. For each fan-out node of  $G$ , if the node has edge  $e$  directed in and edges  $e_1, \dots, e_k$  directed out, then for each  $i$  with  $1 \leq i \leq k$ , there is an edge labeled  $i$  from  $A_e$  to  $A_{e_i}$ , from  $B_e$  to  $B_{e_i}$ , from  $C_e$  to  $C_{e_i}$ , and from  $D_e$  to  $D_{e_i}$ . For each NAND node, with edges  $e'$  and  $e''$  directed in and edge  $e$  directed out, the trees contain the nodes and edges shown in Fig. 3 (to simplify notation,  $A'$  is written for  $A_e$  etc.).

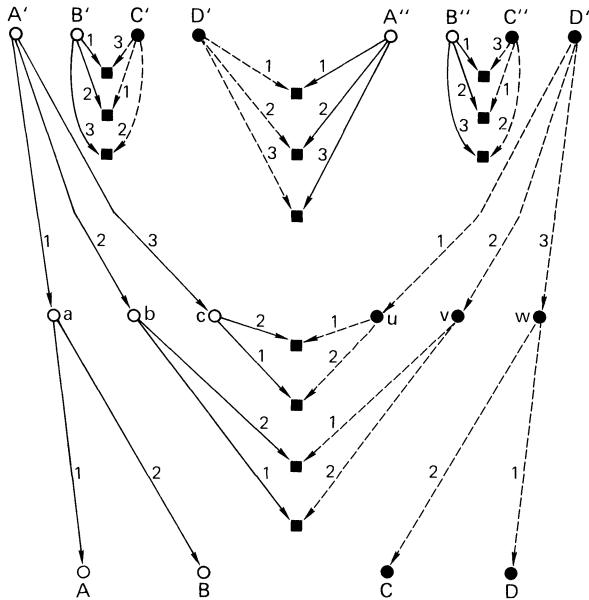


FIG. 3. The transformation of a NAND node used in the proof of Theorem 22. Nodes of  $T_1$  (resp.,  $T_2$ ) are drawn as open circles (resp., solid circles). Edges of  $T_1$  (resp.,  $T_2$ ) are drawn as solid lines (resp., dashed lines). Common leaves which are labeled by variables are drawn as solid squares.

We must also define the labeling of tree nodes. Each nonleaf node is labeled by the function symbol  $f^{(k)}$  where  $k$  is the outdegree of the node. The leaves which are drawn as squares in Fig. 3 are each labeled by a different variable symbol. If  $p$  is the output edge of  $G$ , then  $A_p$  and  $C_p$  are labeled with the constant symbol  $g$ , and  $B_p$  and  $D_p$  are labeled with a different constant symbol  $h$ . This completes the description of the transformation.

We must argue that the output edge of  $G$  has value 1 if and only if  $T_1$  and  $T_2$  are unifiable. To do this it is sufficient to show that the transformations of fan-out nodes and NAND nodes correctly propagate Boolean values according to the encoding

of values by unifications described above. For fan-out nodes this is obvious. To verify this for the NAND construction in Fig. 3, it is helpful to break the construction into two parts. The first part, which consists of the part of the figure above the nodes marked  $a, b, c, u, v, w$  (including these six nodes), computes the numerical sum of the two input values, viewing these values as integers rather than Boolean values. The three possible sums, 0, 1, and 2, are encoded by unifications among  $a, b, c, u, v, w$ . It is easy to check that the four possible values for  $e'$  and  $e''$  force unifications among  $a, b, c, u, v, w$  as follows:

$$\begin{aligned} e' = 0 \text{ and } e'' = 0 & \text{ implies } a \sim u, b \sim v, c \sim w, \\ e' = 0 \text{ and } e'' = 1 & \text{ implies } a \sim v, b \sim w, c \sim u, \\ e' = 1 \text{ and } e'' = 0 & \text{ implies } a \sim v, b \sim w, c \sim u, \\ e' = 1 \text{ and } e'' = 1 & \text{ implies } a \sim w, b \sim u, c \sim v. \end{aligned}$$

We can now forget about the top half of Fig. 3 and just check that these three possible unifications among  $a, b, c, u, v, w$  force the proper unifications among  $A, B, C, D$ . The verification of the following is again straightforward:

$$\begin{aligned} a \sim u, b \sim v, c \sim w & \text{ implies } A \sim C \text{ and } B \sim D (e = 1), \\ a \sim v, b \sim w, c \sim u & \text{ implies } A \sim C \text{ and } B \sim D (e = 1), \\ a \sim w, b \sim u, c \sim v & \text{ implies } A \sim D \text{ and } B \sim C (e = 0). \end{aligned}$$

If the output edge  $p$  of  $G$  has value 0, then an attempt to unify the roots  $r_1$  and  $r_2$  will force the unification of two nodes  $A_p$  and  $D_p$  labeled with different constant symbols. On the other hand, if the output edge has value 1, then the two roots can be unified.  $\square$

In the case that the two terms do not share any variables, we have noted in § 3.3 that unification can be solved in NC if one of the terms is linear. The following easy corollary of Theorem 22 shows that this is in some sense the best possible, since if both terms are barely nonlinear the problem becomes P-complete.

**THEOREM 23.** *Unification is P-complete even if both terms are represented by trees, no variable appears in both terms, each variable appears at most twice in some term and all function symbols have arity  $\leq 2$ .*

*Proof.* As in the previous proof, we allow function symbols with large arities. The proof is by a reduction from the unification problem for linear trees. Let  $T_1$  and  $T_2$  be a given pair of trees representing linear terms, and let  $x_1, x_2, \dots, x_m$  be the variables which appear in both trees. For each  $i$  with  $1 \leq i \leq m$ , replace the single occurrence of  $x_i$  in  $T_1$  by a new variable  $x_{i1}$  and replace the single occurrence of  $x_i$  in  $T_2$  by  $x_{i2}$ . For each  $i$ , we can force  $x_{i1}$  and  $x_{i2}$  to be equal by increasing the arity of the roots from 2 to  $2+m$ , adding a new edge labeled  $i+2$  from the root of  $T_1$  to a new node labeled  $x_{i1}$ , and adding a new edge labeled  $i+2$  from the root of  $T_2$  to a new node labeled  $x_{i2}$ . Clearly, the transformed trees are unifiable if and only if  $T_1$  and  $T_2$  are unifiable.  $\square$

#### REFERENCES

- [1] S. A. AMITSUR AND J. LEVITZKI, *Minimal identities for algebras*, Proc. Amer. Math. Soc., 1 (1950), pp. 449-463.
- [2] W. F. CLOCKSIN AND C. S. MELLISH, *Programming in Prolog*, Springer-Verlag, New York, Berlin, 1981.
- [3] D. COPPERSMITH AND S. WINOGRAD, *On the asymptotic complexity of matrix multiplication*, SIAM J. Comput., 11 (1982), pp. 472-492.

- [4] C. DWORK, P. C. KANELLAKIS, AND J. C. MITCHELL, *On the sequential nature of unification*, J. Logic Programming, 1 (1984), pp. 35–50.
- [5] S. FORTUNE AND J. WYLLIE, *Parallelism in random access machines*, Proc. 10th ACM Symposium on Theory of Computing, 1978, pp. 114–118.
- [6] G. HUET AND D. OPPEN, *Equations and rewrite rules: a survey*, in Formal Language Theory: Perspectives and Open Problems, R. V. Book, ed., Academic Press, New York, 1980.
- [7] R. KOWALSKI, *Predicate logic as a programming language*, Proc. IFIPS, 74 (1974), pp. 569–574.
- [8] R. LADNER, *The circuit value problem is log space complete for P*, SIGACT News, 7 (1975), pp. 18–20.
- [9] J. MALUSZYNSKI AND H. J. KOMOROWSKI, *Unification-free execution of horn-clause programs*, Proc. 2nd IEEE Logic Programming Symposium, 1985, pp. 78–86.
- [10] R. MILNER, *A theory of type polymorphism in programming*, J. Comput. System Sci., 17 (1978), pp. 348–375.
- [11] V. PAN AND J. REIF, *Efficient parallel solution of linear systems*, Proc. 17th Annual ACM Symposium on Theory of Computing, 1985, pp. 143–152.
- [12] M. S. PATERSON AND M. N. WEGMAN, *Linear unification*, J. Comput. System Sci., 16 (1978), pp. 158–167.
- [13] M. O. RABIN, *Probabilistic algorithm for testing primality*, J. Number Theory, 12 (1980), pp. 128–138.
- [14] J. A. ROBINSON, *A machine oriented logic based on the resolution principle*, J. Assoc. Comput. Mach., 12 (1965), pp. 23–41.
- [15] J. T. SCHWARTZ, *Fast probabilistic algorithms for verification of polynomial identities*, J. Assoc. Comput. Mach., 27 (1980), pp. 701–717.
- [16] R. E. TARJAN AND U. VISHKIN, *An efficient parallel biconnectivity algorithm*, SIAM J. Comput., 14 (1985), pp. 862–874.
- [17] R. M. VERMA, T. KRISHNAPRASAD, AND I. V. RAMAKRISHNAN, *An efficient parallel algorithm for term matching*, Dept. of Computer Science, State University of New York, Stony Brook, New York, 1986, manuscript.
- [18] H. YASUURA, *On the parallel computational complexity of unification*, Res. Report ER 83-01, Yajima Laboratories, October 1983.

## THE PROBABILISTIC ANALYSIS OF A HEURISTIC FOR THE ASSIGNMENT PROBLEM\*

DAVID AVIS† AND C. W. LAI‡

**Abstract.** We present a heuristic to solve the  $m \times m$  assignment problem in  $O(m^2)$  time. The assignment problem is formulated as a weighted complete bipartite graph  $G = (S, T, E)$ ,  $|S| = |T| = m$ . For convenience we assume that  $m$  is even. The main procedure in the heuristic is to construct a graph  $G_d = (S, T, E_d)$  which is a subgraph of  $G$ ,  $|E_d| = 4dn$ ,  $n = m/2$ , such that we can find a perfect matching in  $G_d$  with probability at least  $1 - \frac{1}{3}(d/n)^{d^2-4d-1}$ . An  $O(|S||E|)$  exact algorithm is used to find a minimum weight matching  $M$  in  $G_d$ . Any unmatched vertices in  $G$  relative to  $M$  are then matched by a greedy algorithm. The expected value of the total cost of the matching found by the heuristic is shown to be less than six if the costs are independent and identically distributed uniformly in the unit interval. Further, with the above probability, the heuristic produces a solution which is at most six times the optimal solution.

**Key words.** assignment problem, heuristics, probabilistic analysis of algorithms, weighted matchings

**AMS(MOS) subject classifications.** 68Q25, 68R05

**1. Introduction.** Let  $G = (S, T, E)$  be the complete bipartite graph with vertex sets  $S$  and  $T$ , each of cardinality  $m$ , and  $E$  the set of edges between  $S$  and  $T$ . Associated with each edge  $(i, j)$  there is a weight  $c_{i,j}$ . A matching  $M$  in  $G$  is a set of edges such that no two edges in  $M$  are incident to the same vertex of  $G$ . The matching is perfect if each vertex in  $G$  is adjacent to some edge in  $M$ . A minimum weight perfect matching for  $G$  is a perfect matching for which the sum of the edge weights is minimum. The assignment problem is to find such a minimum weight matching.

There are exact algorithms to solve the assignment problem in  $O(m^3)$  time [10]. Karp [7] has found an algorithm that finds an optimal solution in expected time  $O(m^2 \log m)$ . Several heuristics exist for the assignment problem [8], [6], [1], [2]. A survey of heuristics for the weighted matching problem and some applications is contained in [3]. In order to evaluate the solutions produced by these heuristics, it is natural to ask what is the expected size of the optimal solution, given a probability distribution for the edge weights. If the edge weights are independently distributed on the unit interval, Walkup [12] has proved that this expected value is bounded above by the constant 3, for all  $m$ , a very remarkable result. The idea behind Walkup's proof is the construction of a sparse subgraph of  $G$  in which the edges all have low weight. He defines a subgraph  $G_d$ , for some small integer  $d$ , by choosing  $d$  edges randomly from each vertex in  $S$  and  $T$ . In [13] it is shown that with probability approaching one as  $m$  tends to infinity, this graph has a perfect matching. The natural candidate for such a subgraph would be to choose for each vertex, the  $d$  smallest weight edges that are incident with it. Unfortunately this will not work, since the edges chosen in such a way will not be independent. Consider, for example, the minimum weight edge in  $G$ . It will be chosen twice, once by a vertex in  $S$  and once by a vertex in  $T$ . Walkup avoids this problem by using a probabilistic trick: each edge is replaced by two directed edges with opposite directions. The weights on the directed edges are chosen from

---

\* Received by the editors April 15, 1985; accepted for publication (in revised form) November 9, 1987. This work was supported by the Natural Sciences and Engineering Research Council of Canada under grant A3013.

† School of Computer Science, McGill University, Montreal, Quebec, Canada H3A 2K6.

‡ Present address, G.P.O. 2706, Hong Kong.

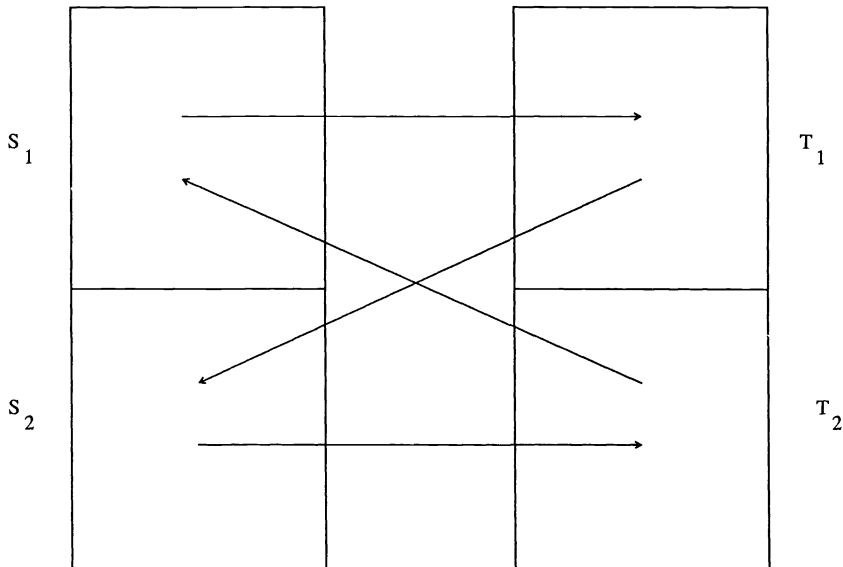
distributions such that the minimum of the two edge weights still has uniform distribution on the unit interval. This is sufficient for proving the result stated, but does not provide a constructive method that can be used to produce a heuristic.

In this paper, we avoid the dependency problem by constructing a different kind of random graph. We prove analogues of Walkup's results for this new graph. It is shown to almost surely have a perfect matching, and it is shown how to construct this graph directly from the given edge weights. We prove that the perfect matching thus found has expected weight at most 6. The heuristic runs in  $O(m^2)$  time and produces a solution which, with probability approaching one, is at most 6 times the optimum value. This is the only known heuristic with a constant ratio bound, with probability approaching one. In § 2 the heuristic is presented along with a complexity analysis. In § 3 we show that the random graph constructed by the heuristic has a perfect matching with probability approaching one. This is followed in § 4 by a proof that the expected weight of the matching produced is less than 6. The results given in this paper were first published in the second author's Master's thesis [9]. This thesis also contains extensive numerical results.

**2. The heuristic.** For convenience we assume that  $m$  is even and set  $n = m/2$ . The heuristic is supplied with an integer parameter  $d$  which must be at least 5. This parameter controls the quality of the solution and the cost of the computation. In practice, setting  $d = 5$  appears to be most satisfactory. The algorithm consists of three steps: the construction of a sparse directed random graph  $G_d$  with  $4dn$  edges; the solution of the maximum cardinality minimum weight matching problem in the undirected graph obtained by ignoring the directions on edges of  $G_d$ ; and the matching of any unmatched vertices in  $G_d$  using a greedy heuristic.

*Step 1.* (Construction of a directed graph  $G_d = (S_1 \cup S_2, T_1 \cup T_2, E_d)$  with  $4dn$  edges; see Fig. 1.)

Partition  $S$  and  $T$  into subsets  $S_1, S_2$  and  $T_1, T_2$ , each of cardinality  $n$ .



—————> Indicates Allowable Edges

FIG. 1. Illustrating the construction of  $G_d$ .

For  $v \in S_i$ ,  $i = 1, 2$

$N(v) = \{w \in T_i \mid c_{v,w} \text{ is one of the } d \text{ smallest edge weights from } T_i \text{ incident } v\}$ .

For  $w \in T_i$ ,  $i = 1, 2$

$N(w) = \{v \in S_{3-i} \mid c_{v,w} \text{ is one of the } d \text{ smallest edge weights from } S_{3-i} \text{ incident } w\}$ .

Set  $E_d = \{(v, w) \mid v \in S_i \cup T_i, w \in N(v), i = 1, 2\}$ .

Step 2. (Find a minimum weight matching in  $G_d$ .)

Apply an  $O(|S||E_d|)$  minimum weight matching algorithm to  $G_d$ , ignoring the directions on the edges. Let  $M_d$  be the resulting matching.

Step 3. (Match the unmatched vertices in  $G_d$  by a greedy algorithm.)

$$S^* = \{\text{unmatched vertices in } S \text{ relative } M_d\},$$

$$T^* = \{\text{unmatched vertices in } T \text{ relative } M_d\}.$$

**while**  $|S^*| \geq 1$  **do**

Choose a vertex  $s \in S^*$  and find a vertex  $t \in T^*$  such that

$$c_{s,t} = \min \{c_{s,v} \mid v \in T^*\}, \quad S^* = S^* - s, \quad T^* = T^* - t, \quad M_d = M_d \cup (s, t).$$

**end.**

Step 1 involves at most  $O(2d|S|)$  operations, Step 2 needs  $O(|S||E_d|) = O(|S|^2)$  operations and Step 3 requires at most  $O(|S|^2)$  operations. The memory space required by  $G$  and  $G_d$  is  $O(|S|^2)$  and  $O(2d|S|)$ , respectively. There are exact algorithms for Step 2 that require  $O(|S|^2)$  space. Therefore the overall time and space requirements for the algorithm are  $O(m^2)$ .

**3. Probability of a perfect matching in  $G_d$ .** In this section, we show that with probability approaching one, the graph  $G_d = (S, T, E_d)$  constructed by the heuristic contains a perfect matching. It is convenient to consider the following random graph model for a random  $2n \times 2n$  directed bipartite graph with vertex sets  $(S_1, S_2; T_1, T_2)$ ,  $|S_i| = |T_i| = n$ ,  $i = 1, 2$ . For  $i = 1, 2$  and each vertex  $v \in S_i$ , select  $d$  neighbours at random in  $T_i$ . For  $i = 1, 2$  and each vertex  $w \in T_i$ , select  $d$  neighbours at random in  $S_{3-i}$ . A graph constructed in such a way will arise with equal probability by an application of Step 1 of the heuristic. This follows from the fact that selecting the  $d$  smallest from a set of  $n$  independent and identically distributed random variables gives a uniform random subset of  $d$  of the random variables. In this section, we deal with the equivalent formulation mentioned above which does not involve edge weights. Finally, in § 4 we reintroduce the edge weights to prove our main result. We will apply the Konig-Hall Theorem (see, for example, Bondy and Murty [4]), to show that  $G_d$  almost surely has a perfect matching. We require the following definitions.

Let  $k$  be a positive integer, satisfying  $d + 1 \leq k \leq 2n - d$ , let  $A$  be a subset of  $T$  with cardinality  $k - 1$  and let  $B$  be a subset of  $S$  with cardinality  $k$ . Set

$$A_i = T_i \cap A, \quad a_i = |A_i|, \quad i = 1, 2,$$

$$B_i = S_i \cap B, \quad b_i = |B_i|, \quad i = 1, 2.$$

For any subset  $U$  of vertices, let

$$\Gamma(U) = \{w : (v, w) \in E_d \text{ for some } v \in U\}.$$

We say that  $(A, B)$  is a *blocking  $k$ -pair* in  $G_d$  if the following hold (see Fig. 2):

$$\Gamma(B_i) \subseteq A_i, \quad \Gamma(T_i - A_i) \subseteq S_{3-i} - B_{3-i}, \quad i = 1, 2.$$

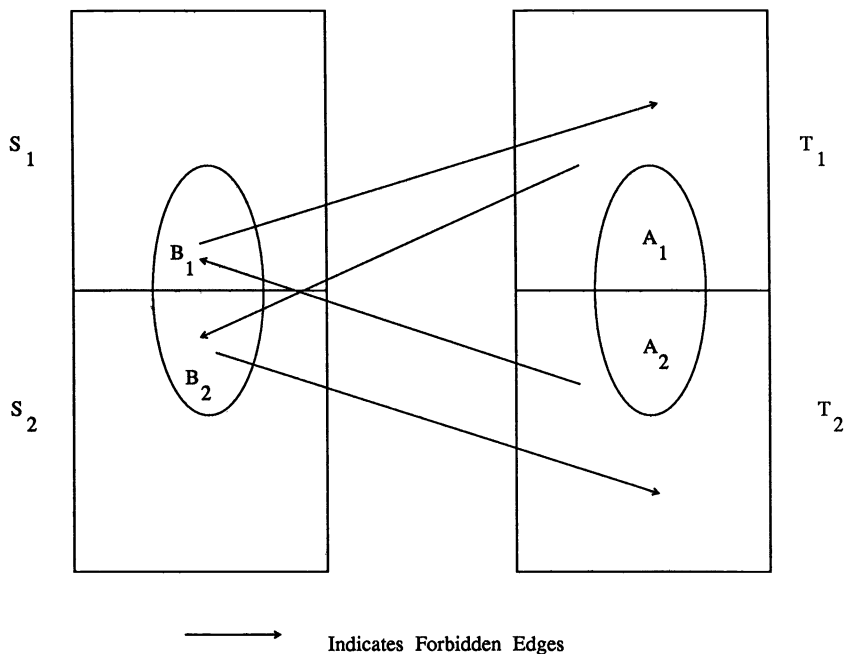


FIG. 2. Illustrating a blocking  $k$ -pair in  $G_d$ .

It can be seen from the construction that certain values of  $a_i$  place restrictions on the corresponding values of  $b_i$ . In particular:

- (i) If  $a_i < d$  then  $b_i = 0$ ;
- (ii) If  $a_i = 0$  then  $d + 1 \leq k \leq n - d$ ;
- (iii) If  $a_1, a_2 \geq d$  then  $b_1, b_2 \leq n - d$ ; and
- (iv)  $a_1 + a_2 = k - 1, b_1 + b_2 = k$ .

We say that the 4-tuple is *valid* if it is consistent with (i)-(iv) above.

By the Konig-Hall Theorem,  $G_d$  has a perfect matching if and only if it has no blocking  $k$ -pair. Let  $P_{a_1, b_1}^{(d)}(k)$  be the probability that  $(A, B)$  is a blocking  $k$ -pair in  $G_d$ . This notation is justified by observing that the probability of the algorithm creating a blocking  $k$ -pair  $(A, B)$  is determined only by the cardinalities  $a_1, a_2, b_1, b_2$ . Since  $a_2$  and  $b_2$  are determined once  $a_1$  and  $b_1$  are fixed, we omit them as subscripts of  $P$ . Then if  $a_1, b_1$  satisfy the above conditions for a blocking  $k$ -pair,

$$(1) \quad P_{a_1, b_1}^{(d)}(k) = \frac{\binom{a_1}{d}}{\binom{n}{d}}^{b_1} \frac{\binom{a_2}{a}}{\binom{n}{d}}^{b_2} \frac{\binom{n-b_2}{d}}{\binom{n}{d}}^{n-a_1} \frac{\binom{n-b_1}{d}}{\binom{n}{d}}^{n-a_2} .$$

We denote by  $\beta^{(d)}(k)$  the expected number of blocking  $k$ -pairs. Then we have

$$(2) \quad \beta^{(d)}(k) = \sum_{b_1+b_2=k} \sum_{a_1+a_2=k-1} \binom{n}{a_1} \binom{n}{a_2} \binom{n}{b_1} \binom{n}{b_2} P_{a_1, b_1}^{(d)}(k).$$

Our main result is the following. Let  $\beta^{(d)}$  denote the expected number of blocking pairs in  $G_d$ .

**THEOREM 1.** For  $n \geq 2d \geq 10$

$$\beta^{(d)} < \frac{1}{3} \left( \frac{d}{n} \right)^{d^2-4d-1} .$$



The theorem proves the claim that  $G_d$  almost certainly has a perfect matching, since by the Markov inequality,

$$P\{G_d \text{ has no perfect matching}\} \leq \beta^{(d)}.$$

In order to prove this result we need a series of combinatorial lemmas.

LEMMA 1 (see [11]). For positive integers  $k, m$  with  $k < m$ ,

(a) 
$$\binom{m}{k} \leq \frac{1}{\sqrt{2\pi k}} \left(\frac{em}{k}\right)^k,$$

(b) 
$$\binom{m}{k} \leq \frac{m^m}{k^k(m-k)^{(m-k)}}. \quad \square$$

LEMMA 2.

$$\binom{p}{j} / \binom{q}{j} \leq \left(\frac{p}{q}\right)^j, \quad 0 \leq j \leq p \leq q. \quad \square$$

LEMMA 3. For  $2 \leq d+1 \leq k < n$ , every valid 4-tuple  $(a_1, a_2, b_1, b_2)$  satisfies:

(a) 
$$\left(\frac{a_1}{n}\right)^{b_1} \left(\frac{a_2}{n}\right)^{b_2} \leq \left(\frac{k-1}{n}\right)^k;$$

(b) 
$$\left(\frac{n-b_2}{n}\right)^{n-a_1} \left(\frac{n-b_1}{n}\right)^{n-a_2} \leq \left(\frac{n-k}{n}\right)^{n-k}.$$

(c) 
$$\left(\frac{a_1}{n}\right)^{b_1} \left(\frac{a_2}{n}\right)^{b_2} \left(\frac{n-b_2}{n}\right)^{n-a_1} \left(\frac{n-b_1}{n}\right)^{n-a_2} \leq \left(\frac{d}{n}\right)^d.$$

(d) If, in addition,  $k \leq n-d$  then

$$\left(\frac{k-1}{n}\right)^k \left(\frac{n-k}{n}\right)^{n-k} \leq \left(\frac{d}{n}\right)^d.$$

*Proof.* The various parts of the lemma are proved in more or less the same way. We just outline their proof, leaving the details to the interested reader.

(a) This is an immediate consequence of the arithmetic-geometric mean inequality (see [11, p. 76]).

(b) Set  $a = a_2, b = b_1, a_1 = k - a - 1$ , and  $b_2 = k - b$ . We first assume that  $a_1, a_2 \geq d$ . Then, using elementary calculus,

$$\begin{aligned} n^{-1}(n-b_1)^{n-a_2}(n-b_2)^{n-a_1} &\leq (n-b)^{n-a}(n-k+b)^{n-k+a} \\ &\leq (n-a)^{n-a}(n-k+a)^{n-k+a} \\ &\leq (n-d)^{n-d}(n-k+d)^{n-k+d} \\ &\leq n^n(n-k)^{n-k}. \end{aligned}$$

Now, suppose that  $a_1 < d$ . Since  $a_1, a_2, b_1, b_2$  is valid,  $b_1 = 0$ ; hence  $b_2 = k$ . Then the left-hand side of  $b$  reduces to

$$\left(\frac{n-k}{n}\right)^{n-a_1} \leq \left(\frac{n-k}{n}\right)^{n-d+1} \leq \left(\frac{n-k}{n}\right)^{n-k}$$

as required. The case  $a_2 < d$  is symmetric.

(c) Let  $h$  denote the left-hand side. Then from the third inequality in the proof of (b), part (a), and elementary calculus, we obtain

$$h \leq \left(\frac{k}{n}\right)^k \left(\frac{n-k+d}{n}\right)^{n-k+d} \leq \left(\frac{d}{n}\right)^d.$$

(d) The proof follows from elementary calculus.  $\square$

The following rather technical lemma will be required in the proof of Lemma 6, but can be skipped without loss of continuity.

LEMMA 4. *If  $1 \leq n/4 \leq k/2 \leq a < k-1$  and  $a < b \leq k \leq n$ , then*

$$h = a^{b-a-1}(k-1-a)^{a-b+1}(n-b)^{(a+b)-k}(n-k+b)^{k-(a+b)} \leq 1.$$

*Proof.* Using standard inequalities for logarithms [11] we obtain

$$\frac{1}{2} \log \left(\frac{a}{k-1-a}\right) \leq \frac{2a-k+1}{k-1} + \frac{1}{3} \left(\frac{2a-k+1}{k-1}\right)^3 / \left(1 - \left(\frac{2a-k+1}{k-1}\right)^2\right),$$

and

$$\frac{1}{2} \log \left(\frac{n-k+b}{n-b}\right) \leq \frac{2b-k}{2n-k}.$$

An elementary calculation shows that

$$(b-a-1) \frac{2a-k+1}{k-1} - (a+b-k) \frac{2b-k}{2n-k} \leq -\frac{(2a-k+1)(2a-k+2)}{3k}.$$

Therefore,

$$\begin{aligned} \frac{1}{2} \log h &\leq \frac{1}{2} \left[ (b-a-1) \log \left(\frac{a}{k-1-a}\right) - (a+b-k) \log \left(\frac{n-k+b}{n-b}\right) \right] \\ &\leq (b-a-1) \left[ \frac{2a-k+1}{k-1} + \frac{1}{3} \left(\frac{2a-k+1}{k-1}\right)^3 / \left(1 - \left(\frac{2a-k+1}{k-1}\right)^2\right) \right] \\ &\quad - (a+b-k) \frac{2b-k}{2n-k} \\ &\leq -\frac{(2a-k+1)(2a-k+2)}{3k} + \frac{(b-a-1)}{3} \frac{(2a-k+1)^2(2a-k+2)}{2ak(2k-2a-2)} \\ &\leq \frac{(2a-k+1)(2a-k+2)}{3k} \left( -1 + \frac{(b-a-1)(2a-k+1)}{4a(k-a-1)} \right) \\ &\leq 0, \end{aligned}$$

since  $2a-k+1 \geq 1$ ,  $k-a-1 \geq b-a-1$  and  $2a > 2a-k+1$ . Therefore  $h \leq 1$ , proving the lemma.  $\square$

We are now ready to proceed with the evaluation of (1). By Lemma 2,

$$P_{a_1, b_1}^{(d)}(k) \leq \left[ \frac{a_1^{b_1} a_2^{b_2} (n-b_2)^{n-a_1} (n-b_1)^{n-a_2}}{n^{2n+1}} \right]^d, \quad a_1, a_2 > 0.$$

Applying Lemma 3(c) repeatedly, we obtain the following set of inequalities parameterized by  $i$ :

$$(3) \quad P_{a_1, b_1}^{(d)}(k) \leq \left(\frac{d}{n}\right)^{d(d-i)} \left[ \frac{a_1^{b_1} a_2^{b_2} (n-b_2)^{n-a_1} (n-b_1)^{n-a_2}}{n^{2n+1}} \right]^i, \\ a_1, a_2 > 0, \quad i = 0, 1, \dots, d.$$

The following bound holds for all  $a_1, a_2$ . If  $a_1 = 0$  or  $a_2 = 0$  it follows from Lemmas 2 and 3(d). Otherwise it follows from Lemma 3(a), (b), (c):

$$(4) \quad P_{a_1, b_1}^{(d)}(k) \leq \left(\frac{d}{n}\right)^{d(d-i)} \left(\frac{k}{n}\right)^{ik} \left(\frac{n-k}{n}\right)^{i(n-k)}, \quad i = 0, 1, \dots, d.$$

We are now able to derive upper bounds for the expected number of blocking  $k$ -pairs. We divide the proof into two parts, depending on whether  $k$  is less than or greater than  $n/2$ .

LEMMA 5. *If  $5 \leq d \leq k \leq n/2$  then*

$$\beta^{(d)}(k) \leq \frac{1}{2\pi k} \left(\frac{d}{n}\right)^{d(d-4)}.$$

*Proof.* From (1) and (4) with  $i = 4$  we obtain

$$\beta^{(d)}(k) \left(\frac{n}{d}\right)^{d(d-4)} \leq \left(\frac{k}{n}\right)^{4k} \left(\frac{n-k}{n}\right)^{4(n-k)} \sum_{b_1+b_2=k} \sum_{a_1+a_2=k-1} \binom{n}{a_1} \binom{n}{a_2} \binom{n}{b_1} \binom{n}{b_2}.$$

Now an elementary calculation shows that

$$\sum_{a_1+a_2=k-1} \binom{n}{a_1} \binom{n}{a_2} \leq \binom{2n}{k-1}$$

and

$$\sum_{b_1+b_2=k} \binom{n}{b_1} \binom{n}{b_2} \leq \binom{2n}{k}.$$

Therefore,

$$(5) \quad \begin{aligned} \beta^{(d)}(k) \left(\frac{n}{d}\right)^{d(d-4)} &\leq \left(\frac{k}{n}\right)^{4k} \left(\frac{n-k}{n}\right)^{4(n-k)} \binom{2n}{k}^2 \frac{k}{2n-k+1} \\ &\leq \left(\frac{k}{n}\right)^{4k} \left(\frac{n-k}{n}\right)^{4(n-k)} \left[ \frac{1}{\sqrt{2\pi k}} \left(\frac{2en}{k}\right)^k \right]^2 \\ &\leq \left(\frac{k}{n}\right)^{2k} \left(1 - \frac{k}{n}\right)^{4(n-n/2)} \frac{(2e)^{2k}}{2\pi k} \\ &\leq 2^{-2k} e^{-2k} \frac{(2e)^{2k}}{2\pi k} \\ &= \frac{1}{2\pi k}. \end{aligned}$$

Inequality (5) follows from Lemma (1a). This completes the proof of Lemma 5.  $\square$

LEMMA 6. *If  $5 \leq d \leq n/2 \leq k \leq n$  then*

$$\beta^{(d)}(k) \leq \left(\frac{k}{e}\right)^2 \left(\frac{d}{n}\right)^{d(d-2)}.$$

*Proof.* At first we assume that  $a_1 > 0, a_2 > 0$ , and  $0 < b_1 \leq b_2 < n$ . Let

$$f = \frac{a_1^{b_1} a_2^{b_2} (n-b_2)^{(n-a_1)} (n-b_1)^{(n-a_2)}}{b_1^{b_1} b_2^{b_2} (n-a_1)^{(n-a_1)} (n-a_2)^{(n-a_2)}}$$

and

$$(6) \quad g = \frac{a_1^{b_1} a_2^{b_2} (n-b_2)^{(n-a_1)} (n-b_1)^{(n-a_2)}}{a_1^{a_1} a_2^{a_2} (n-b_2)^{(n-b_2)} (n-b_1)^{(n-b_1)}} \frac{1}{n^2}.$$

Applying Lemma 1(b) and (3) with  $i = 2$ , we see that

$$\binom{n}{a_1} \binom{n}{a_2} \binom{n}{b_1} \binom{n}{b_2} P_{a_1, b_1}^{(d)}(k) \leq \left(\frac{d}{n}\right)^{d(d-2)} fg.$$

We will show that  $f \leq e^{-2}$  and  $g \leq 1$ .

The bound for  $f$  is obtained by applying the inequality  $(1 + y/x)^x \leq e^y$  to the expression. Therefore,

$$f = \left(1 - \frac{b_1 - a_1}{b_1}\right)^{b_1} \left(1 - \frac{b_2 - a_2}{b_2}\right)^{b_2} \left(1 - \frac{b_2 - a_1}{n - a_1}\right)^{n - a_1} \left(1 - \frac{b_1 - a_2}{n - a_2}\right)^{n - a_2} \leq e^{-2},$$

as  $b_1 + b_2 = k$  and  $a_1 + a_2 = k - 1$ .

The bound for  $g$  is lengthier. We begin by eliminating two variables from (6). Set  $a_2 = a$ ,  $a_1 = k - 1 - a$ ,  $b_2 = b$ ,  $b_1 = k - b$ . Then we have

$$(7) \quad g = \left(\frac{k - 1 - a}{a}\right)^{a - b + 1} \left(\frac{n - b}{n - k + b}\right)^{a + b - k} \frac{a(n - b)}{n^2}.$$

By our initial assumptions  $d \leq a \leq k - d - 1$ ,  $k/2 \leq b \leq k < n$ . We consider two main cases.

*Case 1.*  $a \geq (k - 1)/2$ . Since  $a$  and  $b$  are integers,  $a + b \geq k$ . Clearly

$$\frac{k - 1 - a}{a} \leq 1 \quad \text{and} \quad \frac{n - b}{n - k + b} \leq 1.$$

If  $a \geq b$  then the first exponent of (7) is also greater than one and we are done. If  $a = (k - 1)/2$  we are also done because the first term is equal to one. The remaining case,  $a \geq k/2$ ,  $b > k/2$ , is settled by Lemma 4. Hence  $g \leq 1$ .

*Case 2.*  $a < (k - 1)/2$ . In this case

$$\frac{k - 1 - a}{a} > 1, \quad \frac{n - b}{n - k + b} \leq 1, \quad \text{and} \quad a - b + 1 \leq 0.$$

If in addition,  $a + b - k \geq 0$  or  $b = k/2$  we are done immediately. Otherwise we have

$$a - b + 1 \leq a + b - k \quad \text{and} \quad b - a \leq k - 2a - 1.$$

Therefore, from (7) we obtain

$$g \leq \left(\frac{k - 1 - a}{a} \frac{n - b}{n - k + b}\right)^{a + b - k}.$$

A simple calculation shows that

$$(k - 1 - a)(n - b) - a(n - k + b) = (k - 2a - 1)n + b - (b - a)k \geq 0.$$

Hence,  $g \leq 1$ .

We now show that the initial assumptions can be lifted. We first observe that  $f$  and  $g$  are symmetric in the subscripts 1 and 2. Hence the above results hold if  $b_1 > b_2$ . Next, consider the degenerate case when  $a_1 = 0$  and hence  $a_2 = k - 1$ ,  $b_1 = 0$ ,  $b_2 = k$ . The case  $a_2 = 0$  is identical. A simple argument based on Lemma 1(b) and equation (4) with  $i = 2$  shows that

$$\binom{n}{k - 1} \binom{n}{k} P_{0,0}^{(d)}(k) \leq \frac{1}{2} \left(\frac{d}{n}\right)^{d(d-2)}.$$

The lemma now follows from the definition of  $\beta^{(d)}(k)$  in equation (2) and the observation that the double sum involves at most  $k^2 - k$  terms.  $\square$

We may now prove the main result of this section.

*Proof of Theorem 1.* Suppose  $(A, B)$  is a blocking  $k$ -pair with  $|B| = k > n$ . Then  $(S - B, T - A)$  is a blocking  $(2n - k + 1)$ -pair in the graph  $G_d^* = (T, S, E_d)$ , and  $2n - k + 1 \leq n$ . Therefore, the expected number of blocking pairs is bounded by

$$\begin{aligned} \beta^{(d)} &\leq 2 \left\{ \sum_{k=d+1}^{n/2} \beta^{(d)}(k) + \sum_{k=n/2}^n \beta^{(d)}(k) \right\} \\ &\leq 2 \left\{ \sum_{k=d+1}^{n/2} \left(\frac{d}{n}\right)^{d(d-4)} \frac{1}{2\pi k} + \sum_{k=n/2}^n \frac{k^2}{e^2} \left(\frac{d}{n}\right)^{d(d-2)} \right\} \\ &\leq 2 \left\{ \left(\frac{d}{n}\right)^{d(d-4)} \frac{n}{4\pi d} + \left(\frac{d}{n}\right)^{d(d-2)} \frac{n^3}{2e^2} \right\} \\ &\leq 2 \left(\frac{d}{n}\right)^{d^2-4d-1} \left\{ \frac{1}{4\pi} + \left(\frac{d}{n}\right)^{2d+1} \frac{1}{e^2} \frac{n^3}{2} \right\} \\ &\leq 2 \left(\frac{d}{n}\right)^{d^2-4d-1} \left( \frac{1}{4\pi} + \frac{1}{2e^2} \right). \end{aligned}$$

This completes the proof of the theorem.  $\square$

**4. Expected value of the heuristic solution.** In this section, we derive an upper bound on the expected value of the heuristic solution to the assignment problem described in § 2. Suppose initially that  $G_d$  contains a perfect matching  $M_d$ . We estimate the expected weight of a randomly chosen edge from  $G_d$ . We first recall that the expected value of the  $r$ th-order statistic from  $n$  uniform independent random variables in  $[0, 1]$  is  $r/(n + 1)$  [5]. Therefore, the  $d$  directed edges selected from each vertex by the heuristic have expected weights  $1/(n + 1), 2/(n + 1), \dots, d/(n + 1)$ . An arbitrarily selected edge therefore has expected weight  $(d + 1)/2(n + 1)$ . If  $G_d$  has a perfect matching, then such a matching chosen at random without regard to edge weights will have expected weight  $n(d + 1)/(n + 1)$ . The minimum weight perfect matching in such a graph therefore has expected weight bounded above by this amount. Let  $Z_d$  be the cost of the matching found by the heuristic. Then we have

$$E(Z_d \mid G_d \text{ has a perfect matching}) \leq \frac{n(d + 1)}{n + 1}.$$

However,  $G_d$  may not have a perfect matching. In this case, we take the trivial upper bound of  $2n$  for  $Z_d$ . Hence,

$$\begin{aligned} E(Z_d) &\leq E(Z_d \mid G_d \text{ has a perfect matching}) \\ &\quad + E(Z_d \mid G_d \text{ has no perfect matching})P(G_d \text{ has no perfect matching}) \\ &\leq \frac{n(d + 1)}{n + 1} + \frac{2n}{3} \left(\frac{d}{n}\right)^{d^2-4d-1}, \end{aligned}$$

by Theorem 1. Setting  $d = 5$  we obtain

$$\begin{aligned} E(Z_d) &\leq \frac{6n}{n + 1} + \frac{2n}{3} \left(\frac{5}{n}\right)^4 \\ &\leq 6 \quad \text{if } n \geq 11. \end{aligned}$$

Since  $G_d$  is a subgraph of  $G_{d+1}$ , any matching in  $G_d$  exists also in  $G_{d+1}$ . Therefore,

$$E(Z_{d+1}) \leq E(Z_d).$$

Thus, we have proved the following theorem.

**THEOREM 2.** *If  $n \geq 11$ , and  $d \geq 5$  then  $E(Z_d) \leq 6$ .*

We remark that the bound holds for  $d = 5$  if the heuristic is modified to simply find any maximum cardinality matching in Step 2, rather than one of minimum weight. Such a heuristic is considerably simpler to implement.

**Acknowledgment.** We thank an anonymous referee for a careful reading resulting in, we hope, many clarifications.

#### REFERENCES

- [1] D. AVIS, *Two greedy heuristics for the weighted matching problem*, Congr. Numer. XXI; also in Proc. 9th Southeast Conference on Combinatorics, Graph Theory and Computing, Boca Raton, 1978.
- [2] D. AVIS AND L. DEVROYE, *An analysis of a decomposition heuristic for the assignment problem*, Oper. Res. Lett., 3 (1985), pp. 279-283.
- [3] D. AVIS, *A survey of heuristics for the weighted matching problem*, Networks, 13 (1983), pp. 475-493.
- [4] A. BONDY AND U. S. R. MURTY, *Graph Theory with Applications*, American Elsevier, New York, 1976.
- [5] H. A. DAVID, *Order Statistics*, John Wiley, New York, 1970.
- [6] W. E. DONATH, *Algorithms and average value bounds for assignment problems*, IBM J. Res. Develop. 8 (1969), pp. 380-386.
- [7] R. M. KARP, *An algorithm to solve the  $m \times n$  assignment problem in expected time  $O(mn \log n)$* , Networks, 10 (1980), pp. 143-152.
- [8] J. M. KURTZBERG, *On approximate methods for the assignment problem*, J. Assoc. Comput. Mach., 9 (1962), pp. 419-439.
- [9] C. W. LAI, *A heuristic for the assignment problem and related bounds*, SOCS 81.21, School of Computer Science, McGill University, Montreal, Quebec, Canada, 1981.
- [10] E. L. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.
- [11] D. S. MITRINOVIC, *Analytic Inequalities*, Springer-Verlag, Heidelberg, New York, Berlin, 1970.
- [12] D. W. WALKUP, *On the expected value of a random assignment problem*, SIAM J. Comput., 7 (1979), pp. 440-442.
- [13] ———, *Matching in random regular bipartite graphs*, Discrete Math., 31 (1980), pp. 59-64.

## THE STRUCTURE OF THE STABLE ROOMMATE PROBLEM: EFFICIENT REPRESENTATION AND ENUMERATION OF ALL STABLE ASSIGNMENTS\*

DAN GUSFIELD†

**Abstract.** The stable *roommates* problem is a well-known problem of matching  $2n$  people into disjoint pairs to achieve a certain type of stability. The problem strictly generalizes the better-known stable *marriage* problem. It has been previously shown [Irving and Leather, *SIAM J. Comput.*, 15 (1986), pp. 655–667], that the set of stable marriages, for a given instance, can be compactly represented and this representation has been exploited to yield a number of very efficient algorithms concerned with stable marriage [Irving, Leather, and Gusfield, *J. Assoc. Comput. Mach.*, 34 (1987), pp. 532–543], [Gusfield, *SIAM J. Comput.*, 16 (1987), pp. 111–128], [Gusfield et al., *J. Combin. Theory, Ser. A.*, (1987), pp. 304–309]. In this paper, we generalize the structure of the stable marriages to obtain two efficiently computed, small, implicit representations of the set of all stable roommate assignments, for any given instance. One representation is a partial order  $\Pi$  on  $O(n^2)$  elements such that the stable assignments are in one-one correspondence with certain easily recognized subsets of  $\Pi$ . Partial order  $\Pi$  is a strict generalization of the stable marriage representation of [Irving and Leather, *SIAM J. Comput.*, 15 (1986), pp. 655–667]. The second representation is an efficiently constructed, undirected graph  $G$  with  $O(n^2)$  nodes, such that there exists a one-one correspondence between the *maximal* (not maximum) independent sets of  $G$  and the stable roommate assignments. In either representation,  $G$  or  $\Pi$ , given a set representing a stable assignment, the assignment itself can be constructed in  $O(n^2)$  time. We also give an algorithm to generate each stable assignment for any given instance in  $O(n^2)$  time per assignment. The efficiency of this method depends heavily on special properties of the stable assignment problem developed in this paper. Finally, we give a succinct characterization of the set of all “stable pairs,” those pairs of people who are roommates in at least one stable assignment, and we give an  $O(n^3 \log n)$  time algorithm to find them all.

**Key words.** stable roommates, stable marriage, matching, enumeration, combinatorial algorithm, partial order

**AMS(MOS) subject classifications.** 90C27, 68R10, 68Q25, 06A10

**1. Introduction.** The stable *roommates* problem is a well-known problem of matching  $2n$  people into disjoint pairs to achieve a certain type of stability. The input to the problem is a set of  $2n$  preference lists, one for each person  $i$ , where person  $i$ 's list is a rank ordering (most preferred first) of the  $2n - 1$  people other than  $i$ . A roommate assignment  $A$  is a pairing of the  $2n$  people into  $n$  disjoint pairs. Assignment  $A$  is said to be *unstable* if there are two people who are not paired together in  $A$ , but who each prefer the other to their respective mates in  $A$ ; such a pair is said to *block* assignment  $A$ . An assignment which is not unstable is called *stable*. An instance of the stable roommates problem is called *solvable* if there is at least one stable assignment. It is known [GS], [L], [K], [PTW] that there are unsolvable instances of the stable roommate problem; the problem of finding an efficient algorithm to determine if an instance is solvable was proposed by Knuth [K] and only recently solved by Irving [I].

The stable roommates problem is closely related to, and is a strict generalization of, another well-known problem, the stable *marriage* problem. In the stable marriage problem, the  $2n$  people consist of  $n$  men and  $n$  women, and each pair is constrained to consist of a man and a woman. Each man ranks only the women and each woman

---

\* Received by the editors December 31, 1986; accepted for publication (in revised form) August 3, 1987. This research was funded in part by Office of Naval Research grant N00012-82-K-0184 and National Science Foundation grant MCS-81/05894.

† Department of Electrical and Computer Engineering, Division of Computer Science, University of California, Davis, California 95616.

ranks only the men, and an assignment in this problem is called a *marriage* (from here on, the word “assignment” will be used only for roommate assignment). A marriage  $M$  is unstable if there is a man and a woman who are not married to each other in  $M$ , but who mutually prefer each other to their respective mates in  $M$ . It is easy to reduce an instance of the stable marriage problem to an instance of the stable roommates problem. However, in contrast to the stable roommates problem, it is well known [GS] that every instance of the stable marriage problem is solvable, i.e., has at least one stable marriage, and for any instance, one stable marriage is easy to find.

For a given instance of the stable marriage problem, there may be many distinct stable marriages, and as a function of  $n$ , the numbers of stable marriages can grow exponentially [K]. Despite this exponential growth, for any given problem instance, there exists an extremely small, efficiently computed, implicit representation of the set of all stable marriages for the instance, where any particular stable marriage can be extracted from the representation very quickly. The structure of the stable marriages which gives rise to this implicit representation was first made explicit by Irving and Leather [IL], and, as pointed out in [GILS], can also be seen via a more algebraic approach based on the theory of distributive lattices. This efficiently constructed representation of the set of all stable marriages was exploited in [IL], [ILG], [G], [GILS] to solve a number of problems related to stable marriage. For example, for any given instance of the stable marriage problem, the implicit representation of all the stable marriages can be constructed in  $O(n^2)$  time, and thereafter, each stable marriage can be generated from the representation in  $O(n)$  time per marriage [G], which is time optimal.

In this paper, we generalize the structure of the stable marriages to obtain two efficiently computed, small, implicit representations of the set of all stable roommate assignments, for any given instance. One representation is a partial order  $\Pi$  on  $O(n^2)$  elements such that the stable assignments are in one-one correspondence with certain easily recognized subsets of  $\Pi$ . Partial order  $\Pi$  is a strict generalization of the stable marriage representation given in [IL] and [GILS]. The second representation is an efficiently constructed, undirected graph  $G$  with  $O(n^2)$  nodes, such that there exists a one-one correspondence between the *maximal* (not maximum) independent sets of  $G$  and the stable roommate assignments. In either representation,  $G$  or  $\Pi$ , given a set representing a stable assignment, the assignment itself can be constructed in  $O(n^2)$  time. We then give an algorithm to generate each stable assignment for any given instance in  $O(n^2)$  time per assignment. This compares favorably to the  $O(n^3)$  time, per marriage, method given by Knuth [K] to generate all stable *marriages*. The faster method given in this paper does not follow immediately from the representations above, since the fastest known time to generate maximal independent sets in a general graph on  $O(n^2)$  nodes runs in  $O(n^4)$  time per independent set, and methods based on general partial orders appear even less efficient. Finally, we give a succinct characterization of the set of all “stable pairs,” those pairs of people who are roommates in at least one stable assignment, and we give an  $O(n^3 \log n)$  time Algorithm to find all the stable pairs.

The results and algorithms in this paper are obtained by close examination of Irving’s algorithm [I] which finds *one* stable assignment if there is one, or reports that no stable assignment exists. Hence we will begin by describing Algorithm 1.

## 2. Algorithm I and its execution tree $D$ .

**2.1. Algorithm I.** Algorithm I successively deletes entries from preference lists until either each person has only one entry on its list, or until someone has no entries.



In the first case, the entries specify a stable roommate assignment, and in the second case, there are no stable assignments; the algorithm runs in time  $O(n^2)$ . The algorithm is divided into two phases. In Phase 1, entries are removed from lists, but no stable assignments are affected, i.e., if  $j$  is removed from  $i$ 's list, then  $(i, j)$  is a pair in no stable assignment. In Phase 2, the removed entries may affect stable assignments, but the invariant is maintained at each iteration, that *if* there is a stable assignment in the lists before the current iteration of removals, *then* there is a stable assignment in the lists resulting from the iteration of removals. Hence if any list becomes empty in either phase, there can be no stable assignment. Before describing the algorithm, we need the following definitions.

DEFINITION. The current set of lists at any point in the algorithm is called a *table*.

DEFINITION. Let  $e_i$  denote a person. At any point in the algorithm,  $h_i$  will denote the current head of person  $e_i$ 's list, and  $s_i$  will denote the current second entry on  $e_i$ 's list.

DEFINITION. At any point in Algorithm I, a person  $e_i$  is said to be *semi-engaged* to  $h_i$  if and only if  $e_i$  is the bottom entry on  $h_i$ 's list. A person who is not semi-engaged is called *free*. A person may alternate between being free and semi-engaged.

Note that semi-engagement is not a symmetric relation. However, if everyone is semi-engaged, then it follows easily that the set of list heads is a permutation of the  $2n$  people. We now describe Algorithm I.

PHASE 1 of Algorithm I iterates the following:

1. If there is an empty list, then terminate Algorithm I; there is no stable assignment.
2. Else, if everyone is semi-engaged, then go to Phase 2.
3. Else, pick an arbitrary free person  $e_i$ , and execute the following operations for each person  $k$  who is ranked below  $e_i$  on  $h_i$ 's list in  $T$ : remove  $k$  from  $h_i$ 's list, and remove  $h_i$  from  $k$ 's list.

Note that throughout Phase 1, person  $i$  is on  $j$ 's list if and only if  $j$  is on  $i$ 's list. Hence in a step where  $e_i$  becomes semi-engaged to  $h_i$ , if there is a person  $p$  who is semi-engaged to  $h_i$  just before that step, then  $p$  is (automatically) not semi-engaged to  $h_i$  after that step. This follows since at the start of the step,  $p$  must be below  $e_i$  on  $h_i$ 's list, so during that step,  $p$  is removed from  $h_i$ 's list, and  $h_i$  is removed from  $p$ 's list. After the step,  $p$  might be free, or it might have become (automatically) semi-engaged to the new head of its list.

The set of lists at the end of Phase 1 is called the *Phase 1 table*. It is proved in [I] that if  $j$  is missing from  $i$ 's list in the Phase 1 table, then there are no stable assignments which pair  $i$  to  $j$ . Hence if some list in the Phase 1 table is empty, there are no stable assignments. Otherwise, when Phase 1 terminates with everyone semi-engaged,  $j$  is the head of  $i$ 's list if and only if  $i$  is the bottom of  $j$ 's list, and so the set of head entries of the Phase 1 table are a permutation of the  $2n$  people.

Figure 1(a) gives an instance of the stable roommate problem, and 1(b) shows the three stable assignments for the table. Figure 2(a) shows the Phase 1 table for the example.

PHASE 2. Throughout Phase 2 all the people remain semi-engaged, although who they are semi-engaged to may change. Hence at any point in Phase 2,  $j$  is the head of  $i$ 's list if and only if  $i$  is the bottom of  $j$ 's list. It will also be true that  $i$  is on  $j$ 's list if and only if  $j$  is on  $i$ 's list. Phase 2 starts with the Phase 1 table and removes entries from lists in a way similar to Phase 1, but the selection of lists is more constrained. We first need some definitions.

1	7	2	6	8	5	3	4
2	4	6	5	3	8	1	7
3	5	2	1	7	4	6	8
4	1	7	3	6	5	8	2
5	7	1	8	4	6	2	3
6	7	3	8	4	5	1	2
7	2	8	4	3	5	6	1
8	4	2	3	5	6	7	1

FIG. 1(a). 8-person preference lists.

(1,6)	(1,5)	(1,4)
(2,3)	(2,3)	(2,3)
(7,4)	(7,4)	(7,5)
(8,5)	(8,6)	(8,6)

FIG. 1(b). Three stable roommate assignments.

1	2	6	5	3	4		
2	6	5	3	8	1		
3	5	2	1	7	4	6	
4	1	7	3	6	5	8	
5	7	1	8	4	6	2	3
6	3	8	4	5	1	2	
7	8	4	3	5			
8	4	2	5	6	7		

FIG. 2(a). Phase 1 table.

$E_1$	$H_1$	$S_1$	$E_2$	$H_2$	$S_2$
1	2	6	4	1	7
2	6	5	5	7	1
3	5	2			

FIG. 2(b). Rotation  $R_1 = (E_1, H_1, S_1)$ , Rotation  $R_2 = (E_2, H_2, S_2)$  both exposed in the Phase 1 table. In the Phase 1 table 6, 7, 8 is a tail of  $R_1$ , and  $R_2$  has no tail.

DEFINITION. In a table  $T$ , an *exposed rotation*  $R$  is an ordered subset of people  $E = \{e_1, e_2, \dots, e_r\}$ , such that  $s_i = h_{i+1}$ , for all  $i$  from 1 to  $r$ , where  $i + 1$  is taken modulo  $r$ . Note that since the order of  $E$  is cyclic, the actual selection of which element in  $E$  is named  $e_1$  is arbitrary, but that selection determines the rest of the ordering.

Figure 2(b) shows two rotations that are exposed in the Phase 1 table of the running example.

We will often write " $R = (E, H, S)$ ," where  $H$  is the set of head entries of  $E$  ordered to correspond to the order of  $E$ , and  $S$  is the set of second entries of  $E$ , with corresponding order. Note that, as sets,  $S = H$ , and that, as ordered sets,  $S$  is a (backwards) cyclic rotation of  $H$ ; when that point is central, we will write  $S = H'$ . We will sometimes say that " $e$  is in  $R$ " to mean that  $e$  is the  $E$  set of  $R$ ; we will also say that " $(e, h)$  is a pair in  $R$ " to mean that  $e = e_i$  and  $h = h_i$  for some  $e_i$  in  $E$ .

DEFINITION. If  $R = (E, H, S)$  is an exposed rotation in table  $T$ , then the *elimination* of  $R$  from  $T$  is the following operation: for every  $s_i$  in  $S$ , remove every entry below  $e_i$  in  $s_i$ 's list in  $T$ , i.e., move the bottom of  $s_i$ 's list up to  $e_i$  (from  $e_{i+1}$ ). Then remove  $s_i$  from  $k$ 's list, for each person  $k$  who was just removed from  $s_i$ 's list.

Notice that if all people are semi-engaged in a table  $T$  before a rotation elimination, then all people are semi-engaged after that elimination; hence, one effect of the elimination is to move the head of  $e_i$ 's list down one place, for each  $e_i$  in  $R$ , i.e.,  $e_i$  becomes semi-engaged to the  $s_i$  of table  $T$ . Figure 2(c) shows the table resulting from eliminating  $R_1$  from the Phase 1 table.

1	6	5	3	4				
2	5	3						
3	2	1	7	4	6			
4	1	7	3	6	5	8		
5	7	1	8	4	6	2		
6	3	8	4	5	1			
7	8	4	3	5				
8	4	5	6	7				

FIG. 2(c). Table after eliminating  $R_1$  from the Phase 1 table. Note that  $R_2$  is still exposed, and now has a tail of 3. Rotation  $R_3 = (E_3, H_3, S_3)$ , where  $E_3 = \{2, 6, 7, 8\}$ , is now also exposed.

PHASE 2 of the algorithm is simply:

1. While some person has more than one entry on his list, and no list is empty, find and eliminate a rotation.
2. If every person has exactly one entry on his list, then pairing each person with their head entry specifies a stable assignment.
3. If there is an empty list, then there are no stable assignments.

Figure 2(d) completes the execution of Phase 2, eliminating rotations  $R_3$  and  $R_4$ .

1	6	5						
2	5	3						
3	2	4	6					
4	7	3	6	5	8			
5	1	8	4	6	2			
6	3	8	4	5	1			
7	8	4						
8	4	5	6	7				

FIG. 2(d). Table after eliminating  $R_2$ .  $R_3$  is the only exposed rotation.

1	6	5
2	3	
3	2	
4	7	
5	1	8
6	8	1
7	4	
8	5	6

Table after elimination of  $R_3$ . Now  $R_4, R_5$  are exposed, where  $E_4 = \{1, 8\}$   $E_5 = \{5, 8\}$ .

1	5
2	3
3	2
4	7
5	1
6	8
7	4
8	6

Table after elimination of  $R_4$ .

**2.2. Correctness of Algorithm I.** The correctness of Algorithm I is proved in [I], and will not be fully repeated here. However, we need the statements of the central lemmas that prove correctness, and we need to extend some of them; we will give proofs of the extended lemmas.

**DEFINITION.** If  $T$  is a table, then roommate assignment  $A$  is said to be *contained in*, or *in*,  $T$ , if and only if every pair in  $A$  is in  $T$ , i.e.,  $i$  is on  $j$ 's list, and  $j$  is on  $i$ 's list for each pair  $(i, j)$  in  $A$ .

The following lemmas imply the correctness of Algorithm I.

**LEMMA 2.1 [I].** *If  $T$  is a table (in Phase 2) where no list is empty, and at least one person has more than one entry, then there is a rotation exposed in  $T$ .*

Lemma 2.1 will be proved and extended in the next section.

**LEMMA 2.2 [I].** *Let  $R = (E, H, S)$  be an exposed rotation in  $T$ , and let  $A$  be any stable assignment contained in  $T$ . If  $e_i \in E$  and  $(e_i, h_i)$  is a pair in  $A$ , then  $(e_i, h_i)$  must also be a pair in  $R$ , for every  $e_i$  in  $E$ .*

Lemma 2.2 will be proved and extended in the next section.

**LEMMA 2.3 [I].** *If rotation  $R = (E, H, S)$  is exposed in  $T$ , and there exists a stable assignment in  $T$  where  $e_1 \in E$  pairs with  $h_1$ , then there also exists a stable assignment in  $T$  where  $e_1$  does not pair with  $h_1$ , and by Lemma 2.2, no  $e_i$  pairs with  $h_i$ , for any  $e_i$  in  $E$ .*

**LEMMA 2.4 [I].** *If the algorithm ends with a single entry on each list, then pairing each person to that entry gives a stable assignment.*

**2.3. Extensions of the central lemmas.** We will prove Lemmas 2.1 and 2.2 in order to extend them.

*Proof of Lemma 2.1.* Let  $e_i$  be a person who has at least two entries,  $h_i$  and  $s_i$ , on its list in  $T$ . Since the head entries are a permutation of the people, and  $s_i \neq h_i$ , there must be a person  $e_j$  such that  $s_i = h_j$ . We claim that  $e_j$  must have two or more entries on its list. If not, then  $h_j$  is its only entry, and so  $e_j$  is the only entry on  $h_j$ 's list. To see this, note that  $h_j$  is both the head and bottom entry on  $e_j$ 's list, so  $e_j$  must also be both the head and bottom of  $h_j$ 's list. But,  $h_j = s_i$  which is on  $e_i$ 's list, so  $e_i$  (which cannot be  $e_j$ ) must also be on  $h_j$ 's list, and so both  $h_j$  and  $e_j$  must have at least two entries on their lists. Repeating this argument, we must eventually cycle, in which case a rotation has been found.  $\square$

**DEFINITION.** The proof above gives an (implicit) algorithm for finding a rotation  $R$ , starting from any person  $e$  who has at least two entries on its list in  $T$ . Let  $e_1$  denote the person who is visited twice by the algorithm (i.e., where the cycle is detected). Every person who is visited before the first visit to  $e_1$  is said to be on a *tail* of  $R$ , and the other people are in the *body* of  $R$ .

Note that in a given table, an exposed rotation may have many tails, and in a different table, the same exposed rotation may have different tails. This is illustrated in the example of Fig. 2. We will need the following extension of Lemma 2.1.

**COROLLARY 2.1.** *If  $e$  is a person with two or more entries on its list in table  $T$ , then  $e$  is either in a tail or in the body of a rotation exposed in  $T$ .*

The following lemma extends Lemma 2.2.

**LEMMA 2.5.** *Let  $R$  be a rotation exposed in table  $T$ , and  $(e_i, h_i)$  a pair in  $R$ . If  $A$  is a stable assignment contained in  $T$  where  $e_i$  pairs with  $h_i$ , and if  $(e, h)$  is any pair in either the body of  $R$  or a tail of  $R$ , then  $(e, h)$  must be a pair in assignment  $A$ .*

*Proof.* Let  $e_j$  be a person either in  $R$  or in a tail of  $R$ . If  $e_k$  is any other person such that  $s_k = h_j$ , then in  $A$ ,  $e_k$  must pair with  $h_k$  if  $e_j$  pairs with  $h_j$ ; if not, then  $e_k$  must be paired with a person below  $s_k$  on its list, since  $h_k$  is already paired with  $e_j$ , and  $A$  is in  $T$ . But  $s_k$  is the head of  $e_j$ 's list in  $T$ , so  $e_j$  must be the bottom of  $s_k$ 's list, and

since  $s_k$  is on  $e_k$ 's list,  $e_k$  is on  $s_k$ 's list, and is preferred to  $e_j$  by  $s_k$ . Hence  $e_k$  and  $s_k$  would block  $A$ . It follows that if  $e_j$  is in the body of  $R$ , and if  $e_j$  pairs with  $h_j$  in  $A$ , then every  $e_i$  in  $R$  must pair with  $h_j$  in  $A$ . Now consider any tail of  $R$  (relative to  $T$ ). If  $(e, h, s)$  is the last triple of the tail, then  $s = h_i$  for some  $e_i$  in  $R$ , so  $(e, h)$  must be a pair in  $A$ , and the implication follows backwards along the tail. Hence in  $A$ , each person in the tail must also pair with the head person on their list in  $T$ .  $\square$

**2.4. The execution tree  $D$ .** Algorithm I is guaranteed to produce a stable assignment if there is one. However, in this paper we are concerned with the structure of the set of all the stable assignments for a particular instance; most of what we will deduce will be by examining the possible executions of Algorithm I. Hence we need the following theorem.

**THEOREM 2.1.** *If  $A$  is any stable roommate assignment, then there is an execution of Algorithm I which produces  $A$ .*

*Proof.* Let  $T$  be any table obtained from a (partial) execution of Algorithm I, where stable assignment  $A$  is in  $T$ . If in  $T$ , the head of each person's list is their partner in  $A$ , then, as in the proof of Lemma 2.1, each list has only a single entry, and so  $T$  is the final table of an execution of Algorithm I, and  $A$  is the resulting stable assignment. So, assume that there is a person  $p$  whose partner in  $A$  is not the head element of  $p$ 's list in  $T$ . Hence  $p$ 's list has at least two entries, and, by Corollary 2.1,  $p$  is either in the body or in a tail of a rotation  $R = (E, H, S)$  exposed in  $T$ . We claim that no person  $e_i$  in the body of  $R$  pairs with  $h_i$  in  $A$ . This follows directly from Lemma 2.5, since if  $(e_i, h_i)$  is a pair in  $A$ , then  $p$ 's partner in  $A$  must also be its head entry in  $T$ , contradicting the selection of  $p$ .

We will show that when  $R$  is eliminated from  $T$ , assignment  $A$  is still contained in the resulting table. The elimination of  $R$  from  $T$  can be viewed as a two-step process. First, the head of each element  $e_i$  in  $R$  is moved down one position to  $h_i$ . By the argument in the paragraph above, assignment  $A$  is in the table after these moves. Hence in  $A$ , each  $e_i$  in  $E$  must be paired with  $s_i$  or below in its list, and for the stability of  $A$ , it follows that each  $s_i$  in  $S$  must be paired with  $e_i$  or above, in its list. Hence  $A$  will be in the remaining table if, for each  $s_i$  in  $S$ , we remove all the elements below  $e_i$  in  $s_i$ 's list, and remove  $s_i$  from the lists of each of these elements. But these are exactly the elements that are removed when  $R$  is eliminated from  $T$ . Hence rotation  $R$  can be eliminated from  $T$ , creating a smaller table  $T'$  which still contains the stable assignment  $A$ . The theorem follows by repeating this argument until no rotations remain.  $\square$

**COROLLARY 2.2.** *Let  $R = (E, H, S)$  be an exposed rotation in table  $T$ , and let  $T'$  be the table after eliminating  $R$  from  $T$ . If  $e_i$  is any person in  $E$ , then  $T'$  contains all stable assignments that  $T$  contains, except for those assignments where  $e_i$  mates with  $h_i$ .*

**DEFINITION.** We use  $D$  to refer to the resulting execution tree, when, for a given Phase 1 table, Phase 2 of Algorithm I is executed in all possible ways. Each node  $x$  in  $D$  represents the table  $T(x)$ , which is the current state of the algorithm at node  $x$ . Each edge out of  $x$  is labeled with a rotation which is exposed in  $T(x)$ , and which is the next rotation eliminated from  $T(x)$  on that execution path out of  $x$ . We use  $D(x)$  to denote the subtree of  $D$  rooted at  $x$ .

Note that  $D$  is defined only for the Phase 2 executions. In the remainder of the paper, when we talk about Algorithm I, we will be referring to Phase 2, unless we specifically state otherwise.

*Naive enumeration of all stable assignments.* Given Theorem 2.1, we could generate all stable assignments by forcing all possible executions of Algorithm I. This would be simple to do, but would be terribly inefficient, as it would most often generate the

same stable assignment several times. However, we will show in this paper that an efficiently implemented modification of this naive approach generates each stable assignment exactly once, at a cost of  $O(n^2)$  time per assignment. Although the modification is simple, its proof of correctness and time is not, and most of this paper centers on developing the needed tools for the proof. We will return to the enumeration problem after examining the structure of  $D$ , and the rotations in the next several sections.

**3. Basic lemmas.** In this section, we develop the basic (technical) tools and definitions that will be used in the rest of the paper. Before going on, it is useful to examine the execution tree  $D$  in a running example (see Fig. 3). Three initial observations stand out: first, if  $P$  and  $P'$  are distinct paths in  $D$  that lead to the same stable assignment, then the edges of  $P$  and  $P'$  are marked with the same *set* of rotations, although in different order; second, every path in  $D$  has the same length; and third, many of the rotations seem to come in dual pairs as defined below.

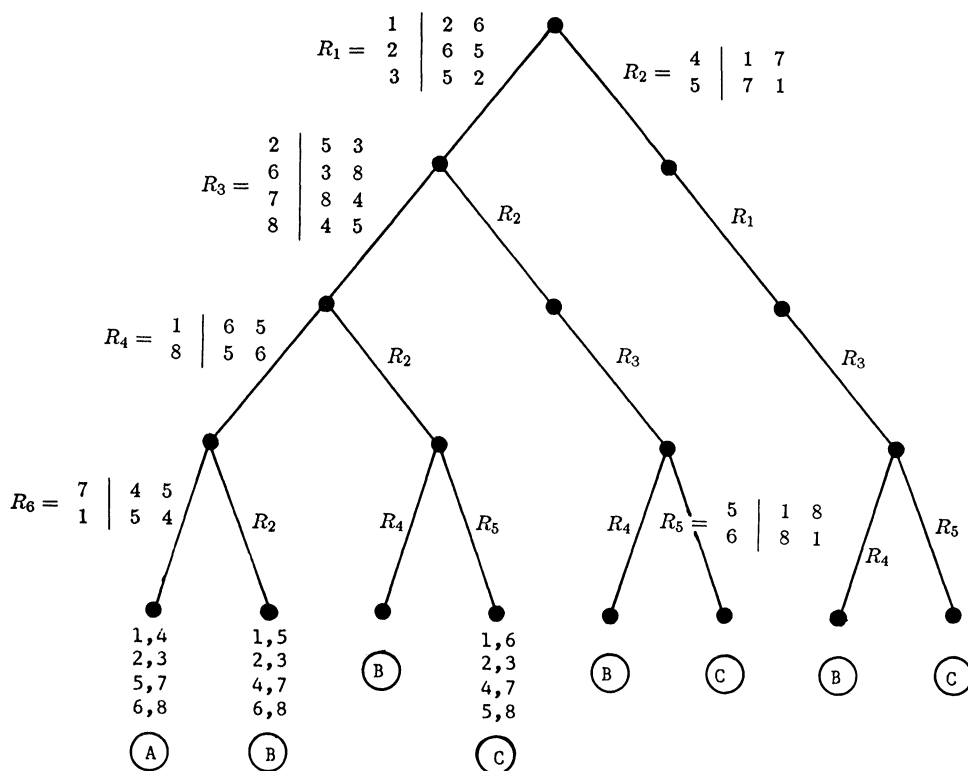


FIG. 3. Tree  $D$  for the example. The tables at the nodes are not shown. All paths have length 4.  $(R_4, R_5)$  and  $(R_2, R_6)$  are each a dual pair of rotations.  $R_1$  and  $R_3$  have no duals.

**DEFINITION.** If  $R = (E, H, S)$  is a rotation in  $D$  then we define  $R^d$  to be the triple  $(S, E, E')$ , where  $S$  and  $E$  have the same order in  $R^d$  as they have in  $R$ . Note that with this definition  $(R^d)^d = R$ . Note also that  $R^d$  has the *form* of a rotation; if  $R^d$  is actually a rotation in  $D$  (i.e., is a rotation exposed in some table  $T$  in  $D$ ), then we call  $R$  and  $R^d$  a *dual pair of rotations*. Any rotation without a dual is called a *singleton rotation*.

In the example, rotations  $R_1$  and  $R_3$  are singletons, and  $(R_4, R_5)$  and  $(R_2, R_6)$  are each a dual pair of rotations. With this terminology, we can make a more precise

observation, which replaces the second and third observations above: Each path from the root to a leaf in  $D$  contains every singleton rotation, and exactly one of each pair of dual rotations. We will prove that these observations are facts which hold for any execution tree  $D$  of Algorithm I, and these facts will then be exploited to reveal the structure of the set of stable roommate assignments. However, we first need several technical definitions and lemmas.

LEMMA 3.1. *If  $R = (E, H, S)$  and  $R^d = (S, E, E')$  are dual rotations that are both exposed in a table  $T$ , then in  $T$  each list of  $E \cup S$  has exactly two elements.*

*Proof.* This follows simply from definition of duals, and the fact that person  $i$  is the head of  $j$ 's list in any table if and only if person  $j$  is the bottom of  $i$ 's list in that table.  $\square$

DEFINITION. For a table  $T$ , the *active part* of  $T$  is the subtable of  $T$  consisting of those lists which contain more than one person.

LEMMA 3.2. *If  $R = (E, H, S)$  and  $R^d = (S, E, E')$  are both exposed in  $T$ , then the active part of the table resulting from eliminating  $R$  from  $T$  is the same as the active part of the table resulting from eliminating  $R^d$  from  $T$ . Further, that active part is just the active part of  $T$  minus the lists of  $E \cup S$ .*

*Proof.* This follows directly from the definition's dual rotations and rotation elimination, and Lemma 3.1 above.  $\square$

DEFINITION. Let  $T$  be a table and  $R = (E, H, S)$  be a rotation. If there is a subset of elements of  $T$  which form a table  $T'$ , such that  $R$  is exposed in  $T'$ , then we say that  $R$  is *embedded* in  $T$ . Note that the definition does not require that some execution of Algorithm I actually expose  $R$ , when started with  $T$ .

LEMMA 3.3. *If  $R$  and  $R^d$  are dual rotations, then  $R$  is embedded in table  $T$  if and only if  $R^d$  is.*

*Proof.* This follows directly from the definition of duals, and the fact that  $i$  is on  $j$ 's list if and only if  $j$  is on  $i$ 's list.  $\square$

DEFINITION. Let  $R$  be a rotation exposed in table  $T$ , and let  $T(R)$  be the table resulting from eliminating  $R$  from  $T$ . If rotation  $R' = (E', H', S')$  is embedded in  $T$  but not in  $T(R)$ , then we say that  $R$  *removes*  $R'$  from  $T$ . Note that for  $R$  to remove  $R'$  from  $T$  all that is required is that  $h'_i$  or  $s'_i$  not appear on  $e'_i$ 's list in  $T(R)$ , for at least one  $e'_i$  in  $R'$ .

DEFINITION. A path  $P$  in  $D$  is said to *contain* the rotations that label the edges of  $P$ .

LEMMA 3.4. *If  $P$  is a path from the root of  $D$  to a node  $x$  in  $D$ , and  $P'$  is a path from the root to a node  $x'$ , and  $P$  and  $P'$  contain the same rotations in different order, then table  $T(x)$  and table  $T(x')$  are identical. Hence a table is determined from the phase 1 table by the set of rotations leading to it, not by their order.*

*Proof.* It is clear from the way that elements are removed in Phase 1 and Phase 2, that at any point in Phase 2, the current table  $T$  is determined by the Phase 1 table and the bottom elements of each list in  $T$ . In Phase 2, the bottom element of person  $i$ 's list is changed only if person  $i$  is in the  $S$  set of an eliminated rotation. Hence if  $i$  is not the second element in any rotation on the path, then  $i$ 's bottom element in  $T$  is its bottom in the Phase 1 table; otherwise, the bottom of  $i$ 's list in  $T$  is given by the person  $p$  who  $i$  most prefers, such that  $i$  is the second element in  $p$ 's list in some rotation on the path to  $T$ .  $\square$

DEFINITION. The set of rotations that appear on a path from the root to a leaf in  $D$  is called a *path set*. We will use this term when the order of the rotations is not important.

Lemma 3.4 shows a mapping from the path sets onto the stable assignments, but Lemma 3.4 does not show that two different path sets cannot generate the same stable assignment. We will later show that the mapping is in fact one-one, i.e., that any two paths in  $D$  which lead to the same table must contain the same set of rotations. This fact, which is more difficult to prove than Lemma 3.4, will be central in the efficient enumeration of the stable assignments.

We are now ready to state and prove the first nontrivial technical lemma.

LEMMA 3.5. *If  $R = (E, H, S)$  and  $R' = (E', H', S')$  are two distinct rotations exposed in a table  $T$ , then  $R$  removes  $R'$  from  $T$  if and only if  $R' = R^d$ . Hence the only way to remove an exposed rotation is to explicitly eliminate it or its dual rotation, if it has one.*

*Proof.* One direction is trivial. If  $R$  and  $R^d$  are dual rotations, then since one is embedded in  $T$  if and only if the other is, the elimination of  $R$  must remove  $R^d$ . To prove the other direction, suppose that  $R$  and  $R'$  are exposed in  $T$ , and that  $R \neq R'$  eliminates  $R'$  from  $T$ . We will show that  $R'$  must be  $R^d$ . When  $R$  is eliminated, person  $q$  is removed from the list of a person  $p$  if and only if  $p = s$ ; or  $q = s$ ; for some  $s_i \in S$ . Clearly, these removals of individual people from  $T$  affect the lists of people in  $E'$  only if  $s_i$  is in  $H'$  (hence in  $S'$ ), or if  $s_i$  is in  $E'$ . To see that the first case is not possible, recall that in table  $T$  the elements in the  $H$  column are a permutation of the  $2n$  people, and that in each rotation  $R$ , the set of  $S$  elements and  $H$  elements in  $R$  are the same. So even though the  $S$  column of table  $T$  is not necessarily a permutation (i.e., a person can appear more than once in the  $S$  column), no person can appear in the  $S$  set of more than one rotation exposed in  $T$ . Hence  $S \cap S' = S \cap H' = \emptyset$ , and so the first case is not possible. Hence the elimination of  $R$  removes  $R'$  from  $T$  only if some  $s_i$  is in  $S \cap E'$ . For ease of discussion, assume without loss of generality that  $s_1 \in S \cap E'$ , and that  $s_1 = e'_j$ .

If  $e_1 \neq h'_j$ , then the change of the bottom of  $s_1$  to  $e_1$  (the consequence of eliminating  $R$ ) will not affect  $R'$ , so we assume also that  $e_1 = h'_j$ . Let  $T(R)$  be the table resulting from eliminating  $R$  in  $T$ . Since the bottom of  $s_1$ 's list moves up to  $e_1$ , which is the head of  $s_1$ 's list in  $T$  (since  $s_1 = e'_j$  and  $e_1 = h'_j$ ),  $s_1$ 's list in  $T(R)$  contains only the single element  $e_1$ . We claim that if  $e'_i \in E'$ , then  $e'_i$ 's list in  $T(R)$  must also contain only a single element. If not, then in  $T(R)$ ,  $e'_i$  is on a tail that leads to no rotation. To see this, note first that  $H'$  cannot be affected by the elimination of  $R$ , and if  $e'_i$  has two elements in its list in  $T(R)$ , then its first two elements in  $T(R)$  are the same as in  $T$ . Hence, following the proof of Lemma 2.1, the unique path from  $e'_i$  in  $T(R)$  is the same as in  $T$ , but that path cannot form a cycle, since it will encounter a member of  $R'$  ( $s_1$  or earlier) which has only one element on its list. Hence if  $e'_i$  has more than one element on its list in  $T(R)$  then it will be on a tail leading to no rotation. But this contradicts Corollary 2.1, so in  $T(R)$  each  $e'_i$  in  $E'$  contains only a single element in its list. Now  $R'$  is exposed in  $T$ , so each  $e'_i$  in  $E'$  has two or more elements on its list in  $T$ , so the effect of eliminating  $R$  in  $T$  is to move the bottom of each  $e'_i$  in  $R'$ . But this is possible only if for each  $e'_i \in E'$ ,  $e'_i = s_k$  and  $h'_i = e_k$ , for some  $e_k$  in  $R$ .

So we now know that if  $R$  removes  $R'$  from  $T$ , then as sets,  $E' = H$ , and  $H' = E = S'$ . This is necessary if  $R' = R^d$ , but in order to actually prove that equality, we need to show that the order inside the sets is correct. We already know that the correspondence between  $E'$  and  $H'$  is correct, i.e., that  $e'_i = s_k$  and  $h'_i = e_k$ , for the same  $k$ . So, assuming without loss of generality that  $s_1 = e'_1$ , we must show that  $s'_i = e_{i+1}$  for each  $i$ , where  $i + 1$  is taken (mod  $r$ ), and  $r$  is the size of  $R$ . To do this, we first note that in  $T$  the list of every element in  $R$  contains exactly two elements; this follows from what we just showed, since in  $T$ , each  $e_i$  in  $R$  is the head of  $s_i$ 's list, so  $s_i$  is the bottom of  $e_i$ 's list,



so each element in  $R$  has exactly two people on its list in  $T$ . But then each  $e_i$  can be only on the list of  $h_i$  or  $s_i$  in  $T$ . Further,  $e_i$  appears once in  $H'$  and once in  $S'$ . Now  $e_i$  is the head of  $s_i$ 's list in  $T$ , so  $e_i$  must be the second element in  $h_i$ 's list in  $T$ . So  $e'_i = s_i = h_{i+1}$ , and the second element on  $h_{i+1}$ 's list is  $e_{i+1}$ , so  $s'_i = e_{i+1}$ , as claimed. Hence if  $R$  removes  $R'$  from  $T$ , then  $R' = R^d$ .  $\square$

Later in the paper we will strengthen this theorem to show that if  $R$  is exposed in  $T$ , and  $R'$  is *embedded* in  $T$ , but perhaps not exposed, then  $R$  removes  $R'$  from  $T$  if and only if  $R' = R^d$ .

**4. The structure of  $D$ .** In this section we examine the structure of  $D$ , as this structure will reveal the structure of the set of stable assignments.

**4.1. Covering rotations.** DEFINITION. Let  $x$  be a node in  $D$  and  $D(x)$  the subtree of  $D$  rooted at  $x$ . The *active part* of  $D(x)$  is the tree  $D(x)$  where at each node  $y$  in  $D(x)$ ,  $T(y)$  is replaced by the active part of  $T(y)$ . Note that the edge labels of  $D(x)$  do not change.

DEFINITION. If  $R$  and  $R^d$  are dual rotations, and path  $P$  in  $D$  contains *either* of them, then we say that  $P$  *covers*  $R$  and  $R^d$ . If  $R$  is a singleton, and  $P$  contains it, then  $P$  covers  $R$ .

LEMMA 4.1. *Let  $x$  be a node in  $D$  with associated table  $T(x)$ . Every path from  $x$  to a leaf in  $D(x)$  covers the same set of rotations.*

*Proof.* Let  $d(x)$  denote the maximum number of edges on any path from  $x$  to a leaf in  $D$ . The theorem will be proved by induction on  $d(x)$ . For  $d(x) = 1$ , if there is only one edge out of  $x$  (i.e., only one rotation exposed in  $T(x)$ ), then the basis is trivially true. If there are two rotations  $R$  and  $R'$  exposed in  $T(x)$ , then, by Lemma 3.5, they must be duals of each other, since eliminating either one of them results in a table with no rotations (i.e., each removes the other). Similarly, there cannot be more than two rotations in  $T(x)$ , since the elimination of any of them removes them all. So the basis is proved.

Assuming that the theorem holds for  $d(x) \leq k$ , let  $x$  be a node in  $D$  where  $d(x) = k + 1$ , and let  $z$  be a child of  $x$  such that  $d(z) = k$ ; by the induction hypothesis, all paths from  $x$  through  $z$  to a leaf must cover the same rotations; let  $P$  be any such path, and let  $R$  be the rotation labeling the edge  $(x, z)$ . If  $R$  is the only rotation exposed in  $T(x)$  then there is nothing to prove, so let  $y$  be another child of  $x$ , and let  $R'$  be the rotation on the edge  $(x, y)$ . We will show that every path from  $x$  through  $y$  to a leaf of  $D(y)$  covers the same set of rotations as  $P$ .

If  $R' = R^d$ , then by Lemma 3.2 (since both  $R$  and  $R^d$  are exposed in  $T(x)$ ), the *active parts* of  $T(z)$  and  $T(y)$  are identical, and hence the active parts of  $D(z)$  and  $D(y)$  are identical. Further,  $d(z) = k$ , and  $d(y) \leq k$ , so each path from  $z$  covers the same rotations, and each path from  $y$  covers the same rotations, so, since the subtrees from  $z$  and  $y$  are identical, any path from  $z$  must cover the same rotations as any path from  $y$ . Then every path from  $x$  through  $y$  covers the same rotations as  $P$ .

If  $R' \neq R^d$ , then, by Lemma 3.5,  $R$  is still exposed in table  $T(y)$ , and  $R'$  is still exposed in table  $T(z)$ . Let  $z'$  be the node associated with the table obtained by eliminating  $R'$  from  $T(y)$ , and let  $y'$  be the node associated with the table obtained by eliminating  $R$  from  $T(y)$ ; and let  $P(z')$  and  $P(y')$  be paths from  $x$  to leafs in  $D$  that pass through  $z'$  and  $y'$ , respectively. Now the set of rotations on the path from the root of  $D$  to  $z'$  is exactly the same as the set of rotations on the path to  $y'$ , hence by Lemma 3.4,  $T(z')$  is identical to  $T(y')$ , and so  $D(z') = D(y')$ . It follows, as in the case above, that  $P(z')$  and  $P(y')$  cover the same set of rotations. But,  $P(z')$  covers the same set as  $P$ , and since  $d(y) \leq k$ , any path out of  $y$  covers the same set of rotations

as  $P(y')$ , hence covers the same set as  $P$ . Node  $y$  was an arbitrary child of  $x$  such that  $y \neq z$ , so the theorem is proved.  $\square$

By definition, every rotation is exposed somewhere in  $D$ ; hence the major consequence of this theorem is the following theorem.

**Path Theorem.**

**THEOREM 4.1.** *Every path from the root of  $D$  to a leaf covers all the rotations. Further, since no path can contain both a rotation and its dual, each path contains every singleton and exactly one of each dual pair of rotations.*

**COROLLARY 4.1.** *Every path in  $D$  from the root to a leaf has the same length.*

Hence the observations in the example hold in general.

We can now strengthen Lemma 3.5.

**COROLLARY 4.2.** *If  $R$  is exposed in table  $T$ , and  $R' \neq R$  is embedded in  $T$ , then  $R$  removes  $R'$  if and only if  $R' = R^d$ .*

*Proof.* Clearly,  $R$  removes  $R^d$  whether  $R^d$  is exposed or not. To prove the converse, let  $x$  be a node in  $D$  with associated table  $T(x)$ , and let  $y$  be the child of  $x$  obtained by eliminating  $R$  from  $T(x)$ . Since  $R'$  is embedded in  $T(x)$ , neither  $R'$  nor  $(R')^d$  are on the path from the root to  $x$ . If  $R$  removes  $R'$ , it removes  $(R')^d$  also, so neither of these rotations is on any path from  $x$  to a leaf. Hence to avoid contradicting Theorem 4.1, it follows that  $R' = R^d$ .  $\square$

**5. The structure of the rotations and stable assignments.** In this section we derive two compact representations of the set of all stable assignments. We first need a few more technical observations.

**5.1. Unique elimination.**

**DEFINITION.** Let  $e_i, h_i, s_i$  be a triple in rotation  $R$ ; hence when  $R$  is eliminated from any table it is exposed in, the bottom of  $s_i$ 's list moves from  $e_{i+1}$  to  $e_i$ , where  $i+1$  is taken mod  $r$ . Let  $A(R, i)$  denote the set of people on  $s_i$ 's original list between  $e_i$  and  $e_{i+1}$ , including  $e_i$  but excluding  $e_{i+1}$ . Similarly, let  $B(R, i)$  be the people between  $e_i$  and  $e_{i+1}$ , including  $e_{i+1}$  but excluding  $e_i$ .

**LEMMA 5.1.** *For any  $e_i$  in  $R$ ,  $R$  is the only rotation whose elimination moves the bottom of  $s_i$ 's list to a person in  $A(R, i)$ , and is the only rotation whose elimination moves the bottom of  $s_i$ 's list from a person in  $B(R, i)$ .*

*Proof.* Let  $R'$  be a different rotation whose elimination moves the bottom of  $s_i$ 's list to a person  $p$  in  $A(R, i)$  from a person  $q$ . Clearly, since  $e_i$  is above  $q$ , and  $p$  is above  $e_{i+1}$ , no path in  $D$  can contain both  $R$  and  $R'$ . Further,  $R^d$  (if it is a rotation) cannot precede  $R'$  on any path, since  $R^d$  moves the head of  $s_i$ 's list to  $e_{i+1}$ , which is below  $p$ . Similarly,  $R'$  cannot precede  $R^d$  on any path, since it moves  $s_i$ 's bottom to  $p$ , which is above  $e_{i+1}$ . But every path contains either  $R$  or  $R^d$ , so no path contains  $R'$ , contradicting the definition of a rotation. The proof for moves from  $B(R, i)$  is similar.  $\square$

**COROLLARY 5.1.** *A person  $p$  is the  $H$  element of person  $q$ 's list in at most one rotation, and is the  $S$  element of  $q$ 's list in at most one rotation. Hence there is at most one rotation whose elimination moves the head of  $q$ 's list to  $p$ , and there is at most one rotation whose elimination moves the bottom of  $p$ 's list to  $q$ .*

**COROLLARY 5.2.** *If  $p \neq e_i$ , and  $p \in A(R, i)$ , then  $s_i$  can never be paired with  $p$  in any stable roommate assignment.*

*Proof.* Consider any path  $P$  where  $p$  is paired with  $s_i$  in the resulting assignment. Since  $s_i$  prefers  $p$  to  $e_{i+1}$ ,  $p$  is not the bottom of  $s_i$ 's list in the Phase 1 table. Hence, somewhere on  $P$ ,  $p$  must become the bottom of  $s_i$ 's list. By the nature of rotation eliminations, for any person  $z$ , the only way that the bottom of  $z$ 's list can change is

by the elimination of a rotation  $R = (E, H, S)$ , where  $z$  is in  $S$ . Hence for  $p$  to become the bottom of  $s_i$ 's list, some rotation must have explicitly moved the bottom of the list to  $p$ . But this contradicts Lemma 5.1 above.  $\square$

*Necessary elimination.* Let  $R$  be a rotation with the triple  $e_i, h_i, s_i$  in  $R$ . If  $p \neq h_i$ , and  $p$  is above  $s_i$  in  $e_i$ 's list in the Phase 1 table, then  $R$  will never be exposed until  $p$  is removed from  $e_i$ 's list.

LEMMA 5.2. *Let  $p$  be a person who must be removed from  $e_i$ 's list before  $R$  is exposed. There exists a unique rotation  $R'$ , such that  $R'$  appears before  $R$  on every path that contains  $R$ , and such that of all the rotations which appear before  $R$  on any path in  $D$ ,  $R'$  is the only one whose elimination removes  $p$  from  $e_i$ 's list.*

*Proof.* From examination of Algorithm I, there are only two ways in which  $p$  is removed from  $e_i$ 's list: either  $p$  is removed when the bottom of  $e_i$ 's list moves up above  $p$ , or when the bottom of  $p$ 's list moves up above  $e_i$ . If  $p$  is removed by the first case event, then  $R$  is not embedded in the table after  $e_i$ 's bottom moves above  $p$ , since  $p$  is above  $s_i$ . Hence  $p$  is removed by the first case event only on paths that do not contain  $R$ . By Lemma 5.1, the second case can happen only when a particular unique rotation,  $R'$ , is eliminated. Since  $p$  must be removed from  $e_i$ 's list before  $R$  can be exposed,  $R'$  must precede  $R$  on any path that contains  $R$ .  $\square$

DEFINITION. If  $p$  must be removed from  $e_i$ 's list before  $R$  is exposed, and if  $R'$  is the (unique) rotation discussed in Lemma 5.2, then we say that  $R'$  explicitly precedes  $R$ .

**5.2. The partial order  $\Pi^*$  and the structure of the stable assignments.**

DEFINITION. Let  $\Pi^*$  be the reflexive transitive closure of the above relation of explicit precedence. It is clear that  $\Pi^*$  is a partial order on the rotations. Figure 4 shows the Hasse diagram of  $\Pi^*$  for the running example.

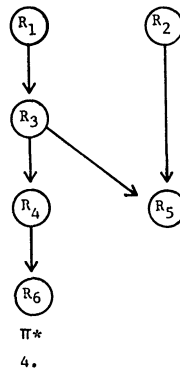


FIG. 4

DEFINITION. In partial order  $\Pi^*$ , a subset  $C$  of rotations is called *closed* if and only if it is closed under the predecessor relation, i.e., if  $R$  is in  $C$ , and  $R'$  precedes  $R$  in  $\Pi^*$ , then  $R'$  is in  $C$ .

LEMMA 5.3. *There is a one-one correspondence between the path sets in  $D$  and the set of those closed subsets of  $\Pi^*$  which contain all the singleton rotations, and contain exactly one of each dual pair of rotations.*

*Proof.* One direction is trivial. Let  $C$  be a path set in  $D$ , and let  $P$  be any of the paths in  $D$  containing path set  $C$ . We claim that  $C$  forms a closed subset in  $\Pi^*$  of the required type. We know that each path in  $D$  contains all the singleton rotations and exactly one of each dual pair of rotations; hence we only need to show that  $C$  is closed

in  $\Pi^*$ . But, by Lemma 5.2 above, any rotation that precedes  $R \in C$  must be contained on  $P$ ; hence  $C$  is closed. Conversely, let  $C$  be a closed set of the required type; we will show that there is a path  $P$  in  $D$  which contains  $C$  exactly. First, the maximal elements  $C_0 \subseteq C$  (those with no predecessors in  $\Pi^*$ ) must be exposed rotations in the Phase 1 table, and since only one of any dual pair is in  $C$ , there is a subpath from the root of  $D$  consisting of the rotations  $C_0$ . After the  $C_0$  rotations are eliminated, the elements  $C_1 \subset C$  whose only predecessors are in  $C_0$  must now be exposed, and (since only one of each dual pair is in  $C$ ) each remains exposed until eliminated, and hence there is a path from the root consisting of  $C_0$  followed by  $C_1$ . Continuing in this way, there is a path from the root to a leaf in  $D$  consisting of the rotations in  $C$ .  $\square$

The proof of the following is essentially the same as the second part of the above proof of Lemma 5.3.

**COROLLARY 5.3.** *If  $C$  is a closed set in  $\Pi^*$ , and is contained in some path  $P$  in  $D$ , then there exists a path  $P'$  in  $D$  containing the same rotations as  $P$ , such that in  $P'$ , all rotations in  $C$  appear before any rotations not in  $C$ . Further, the internal order of the rotations in  $C$  is the same on both paths, as is the internal order of the rotations not in  $C$ , i.e., in  $P'$  the rotations in  $C$  simply move above the non- $C$  rotations, but keep their same internal order.*

Since each path set maps to a unique stable assignment, Lemma 5.3 implies that each closed subset in  $\Pi^*$  of the required type maps to a unique stable assignment. We now show that distinct closed subset map to distinct stable assignments.

**LEMMA 5.4.** *Let  $C$  and  $C'$  be two distinct closed subsets of  $\Pi^*$  which both contain all the singletons and exactly one of each dual pair of rotations. Then, as path sets,  $C$  and  $C'$  yield distinct stable assignments.*

*Proof.* Suppose to the contrary that both the path sets  $C$  and  $C'$  produce the same stable assignment  $A$ . We claim that the minimal rotations in  $C$  must be in  $C'$ . Let  $R = (E, H, S)$  be a minimal rotation in  $C$ , and let  $e_i$  be in  $E$ . Then in  $A$ ,  $e_i$  must be mated to  $s_i$ . To see this, note that the elimination of  $R$  moves  $e_i$ 's head to  $s_i$ , and every rotation that moves  $e_i$ 's head below  $s_i$  (and moves  $s_i$ 's bottom above  $e_i$ ) must be preceded by  $R$ . But  $R$  is a minimal rotation in  $C$ , so the head of  $e_i$ 's list must end up at  $s_i$ , and  $e_i$  is mated to  $s_i$  in  $A$ . Now  $s_i$  is not the head of  $e$ 's list in the Phase 1 table, and the head of any list  $e$  can change only by the elimination of a rotation  $R^* = (E^*, H^*, S^*)$  where  $e$  is in  $E^*$ , so  $C'$  must contain a rotation that moves the head of  $e_i$ 's list to  $s_i$ . But by Corollary 5.1, there is only one rotation,  $R$ , that moves the head of  $e_i$  to  $s_i$ . Hence  $R$ , and all minimal rotations in  $C$ , must also be in  $C'$ . But since  $C$  and  $C'$  are closed, all the predecessors of the minimal rotations in  $C$  must also be in  $C$ , and hence in  $C'$ . But any closed subset is precisely the set of its minimal elements, plus the predecessors of those minimal elements. Hence  $C \subseteq C'$ , but since they both have the same cardinality,  $C = C'$ .  $\square$

The following two theorems connect the preceding lemmas and summarize what we now know about the structure of the stable assignments.

**THEOREM 5.1.** *There is a one-one correspondence between stable assignments and those closed sets in  $\Pi^*$  which contain every singleton rotation and exactly one of each dual pair.*

Hence  $\Pi^*$  is a small ( $O(n^2)$  size) representation of the set of all stable assignments. We will later discuss how to efficiently construct  $\Pi^*$ , and how to construct a stable assignment from a closed subset of the correct type.

**THEOREM 5.2.** *There is a one-one correspondence between path sets in  $D$  and stable assignments.*

Theorem 5.2 is Lemma 3.4 and its converse, and is one of the keys to efficient enumeration of the stable assignments. Before discussing enumeration, we derive an alternative representation for the stable assignments.

**5.3. Independent set representation of stable assignments.** In this section, we present a second compact representation of the set of all stable assignments. We first need some additional observations.

**5.3.1. Refining  $\Pi^*$ .**

LEMMA 5.5. *Let  $(R, R^d)$  and  $(R_1, R_1^d)$  be two dual pairs of rotations, and  $R'$  a singleton rotation. Then:*

(1) *Neither  $R$  nor  $R^d$  can precede  $R'$  in  $\Pi^*$ , i.e., only a singleton rotation can precede a singleton.*

(2)  *$R$  precedes  $R_1$  in  $\Pi^*$  if and only if  $R_1^d$  precedes  $R^d$  in  $\Pi^*$ .*

*Proof.* Since each singleton rotation is on every path in  $D$ , any rotation which precedes a singleton rotation in  $\Pi^*$  must be on every path in  $D$ . So if  $R$  precedes  $R'$ ,  $R$  is on every path, and  $R^d$  is on no paths in  $D$ , contradicting the assumption that  $R^d$  is a rotation; so the first fact is proved. For the second fact, observe first that since  $R$  precedes  $R_1$ , no path can contain both  $R_1$  and  $R^d$ , so any path containing  $R^d$  contains  $R_1^d$ . We must show that in any such path,  $R_1^d$  appears before  $R^d$ . Let  $P$  be a path containing  $R^d$ , and consider the point  $x$  where  $R^d$  is eliminated. Since no path can contain both  $R_1$  and  $R^d$ ,  $R_1$  cannot be exposed anywhere in  $D(x)$ , the subtree of  $D$  below  $x$ . But Corollary 6.2, which will be proven later, states that if any nonsingleton rotation is embedded in  $T(x)$ , then both it and its dual are exposed somewhere in  $D(x)$ . Hence it must be that  $R_1^d$  is not embedded in  $T(x)$ , so  $R_1^d$  must appear on  $P$  before  $R^d$ , and fact 2 is proved.  $\square$

**5.3.2. Graph  $G$ .** We define an undirected graph  $G$  as follows. There exists one node in  $G$  for each nonsingleton rotation, and two rotations  $R_1$  and  $R_2$  are connected by an edge in  $G$  if and only if there exists a rotation  $R$  (possibly  $R_1$  or  $R_2$ , since a node precedes itself by definition) such that in  $\Pi^*$ ,  $R$  precedes  $R_1$  and  $R^d$  precedes  $R_2$ . It follows that  $R$  and  $R^d$  are connected for each dual pair  $(R, R^d)$  and if two rotations are adjacent in  $G$ , then they cannot appear together on any path in  $D$ .

LEMMA 5.6. *Every maximal independent set in  $G$  contains exactly one node from each dual pair of rotations.*

*Proof.* First, no independent set can contain both nodes of a dual pair. For the other side, let  $S$  be an independent set in  $G$  which does not contain either  $R$  or  $R^d$ . If neither  $R$  nor  $R^d$  can be added to  $S$ , then there must be a node  $R_1$  in  $S$  such that  $(R, R_1)$  is an edge in  $G$ , and there must be a node  $R_2$  in  $S$  such that  $(R^d, R_2)$  is an edge in  $G$ . But then there exists a rotation  $R_3$  such that  $R_3$  precedes  $R$ , and  $R_3^d$  precedes  $R_1$ , so  $R_1^d$  precedes  $R_3$  (by Lemma 5.5(2)), and it follows that  $R_1^d$  precedes  $R$ . Also, there exists a rotation  $R_4$  such that  $R_4$  precedes  $R^d$  and  $R_4^d$  precedes  $R_2$ , so  $R$  precedes  $R_4^d$  precedes  $R_2$ . So  $R_1^d$  precedes  $R$  precedes  $R_2$ . But then  $R_1$  and  $R_2$  would be connected in  $G$ . Hence it cannot happen that neither  $R$  nor  $R^d$  could be added to  $S$  to create a larger independent set, and continuing in this way, the lemma is proved.  $\square$

DEFINITION. Let  $\Pi$  be the partial order  $\Pi^*$  with the singletons removed.

The following is directly implied by Theorem 5.1 and Lemma 5.5 (1).

LEMMA 5.7. *Let  $\Sigma$  be the set of all singleton rotations in  $\Pi^*$ . A set of rotations  $C$  is closed in  $\Pi^*$  if and only if  $C - \Sigma$  is closed in  $\Pi$ . Hence there is a one-one correspondence between the stable assignments and the closed subsets of  $\Pi$  that contain exactly one of each dual pair.*

We can now show a one-one correspondence between the maximal independent sets in  $G$  and the stable assignments.

LEMMA 5.8. *Any closed subset  $C$  in  $\Pi$  that contains exactly one of each dual pair, is a maximal independent set in  $G$ .*

*Proof.* No path in  $D$  can contain two rotations which are connected in  $G$ . But since  $C$  is closed in  $\Pi$ ,  $C \cup \Sigma$  corresponds to a path set from  $D$ , hence to at least one path in  $D$ . Hence no rotations in  $C$  can be connected in  $G$ , and  $C$  is an independent set in  $G$ . It is maximal, because it has one rotation from each dual pair, and no independent set in  $G$  has more.  $\square$

LEMMA 5.9. *Any maximal independent set in  $G$  is a closed set in  $\Pi$ .*

*Proof.* Let  $S$  be a maximal independent set in  $G$ , and let  $R$  be any rotation in  $S$ . Suppose  $R'$  precedes  $R$  in  $\Pi$ . Then, since  $R'^d$  precedes itself,  $R$  and  $R'^d$  are connected in  $G$ , so  $R'^d$  is not in  $S$ . But  $S$  must contain one of each dual pair, so  $S$  contains  $R'$ . It follows that  $S$  is closed in  $\Pi$ .  $\square$

In summary, we have Theorem 5.3.

THEOREM 5.3. *There is a one-one correspondence between the maximal independent sets in  $G$  and the set of stable assignments.*

Hence,  $G$  is another small implicit representation of the set of all stable assignments. We can simplify the definition of  $G$  with the following lemma.

LEMMA 5.10. *Rotations  $R$  and  $R'$  are connected in  $G$  if and only if  $R^d$  precedes  $R'$  in  $\Pi$ .*

*Proof.* First, if  $R^d$  precedes  $R'$ ,  $R$  and  $R'$  are connected in  $G$ , since  $R$  precedes itself. Conversely, if  $R$  and  $R'$  are connected in  $G$ , then there is a rotation  $R^*$  such that  $R^*$  precedes  $R$  and  $R^{*d}$  precedes  $R'$ . But by Lemma 5.5,  $R^*$  precedes  $R$  implies that  $R^d$  precedes  $R^{*d}$  so  $R^d$  precedes  $R'$ .  $\square$

The above lemma allows a more efficient method to construct  $G$ : we can determine whether to connect two nodes by looking directly at a single entry in the precedence relation, rather than looking for a third rotation to satisfy the initial requirement for connecting two nodes in  $G$ . It turns out that  $G$  can be constructed in time  $O(n^4)$ ; details are omitted, but are very similar to the construction of a related partial order  $\Pi$  given in [G].

**5.4. Equivalent sets and representations.** We now know that the path sets of  $D$ , the closed subsets of  $\Pi^*$  that contain each singleton and exactly one of each dual pair, and the maximal independent sets of  $G$  augmented with the singletons, are all exactly the same sets of rotations, and these sets are in one-one correspondence with the stable assignments. We will see later that each such set can be used to generate the associated stable assignment in  $O(n^2)$  time.

**6. Efficient enumeration of all stable assignments.** We now discuss an efficient method to generate all stable assignments. The method is a modification of a very general, naive method that is often suggested as a way to enumerate constrained sets; the naive method is rarely efficient, but is efficient in the stable assignment problem. The efficiency here is a consequence of very special properties of the stable assignment problem, and the way the enumeration is implemented.

By Theorem 2.1, the stable assignments can be generated by forcing all execution behaviors of Algorithm I. This would not be guaranteed to be efficient, since the same assignment would likely be generated many times. However, by Theorem 5.2, we need not generate each path in  $D$ , but only each path set. One approach is to use graph  $G$  to generate each maximal independent set (path set) in  $G$ , and, as we will show below, then use the path set to generate the associated stable assignment. The fastest known methods to generate all maximal independent sets in a general graph appear in [LLR], but applying those methods to  $G$  yields a time bound of  $O(n^4)$  per independent set. Alternatively, we could try to generate all the closed subsets of  $\Pi^*$  which contain all

singletons and exactly one of each dual pair. However, general techniques for such constrained enumeration in general partial orders seem very inefficient. Here, we will use a different approach, heavily exploiting results about the structure of  $D$  and the rotations, to obtain a method which, after  $\Pi^*$  is constructed, will generate each path set, and each stable assignment, in  $O(n^2)$  time per assignment. This compares favorably with the  $O(n^3)$  methods given in [K] and [MW] to construct each stable *marriage*.<sup>1</sup> In the first section we will present the method and prove it correct, and in the second section we will discuss the time it requires. We first need the following definitions.

DEFINITION. Let SR be a set of rotations. For each person  $p$ , let  $SR(p)$  be the highest person on  $p$ 's list such that  $SR(p) = e_i$  and  $p = s_i$  in some rotation  $R = (E, H, S)$  in SR.

DEFINITION. Let  $T$  be a table and SR be a set of rotations. The *elimination of SR from  $T$*  is the following: For each person  $p$ , delete all people in  $p$ 's list below  $SR(p)$ , if there are any, and delete  $p$  from the lists of all of those deleted people, if  $p$  exists in those lists.

Note that the definition does not require that all the rotations in SR be embedded in  $T$ , or that SR is contained in any path set.

DEFINITION. For a rotation  $R$ , define  $\Pi^*(R)$  as the set of rotations consisting of  $R$  and all the predecessors of  $R$  in  $\Pi^*$ .

**6.1. The dual enumeration method.** The idea of the method is to simulate Algorithm I forcing it to generate each path set, and associated stable assignment, exactly once. We will represent the simulation by a binary tree  $B$ . As in tree  $D$ , each node  $x$  in  $B$  will represent a table; when node  $x$  is a nonleaf, one edge out of  $x$  will be labeled by a single rotation which is exposed in  $T(x)$ , but the other edge, if it exists, will be labeled by a set of rotations, not necessarily embedded in  $T(x)$ . We will call the first edge the *left* edge and the second edge the *right* edge. If the left edge  $(x, y)$  in  $B$  is labeled by  $R$ , then the table  $T(y)$  at node  $y$  is obtained from  $T(x)$  by eliminating  $R$  from  $T(x)$ . When  $R$  is a singleton, then there will be no right edge out of  $x$ , but when  $R$  is a nonsingleton, then the right edge  $(x, z)$  will exist and will be labelled with

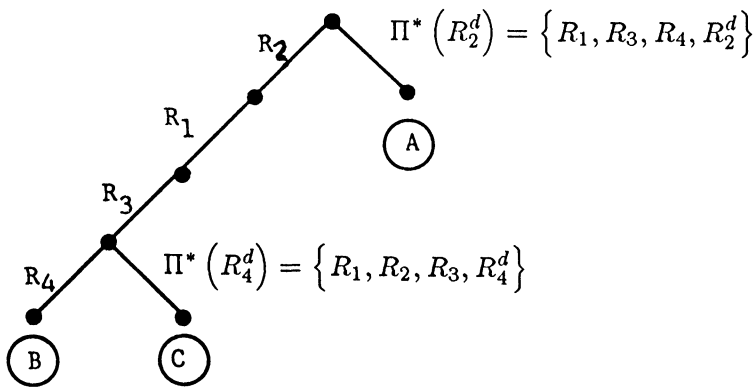


FIG. 5. Tree  $B$  given by the dual enumeration method. The labels at the leaves refer to assignments given in Fig. 3.

<sup>1</sup> It is reported in [K] that the time is  $O(n^2)$ , per marriage, but this is incorrect. Constructions appear in the appendix of [G1] showing that the algorithm can take  $\Omega(n^3k/[\log k^2])$  time for  $k$  stable marriages.

$\Pi^*(R^d)$ , and the resulting table  $T(z)$  will be the table obtained by the elimination of the set  $\Pi^*(R^d)$  from  $T(x)$ . Each leaf of  $B$  will contain a table with no exposed rotations. The simulation begins with a node representing the phase one table, and each node  $x$  in  $B$  is expanded by arbitrarily choosing an exposed rotation  $R$  in  $T(x)$ , eliminating  $R$  from  $T(x)$  on the left edge out of  $x$ , and, if  $R$  is a nonsingleton, eliminating  $\Pi^*(R^d)$  from  $T(x)$  on the right edge out of  $x$ . A table with no exposed rotations is not expandable; we will show that such a table must specify a stable assignment. The simulation ends when no unexpanded nodes in  $B$  are expandable. We call this method the *dual enumeration* method. Figure 5 illustrates this method on the running example.

### 6.1.1. Correctness of the dual enumeration method.

LEMMA 6.1. *The dual enumeration method never generates a path set (stable assignment) more than once.*

*Proof.* Let  $x$  be a nonleaf node in  $B$ , and let  $R$  be the rotation on the left edge out of  $x$ . Consider the leaves in the subtree rooted at  $x$ , and the path sets associated with those leaves. Those path sets are divided into those sets containing  $R$  and those containing  $R^d$  (none, if  $R$  is a singleton). Since no path set contains both a rotation and its dual, there is no intersection between these two sets of path sets. Applying this fact inductively upward from the leaves to the root in  $B$ , it follows that no path set is generated more than once. Theorem 5.2 then implies that no stable assignment is generated more than once.  $\square$

LEMMA 6.2. *Every stable assignment (path set) is generated at least once by the above method.*

*Proof.* Let  $x$  be a nonleaf node in  $B$  which is also a node in  $D$  (the root of  $B$  is such a node, although we will later see that all nodes of  $B$  are in  $D$ ). Let  $A$  be a stable assignment contained in  $T(x)$ . Let  $R$  be the exposed rotation eliminated from  $T(x)$  on the left edge out of  $x$  in  $B$ , and let  $T(y)$  be the resulting table. Suppose first that the path set for  $A$  contains  $R$ . Then by Corollary 2.2, and the fact that  $T(x)$  is in  $D$  as well as in  $B$ , it follows that node  $y$  is in both  $D$  and  $B$ , and assignment  $A$  is contained in table  $T(y)$ .

Now suppose that the path set for  $A$  contains  $R^d$ , and let  $w$  be any node in  $D(x)$ , the subtree of  $x$  in  $D$ , such that  $T(w)$  contains  $A$  and such that the edge into  $w$  is labeled with  $R^d$ . Clearly, every path from  $x$  which leads to a leaf labeled with  $A$  contains such a node  $w$ . Also, every rotation in  $\Pi^*(R^d)$  must be on the path in  $D$  from the root to  $w$ . Now, as in the proof of Corollary 5.3, a rotation  $R^d$  is exposed if all of its predecessors in  $\Pi^*$  have been eliminated and  $R$  has not been removed, so there must also be a path in  $D$  from  $x$  to a node  $z$ , containing exactly those rotations in  $\Pi^*(R^d)$  that are on the path from  $x$  to  $w$  (i.e., this second path is obtained by deleting all the rotations on the first path from  $x$  that are not in  $\Pi^*(R^d)$ ). Further, the elements of  $T(w)$  are all contained in  $T(z)$ , since the rotations leading to  $w$  are a superset of those leading to  $z$ . Hence  $A$  must be contained in  $T(z)$  as well as in  $T(w)$ . But  $T(z)$  is exactly the table obtained in  $B$  by the elimination of  $\Pi^*(R^d)$  from  $T(x)$ . Hence, if the path set for  $A$  contains  $R^d$ , then the right child of node  $x$  in  $B$  is the above node  $z$  in  $D$ , and  $T(z)$  in  $B$  contains assignment  $A$ .

Now the phase one table is in both  $D$  and  $B$ , and so by iterating the above arguments downward from the root, we see that there is a path in  $B$  consisting of nodes in both  $D$  and  $B$ , such that assignment  $A$  is contained in each of the tables on the nodes in the path. Since the nodes on this path are in  $D$ , and the table sizes decrease with each edge on the path, the path ends with a table containing assignment  $A$  exactly. Hence assignment  $A$  appears at some leaf of tree  $B$ .  $\square$



The above lemmas show that every stable assignment is generated exactly once, but this does not prove the correctness of the dual method, since we have not proved that the method never generates tables that are not in  $D$ . If it did, then it could generate assignments that are not stable, or it could generate tables that have “rotations” that are not in  $D$ . Lemma 6.2 does not show that every table in  $B$  is also in  $D$ , although it is immediate that if  $x$  is in  $B$  and  $D$ , then  $y$ , the left child of  $x$  in  $B$ , is also in  $D$ . What remains to show is that the right child of  $x$  in  $B$  is necessarily in  $D$ . The point is that even though there is an assignment in  $T(x)$  whose path set contains the nonsingleton rotation  $R$ , which is exposed in  $T(x)$ , we do not know for sure that there is an assignment in  $T(x)$  whose path set contains  $R^d$ . If there is no such assignment, then the dual enumeration method will either generate a table which does not specify an assignment but which has no exposed rotations (in which case it will have done excess work), or it will generate an assignment which is not stable. Neither of these things can happen if each table in  $B$  is also in  $D$ . To prove that each table of  $B$  is in  $D$ , we show that for any node  $x$  in  $D$ , if  $R$  is exposed in  $T(x)$ , then  $R^d$  is exposed in some table in  $D(x)$ , the subtree of  $D$  rooted at  $x$ . This implies that  $T(x)$  contains a stable assignment whose path set contains  $R^d$  in  $D(x)$ , if  $R$  is exposed in  $T(x)$ . Of course, since  $R$  is exposed in  $T(x)$ ,  $R^d$  appears below  $x$  on any of these paths.

**THEOREM 6.1.** *Let  $R$  and  $R^d$  be dual rotations, and  $x$  a point in  $D$ . If  $R$  is exposed in  $T(x)$ , then  $R^d$  is exposed in  $D(x)$ . Hence if  $R$  is exposed in  $T(x)$ , then there is a stable assignment in  $T(x)$  whose path set contains  $R^d$  as well as one which contains  $R$ .*

To prove this theorem we first need the following.

**LEMMA 6.3.** *If  $P$  is a path from the root of  $D$  to a node  $x$  in  $D$ , and  $P'$  is a path from the root to a node  $x'$ , and  $P$  and  $P'$  cover the same set of rotations, then the active parts of table  $T(x)$  and table  $T(x')$  are identical, and hence the active parts of  $D(x)$  and  $D(x')$  are also identical.*

*Proof.* Note first that since no path can contain both rotations in a dual pair, the length of  $P$  and  $P'$  are the same. Let  $dP$  and  $dP'$  be the parts of  $P$  and  $P'$ , respectively, after the point  $v$  in  $D$  where  $P$  and  $P'$  diverge. The proof of the lemma is by induction of the length of  $dP$  (which is, of course, also the length of  $dP'$ ). For length of one,  $dP$  must contain  $R$  while  $dP'$  contains  $R^d$ , for some dual pair of rotations. Then in  $T(x)$ , both  $R$  and  $R^d$  are exposed and the basis follows from Lemma 3.2. Now assuming the theorem holds for  $dP$  of length  $k$ , consider  $dP$  of length  $k+1$ , and let  $R$  and  $R'$  be the first rotations on  $dP$  and  $dP'$  respectively, and let  $\nu_R$  and  $\nu_{R'}$  be the first nodes below  $v$  on these paths (see Fig. 6(a)). If  $R' = R^d$ , then the active tables are the same after eliminating either rotation, and hence there must be a path from  $\nu_R$  that is identical to the part of  $dP'$  starting at  $\nu_{R'}$ . Hence the table  $T$  at the end of that path is  $T(x')$ . But, by the induction hypothesis, the active part of  $T(x)$  is the same as the active part of  $T$ ; hence the theorem follows in this case.

Now suppose that  $R' \neq R^d$ . There are two cases to consider: either  $R$  is on  $dP'$ , or  $R^d$  is on  $dP'$ .

Let  $w$  be the point on  $dP'$  where  $R$  (in the first case) or  $R^d$  (in the second case) is eliminated. Since both  $R$  and  $R'$  are exposed at  $v$ ,  $R$  must be exposed at every table on  $dP'$  down to  $w$ .

In the first case, consider the edge on  $dP'$  into  $w$ , and let  $R^*$  be the rotation eliminated there (see Fig. 6(b)). If instead of eliminating  $R^*$ ,  $R$  is eliminated at that point,  $R^*$  will not be removed (since  $R^* \neq R^d$ ). Hence the path which is identical to  $dP'$  except that the order of  $R^*$  and  $R$  is reversed, is in fact a path from  $v$ ; call that path  $dP^*$ . Now  $dP'$  and  $dP^*$  contain exactly the same rotations, so, by Lemma 3.4, the table at the end of  $dP^*$  is  $T(x')$ . Repeating this argument up the length of  $dP'$ ,

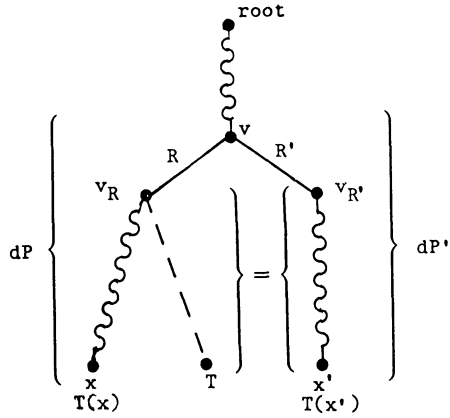


FIG. 6(a). Proof of Lemma 6.3, when  $R' = R^d$ . Wavy lines are known paths in  $D$ , solid edges are known single edges in  $D$ , labeled dashed lines are inferred edges, and unlabeled dashed lines are inferred paths in  $D$ .

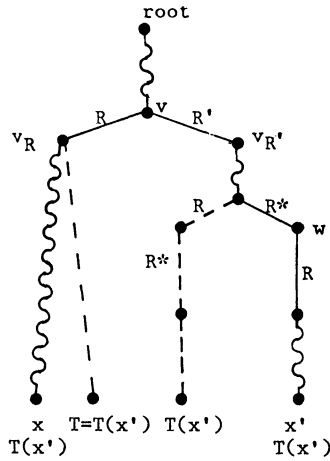


FIG. 6(b).  $R' \neq R^d$  and  $R^d$  is on  $dP'$ .

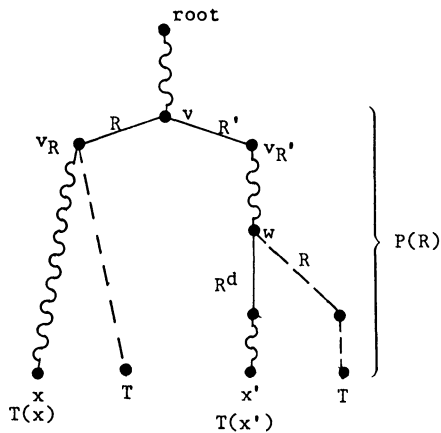


FIG. 6(c).  $R' \neq R^d$  and  $R^d$  is on  $dP'$ .

moving  $R$  up at each step and leaving the rest of the path the same, it follows that there is a path from  $v$  through  $\nu_R$  which contains the same rotations as  $dP'$ , and hence ends with table  $T = T(x')$ . But, by the inductive hypothesis, as above, the active parts of  $T(x)$  are identical to the active parts of  $T$ , and hence of  $T(x')$ .

In the second case, when  $R^d$  is on path  $dP'$ , there must be a path,  $P(R)$ , from  $v$  through  $\nu_R$ , which is identical to  $dP'$ , except that  $R$  replaces  $R^d$  (see Fig. 6(c)). This follows from Lemma 3.2, and the fact that  $R$  is exposed at  $w$ . But now  $dP'$  and  $P(R)$  diverge below  $v$ ; hence by the induction hypothesis, the active part of the table,  $T$ , at the end of  $P(R)$  is identical to the active part of  $T(x')$ . Now we can repeat the step argument of the first case, moving  $R$  up  $P(R)$  to  $v$ , and conclude that there is a path from  $v$  through  $\nu_R$  which contains exactly the same rotations as  $P(R)$ . Then, by the inductive hypothesis, the active part of  $T(x)$  is the same as the active part of  $T$ , which is the same as the active part of  $T(x')$ .  $\square$

*Proof of Theorem 6.1.* Let  $z$  be the closest ancestor of  $x$  such that  $R^d$  is exposed in  $D(z)$ , and let  $y$  (possibly  $x$ ) be the child of  $z$  on the path from  $z$  to  $x$ . Let  $R_1$  be the rotation on the  $(z, y)$  edge. Let  $P$  be a path from  $z$  to a leaf, where  $P$  contains  $R^d$ , and let  $R_2$  be the first rotation on  $P$  (see Fig. 7(a)). If  $R_1 = R_2^d$ , then the active tables after eliminating either rotation are the same, so the subtrees below those two points must be the same; hence  $D(y)$  must contain  $R^d$ . So, assume that  $R_1 \neq R_2^d$ . Neither  $R_1$  nor  $R_1^d$  is on the path from the root to  $z$ , so either  $R_1$  or  $R_1^d$  must be on  $P$ , say at a point  $w$ . Note that in either case,  $R_1$  is exposed at  $w$ . Hence if  $R^d$  is before  $w$  on  $P$ , then we can assume  $P$  contains  $R_1$ . If  $P$  contains  $R_1^d$  and  $R^d$  is after  $w$  (see Fig. 7(b)), then consider the effect of eliminating  $R_1$  at  $w$ ; the resulting active table is the same as after eliminating  $R_1^d$ , so there is a path from  $w$  which contains  $R^d$ . So we can always assume that  $P$  contains  $R_1$  and  $R^d$ . But now, we can move  $R_1$  up a step at a time, as in the preceding proof, concluding that there is a path from  $z$  through  $y$  that contains  $R^d$ . This contradicts the selection of  $z$ , and proves the theorem.  $\square$

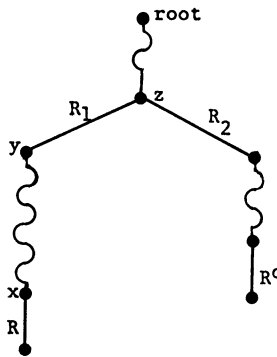
Theorem 6.1 completes the proof of correctness of the dual enumeration method. In summary, we have Theorem 6.2.

**THEOREM 6.2.** *The dual enumeration method generates each stable assignment exactly once, and the table at each leaf in  $B$  specifies a stable assignment.*

There are several useful corollaries of Theorem 6.1.

**COROLLARY 6.1.** *For any table  $T(x)$  in  $D$ , if  $R$  is the only rotation exposed in  $T(x)$ , then  $R$  is a singleton.*

**COROLLARY 6.2.** *If  $R$  and  $R^d$  are duals embedded in  $T(x)$ , then both are exposed somewhere in  $D(x)$ .*



7a.

FIG. 7(a)

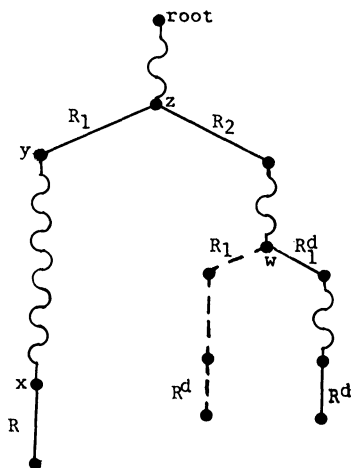


FIG. 7(b).  $R^d$  is on  $P$  after  $w$ .

Recall that the proof of Lemma 5.5 (2) depended on Corollary 6.2, so Lemma 5.5 is now fully proved.

**COROLLARY 6.3.** *If  $R$  and  $R^d$  are duals, then there is a point  $x$  in  $D$  where both  $R$  and  $R^d$  are exposed in the table  $T(x)$ .*

**COROLLARY 6.4.** *If  $R$  is a nonsingleton rotation, and  $e_i \in R$ , then there is a stable roommate assignment where each  $e_i$  is paired with  $h_i$ , and also one where each  $e_i$  is paired with  $s_i$ .*

*Proof.* By Lemma 3.1, at the point  $x$  where both  $R = (E, H, S)$  and  $R^d$  are exposed, each list in  $E \cup S$  contains exactly two elements, and eliminating  $R^d$  makes  $h_i$  the only element on  $e_i$ 's list, and eliminating  $R$  makes  $s_i$  the only element on  $e_i$ 's list. With either elimination, the algorithm is guaranteed to find a stable assignment in the resulting table.  $\square$

**6.2. Complexity analysis.** In this section we show that the dual enumeration method works in time  $O(n^3 \log n + kn^2)$  to enumerate  $k$  stable assignments. The  $O(n^2 \log n)$  term is the time needed to find all the rotations in  $D$  and to construct  $\Pi^*$ . Thereafter, each stable assignment is generated in  $O(n^2)$  time, per assignment. To prove the above bound, we first examine the computational steps needed before and in the dual method: how to find all the rotations and recognize the singletons, how to eliminate a set of rotations, how to generate “enough” of  $\Pi^*$ , and how to find  $\Pi^*(R)$  for a rotation  $R$ . We then show how to charge the work during the dual algorithm, and prove the above time bound.

**6.2.1. Finding all the rotations in  $O(n^3 \log n)$  time.** Even though the number of stable assignments, hence the size of  $D$ , grows exponentially in  $n$ , the Path Theorem, and the fact that Algorithm I runs in  $O(n^2)$  time, imply that there can be at most  $O(n^2)$  rotations in  $D$ . In fact, Corollary 5.1 shows that there can be at most  $n(n-1)$  rotations. We show here that all the rotations in  $D$  can be found in  $O(n^3 \log n)$  time. To do this, we run Algorithm I once, following a path  $P$  in  $D$ , finding the rotations on  $P$ . By the Path Theorem,  $P$  covers all the rotations on  $D$ , but we still have to determine which are singletons and which have duals. If  $R$  is a rotation on  $P$ , then

we can test if  $R^d$  is a rotation by simply returning to the point on  $P$  where  $R$  is eliminated and successively choosing and eliminating any rotation other than  $R$ . By Corollary 4.2, we will either expose and eliminate  $R^d$ , or we will have a table where only  $R$  is exposed; in the latter case,  $R$  must be a singleton, by Corollary 6.1. There are at most  $n(n-1)$  rotations on  $P$ , and each run of Algorithm I costs  $O(n^2)$  time, so although the size of  $D$  can be exponential in  $n$ , all rotations in  $D$  can be found in  $O(n^4)$  time. Of course, in practice this procedure can be sped up by noting at each step which other rotations are exposed.

We can speed up the above method to run in time  $O(n^3 \log n)$ . The idea is that we can find  $2n$  chains in  $\Pi^*$  that contain all the rotations on path  $P$ ; there is one chain for each person  $p$ , and it simply consists of the ordered list of the rotations on  $P$  that move the head of  $p$ 's list. The rotations in the chain are ordered by the relative order that they appear on  $P$ . It is easy to see that if rotation  $R$  moves  $p$ 's head before rotation  $R'$  does, on  $P$ , then  $R$  precedes  $R'$  in the partial order  $\Pi^*$ . Now Lemma 5.5 (1) says that in  $\Pi^*$  only singleton rotations can precede a singleton, so this implies that there is a point on each chain where all the rotations above the point are singletons, and all below it are nonsingletons; we search for that point using binary search. Each query in the search costs  $O(n^2)$  time, as above, and since there are only  $n(n-1)$  rotations per chain, the breakpoint for each chain is found in  $O(n^2 \log n)$  time. There are only  $2n$  chains, so at most  $O(n^3 \log n)$  total time is required. Since the form of the dual rotations (not on  $P$ ) are known, and the total size of their description is  $O(n^2)$ , once the singletons have been identified, the rotations not on  $P$  can be generated in  $O(n^2)$  total time.

**6.2.2. Time needed for the elimination of a set of rotations.** Any arbitrary set of rotations  $SR$  can be eliminated from any table  $T$  in  $O(n^2)$  time. The key is that the total size of the description of  $SR$  is  $O(n^2)$ , so a simple scan through the rotations in  $SR$  finds  $SR(p)$  for each person  $p$ , and this takes  $O(n^2)$  time in total. Further, any table is of size  $O(n^2)$ , and when  $SR$  is eliminated, each element to be removed can be found and removed in constant time, so the elimination of the elements only requires  $O(n^2)$  time. This also shows that given a path set, the associated stable assignment can be generated in  $O(n^2)$  time, by simply eliminating the path set from the phase one table.

**6.2.3. Time needed to “construct”  $\Pi^*$  and to find  $\Pi^*(R)$ .** The partial order  $\Pi^*$  has  $O(n^2)$  elements, and if we represent  $\Pi^*$  as a directed graph,  $DG$ , where each node is an element in  $\Pi^*$  and each edge corresponds to a pair in the relation, then there might be as many as  $\Omega(n^4)$  edges in the graph. However, in the dual enumeration method, we only need to know  $\Pi^*$  in order to find the predecessors of a rotation. Hence it will suffice to know any subgraph of  $DG$  whose transitive closure is  $\Pi^*$ . By definition, the Hasse diagram of  $\Pi^*$  is the smallest such subgraph. It turns out that the Hasse diagram has at most  $O(n^2)$  edges, and there is a supergraph,  $DG^*$ , of the Hasse diagram, which also contains only  $O(n^2)$  edges, and  $DG^*$  can be found from the set of rotations in  $O(n^2)$  time. Then since  $DG^*$  has only  $O(n^2)$  nodes and edges, given any rotation  $R$ , we can find  $\Pi^*(R)$  in  $O(n^2)$  time by backwards search from  $R$  in the obvious way.

Summarizing, we have Lemma 6.4.

**LEMMA 6.4.** *There exists a directed acyclic subgraph,  $DG^*$ , of  $DG$  containing all the nodes of  $DG$ , such that  $R$  leads to  $R'$  by a path in  $DG^*$  if and only if  $R$  is connected by a directed edge to  $R'$  in  $DG$ . Further,  $DG^*$  has  $O(n^2)$  edges, and it can be constructed from the set of rotations in  $O(n^2)$  time.*

*Proof.* Let  $e_i, h_i, s_i$  be a triple in rotation  $R$ . Recall that when  $R$  is eliminated from any table it is exposed in, the bottom of  $s_i$ 's list moves from  $e_{i+1}$  to  $e_i$ , where  $i+1$  is taken mod  $|R|$ . Recall also that  $A(R, i)$  is the set of people on  $s_i$ 's original list between  $e_i$  and  $e_{i+1}$ , including  $e_i$  but excluding  $e_{i+1}$ . In order to "construct"  $\Pi^*$ , we first examine each rotation  $R = (E, H, S)$ , and for each person  $s_i$  in  $S$ , we find  $A(R, i)$  and then for each person  $p$  in  $A(R, i)$ , we mark  $s_i$  in  $p$ 's list with rotation  $R$ . All of these markings can be done in  $O(n^2)$  time, since each  $A(R, i)$  is a contiguous list of elements in  $s_i$ 's original list, and by Lemma 5.1, no two  $A(R, i)$  sets intersect.

The rest of the construction method, and the proof of its correctness and time, essentially appear in [G] § 4.2 where enumeration for the stable *marriage* problem is discussed. In the stable marriage problem, the set of stable marriages are also represented by a supergraph of the Hasse diagram of a particular partial order. That supergraph has  $O(n^2)$  nodes and edges and is constructed from a marked table similar to the one above. What is important is that the only properties of the marked table that are needed in the construction of the supergraph of [G], and to prove the size and time bounds, also hold for the above marked table used here. The reader is referred to [G] for the complete details.  $\square$

**6.2.4. Time needed for the dual enumeration method.**

**THEOREM 6.3.** *Given  $DG^*$  as above, each stable assignment can be generated in  $O(n^2)$  time, per assignment. In fact, the assignments can be generated on-line in this time.*

*Proof.* Consider a node  $x$  in  $B$ , and define  $x$  as a left node if it is the root of  $B$ , or if the edge into it from its parent is a left edge. From each leaf which is also a left node there is a unique maximal path upward from the leaf consisting only of left nodes (see Fig. 8). For example, the path from the leftmost leaf in  $B$  runs to the root of  $B$ . These paths are edge disjoint, and cover every left edge in  $B$ . Now consider the top node  $x$  of one of these paths. Starting from this top node  $x$ , the work of the dual enumeration method along the path down to the associated leaf, consists of an execution of Algorithm I starting from  $T(x)$ , which is a subtable of the Phase 1 table. Hence the total time for the work along this path is  $O(n^2)$ . But these paths are disjoint, each ends at a distinct leaf of  $B$ , hence at a distinct stable assignment, and these paths cover all the left edges in  $B$ . Hence the total work of the dual enumeration method along the left edges of  $B$  is  $O(n^2)$ , per assignment.

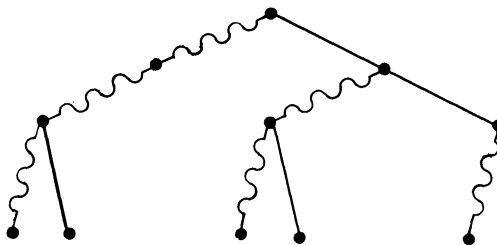


FIG. 8. Schematic tree  $B$ . The maximal paths of left edges are drawn with wavy lines.

The work on any right edge out of any node  $x$  in  $B$  consists of finding  $\Pi^*(R^d)$  for a given  $R^d$ , and eliminating  $\Pi^*(R^d)$  from  $T(x)$ . As shown above, each of these

operations can be done in  $O(n^2)$  time. If  $(x, y)$  is a right edge, and  $y$  is a leaf, then we can charge the  $O(n^2)$  time for edge  $(x, y)$  to the assignment at node  $y$ . If  $(x, y)$  is a right edge and  $y$  is not a leaf, then  $y$  is the top node of one of the maximal left paths discussed above. We charge the work on  $(x, y)$  to the assignment at the end of that path. Clearly, no assignment gets charged for more than one right edge, so the work for all the right edges in  $B$  is also  $O(n^2)$ , per assignment.

In order to make the time bound on-line, simply expand  $B$  in a depth-first manner, going left whenever possible. In such a traversal, no more than  $O(n^2)$  time can pass between the generation of new stable assignments.  $\square$

In the above analysis, the time for the left edges can be accounted for more closely, to get an  $O(n)$  time bound per assignment for the left edges. However, the work for the right edges remains  $O(n^2)$  per assignment, and it is an open question whether this can be substantially reduced.

### 7. Characterizing the stable pairs.

**DEFINITION.** If persons  $i$  and  $j$  are paired together in some stable assignment, then they are called a *stable pair*. If they are paired together in all assignments, then they are called a *fixed pair*.

Knuth [K] mentions the utility of knowing the stable pairs in the stable marriage problem, and in [G] it is shown how to find all the stable pairs for the marriage problem in  $O(n^2)$  time. Here we examine the equivalent question for the stable roommate problem.

**DEFINITION.** Let  $\Sigma$  be the set of singleton rotations, and let  $P$  be a subpath from the root in  $D$  containing all and only the singleton rotations. Let  $x^*$  be the end of path  $P$ .

By Lemmas 5.3 and 5.5, it is clear that such a  $P$  exists, and that all stable assignments are in  $D(x^*)$ . In other words, if all the singleton rotations are eliminated as a set from the phase one table,  $T(x^*)$  results, and it contains all stable assignments.

**LEMMA 7.1.** *Person  $i$  is in a fixed pair if and only if  $i$ 's list in  $T(x^*)$  has only a single entry.*

*Proof.* First, since  $D(x^*)$  contains all the stable assignments, if  $i$ 's list in  $T(x^*)$  contains only one entry,  $j$ , then  $(i, j)$  is a pair in every stable assignment. If  $i$ 's list in  $T(x^*)$  is not a single entry, then  $i$  must be in the  $E$  set of a rotation  $R$ , and in the  $H = S$  sets of rotation  $R^d$ . Hence by Corollary 6.4,  $i$  is in at least two stable pairs, and so is not in a fixed pair.  $\square$

We claim that if  $i$  is not in a fixed pair, then  $i$  can mate with person  $p$  only if  $p$  is in the  $S$  set for  $i$  in some nonsingleton rotation  $R$ , or if  $i$  is in the  $S$  set for  $p$  in  $R^d$ . This follows easily from the workings of Algorithm I, and the fact that  $T(x^*)$  is generated by the singletons. Combining this observation with Corollary 6.4 gives Theorem 7.1.

**THEOREM 7.1.** *If  $(e_i, j)$  is not a fixed pair, then it is a stable pair if and only if  $j = h_i$  or  $j = s_i$  in some nonsingleton rotation  $R = (E, H, S)$ .*

Hence the nonfixed stable pairs can be found immediately from the nonsingleton rotations, and the fixed pairs can be found in  $O(n^2)$  time from the singletons, by eliminating the singletons from the Phase 1 table. Hence the set of all stable pairs can be found in  $O(n^3 \log n)$ , and any speed up, down to  $O(n^2)$  time, in finding the rotations will speed up finding the stable pairs. It is not difficult, using Theorem 7.1, to find all the fixed pairs in  $O(n^3)$  time, but it is open whether this leads to a faster way of finding all the stable pairs, or of finding the singletons.

### 8. Specializing the roommate structure to marriage.

DEFINITION. Let  $MP$  be an instance of the stable marriage problem on  $n$  men  $M$ , and  $n$  women  $W$ . Let  $RMP$  be an instance of the stable roommate problem on  $M \cup W$ , obtained from  $MP$  by adding to the end of each man (woman)  $p$ 's list all the men (women) other than  $p$ .

It is easy to prove that each stable assignment in  $RMP$  is a stable marriage in  $MP$  and conversely. Hence the structure of the stable roommate problem applies to the marriage problem, and it is of interest to see how the general structure specializes in the case of stable marriage. In particular, it is interesting to compare the results here to those of [IL], where a structure of the set of stable marriages was first obtained. The results in this paper specialize to those in [IL] for the marriage problem. However, the structure of stable marriages is somewhat simpler than the structure of stable roommate assignments, and this allows faster algorithms to construct it, and a simpler view of how to construct stable marriages from the partial order(s). Hence the structure resulting from specializing the roommate structure to stable marriage at first appears different than that in [IL]. In the next paragraphs we sketch additional observations that connect the structures and the expositions.

The following facts are stated without proof:

(1) In  $\Pi^*$  resulting from  $RMP$ , there are no singleton rotations.  
 (2) If  $R = (E, H, S)$  is a rotation in  $\Pi^*$  resulting from  $RMP$ , then all people in  $E$  have the same sex, and all people in  $H$  have the opposite sex. Hence the  $E$  set in  $R$  is male if and only if the  $E$  set in  $R^d$  is female, and the rotations partition naturally into two equal-sized sets of rotations, one in which each rotation has a male  $E$  set and one in which each rotation has a female  $E$  set. We call the first type the male rotations and the second type the female rotations.

(3) In every pair in relation  $\Pi^*$  resulting from  $RMP$ , the two rotations have the same sex. Hence, the partial order  $\Pi^*$  in  $RMP$  is composed of two disjoint partial orders:  $M\Pi^*$  containing the male rotations, and  $F\Pi^*$  containing the female rotations.

(4) Let  $C$  be a closed set in  $M\Pi^*$ , and let  $C'$  be the rotations in  $M\Pi^* - C$ , and let  $C'^d$  be the duals, in  $F\Pi^*$ , of the rotations in  $C'$ . If  $C$  is closed in  $M\Pi^*$ , then  $C'^d$  is closed in  $F\Pi^*$ , and hence their union is closed in  $\Pi^*$ , and since the union contains exactly one of each dual pair, it represents a stable assignment in  $RMP$  (stable marriage in  $MP$ ). The converse is obviously also true. Hence, there is a one-one correspondence between the closed sets in  $M\Pi^*$  (without further constraints) and the stable marriages of  $MP$ .

This paper introduced the concepts of dual rotation and singleton rotation. Facts 1 and 4 explain why these concepts were not needed in the structure of stable marriage. Although the exposition is quite different here, the representation of the set of stable marriages given in [IL] can be expressed as  $M\Pi^*$ . Facts 3 and 4 help "explain" why the representation of stable marriages is simpler than the representation of stable assignments: each closed subset in  $M\Pi^*$  represents a stable marriage, and conversely, while in the general case of stable assignment, only particular, highly constrained closed subsets in  $\Pi^*$  represent stable assignments. This also partly "explains" why each stable marriage can be constructed in  $O(n)$  time in [G], while in this paper,  $O(n^2)$  time for each stable assignment is the best bound obtained. Hence what makes the roommate problem more involved is the existence of singleton rotations, and the fact that the nonsingletons do not partition in a nice way, as they do in the marriage problem. As a final comment, note that given fact 1, all rotations in  $M\Pi^*$  can be obtained from a single pass through Algorithm I in  $O(n^2)$  time. This bound was first



obtained for the stable marriage problem by a very different method and argument in [G].

**9. Open problems.** First, given the  $O(n)$ -time method in [G] to enumerate each stable marriage, compared to the  $O(n^2)$  method for the more general problem of stable assignment, a natural problem is to bring down the time for enumerating assignments, or to more fully explain why the stable assignment problem is more complex than the marriage problem. Second, the  $O(n^3 \log n)$  bound for finding all the rotations, and hence the stable pairs, seems too large. We conjecture that  $O(n^2)$  is the correct bound, and that the singletons should be recognizable after a single execution of Algorithm I. It seems likely that there are additional structural observations about rotations that will lead to this time bound. For example, it is an easy corollary of Theorem 6.1 that if  $R = (E, H, S)$  is a rotation such that  $E \cap H \neq \emptyset$ , then  $R$  is a singleton rotation. In the example presented in this paper, that fact identifies the singletons, but it is not true in general that for every singleton rotation  $E \cap H \neq \emptyset$ . Finally, there is the more general question of what algebraic structure is generated by the set of stable assignments, under some reasonable relation. It is known that the stable marriages generate a distributive lattice under the relation of "domination" [K], and as pointed out in [GILS], this is the essential key to the efficient representation of the set of stable marriages. Further, there is a good sized class of interesting combinatorial problems each of whose solution sets generate a distributive lattice over some natural relation. It then follows (see [IR] or [N]) that for each of these problems, the solution sets can be represented by a compact partial order where the solutions are in one-one correspondence with the closed subsets of the partial order, although the time needed to find the partial order and to extract the solutions differs in each case. This partial order representation has many algorithmic uses; applications in stable marriage appear in [IL], [ILG] and [GILS], but there are many other applications in other combinatorial problems (see [IR] for a good bibliography). The stable assignments are not known to be representable as closed subsets in a partial order, but their representation in  $\Pi$  (closed subsets which contain exactly one of each dual pair) is certainly related, and it raises the question of whether an algebraic study of this structure would be fruitful. In particular: what sort of algebraic structure do these closed subsets of  $\Pi$  generate, and is there a natural relation that ties them together? Is there an interesting general class of problems with this structure? Can such problems be reduced to problems of the first (seemingly simpler) type? Does this more general algebraic approach lead to more efficient or generalizable algorithms?

**History and acknowledgments.** The structure  $\Pi^*$ , the material in §§ 1-5.2, and in §§ 7 and 8, were first derived and presented in [G2]. That report then used  $\Pi^*$  to efficiently reduce the roommate problem to the marriage problem. However, the "proof" of that reduction was in error, and the main corollary of the reduction, the claimed "partition theorem," is not true. I thank Sally Floyd for first catching this error and Robert Irving for finding the same error. The present paper, with its focus on enumeration, first appeared as [G3]. Independently, Robert Irving [I1] developed a different approach leading to a structure similar to  $\Pi^*$ , and an enumeration method similar to the method presented in this paper. His report also gives a solution to the minimum regret stable roommate problem.

I want to thank the students in the fall 1985 Yale computer science 260 class, who generated many useful execution trees at the beginning of this research; the trees of A. Zoler and E. Winters were particularly helpful, the latter demonstrating a counter-

example to an early conjecture. I thank David Warren who acted as a sounding-board, and who contributed the version of Phase 1 of Algorithm I that appears in this paper. Particular thanks go to Dana Angluin, who listened to many parts of this work as it developed. Finally, I thank Carrie Shepard, my stable roommate and partner in stable marriage.

## REFERENCES

- [GS] D. GALE AND L. SHAPLEY, *College admissions and the stability of marriage*, Amer. Math. Monthly, 69 (1962), pp. 9–15.
- [GS84] D. GALE AND M. SOTOMAYOR, *Some remarks on the stable matching problem*, Discrete Appl. Math, 11 (1985), pp. 223–232.
- [G] D. GUSFIELD, *Three fast algorithms for four problems in stable marriage*, SIAM J. Comput. 16 (1987), pp. 111–128.
- [G1] ———, *Three fast algorithms for four problems in stable marriage*, Tech. Report TR-407, Dept. of Computer Science, Yale University, New Haven, CT, July 1985.
- [G2] ———, *Roommate stability leads to marriage: the structure of the stable roommate problem*, Tech. Report TR-435, Dept. of Computer Science, Yale University, New Haven, CT, November 1985.
- [G3] ———, *The structure of the stable roommate problem: efficient representation and enumeration of all stable assignments*, Tech. Report 482, Dept. of Computer Science, Yale University, New Haven, CT, June 1986.
- [GILS] D. GUSFIELD, R. IRVING, P. LEATHER, AND M. SAKS, *Every finite distributive lattice is a set of stable matchings for a small stable marriage*, J. Combin. Theory, Ser. A, 44 (1987), pp. 304–309.
- [I] R. IRVING, *An efficient algorithm for the stable room-mates problem*, J. Algorithms, 6 (1985) pp. 577–595.
- [I1] ———, *On the stable room-mates problem*, Res. Report CSC/86/R5, Department of Computing Science, University of Glasgow, July 1986.
- [IL] R. IRVING AND P. LEATHER, *The complexity of counting stable marriages*, SIAM J. Comput., 15 (1986), pp. 655–667.
- [ILG] R. IRVING, P. LEATHER, AND D. GUSFIELD, *An efficient algorithm for the “optimal” stable marriage*, J. Assoc. Comput. Mach., 34 (1987), pp. 532–543.
- [IR] M. IRI, *Structural theory for the combinatorial systems characterized by submodular functions*, in Progress in Combinatorial Optimization, Academic Press, Canada, 1984.
- [K] D. E. KNUTH, *Mariages Stables*, Les Presses de L’Université de Montreal, 1976. (In French.)
- [L] E. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.
- [LLR] E. LAWLER, J. K. LENSTRA, AND A. H. G. RINNOOY-KAN, *Generating all maximal independent sets: NP-hardness and polynomial-time algorithms*, preprint.
- [MW] D. MCVITIE AND L. B. WILSON, *The stable marriage problem*, CACM, 14 (1971), pp. 486–490.
- [N] M. NAKAMURA, *Boolean sublattices connected with minimization problems on matroids*, Math. Programming, 22 (1982), pp. 117–120.
- [PTW] G. PÓLYA, R. E. TARJAN, AND D. R. WOODS, *Notes on Introductory Combinatorics*, Birkhäuser-Verlag, Boston, 1983.

## PARALLEL MERGE SORT\*

RICHARD COLE†

**Abstract.** We give a parallel implementation of merge sort on a CREW PRAM that uses  $n$  processors and  $O(\log n)$  time; the constant in the running time is small. We also give a more complex version of the algorithm for the EREW PRAM; it also uses  $n$  processors and  $O(\log n)$  time. The constant in the running time is still moderate, though not as small.

**Key words.** sort, merge, parallel algorithm, sample

**AMS(MOS) subject classifications.** 68E05, 68C05, 68C25

**1. Introduction.** There are a variety of models in which parallel algorithms can be designed. For sorting, two models are usually considered: circuits and the PRAM; the circuit model is the more restrictive. An early result in this area was the sorting circuit due to Batcher [B-68]; it uses time  $\frac{1}{2} \log^2 n$ . More recently, Ajtai et al. [AKS-83] gave a sorting circuit that ran in  $O(\log n)$  time; however, the constant in the running time was very large (we will refer to this as the AKS network). The huge size of the constant is due, in part, to the use of expander graphs in the circuit. The recent result of Lubotzky et al. [LPS-86] concerning expander graphs may well reduce this constant considerably; however, it appears that the constant is still large [CO-86], [Pa-87].

The PRAM provides an alternative, and less restrictive, computation model. There are three variants of this model that are frequently used: the CRCW PRAM, the CREW PRAM, and the EREW PRAM; the first model allows concurrent access to a memory location both for reading and writing, the second model allows concurrent access only for reading, while the third model does not allow concurrent access to a memory location. A sorting circuit can be implemented in any of these models (without loss of efficiency).

Preparata [Pr-78] gave a sorting algorithm for the CREW PRAM that ran in  $O(\log n)$  time on  $(n \log n)$  processors; the constant in the running time was small. (In fact, there were some implementation details left incomplete in this algorithm; this was rectified by Borodin and Hopcroft in [BH-85].) Preparata's algorithm was based on a merging procedure given by Valiant [V-75]; this procedure merges two sorted arrays, each of length at most  $n$ , in time  $O(\log \log n)$  using a linear number of processors. When used in the obvious way, Valiant's procedure leads to an implementation of merge sort on  $n$  processors using  $O(\log n \log \log n)$  time. More recently, Kruskal [K-83] improved this sorting algorithm to obtain a sorting algorithm that ran in time  $O(\log n \log \log n / \log \log \log n)$  on  $n$  processors. (The basic algorithm was Preparata's; however, a different choice of parameters was made.) In part, Kruskal's algorithm depended on using the most efficient versions of Valiant's merging algorithm; these are also described in Kruskal's paper.

More recently, Bilardi and Nicolau [BN-86] gave an implementation of bitonic sort on the EREW PRAM that used  $n/\log n$  processors and  $O(\log^2 n)$  time. The constant in the running time was small.

---

\* Received by the editors May 10, 1987; accepted for publication (in revised form) July 21, 1987. This work was supported in part by an IBM Faculty Development Award, by National Science Foundation grants DCR-84-01633 and CCR-8702271, and by Office of Naval Research grant N00014-85-K-0046. This is a revised version of the paper "Parallel Merge Sorts," appearing in Proceedings of the 27th Annual Symposium on Foundations of Computer Science, October 27-29, 1986, Toronto, Canada, © 1986 by IEEE.

† Courant Institute of Mathematical Sciences, New York University, New York, New York 10012.

In the next section, we describe a simple CREW PRAM sorting algorithm that uses  $n$  processors and runs in time  $O(\log n)$ ; the algorithm performs just  $5/2n \log n$  comparisons. In § 3, we modify the algorithm to run on the EREW PRAM. The algorithm still runs in time  $O(\log n)$  on  $n$  processors; however, the constant in the running time is somewhat less small than for the CREW algorithm. We note that apart from the AKS sorting network, the known deterministic EREW sorting algorithms that use about  $n$  processors all run in time  $O(\log^2 n)$  (these algorithms are implementations of the various sorting networks such as Batcher's sort). Our algorithms will not make use of expander graphs or any related constructs; this avoids the huge constants in the running time associated with the AKS construction.

The contribution of this work is twofold: first, it provides a second  $O(\log n)$  time,  $n$  processor parallel sorting algorithm (the first such algorithm is implied by the AKS sorting circuit); second, it considerably reduces the constant in the running time (by comparison with the AKS result). Of course, AKS is a sorting circuit; this work does not provide a sorting circuit.

In § 4, we show how to modify the CREW algorithm to obtain CRCW sorting algorithms that run in sublogarithmic time. We will also mention some open problems concerning sorting on the PRAM model in sublogarithmic time. In § 5, we consider a parametric search technique due to Megiddo [M-83]; we show that the partial improvement of this technique in [C-87b] is enhanced by using the EREW sorting algorithm.

**2. The CREW algorithm.** By way of motivation, let us consider the natural tree-based merge sort. Consider an algorithm for sorting  $n$  numbers. For simplicity, suppose that  $n$  is a power of 2, and all the items are distinct. The algorithm will use an  $n$ -leaf complete binary tree. Initially, the inputs are distributed one per leaf. The task, at each internal node  $u$  of the tree, is to compute the sorted order for the items initially at the leaves of the subtree rooted at  $u$ . The computation proceeds up the tree, level by level, from the leaves to the root, as follows. At each node we compute the merge of the sorted sets computed at its children. Use of the  $O(\log \log n)$  time,  $n$  processor merging algorithm of Valiant, will yield an  $O(\log n \log \log n)$  time,  $n$  processor sorting algorithm. In fact, we know there is an  $\Omega(\log \log n)$  time lower bound for merging two sorted arrays of  $n$  items using  $n$  processors [BH-85]; thus we do not expect this approach to lead to an  $O(\log n)$  time,  $n$  processor sorting algorithm.

We will not use the fast  $O(\log \log n)$  time merging procedure; instead, we base our algorithm on an  $O(\log n)$  time merging procedure, similar to the one described in the next few sentences. The problem is to merge two sorted arrays of  $n$  items. We proceed in  $\log n$  stages. In the  $i$ th stage, for each array, we take a sorted sample of  $2^{i-1}$  items, comprising every  $n/2^{i-1}$ th item in the array. We compute the merge of these two samples. Given the results of the merge from the  $i-1$ st stage, the merge in the  $i$ th stage can be computed in constant time (this, or rather a related result, will be justified later).

At present, this merging procedure merely leads to an  $O(\log^2 n)$  time sorting algorithm. To obtain an  $O(\log n)$  time sorting algorithm we need the following key observation:

The merges at the different levels of the tree can be pipelined.

This is plausible because merged samples from one level of the tree provide fairly good samples at the next level of the tree. Making this statement precise is the key to the CREW algorithm.

We now describe our sorting algorithm. The inputs are placed at the leaves of the tree. Let  $u$  be an internal node of the tree. The task, at node  $u$ , is to compute  $L(u)$ , the sorted array that contains the items initially at the leaves of the subtree rooted at  $u$ . At intermediate steps in the computation, at node  $u$ , we will have computed  $UP(u)$ , a sorted subset of the items in  $L(u)$ ;  $UP(u)$  will be stored in an array also. The items in  $UP(u)$  will be a rough sample of the items in  $L(u)$ . As the algorithm proceeds, the size of  $UP(u)$  increases, and  $UP(u)$  becomes a more accurate approximation of  $L(u)$ . (Note that at each stage we use a different array for  $UP(u)$ .)

We explain the processing performed in one stage at an arbitrary internal node  $u$  of the tree. The array  $UP(u)$  is the array at hand at the start of the stage;  $NEWUP(u)$  is the array at hand at the start of the next stage, and  $OLDUP(u)$  is the array at hand at the start of the previous stage, if any. Also, in each stage, we will create an array  $SUP(u)$  (short for  $SAMPLEUP(u)$ ) at node  $u$ ;  $NEWSUP(u)$ ,  $OLDSUP(u)$  are the corresponding arrays in respectively, the next, and previous, stage.  $SUP(u)$  is a sorted array comprising every fourth item in  $UP(u)$ , measured from the right end; i.e., if  $|UP(u)| = m$ , then  $SUP(u)$  contains the items of rank  $m - 3 - 4i$  in  $UP(u)$ , for  $0 \leq i < \lfloor m/4 \rfloor$ . At each stage, for each node  $u$ , the computation comprises the following two phases.

- (1) Form the array  $SUP(u)$ .
- (2) Let  $v$  and  $w$  be  $u$ 's children. Compute  $NEWUP(u) = SUP(v) \cup SUP(w)$ , where  $\cup$  denotes merging.

There are some boundary cases where we need to change Phase 1. (For example, initially, the  $UP$  arrays each contain one or zero items. Thus, the  $SUP$  arrays would all be empty and the algorithm would do nothing.) In view of this, we establish the following goal: at each stage, so long as  $0 \neq |UP(u)| \neq |L(u)|$ , the size of  $NEWUP(u)$  is to be twice the size of  $UP(u)$ . At this point, some definitions will be helpful. A node is *external* if  $|UP(u)| = |L(u)|$ , and it is *inside* otherwise. Phases 1 and 2, above, are performed at each inside node. At external nodes, Phase 2 is not performed and Phase 1 is modified as follows. For the first stage in which  $u$  is external, Phase 1 is unchanged. For the second stage,  $SUP(u)$  is defined to be every second item in  $UP(u)$ , in sorted order. And for the third stage,  $SUP(u)$  is defined to be every item in  $UP(u)$ , in sorted order. It should be clear that we have achieved our goal, namely, the following lemma.

LEMMA 1. *While  $0 < |UP(u)| < |L(u)|$ ,  $|NEWUP(u)| = 2|UP(u)|$ .*

It is also clear that 3 stages after node  $u$  becomes external, node  $t$ , the parent of  $u$ , also becomes external. We conclude the following.

LEMMA 2. *The algorithm has  $3 \log n$  stages.*

It remains for us to show how to perform the merges needed for Phase 2. We will show that they can be performed in constant time using  $O(n)$  processors. This yields the  $O(\log n)$  running time for the sorting algorithm.

A few definitions will be helpful. Let  $e, f, g$  be three items, with  $e < g$ .  $f$  is *between*  $e$  and  $g$  if  $e \leq f$  and  $f < g$ ; we also say that  $e$  and  $g$  *straddle*  $f$ . Let  $L$  and  $J$  be sorted arrays. Let  $f$  be an item in  $J$ , and let  $e$  and  $g$  be the two adjacent items in  $L$  that straddle  $f$  (if necessary, we let  $e = -\infty$  or  $g = \infty$ ); then the *rank* of  $f$  in  $L$  is defined to be the rank of  $e$  in  $L$  (if  $e = -\infty$ ,  $f$  is defined to have rank 0). We define the range  $[e, g)$  to be the interval *induced* by item  $e$  (including the cases  $e = -\infty$  and  $g = \infty$ ).  $L$  is a *c-cover* of  $J$  if each interval induced by an item in  $L$  contains at most  $c$  items from  $J$ . We also say that the items from  $J$  in the range  $[e, g)$  are *contained* in  $e$ 's interval. We define  $L$  to be ranked in  $J$  (denoted  $L \rightarrow J$ ) if for each item in  $L$  we know its rank in  $J$ , and we define  $L$  and  $J$  to be *cross-ranked* (denoted  $L \times J$ ) if both  $L \rightarrow J$  and  $J \rightarrow L$ .

We will need the following observation to show that the merge can be performed in  $O(1)$  time:

$OLDSUP(v)$  is a 3-cover for  $SUP(v)$  for each node  $v$ ; as  $UP(u) = OLDSUP(v) \cup OLDSUP(w)$ , we deduce  $UP(u)$  is a 3-cover for  $SUP(v)$ ; similarly,  $UP(u)$  is a 3-cover for  $SUP(w)$ .

This will be shown in Corollary 1, below. But first, we describe the merge (Phase 2).

We need some additional information in order to perform the merge quickly. Specifically, we assume  $UP(u) \rightarrow SUP(v)$ ,  $UP(u) \rightarrow SUP(w)$  are available. Using these rankings, in Step 1 we compute  $NEWUP(u)$ , and in Step 2 we compute  $NEWUP(u) \rightarrow NEWSUP(v)$  and  $NEWUP(u) \rightarrow NEWSUP(w)$ .

*Step 1—computing  $NEWUP(u)$ .* Let  $e$  be an item in  $SUP(v)$ ; the rank of  $e$  in  $NEWUP(u) = SUP(v) \cup SUP(w)$  is equal to the sum of its ranks in  $SUP(v)$  and  $SUP(w)$ . So to compute the merge we cross-rank  $SUP(v)$  and  $SUP(w)$  (the method is given in the following two paragraphs). At this point, for each item  $e$  in  $SUP(v)$ , besides knowing its rank in  $NEWUP(u)$ , we know the two items  $d$  and  $f$  in  $SUP(w)$  that straddle  $e$ , and we know the ranks of  $d$  and  $f$  in  $NEWUP(u)$  (these will be needed in Step 2). For each item in  $NEWUP(u)$  we record whether it came from  $SUP(v)$  or  $SUP(w)$  and we record the ranks (in  $NEWUP(u)$ ) of the two straddling items from the other set.

Let  $e$  be an item in  $SUP(v)$ ; we show how to compute its rank in  $SUP(w)$ . We proceed in two substeps.

*Substep 1.* For each item in  $SUP(v)$  we compute its rank in  $UP(u)$ . This task is performed by processors associated with the items in  $UP(u)$ , as follows. Let  $y$  be an item in  $UP(u)$ . Consider the interval  $I(y)$  in  $UP(u)$  induced by  $y$ , and consider the items in  $SUP(v)$  contained in  $I(y)$  (there are at most three such items by the 3-cover property). Each of these items is given its rank in  $UP(u)$  by the processor associated with  $y$ . Substep 1 takes constant time if we associate one processor with each item in the  $UP$  array at each inside node.

*Substep 2.* (See Fig. 1.) For each item  $e$  in  $SUP(v)$  we compute the rank of  $e$  in  $SUP(w)$ . We determine the two items  $d$  and  $f$  in  $UP(u)$  that straddle  $e$ , using the rank computed in Substep 1. Suppose that  $d$  and  $f$  have ranks  $r$  and  $t$ , respectively, in  $SUP(w)$ . Then all items of rank  $r$  or less are smaller than item  $e$  (recall we assumed that all the inputs were distinct), while all items of rank greater than  $t$  are larger than item  $e$ ; thus the only items about which there is any doubt as to their size relative to  $e$  are the items with rank  $s$ ,  $r < s \leq t$ . But there are at most three such items by the 3-cover property. By means of at most two comparisons, the relative order of  $e$  and these (at most) three items can be determined. So Substep 2 requires constant time if we associate one processor with each item in each  $SUP$  array.

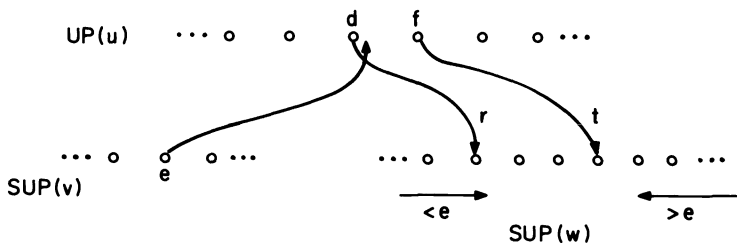


FIG. 1

*Step 2—maintaining ranks.* For each item  $e$  in  $\text{NEWUP}(u)$ , we want to determine its rank in  $\text{NEWSUP}(v)$  (and in  $\text{NEWSUP}(w)$ , using an analogous method). We start by making a few observations. Given the ranks for an item from  $\text{UP}(u)$  in both  $\text{SUP}(v)$  and  $\text{SUP}(w)$  we can immediately deduce the rank of this item in  $\text{NEWUP}(u) = \text{SUP}(v) \cup \text{SUP}(w)$  (the new rank is just the sum of the two old ranks). Similarly, we obtain the ranks for items from  $\text{UP}(v)$  in  $\text{NEWUP}(v)$ . This yields the ranks of items from  $\text{SUP}(v)$  in  $\text{NEWSUP}(v)$  (for each item in  $\text{SUP}(v)$  came from  $\text{UP}(v)$ , and  $\text{NEWSUP}(v)$  comprises every fourth item in  $\text{NEWUP}(v)$ ). Thus, for every item  $e$  in  $\text{NEWUP}(u)$  that came from  $\text{SUP}(v)$  we have its rank in  $\text{NEWSUP}(v)$ ; it remains to compute this rank for those items  $e$  in  $\text{NEWUP}(u)$  that came from  $\text{SUP}(w)$ .

Recall that for each item  $e$  from  $\text{SUP}(w)$  we computed the straddling items  $d$  and  $f$  from  $\text{SUP}(v)$  (in Step 1). (See Fig. 2.) We know the ranks  $r$  and  $t$  of  $d$  and  $f$ , respectively, in  $\text{NEWSUP}(v)$  (as asserted in the previous paragraph). Every item of rank  $r$  or less in  $\text{NEWSUP}(v)$  is smaller than  $e$ , while every item of rank greater than  $t$  is larger than  $e$ ; thus, the only items about which there is any doubt concerning their size relative to  $e$  are the items with rank  $s$ ,  $r < s \leq t$ . But there are at most three such items by the 3-cover property. As before, the relative order of  $e$  and these (at most) three items can be determined by means of at most two comparisons. Thus, Step 2 takes constant time if we associate a processor with each item in the  $\text{NEWUP}$  array at each inside node.

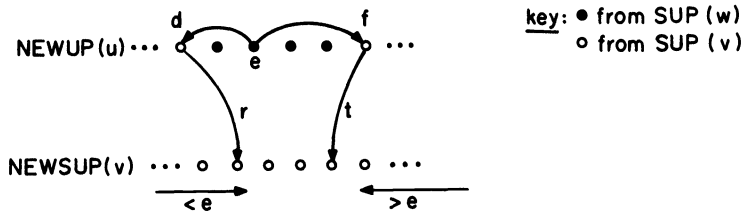


FIG. 2

It remains to prove the 3-cover property (Corollary 1 to Lemma 3) and to determine the complexity of the algorithm (Lemmas 4 and 5).

**LEMMA 3.** *Let  $k \geq 1$ . In each stage, any  $k$  adjacent intervals in  $\text{SUP}(u)$  contain at most  $2k + 1$  items from  $\text{NEWSUP}(u)$ .*

*Proof.* We prove the result by induction on the (implicit) stage number. The claim is true initially, for when  $\text{SUP}(u)$  first becomes nonempty, it contains one item and  $\text{NEWSUP}(u)$  contains two items, and when  $\text{SUP}(u)$  is empty,  $\text{NEWSUP}(u)$  contains at most one item.

*Inductive step.* We seek to prove that  $k$  adjacent intervals in  $\text{SUP}(u)$  contain at most  $2k + 1$  items from  $\text{NEWSUP}(u)$ , assuming that the result is true for the previous stage, i.e., that for all nodes  $u'$ , for all  $k' \geq 1$ ,  $k'$  intervals in  $\text{OLDSUP}(u')$  contain at most  $2k' + 1$  items from  $\text{SUP}(u')$ .

We first suppose that  $u$  is not external at the start of the current stage. (See Fig. 3.) Consider a sequence of  $k$  adjacent intervals in  $\text{SUP}(u)$ ; they cover the same range as some sequence of  $4k$  adjacent intervals in  $\text{UP}(u)$ . Recall that  $\text{UP}(u) = \text{OLDSUP}(v) \cup \text{OLDSUP}(w)$ . The  $4k$  intervals in  $\text{UP}(u)$  overlap some  $h \geq 1$  adjacent intervals in  $\text{OLDSUP}(v)$  and some  $j \geq 1$  adjacent intervals in  $\text{OLDSUP}(w)$ , with  $h + j = 4k + 1$ . The  $h$  intervals in  $\text{OLDSUP}(v)$  contain at most  $2h + 1$  items from  $\text{SUP}(v)$ , by the inductive hypothesis, and likewise, the  $j$  intervals in  $\text{OLDSUP}(w)$  contain at

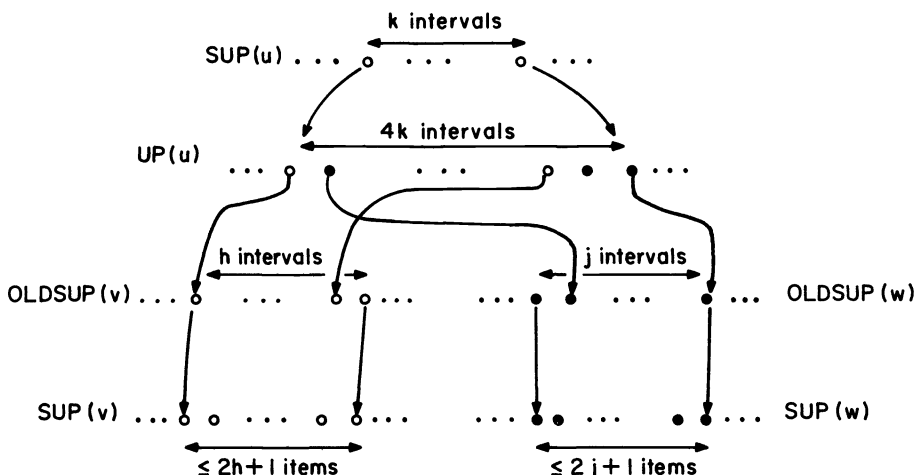


FIG. 3

most  $2j + 1$  items from  $SUP(w)$ . Recall that  $NEWSUP(u) = SUP(v) \cup SUP(w)$ . Thus the  $4k$  intervals in  $UP(u)$  contain at most  $2h + 2j + 2 = 8k + 4$  items from  $NEWSUP(u)$ . But  $NEWSUP(u)$  comprises every fourth item in  $NEWSUP(u)$ ; thus the  $k$  adjacent intervals in  $SUP(u)$  contain at most  $2k + 1$  items from  $NEWSUP(u)$ .

It remains to prove the lemma for the first and second stages in which  $u$  is external (for the third stage in which  $u$  is external there is no  $NEWSUP(u)$  array, and hence no  $NEWSUP(u)$  array). Here we can make the following stronger claim concerning the relationship between  $SUP(u)$  and  $NEWSUP(u)$ :  $k$  adjacent intervals in  $SUP(u)$  contain exactly  $2k$  items from  $NEWSUP(u)$  and every item in  $SUP(u)$  occurs in  $NEWSUP(u)$ . This is readily seen. Consider the first stage in which  $u$  is external.  $SUP(u)$  comprises every fourth item in  $UP(u) = L(u)$  and  $NEWSUP(u)$  comprises every second item in  $UP(u)$ . Clearly the claim is true for this stage; the argument is similar for the second stage.  $\square$

Taking  $k = 1$  we obtain the following.

**COROLLARY 1.**  $SUP(u)$  is a 3-cover of  $NEWSUP(u)$ .

*Remark.* An attempt to prove Lemma 3, in the same way, with a sampling strategy of every second (rather than every fourth) item will not succeed. This explains why we chose the present sampling strategy. It is not the only strategy that will work (another possibility is to sample every eighth item, or even to use a mixed strategy, such as using samples comprising every second and every fourth item, respectively, at alternate levels of the tree); however, the present strategy appears to yield the best constants.

We turn to the analysis of the algorithm. We start by computing the total number of items in the UP arrays. If  $|UP(u)| \neq 0$  and  $v$  is not external, then  $2|UP(u)| = |NEWSUP(u)| = |SUP(v)| + |SUP(w)| = \frac{1}{4}(|UP(v)| + |UP(w)|) = \frac{1}{2}|UP(v)|$ ; that is:

*Observation.*  $|UP(u)| = \frac{1}{4}|UP(v)|$ . So the total size of the UP arrays at  $u$ 's level is  $\frac{1}{8}$  of the size of the UP arrays at  $v$ 's level, if  $v$  is not external.

The observation need not be true at external nodes  $v$ . It is true for the first stage in which  $v$  is external; but for the second stage,  $|UP(u)| = \frac{1}{2}|UP(v)|$ , and so the total size of the UP arrays at  $u$ 's level is  $\frac{1}{4}$  of the total size of the arrays at  $v$ 's level; likewise, for the third stage,  $|UP(u)| = |UP(v)|$ , and so the total size of the UP arrays at  $u$ 's level is  $\frac{1}{2}$  of the total size of the UP arrays at  $v$ 's level.



Thus, on the first stage in which  $v$  is external, the total size of the UP arrays is bounded above by  $n + n/8 + n/64 + \dots = n + n/7$ ; on the second stage, by  $n + n/4 + n/32 + \dots = n + 2n/7$ ; on the third stage, by  $n + n/2 + n/16 + \dots = n + 4n/7$ . Similarly, on the first stage, the total size of the SUP arrays (and hence of the NEWUP arrays at inside nodes) is bounded above by  $n/4 + n/32 + n/256 + \dots = 2n/7$ ; on the second stage, by  $n/2 + n/16 + n/128 + \dots = 4n/7$ ; on the third stage, by  $n + n/8 + n/64 + \dots = 8n/7$ .

We conclude that the algorithm needs  $O(n)$  processors (so as to have a processor standing by each item in the UP, SUP, and NEWUP arrays) and takes constant time. Let us count precisely how many comparisons the algorithm performs.

LEMMA 4. *The algorithm performs  $15/4n \log n$  comparisons.*

*Proof.* Comparisons are performed in Substep 2 of Step 1 and in Step 2. In Step 1, at most 2 comparisons are performed for each item in each SUP array. Over a sequence of three successive stages this is  $2 \cdot (2n/7 + 4n/7 + 8n/7) = 4n$  comparisons. In Step 2, at most 2 comparisons are performed for each item in the NEWUP array at each inside node. Over a sequence of three successive stages this is also  $4n$  comparisons. However, we have overcounted here; on the third stage, in which node  $u$  becomes external, we do not perform any comparisons for items in NEWUP( $u$ ); this reduces the cost of Step 2 to  $2n$  comparisons.

So we have a total of at most  $6n$  comparisons for any three successive stages. However, we are still overcounting; we have not used the fact that during the second and third stages in which node  $v$  is external, SUP( $v$ ) is a 2-cover of NEWSUP( $v$ ) and every item in SUP( $v$ ) occurs in NEWSUP( $v$ ) (see the proof of Lemma 3). This implies that in Step 1, for each item in array SUP( $v$ ), in the second or third stage in which  $v$  is external, at most one comparison need be made (and not two). This reduces the number of comparisons in Step 1, over a sequence of three stages, by  $n/2 + n = 3/2n$ . Likewise, in Step 2, for each item in array NEWUP( $u$ ), in the first or second stages in which the children of  $u$  are external nodes, at most one comparison is performed. This reduces the number of comparisons in Step 2, over a sequence of three stages, by  $n/4 + n/2 = 3/4n$ . Thus the total number of comparisons, over the course of three successive stages, is  $5/2n$  for Step 1, and  $5/4n$  for Step 2, a total of  $15/4n$  comparisons.  $\square$

In order to reduce the number of comparisons to  $5/2n \log n$ , we need to modify the algorithm slightly. More specifically, we modify Step 1, as follows, so that it performs a total of  $5/4n \log n$  comparisons, rather than  $5/2n \log n$  comparisons. When computing the rank of each item from SUP( $v$ ) (respectively, SUP( $w$ )) in SUP( $w$ ) (respectively, SUP( $v$ )), we will allow only the items in SUP( $v$ ) to perform comparisons (or rather, processors associated with these items). We compute the ranks for items from SUP( $v$ ) as before. To obtain the ranks for items from SUP( $w$ ) we need to change both substeps. We change Substep 1 as follows. For each item  $h$  in SUP( $w$ ), we compute its rank  $r$  in UP( $u$ ) as before (the old Substep 1). Let  $k$  be the item of rank  $r$  in UP( $u$ ), and let  $s$  be the rank of  $k$  in SUP( $v$ ). We also store the rank  $s$  with item  $h$ .  $s$  is a good estimate of the rank of  $h$  in SUP( $v$ ); it is at most three smaller than the actual rank. We change Substep 2 as follows. (See Fig. 4.) Item  $e$  in SUP( $v$ ), of rank  $t$ , communicates its rank to the following, at most three, receiving items in SUP( $w$ ): those items with rank  $t$  in SUP( $v$ ) which at present store a smaller estimate for this rank. (These items are determined as follows. Let  $d$  and  $f$  be the items from UP( $u$ ) that straddle  $e$ . Let  $g$  be the successor of  $e$  in SUP( $v$ ). The receiving items are exactly those items straddled both by  $e$  and  $g$ , and by  $d$  and  $f$ ; the second constraint implies that there are at most three receiving items for  $e$ , by the 3-cover property.) For those items  $h$  in SUP( $w$ ) that

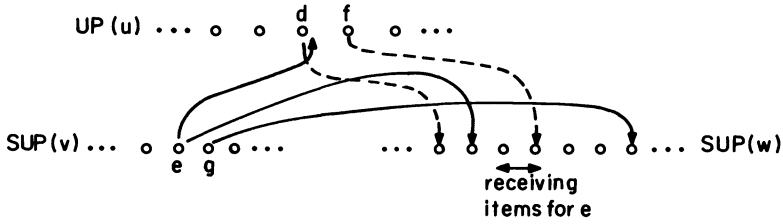


FIG. 4

do not receive a new rank from an item in  $SUP(v)$ , the rank  $s$  computed in the modified Substep 1 is the correct rank.

This new procedure reduces the number of comparisons in Step 1 by a factor of 2, and leaves unaltered the number of comparisons in Step 2. We conclude the following.

LEMMA 5. *The algorithm performs  $5/2n \log n$  comparisons.*

We have shown the following.

THEOREM 1. *There is a CREW PRAM sorting algorithm that runs in  $O(\log n)$  time on  $n$  processors, performing at most  $5/2n \log n$  comparisons.*

Remark. The algorithm needs only  $O(n)$  space. For although each stage requires  $O(n)$  space, the space can be reused from stage to stage.

**3. The EREW algorithm.** The algorithm from § 2 is not EREW at present. While it is possible to modify Step 1 of a phase so that it runs in constant time on an EREW PRAM, the same does not appear to be true for Step 2. Since we use a somewhat different merging procedure here, we will not explain how Step 1 can be modified. However, we do explain the difficulty faced in making Step 2 run in constant time on an EREW PRAM. The explanation follows. Consider  $NEWUP(u)$  and consider  $e$  and  $g$ , two items adjacent in  $SUP(v)$ ; suppose that in  $NEWUP(u)$ , between  $e$  and  $g$ , there are many items  $f$  from  $SUP(w)$ . Let  $f'$  be an item in  $NEWSUP(v)$ , between  $e$  and  $g$ . (See Fig. 5.) The difficulty is that for each item  $f$  we have to decide the relative order of  $f$  and  $f'$ ; furthermore, the decision must be made in constant time, without read conflicts, for every such item  $f$ . This cannot be done. To obtain an optimal logarithmic time EREW sorting algorithm we need to modify our approach. Essentially, the modification causes this difficult computation to become easy by precomputing most of the result.

We now describe the EREW algorithm precisely. We use the same tree as for the CREW algorithm. At each node  $v$  of the tree we maintain two arrays:  $UP(v)$  (defined as before), and  $DOWN(v)$  (to be defined). We define the array  $SUP(v)$  as before; we introduce a second sample array,  $SDOWN(v)$ : it comprises every fourth item in

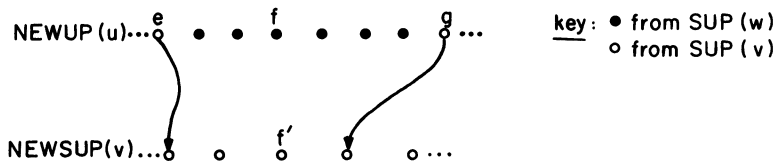


FIG. 5

$\text{DOWN}(v)$ . Let  $u$ ,  $w$ ,  $x$ , and  $y$  be, respectively, the parent, sibling, left child, and right child of  $v$ . A stage of the algorithm comprises the following three steps, performed at each node  $v$ .

- (1) Form the arrays  $\text{SUP}(v)$  and  $\text{SDOWN}(v)$ .
- (2) Compute  $\text{NEWUP}(v) = \text{SUP}(x) \cup \text{SUP}(y)$ .
- (3) Compute  $\text{NEWDOWN}(v) = \text{SUP}(w) \cup \text{SDOWN}(u)$ .

We will need to maintain some other arrays in order to perform the merges in constant time; namely, the arrays  $\text{UP}(v) \cup \text{SDOWN}(v)$  and  $\text{SUP}(v) \cup \text{SDOWN}(v)$ . It is useful to note that  $\text{SDOWN}(v)$  is a 3-cover of  $\text{NEWDOWN}(v)$ ; the proof of this result is identical to the proof of the 3-cover property for the  $\text{SUP}$  arrays (given in Lemma 3 and Corollary 1).

We describe the new merging procedure. Assume that  $J$  and  $K$  are two sorted arrays of distinct items,  $J$  and  $K$  having no items in common. We show how to compute  $J \times K$  in constant time using a linear number of processors (this yields  $L = J \cup K$ ), supposing that we are given the following arrays and rankings (see Fig. 6):

- (i) Arrays  $SJ$  and  $SK$  that are 3-covers for  $J$  and  $K$ , respectively.
- (ii)  $SJ \times SK$  and  $SL = SJ \cup SK$ .
- (iii)  $SK \rightarrow J$  and  $SJ \rightarrow K$ .
- (iv)  $SJ \rightarrow J$  and  $SK \rightarrow K$ .

We will also compute  $SL \rightarrow L$ .

We note that the interval  $I$  between two adjacent items,  $e$  and  $f$ , from  $SL = SJ \cup SK$  contains at most three items from each of  $J$  and  $K$ . In order to cross-rank  $J$  and  $K$ , it suffices, for each such interval, to determine the relative order of the (at most) six items it contains. To carry out this procedure we associate one processor with each interval in the array  $SL$ . The number of intervals is one larger than the number of items in this array. The cross-ranking proceeds in two substeps: for each interval  $I$  in  $SL$ , in Substep 1 we identify the two sets of (at most) 3 items contained in  $I$ , and in Substep 2 we compute the cross-rankings for the items contained in  $I$ .

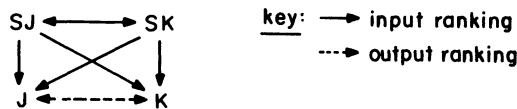


FIG. 6

**Substep 1.** The (at most) three items from  $J$  are those straddled by  $e$  and  $f$ . If  $e$  is in  $SJ$  (respectively,  $SK$ ) we determine the leftmost of these (at most) three items using  $SJ \rightarrow J$  (respectively,  $SK \rightarrow J$ ); the rightmost item is obtained similarly. The (at most) three items from  $K$  are computed analogously.

**Substep 2.** This substep is straightforward; for each interval in  $SL$ , it requires at most five comparisons, and a constant number of other operations.

**Computing  $SL \rightarrow L$ .** For each item  $e$  in  $SL$ , we simply add its ranks in  $J$  and  $K$ , which yields its rank in  $L$  (these ranks are obtained from  $SJ \rightarrow J$  and  $SJ \rightarrow K$  if  $e$  is from  $SJ$ , and from  $SK \rightarrow J$  and  $SK \rightarrow K$  if  $e$  is from  $SK$ ).

*Remark.* If  $SJ$  (respectively,  $SK$ ) is a 4-cover of  $J$  (respectively,  $K$ ) but  $SJ$  (respectively,  $SK$ ) is contained in  $J$  (respectively,  $K$ ) then (essentially) the same algorithm can be used, since the interior of any interval in  $SL$  will still contain at most three items from  $J$  and at most three items from  $K$ .

We return to the EREW sorting algorithm. We suppose that the following rankings are available at the start of a phase at each node  $v$  (see Fig. 7):

- (a)  $OLDSUP(x) \times OLDSUP(y)$ .
- (b)  $OLDSUP(v) \rightarrow SUP(v)$ .
- (c)  $OLDSUP(w) \times OLDSDOWN(u)$ .
- (d)  $OLDSDOWN(v) \rightarrow SDOWN(v)$ .
- (e)  $SUP(v) \times SDOWN(v)$ .
- (f)  $UP(v) \times SDOWN(v)$ .
- (g)  $SUP(v) \times DOWN(v)$ .

In addition, we note that since  $DOWN(v) = OLDSUP(w) \cup OLDSDOWN(u)$ , and as we have  $SUP(v) \times DOWN(v)$  from (g), we can immediately obtain:

- (h)  $OLDSUP(w) \rightarrow SUP(v)$ .
- (i)  $OLDSDOWN(u) \rightarrow SUP(v)$ .

Likewise, from  $UP(v) = OLDSUP(x) \cup OLDSUP(y)$  and from (f),  $UP(v) \times SDOWN(v)$ , we obtain:

- (j)  $OLDSUP(x) \rightarrow SDOWN(v)$ ,  $OLDSUP(y) \rightarrow SDOWN(v)$ .

The computation of (a)-(g) for the next stage at node  $v$  proceeds in five steps. In Step 1 we compute (a) and (b), in Step 2, (c) and (d), in Step 3, (e), in Step 4, (f), and in Step 5, (g).

*Remark.* We note that all the items in  $DOWN(u)$  come from outside the subtree rooted at  $u$  (this is easily verified by induction). This implies that  $SUP(w)$  and  $SDOWN(u)$  have no items in common, and likewise for  $UP(v)$  and  $DOWN(v)$ . Thus, all the cross-rankings  $J \times K$  that we compute below obey the assumption that  $J$  and  $K$  contain no items in common.

*Step 1.* Compute  $SUP(x) \times SUP(y)$  (yielding  $NEWUP(v)$ ). The computation also yields  $UP(v) \rightarrow NEWUP(v)$ , and hence  $SUP(v) \rightarrow NEWSUP(v)$ . (See Fig. 8.) We already have:

- (i)  $OLDSUP(x) \times OLDSUP(y)$  (from (a) at node  $v$ ).
- (ii)  $OLDSUP(x) \rightarrow SUP(y)$  (from (h) at node  $y$ ).
- (iii)  $OLDSUP(y) \rightarrow SUP(x)$  (from (h) at node  $x$ ).

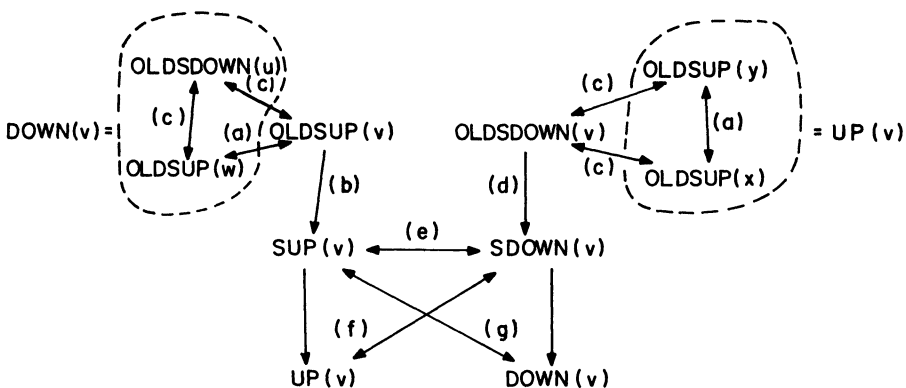


FIG. 7

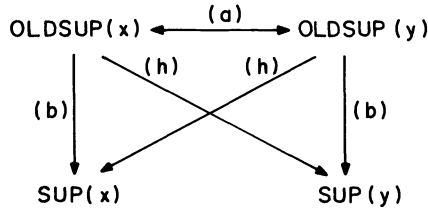


FIG. 8. Step 1.

(iv)  $OLDSUP(x) \rightarrow SUP(x)$  (from (b) at node  $x$ ).

(v)  $OLDSUP(y) \rightarrow SUP(y)$  (from (b) at node  $y$ ).

Step 2. Compute  $SUP(w) \times SDOWN(u)$  (yielding  $NEWDOWN(v)$ ). This computation also yields  $DOWN(v) \rightarrow NEWDOWN(v)$ , and hence  $SDOWN(v) \rightarrow NEWSDOWN(v)$ . (See Fig. 9.) We already have:

(i)  $OLDSUP(w) \times OLDSDOWN(u)$  (from (c) at node  $v$ ).

(ii)  $OLDSUP(w) \rightarrow SDOWN(u)$  (from (j) at node  $u$ ).

(iii)  $OLDSDOWN(u) \rightarrow SUP(w)$  (from (i) at node  $w$ ).

(iv)  $OLDSUP(w) \rightarrow SUP(w)$  (from (b) at node  $w$ ).

(v)  $OLDSDOWN(u) \rightarrow SDOWN(u)$  (from (d) at node  $u$ ).

Step 3. Compute  $NEWSUP(v) \times NEWSDOWN(v)$ . (See Fig. 10.) We already have:

(i)  $SUP(v) \times SDOWN(v)$  (from (e) at node  $v$ ).

(ii)  $SUP(v) \times NEWSDOWN(v)$ , and hence  $SUP(v) \rightarrow NEWSDOWN(v)$  (this is obtained from:  $SUP(v) \times SUP(w)$ , from Step 1 at node  $u$ , and  $SUP(v) \times SDOWN(u)$ , from Step 2 at node  $w$ , yielding  $SUP(v) \times [SUP(w) \cup SDOWN(u)] = SUP(v) \times NEWSDOWN(v)$ ).

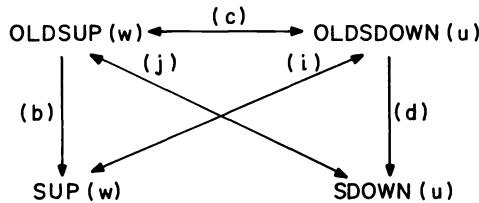


FIG. 9. Step 2.

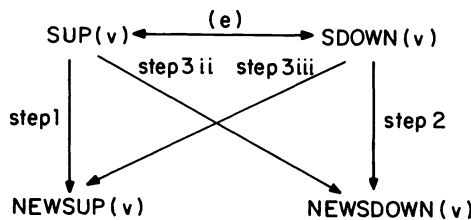


FIG. 10. Step 3.

- (iii)  $NEWSUP(v) \times SDOWN(v)$ , and hence  $SDOWN(v) \rightarrow NEWSUP(v)$  (this is obtained from  $SUP(x) \times SDOWN(v)$ , from Step 2 at node  $y$ , and  $SUP(y) \times SDOWN(v)$ , from Step 2 at node  $x$ , yielding  $[SUP(x) \cup SUP(y)] \times SDOWN(v) = NEWSUP(v) \times SDOWN(v)$ ).
  - (iv)  $SUP(v) \rightarrow NEWSUP(v)$  (from Step 1 at node  $v$ ).
  - (v)  $SDOWN(v) \rightarrow NEWSDOWN(v)$  (from Step 2 at node  $v$ ).
- Step 4. Compute  $NEWSUP(v) \times NEWSDOWN(v)$ . (See Fig. 11.) We already have:
- (i)  $NEWSUP(v) \times SDOWN(v)$  (from Step 3(iii) at node  $v$ ).
  - (ii)  $SDOWN(v) \rightarrow NEWSUP(v)$  (from Step 3(iii) at node  $v$ ).
  - (iii)  $NEWSUP(v) \rightarrow NEWSDOWN(v)$  (from Step 3 at node  $v$ ).
  - (iv)  $NEWSUP(v) \rightarrow NEWSUP(v)$ .
  - (v)  $SDOWN(v) \rightarrow NEWSDOWN(v)$  (from Step 2 at node  $v$ ).

(Here  $NEWSUP(v)$  is a 4-cover of  $NEWSUP(v)$ , contained in  $NEWSUP(v)$ ; as explained in the remark following the merging procedure, this leaves the complexity of the merging procedure unchanged.)

- Step 5. Compute  $NEWSUP(v) \times NEWSDOWN(v)$ . (See Fig. 12.) We already have:
- (i)  $SUP(v) \times NEWSDOWN(v)$  (from Step 3(ii) at node  $v$ ).
  - (ii)  $SUP(v) \rightarrow NEWSDOWN(v)$  (from Step 3(ii) at node  $v$ ).
  - (iii)  $NEWSDOWN(v) \rightarrow NEWSUP(v)$  (from Step 3 at node  $v$ ).
  - (iv)  $SUP(v) \rightarrow NEWSUP(v)$  (from Step 1 at node  $v$ ).
  - (v)  $NEWSDOWN(v) \rightarrow NEWSDOWN(v)$ .

We conclude that each stage can be performed in constant time, given one processor for each item in each array named in (i) of each step, plus one additional processor per array.

It remains to show that only  $O(n)$  processors are needed by the algorithm. This is a consequence of the following linear bound on the total size of the DOWN arrays.

LEMMA 6.  $|DOWN(v)| \leq 16/31 |SUP(v)|$ .

*Proof.* This is readily verified by induction on the stage number.  $\square$

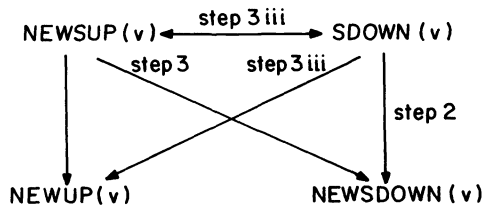


FIG. 11. Step 4.

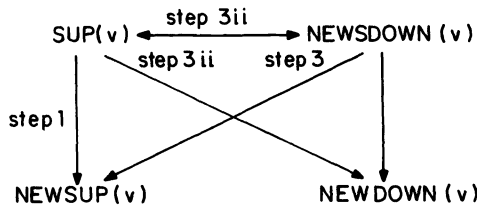


FIG. 12. Step 5.

**COROLLARY 2.** *The total size of the DOWN arrays, at any one stage, is  $O(n)$ .*

This algorithm, like the CREW algorithm, has  $3 \log n$  stages. We conclude the following.

**THEOREM 2.** *There is an EREW PRAM algorithm for sorting  $n$  items; it uses  $n$  processors and  $O(n)$  space, and runs in  $O(\log n)$  time.*

We do not give the elaborations to the merging procedure required to obtain a small total number of comparisons (as regards constants). We simply remark that it is not difficult to reduce the total number of comparisons to less than  $5n \log n$  comparisons; however, as there is no corresponding reduction in the number of other operations, this complexity bound gives a misleading impression of efficiency (which is why we do not strive to attain it).

**4. A sublogarithmic time CRCW algorithm.** References [AAV-86] and [AV-87] give tight upper and lower bounds for sorting in the parallel comparison model; using  $p \geq n$  processors to sort  $n$  items the bound is  $\Theta(\log n / \log 2p/n)$ . In addition, there is a lower bound of  $\Omega(\log n / \log \log n)$  for sorting  $n$  items with a polynomial number of processors in the CRCW PRAM model [BH-87]. We give a CRCW algorithm that uses time  $O(\log n / \log \log 2p/n)$  for  $2n \leq p \leq n^2$ . It is not clear which, if any, of the upper and lower bounds are tight for the CRCW PRAM model.

We describe the algorithm; it is very similar to the CREW algorithm. Let  $r = p/n$ . It is convenient to assume that  $n$  is a power of  $r$  (the details of the general case are left to the reader). There are three major changes to the CREW algorithm. First, rather than use a binary tree to guide the merges, we use an  $r$ -way tree. This tree has height  $h = \log n / \log r$ . Second, we define the array  $SUP(v)$  to comprise every  $r^2$  item in  $UP(v)$ , rather than every fourth item; again, at the external nodes we need a special definition, namely:  $SUP(v)$  comprises every  $r^2$  item in the first stage  $v$  is external, every  $r$ th item in the second stage, and every item in the third stage. Third, the array  $NEWUP(v)$  is defined to be the  $r$ -way merge of the  $SUP$  arrays at its  $r$  children. As before, the algorithm will have  $3h = 3 \log n / \log r$  stages. But here, rather than use constant time, each stage will take  $O(\log r / \log \log 2r)$  time.

To obtain the  $r$ -way merge we perform  $\frac{1}{2}r(r-1)$  pairwise merges of the  $r$   $SUP$  arrays. To obtain the rank of an item in the array  $NEWUP(v)$  we sum its ranks in each of the  $r$   $SUP$  arrays. We use  $r$  processors to compute this sum; they take  $O(\log r / \log \log r)$  time on a CRCW PRAM using the summation algorithm from [CV-87].

Before explaining how to perform a single pairwise merge it is useful to prove a cover property.

**LEMMA 7.**  *$k$  intervals in  $SUP(v)$  contain at most  $rk + 1$  items in  $NEWSUP(v)$ .*

*Proof.* The proof is very similar to that of Lemma 2; the details are left to the reader.  $\square$

**COROLLARY 3.**  *$SUP(v)$  is an  $(r+1)$ -cover of  $NEWSUP(v)$ .*

We perform the merges essentially as in the CREW algorithm. Here, at the start of a stage, we need to assume that for each node  $u$ , for each item in  $UP(u)$  we have its rank in the  $SUP$  arrays at all  $r$  of  $u$ 's children. Let  $v, w$  be children of  $u$ . We proceed in two steps, as in the CREW algorithm.

In Step 1, we start by dividing each pairwise merge into a collection of merging subproblems, each of size at most  $2(r+1)$ ; each subproblem is then solved using Valiant's merging algorithm [V-75]. To divide a merge into subproblems, we exploit the fact that  $UP(u)$  is an  $(r+1)$ -cover of  $SUP(v)$ , as follows. For each child  $v$  of  $u$ , we label each item in  $SUP(v)$  with its rank in  $UP(u)$  (this is carried out in constant

time by providing each item in  $UP(u)$  with  $r(r+1)$  processors). A subproblem is defined by those items labeled with the same rank; it comprises two subarrays each containing at most  $r+1$  items. For the problem of merging  $SUP(v)$  and  $SUP(w)$ , for any item  $e$  in  $SUP(v)$  (respectively,  $SUP(w)$ ) the boundaries of its subproblem can be found as follows: let  $f$  and  $g$  be the items in  $UP(u)$  straddling  $e$  (obtained using the rank of  $e$  in  $UP(u)$ ); the ranks of  $f$  and  $g$  in  $SUP(v)$ ,  $SUP(w)$  yield the boundaries of the subproblem. To ensure that the merges performed using Valiant's algorithm each take  $O(\log \log 2r)$  time, we need to provide a linear number of processors for each merge. Since each item in  $SUP(v)$  participates in exactly  $r-1$  merges, it suffices to allocate  $r-1$  processors to each item in each  $SUP$  array. This gives us  $NEWUP(u)$ .

In Step 2, ranking the items from  $NEWUP(u) - SUP(v)$  in  $NEWSUP(v)$  for each child  $v$  of  $u$ , we proceed as follows. Consider an interval  $I$  induced by an item  $e$  from  $SUP(v)$ . For each such interval  $I$ , we divide each collection of items from  $NEWUP(u) - SUP(v)$ , contained in  $I$ , into sets of  $r$  contiguous items, with possibly one smaller set per interval. Using Valiant's algorithm, we merge each such set with the at most  $r+1$  items in  $NEWSUP(v)$  contained in  $I$ . If we allocate  $r$  processors to each merging problem, they will take  $O(\log \log 2r)$  time. To allocate the processors, we assign  $2(r-1)$  processors to each item in  $NEWUP(u)$ . Each item participates in  $r-1$  merging problems. In a merging problem, if the item is part of a set of size  $r$ , the item uses one of its assigned processors. If the item is part of a set of size  $< r$ , the item takes one processor from the item  $e$  defining the interval  $I$ . Each item  $e$  contributes at most  $r-1$  processors to a merging problem in the latter manner, thus it suffices to provide  $2(r-1)$  processors to each item in  $NEWUP(u)$ .

We conclude the following.

**THEOREM 3.** *There is a CRCW sorting algorithm for the CRCW PRAM that uses  $2n \leq p \leq n^2$  processors and runs in time  $O(\log n / \log \log 2p/n)$ .*

**5. A parametric search technique.** The reader is warned that this section is not self-contained. We recall Megiddo's parametric search technique [M-83] and its improvement in many instances in [C-87b]. The improvement was an asymptotic improvement, but was not practical for it was based on the AKS sorting network. As we will explain, the role played by the AKS network can be replaced by the EREW sorting algorithm from § 3.

In a nutshell, the procedure of [C-87b] can be described as follows. A comparison-based sorting algorithm is executed; however each "comparison" is an expensive operation costing  $C(n)$  time, where  $n$  is the size of the problem at hand. In addition, the comparisons have the property that they can be "batched": given a set of  $c$  comparisons, one of them can be evaluated, and the result of this evaluation resolves further  $c/2$  comparisons, in an additional  $O(c)$  time. Examples of search problems (called *parametric search problems*), mostly geometric search problems, for which this approach is fruitful, include [M-83], [C-87a], [C-87b], [CSS-88]. Megiddo showed that parallel sorting algorithms, executed sequentially, provide good sorting algorithms for this type of problem; the reason is that a parallel sorting algorithm naturally batches comparisons.

In [C-87b] it was shown how to achieve a running time of  $O(n \log n + \log n C(n))$  for the parametric search problems, when using a depth  $O(\log n)$  sorting network as the sorting algorithm. (Briefly, the solution required  $O(\log n)$  "comparisons" to be evaluated; the overhead for running the sorting algorithm and selecting the comparisons to be evaluated was  $O(n \log n)$  time.) It was also observed that to achieve this result it sufficed to have a comparison-based algorithm which could be represented as an



$O(\log n)$  depth,  $O(n)$  width, directed acyclic graph, having bounded indegree, where each node of the graph represented a comparator and the edges carried the outputs of the comparators.

In fact, a slightly more general claim holds. We start by defining a *computation graph* corresponding to an EREW PRAM algorithm on a given input. We define a parallel EREW PRAM computation to proceed in a sequence of steps of the following form. In each step, every processor performs at most  $b$  (a constant) reads, followed by at most  $b$  writes; a constant number of arithmetic operations and comparisons are intermixed (in any order) with the reads and writes. We represent the computation of the algorithm on a given input as a computation graph; the graph has depth  $2T$  (where  $T$  is the running time of the algorithm) and width  $M + P$  (where  $M$  is the space used by the algorithm and  $P$  is the number of processors used by the algorithm). In the computation graph each node represents either a memory cell at a given step, or a processor at a given step. There is a directed edge  $(\langle m, t \rangle, \langle p, t \rangle)$  if processor  $p$  reads memory cell  $m$  at step  $t$ ; likewise there is a directed edge  $(\langle p, t \rangle, \langle m, t + 1 \rangle)$  if processor  $p$  writes to memory cell  $m$  at step  $t$ . If no write is made to memory cell  $m$  at step  $t$ , there is a directed edge  $(\langle m, t \rangle, \langle m, t + 1 \rangle)$ .

Suppose we restrict our attention to algorithms such that at the start of the algorithm, for each memory cell (at step  $t + 1$ ) we know whether the in-edge (in the computation graph) comes from a processor (at step  $t$ ) or a memory cell (at step  $t$ ). For a sorting algorithm of this type, that runs in time  $O(\log n)$  on  $n$  processors using  $O(n)$  space, we can still achieve a running time of  $O(n \log n + \log n C(n))$  for the parametric search problems. (To understand this, it is necessary to read [C-87b, §§ 1–3]. The reason the result holds is that we can determine when a memory cell is active, to use the terminology of [C-87b], and thus play the game of § 2 of [C-87b] on the computation graph. In general, if we do not have the condition on the in-edges for memory cells, it is not clear how to determine if a memory cell is active. As explained in § 3 of [C-87b], given a solution to the game of § 2, we can readily obtain a solution to the parametric search problem).

*Remark.* The computation graph need not be the same for all inputs of size  $n$ . In addition, the graph does not have to be explicitly known at the start of the sorting algorithm.

Next, we show that the EREW sorting algorithm satisfies the conditions of the previous paragraph. Each of the five steps for one stage of the EREW algorithm comprises the computation of the cross-ranks of two arrays. The computation of the cross-ranks proceeds in two substeps; in the first substep, each processor performs a constant number of reads; in the second substep, each processor performs a constant number of writes.

We conclude that the result of [C-87b] can be achieved with a much smaller constant, thereby making the paradigm less impractical.

**Acknowledgments.** Thanks to Richard Anderson and Allen Van Gelder for commenting on an earlier version of this result. In particular, Allen Van Gelder suggested the term *cross-rank*. Many thanks to Jeannette Schmidt for questions and comments on several earlier versions of the result. Thanks also to Alan Siegel and Michelangelo Grigni for suggestions that simplified the presentation. Finally, I am grateful to Zvi Kedem, Ofer Zajicek, and Wayne Berke, all of whom read the penultimate version of the paper and provided a variety of useful comments. I am also very grateful to the referee for pointing out a serious error in the original version of § 3.

## REFERENCES

- [AKS-83] M. AJTAI, J. KOMLOS, AND E. SZEMEREDI, *An  $O(n \log n)$  sorting network*, *Combinatorica*, 3 (1983), pp. 1-19.
- [AAV-86] N. ALON, Y. AZAR, AND U. VISHKIN, *Tight complexity bounds for parallel comparison sorting*, Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 502-510.
- [AV-87] Y. AZAR AND U. VISHKIN, *Tight comparison bounds on the complexity of parallel sorting*, *SIAM J. Comput.*, 3 (1987), pp. 458-464.
- [B-68] K. E. BATCHER, *Sorting networks and their applications*, Proc. AFIPS Spring Joint Summer Computer Conference, 1968, pp. 307-314.
- [BH-85] A. BORODIN AND J. HOPCROFT, *Routing, merging and sorting on parallel models of computation*, *J. Comput. Syst. Sci.*, 30 (1985), pp. 130-145.
- [BH-87] P. BEAME AND J. HASTAD, *Optimal bounds for decision problems on the CRCW PRAM*, Proc. 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 83-93.
- [BN-86] G. BILARDI AND A. NICOLAU, *Bitonic sorting with  $O(n \log n)$  comparisons*, Proc. 20th Annual Conference on Information Sciences and Systems, 1986, pp. 336-341.
- [C-87a] R. COLE, *Partitioning point sets in arbitrary dimension*, *Theoret. Comput. Sci.*, 49 (1987), pp. 239-265.
- [C-87b] ———, *Slowing down sorting networks to obtain faster sorting algorithms*, *J. Assoc. Comput. Mach.*, 34 (1987), pp. 200-208.
- [CO-86] R. COLE AND C. O'DUNLAING, *Notes on the AKS sorting network*, Computer Science Dept. Tech. Report #243, Courant Institute of Mathematical Sciences, New York University, New York, 1986.
- [CSS-88] R. COLE, J. SALOWE, AND W. L. STEIGER, *Optimal slope selection*, to appear, Proc. 15th ICALP, Tampere, Finland, 1988.
- [CV-87] R. COLE AND U. VISHKIN, *Faster optimal prefix sums and list ranking*, Computer Science Department Tech. Report #277, Courant Institute of Mathematical Sciences, New York University, New York, 1987; *Inform. and Computation*, to appear.
- [LPS-86] A. LUBOTZKY, R. PHILLIPS, AND P. SARNAK, *Ramanujan conjecture and explicit constructions of expanders and super-concentrators*, Proc. 18th Annual Symposium on Theory of Computing, 1986, pp. 240-246.
- [K-83] C. KRUSKAL, *Searching, merging, and sorting in parallel computation*, *IEEE Trans. Comput.*, 32 (1983), pp. 942-946.
- [M-83] N. MEGIDDO, *Applying parallel computation algorithms in the design of serial algorithms*, *J. Assoc. Comput. Mach.*, 30 (1983), pp. 852-865.
- [Pa-87] M. S. PATERSON, *Improved sorting networks with  $O(\log n)$  depth*, Dept. of Computer Science Res. Report RR89, University of Warwick, England, 1987.
- [Pr-78] F. P. PREPARATA, *New Parallel-Sorting Schemes*, *IEEE Trans. Comput.*, C-27 (1978), pp. 669-673.
- [V-75] L. VALIANT, *Parallelism in comparison problems*, *SIAM J. Comput.*, 4 (1975), pp. 348-355.

## A NATURAL NP-COMPLETE PROBLEM WITH A NONTRIVIAL LOWER BOUND\*

ETIENNE GRANDJEAN†

**Abstract.** Let  $SAT_{<}(\mathbb{N})$  denote the following problem. *Instance.* A conjunction  $\varphi$  of (in)equalities  $t_1 = t_2$  or  $t_1 < t_2$ , where  $t_1, t_2$  are terms of the form  $f_1 f_2 \cdots f_s(e)$ , where  $e \in \mathbb{N}$ ,  $s \geq 0$  and each  $f_i$  is a monadic function symbol. *Question.* Is  $\varphi$  satisfiable on  $\mathbb{N}$ ?

Let  $SAT_{<}^{2,2}(\mathbb{N})$  denote the following subproblem of  $SAT_{<}(\mathbb{N})$  defined by the following restriction: we assume that  $0 \leq s \leq 2$  and each  $f_i \in \{g_1, g_2\}$ . These two problems are NP-complete. We show that they are solved by a Turing machine using a polynomial number of deterministic steps and only  $n$  nondeterministic steps. This is nearly optimal since we prove that any problem in  $NTIME(n)$  is reducible in deterministic time  $O(n)$  to  $SAT_{<}(\mathbb{N})$  (respectively,  $SAT_{<}^{2,2}(\mathbb{N})$ ). It follows from the result  $\cup_c DTIME(cn) \subsetneq NTIME(n)$  of Paul et al. [*Proc. 24th Annual IEEE Symposium on Foundations of Computer Sciences*, 1983, pp. 429-438] that these problems are not in  $\cup_c DTIME(cn)$ . Further, we show that  $SAT_{<}^{2,2}(\mathbb{N})$  and  $SAT_{<}(\mathbb{N})$  belong to  $\Sigma_2(n)$ , the class of problems solved in time  $O(n)$  by alternating Turing machines using one alternation. They are the first natural problems proved to be in  $\Sigma_2(n) - \cup_c DTIME(cn)$ .

**Key words.** NP-complete problem, determinism, nondeterminism, Turing machine, alternating Turing machine, linear time reduction, random access machine, spectrum of first-order sentence

AMS(MOS) subject classification. 68Q

**1. Introduction.** In his Turing Award Lecture [Co2] S. Cook notes, "... the record for proving lower bounds on problems of small complexity is appalling. In fact there is no nonlinear time lower bound known on a general purpose computation model for any natural problem in NP, in particular, for any of the 300 problems listed in [GaJo]. Of course, one can prove by diagonal arguments the existence of problems in NP requiring time  $n^k$  for any fixed  $k$ ."

Recently, Adachi et al. [AIK] have proved lower bounds on game problems (in directed graphs) which are polynomial time complete. For instance, they show that "the cat and  $K$ -mouse game" introduced by [ChSt] is not in  $DTIME(n^{k-\epsilon})$  for any  $\epsilon > 0$  and integers  $K, k$  with  $K \geq 4k + 1 > 5$ . They essentially prove that each problem in  $DTIME(n^k)$  is reducible to the game problem in time  $O(n \log n)$ ; the result follows by the well-known hierarchy of  $DTIME$  classes (see [HoU1]). These seem to be the first natural problems in NP (in fact, in  $P = \cup_k DTIME(n^k)$ ) on which a nonlinear time lower bound is proved. However, no precise (polynomial time) upper bound is stated and we think that the stated lower bound is not optimal.

As Cook notes, it is surprising that none of the many classical NP-complete problems has a known time lower bound. One reason seems to be that the reductions of problems in NP to the generic NP-complete problems, i.e., the satisfiability of Boolean formulas (SAT) and its restriction, 3-satisfiability (3-SAT), are not efficient enough. The best-known reduction (stated by [Sr]) is as follows: there is a fixed  $k$  such that each problem in  $NTIME(n)$  is reducible to 3-SAT in deterministic time  $O(n(\log_2 n)^k)$ . Because of our lack of knowledge about relationships between  $NTIME$  classes and  $DTIME$  classes, we cannot deduce (from this reduction) any nontrivial time lower bound on problem SAT (respectively, 3-SAT). However, [PPST] obtained a very nice result about this:  $\cup_c DTIME(cn) \subsetneq NTIME(n)$ .

This is the first result which states that nondeterminism increases the power of usual (i.e., one-dimensional multitape) Turing machines. It follows that a hypothetical

\* Received by the editors October 16, 1985; accepted for publication (in revised form) July 31, 1987.

† Université de Caen, Département d'Informatique, 14032 Caen, Cedex, France.

linear time reduction of  $\text{NTIME}(n)$  to SAT (i.e., a slight improvement of the known reduction) would imply that  $\text{SAT} \notin \cup_c \text{DTIME}(cn)$ . Unhappily, such a reduction seems to be improbable (or inaccessible) for it would also imply that each problem in  $\text{NTIME}(n)$  could be solved (as problem SAT) by a Turing machine using only  $n(\log_2 n)^k$  deterministic steps and  $n/\log_2 n$  nondeterministic steps (a Boolean formula of length  $n$  contains  $O(n/\log_2 n)$  distinct variables).

In fact, some versions of the *tiling problem* (cf. [Le]) are NP-complete and can be proved to be  $\text{NTIME}(n)$ -hard (with respect to deterministic linear time reductions) by a straightforward argument. Tiling is essentially a direct encoding of a nondeterministic Turing machine (with one tape and several heads): each tile of a tiling describes one cell at one instant; so a computation of length  $n$  of a Turing machine is encoded by an  $n \times n$  tiling. It must be clear that the following problem, denoted TILE, has the desired properties:

*Instance.* A finite set of tiles  $T$  and two sequences  $T_1, T_2, \dots, T_n$  and  $T'_1, T'_2, \dots, T'_n$  of  $n$  tiles of  $T$ .

*Question.* Is there an  $n \times n$  tiling (with tiles of  $T$ ) whose first row and last row are  $T_1, T_2, \dots, T_n$  and  $T'_1, T'_2, \dots, T'_n$ , respectively? (Note.  $T_1, \dots, T_n$  and  $T'_1, \dots, T'_n$  describe the tape at the first and  $n$ th instant, respectively).

However, we think that such a (linear) lower bound result for TILE (or for any similar tiling problem) is not at all optimal and then is not very interesting: it seems that any decision algorithm of problem TILE needs  $n^2$  nondeterministic steps (in order to guess an  $n \times n$  tiling).

In the present paper, we study a new problem, denoted  $\text{SAT}_{<}(\mathbb{N})$  defined as follows.

*Instance.* A set SYMB of monadic function symbols. A conjunction  $\varphi$  of (in)equalities  $t_1 = t_2$  or  $t_1 < t_2$ , where  $t_1, t_2$  are terms of the form  $f_1 f_2 \dots f_s(e)$ , where  $s \geq 0$ , each  $f_i \in \text{SYMB}$  and  $e \in \mathbb{N}$ .

*Question.* Is there a structure  $\langle \mathbb{N}, g \rangle_{g \in \text{SYMB}}$  on domain  $\mathbb{N}$  which satisfies conjunction  $\varphi$ ?

Let  $\text{SAT}_{<}^{2,2}(\mathbb{N})$  (respectively,  $\text{SAT}_{\geq}^{2,2}(\mathbb{N})$ ) denote the following subproblem of  $\text{SAT}_{<}(\mathbb{N})$ : we assume that each  $f_i \in \{g_1, g_2\}$  (respectively,  $f_i \in \{g_1, g_2\}$ ) and  $0 \leq s \leq 2$ .

Let  $\varphi$  be an instance of  $\text{SAT}_{<}(\mathbb{N})$  of length  $n$  and  $a_0, a_1 \dots a_{m-1}$  be the list of integers which occur in  $\varphi$ . Clearly  $\varphi \in \text{SAT}_{<}(\mathbb{N})$  if and only if there is a model of  $\varphi$  (on domain  $\mathbb{N}$ ), where all terms and subterms which occur in  $\varphi$  have values in the union of intervals  $\cup_{i < m} [a_i - n, a_i + n]$ . This remark proves that problem  $\text{SAT}_{<}(\mathbb{N})$  is in NP.

Our main results (Theorems 6.3 and 6.4) are as follows: each problem in  $\text{NTIME}(n)$  is reducible to problem  $\text{SAT}_{<}(\mathbb{N})$  (respectively,  $\text{SAT}_{\geq}^{2,2}(\mathbb{N})$ ) in time  $O(n)$  on a deterministic Turing machine, in fact, a one-tape transducer (in particular problems  $\text{SAT}_{<}(\mathbb{N})$  and  $\text{SAT}_{\geq}^{2,2}(\mathbb{N})$  are NP-complete).

Hence  $\text{SAT}_{<}(\mathbb{N})$  and  $\text{SAT}_{\geq}^{2,2}(\mathbb{N})$  are not in  $\cup_c \text{DTIME}(cn)$ . Our main result seems to be optimal since these problems are solved by a Turing machine using a polynomial number of deterministic steps and only  $n$  nondeterministic steps (we also prove that  $\text{SAT}_{<}(\mathbb{N}) \in \text{NTIME}(n(\log_2 n)^2)$  and  $\text{SAT}_{\geq}^{2,2}(\mathbb{N}) \in \text{NTIME}(n \log_2 n)$ ). The linear nature of these problems is stated by the following result:  $\text{SAT}_{\geq}^{2,2}(\mathbb{N})$  and  $\text{SAT}_{<}(\mathbb{N})$  belong to class  $\Sigma_2(n)$ , i.e., are solved in time  $O(n)$  by an alternating Turing machine so that existential steps precede universal steps.

We feel that problem  $\text{SAT}_{<}(\mathbb{N})$  has an easy, natural formulation: it concerns one of the simplest classes of quantifier-free formulas interpreted over integers. The use of function symbols is essential for the NP-completeness of this problem: we can

(trivially) decide in deterministic polynomial time the satisfiability problem of a conjunction  $\varphi$  of signed atomic formulas (where individual variables are replaced by integers) if  $\varphi$  includes no function symbols but only relation symbols. Problems  $\text{SAT}_{<}(\mathbb{N})$  and  $\text{SAT}_{\leq}^{2,2}(\mathbb{N})$  are the first *natural* problems which separate classes  $\bigcup_c \text{DTIME}(cn)$  and  $\Sigma_2(n)$ ; the existence of such problems trivially follows from inclusion  $\bigcup_c \text{DTIME}(cn) \not\subseteq \text{NTIME}(n)$ .

Let us mention that the linear time reduction (by one-tape transducer) we use has been defined by Dewdney [De] who proves that some natural NP-complete problems (for instance, 3-SAT and 3-COLOURABILITY) are equivalent via this reduction.

The proof of our main result is long and rather technical; it uses intermediate results about the two following notions:

(i) The NRAM (Nondeterministic Random Access Machine), where the only arithmetical operation is the addition of unit (see [Se], [Mo1], [Mo2]).

(ii) The (generalized) spectrum of a first-order sentence.

(i) Let  $\text{NRAM}(T(n))$  denote the class of languages accepted by an NRAM in time  $T(n)$ . We use a technical version of a result of [Mo2]:

$$\text{NTIME}(T(n)) \subset \bigcup_c \text{NRAM}(cT(n)/\log_2 T(n)).$$

(Monien assumes  $T(n) \geq n \log_2 n$ ; we take  $T(n) = n$ ). The idea is essentially to use the ability of an NRAM to do operations on integers in one step and to guess an integer in one step: more precisely, by first precomputing the table of all possible  $\varepsilon \log_2 T$  moves of a (nondeterministic) Turing machine (NTM) (for a sufficiently small  $\varepsilon$ , the precomputation requires time  $O(T/\log_2 T)$ ), the NRAM simulates  $\varepsilon \log_2 T$  moves of the NTM in  $O(1)$  steps by consulting the table only once.

(ii) We use a result of [Gr2] stating that a set of positive integers (respectively, a set of finite structures) accepted by an NRAM in time  $O(m)$ , where  $m$  is the input integer (respectively, the cardinality of the domain of the input structure), is the spectrum (respectively, generalized spectrum) of a first-order sentence  $\varphi$  with only one variable and relation and function symbol of arity  $\leq 1$ .

Lastly, by unrolling the first-order sentence  $\varphi$  on domain  $m$  (it gives a formula of length  $O(m \log_2 m)$  because  $\varphi$  has only one variable) we obtain a linear time reduction to problem  $\text{SAT}_{<}(\mathbb{N})$ , i.e., our main result; we get rid of disjunctions of the first-order sentence by adding function symbols; by some encoding techniques, we state a linear time reduction to  $\text{SAT}_{\leq}^{2,2}(\mathbb{N})$ .

The structure of the paper is as follows. In § 2, we give some notation and definitions, in particular we present NRAMs. In § 3, we prove several upper bounds on problems  $\text{SAT}_{<}(\mathbb{N})$  and  $\text{SAT}_{\leq}^{2,2}(\mathbb{N})$ . The rest of the paper consists of a proof of the main result: § 4 presents the simulation of NTMs by NRAMs; in § 5, computations of NRAMs are described by one-variable first-order sentences; last, in § 6, we prove the linear time reduction of  $\text{NTIME}(n)$  to  $\text{SAT}_{<}(\mathbb{N})$  and  $\text{SAT}_{\leq}^{2,2}(\mathbb{N})$ . The proof that  $\text{SAT}_{<}(\mathbb{N}) \in \Sigma_2(n)$  is given in the Appendix.

**2. Preliminaries.** We use the usual notation and definitions in computational complexity (see [HoU1]). Our models of computation are *multitape Turing machines* (for short, TM), where every tape is one-dimensional; more precisely, a TM has one read-only tape for input, several read-write tapes (the *worktapes*) and in case it computes a function, one write-only tape for output. A deterministic (respectively, nondeterministic) TM is abbreviated as a DTM (respectively, an NTM). A *one-tape transducer* is a DTM which computes a function with only one worktape. Let  $\text{DTIME}(T(n))$  (respectively,  $\text{NTIME}(T(n))$ ) denote the class of languages accepted in time  $T(n)$  by a DTM (respectively, NTM).

We also use the *nondeterministic random access machines* (for short, NRAM) defined in [Se], [Mo1], [Mo2], [Gr2]. An NRAM consists of a finite program which operates on a sequence of registers  $R_0, R_1, R_2, \dots$ . Each register can store any natural number. The program is a finite sequence of instructions, labeled inst 0, inst 1,  $\dots$ , inst  $l$ , of the following types:

- |                            |   |
|----------------------------|---|
| (1) Read ( $R_i$ )         | (7) $R_i := \langle R_j \rangle$                                |
| (2) $R_i := 0$             | (8) $\langle R_i \rangle := R_j$                                |
| (3) $R_i := R_i + 1$       | (9) Go to inst $i_0$ or $i_1$                                   |
| (4) $R_i := R_i \dot{-} 1$ | (10) If $R_i = R_j$ then go to inst $i_0$ else go to inst $i_1$ |
| (5) Guess ( $R_i$ )        | (11) Accept   |
| (6) $R_i := R_j$           | (12) Reject   |

$\langle R_i \rangle$  denotes the register pointed to by register  $R_i$  (i.e., the address of  $\langle R_i \rangle$  is the content of  $R_i$ ). The effect of instructions (2)–(4), (6)–(8), and (10)–(12) is evident (the value of  $x \dot{-} y$  is  $x - y$  if  $x \geq y$ ; if  $x < y$ , it is 0). The control of the program is transferred from one instruction to the next one, except after instructions (9), (10). Instructions (5) and (9) are nondeterministic; the meaning of Guess ( $R_i$ ) is: guess any integer to be stored in  $R_i$ .

At the beginning of the computation of an NRAM, the control of the program points to inst 0, the content of each register is 0 and a sequence of integers  $U_0, U_1, \dots, U_{m-1}, 0$  with  $U_j > 0, j < m$ , serve as inputs. The instruction Read ( $R_i$ ) causes the NRAM to transfer the first integer  $U_j$  which has not been read in up to this time into register  $R_i$ . We assume that the execution of any instruction only requires one time unit.

A *simple* NRAM (see [Gr2]) is similar to the NRAM defined above with the following changes: it has two accumulators, i.e., special register denoted  $a$  and  $b$ , and its instructions are the following:

- |                              |   |
|------------------------------|---|
| (1) Read ( $a$ )             | (7) $b := a$  |
| (2) $a := 0$                 | (8) $\langle a \rangle := b$                                |
| (3) $a := a + 1$             | (9) Go to inst $i_0$ or $i_1$                               |
| (4) $a := a \dot{-} 1$       | (10) If $a = b$ then go to inst $i_0$ else go to inst $i_1$ |
| (5) Guess ( $a$ )            | (11) Accept   |
| (6) $a := \langle a \rangle$ | (12) Reject   |

$\langle a \rangle$  denotes the register pointed to by accumulator  $a$ . In fact, only accumulator  $a$  is used for accessing the sequence of registers (like the control head of a Turing machine).

LEMMA 2.1. *Each NRAM is simulated by a simple NRAM in linear time.*

*Proof.* The proof is easy (see [Se]).  $\square$

Let us define some generalized NRAMs. A *multitape* NRAM operates on finitely many sequences of registers  $R_0, R_1, \dots, R'_0, R'_1, \dots, R''_0, R''_1, \dots$ . A *multidimensional* NRAM has one or several multidimensional sequences of registers  $R(j_1, j_2, \dots, j_s)$ ; an address  $j_1, \dots, j_s$  is an  $s$ -tuple of natural numbers.

LEMMA 2.2. *Each multitape (respectively, multidimensional) NRAM is simulated by an ordinary NRAM in linear time.*

*Proof.* See [Mo1], [Mo2], [Gr2].  $\square$

We will use *alternating Turing machines* (see [CKS])  $M$  with a bounded number of alternations:  $M$  is a  $\Pi_1$  (respectively,  $\Sigma_2$ ) Turing machine if it has only universal

states (respectively, if in any computation of  $M$  there is only one alternation and existential steps precede universal steps).

Let  $\Pi_1(n)$  (respectively,  $\Sigma_2(n)$ ) denote the class of languages accepted by  $\Pi_1$  (respectively,  $\Sigma_2$ ) Turing machines in time  $O(n)$ .

In the proof of our main result, it is convenient to use a one-to-one representation of positive integers: the *dyadic* notation on the alphabet  $\{1, 2\}$ ; an integer  $e > 0$  such that

$$e = \sum_{i=0}^l \alpha_i 2^i,$$

where  $\alpha_i \in \{1, 2\}$ , is represented (in dyadic notation) by the word  $\alpha_l \alpha_{l-1} \cdots \alpha_1 \alpha_0$ . Similarly, we define the  $d$ -adic notation on the alphabet  $\{1, 2, \dots, d\}$  for  $d \geq 2$ .

The complexity of problem  $\text{SAT}_{<}(\mathbb{N})$  which involves a representation of integers is not modified if we prefer the usual binary notation (for example) rather than the dyadic because of the following easy lemma.

**LEMMA 2.3.** *There is a DTM which transforms a positive integer in dyadic (respectively, binary) notation into binary (respectively, dyadic) notation with  $O(n)$  moves (where  $n$  is the length of the representation).*

Let  $\text{length}(e)$  denote the length of the dyadic representation of an integer  $e > 0$  (convention:  $\text{length}(0) = 1$ ).

*Notation.* For a real number  $r > 0$ , let  $\log r$  denote the logarithm of  $r$  in base 2 and let  $\lceil r \rceil$  and  $\lfloor r \rfloor$  denote the least integer  $n_0 \geq r$  and the greatest integer  $n_1 \leq r$ , respectively.

We will use the usual notation and definitions in first-order logic and model theory (see, for example, [ChKe, Chap. 1]).

**3. Upper bounds of complexity.** For the purpose of giving an efficient algorithm to solve the problem  $\text{SAT}_{<}(\mathbb{N})$  (respectively,  $\text{SAT}_{<}^{2,2}(\mathbb{N})$ ) let us transform each instance  $\varphi$  of this problem into a more manageable formula  $\varphi'$ . Let  $e_0 = 0 < e_1 < \dots < e_{m-1}$  be the increasing list of distinct integers which occur in  $\varphi$  (we add 0 to the list). Let  $\varphi'$  denote the conjunction of (in)equalities obtained from conjunction  $\varphi$  by replacing each integer  $e_i$  by  $e'_i$ , where integers  $e'_i$  ( $i = 0, \dots, m-1$ ) are defined as follows:  $e'_0 = e_0 = 0$  and  $e'_{i+1} - e'_i = \min(n, e_{i+1} - e_i)$ , where  $n = \text{length}(\varphi)$ . Hence  $e'_i \leq (m-1)n \leq n^2 - n$  for  $i \leq m-1$ . In the sequel, formula  $\varphi'$  will also be denoted  $\text{SIMPLE}(\varphi)$ .

**LEMMA 3.1.** *Let us adopt the above notation (in particular  $\varphi' = \text{SIMPLE}(\varphi)$ ).*

(i)  $\varphi$  is satisfiable if and only if  $\varphi'$  is satisfied by a structure where all terms and subterms of  $\varphi'$  have values  $< n^2$  (respectively, if and only if  $\varphi'$  is satisfiable).

(ii)  $\text{length}(\varphi') \leq n$  and  $\text{length}(e'_i) = O(\log n)$  for  $i < m$ .

(iii) The sorted sequence  $e_0, e_1, \dots, e_{m-1}$  is obtained from input  $\varphi$  in time  $O(n \log n)$  on a DTM.

(iv) The sequence  $(e_0, e'_0)(e_1, e'_1) \cdots (e_{m-1}, e'_{m-1})$  is computable on a DTM in time  $O(n_0)$ , where  $n_0$  is the length of the input sequence  $e_0, e_1, \dots, e_{m-1}$ .

(v) Formula  $\varphi'$  is computable from input  $\varphi$  in time  $O(n \log n)$  on a DTM.

*Proof.* (i) Assume  $\varphi$  is satisfied by a structure  $\mathcal{A}$ . We associate to it the following "compressed" structure  $\mathcal{A}'$ : replace each  $e_i$  by  $e'_i$ ; transform the distinct values  $v_1 < v_2 < \dots < v_k$  ( $k \leq n$ ) of the other terms and subterms of  $\varphi$  into the respective values  $v'_1 < v'_2 < \dots < v'_k$  defined by the following induction on integer  $j$  ( $1 \leq j \leq k$ ):

if  $v_j = e_i$  (for some  $i$ ) then  $v'_j := e'_i$

else (let  $i$  be the integer such that  $i < m-1$  and

- $e_i < v_j < e_{i+1}$  or such that  $i = m - 1$  and  $e_i < v_j$   
 if  $e_i < v_{j-1} < v_j$  then  $v'_j := v'_{j-1} + 1$   
 else (i.e., in case  $j = 1$  or in case  $v_{j-1} \leq e_i < v_j$ )  $v'_j := e'_i + 1$ .

Clearly  $\mathcal{A}'$  satisfies  $\varphi'$  and (i) follows because the (in)equalities between the concerned values are all preserved.

- (ii) Clear from the fact that  $e'_i \leq e_i$  and  $e'_i < n^2$  for  $i < m$ .
- (iii) Use one of the standard sorting methods [AHU].
- (iv) Easy because addition of integers is computable in linear time.
- (v)  $\varphi'$  is computed as follows:

- (1) Compute the sequence  $(e_0, e'_0) \cdots (e_{m-1}, e'_{m-1})$  which corresponds to  $\varphi$ .
- (2) Sort the list of (in)equalities of  $\varphi$  according to the integers which occur in their first members (it requires time  $O(n \log n)$ ). In each (in)equality replace the integers  $e_i$  which occur in the first member by  $e'_i$  by using the sequence  $(e_0, e'_0) \cdots (e_{m-1}, e'_{m-1})$  (this requires time  $O(n)$ ).
- (3) It is similar to procedure (2) but we consider second members of (in)equalities instead of first members.  $\square$

The following proposition expresses that problems  $\text{SAT}_{<}(\mathbb{N})$  and  $\text{SAT}_{\geq}^{2,2}(\mathbb{N})$  can be solved in “quasi-linear” time (cf. [Sr]).

**PROPOSITION 3.2.**  $\text{SAT}_{<}(\mathbb{N}) \in \text{NTIME}(n(\log n)^2)$  and  $\text{SAT}_{\geq}^{2,2}(\mathbb{N}) \in \text{NTIME}(n \log n)$ . More precisely, problem  $\text{SAT}_{<}(\mathbb{N})$  (respectively,  $\text{SAT}_{\geq}^{2,2}(\mathbb{N})$ ) is solved by a Turing machine using  $n \log n$  (respectively,  $n$ ) nondeterministic steps and  $n(\log n)^2$  (respectively,  $n \log n$ ) deterministic steps.

*Proof.* The decision algorithm is the following. Let  $\varphi$  be an instance of  $\text{SAT}_{<}(\mathbb{N})$  (respectively,  $\text{SAT}_{\geq}^{2,2}(\mathbb{N})$ ), of length  $n$ .

- (1) Compute the transformed formula  $\varphi'$  (cf. Lemma 3.1(v)).
- (2) Delete each double occurrence of a same (in)equality in  $\varphi'$  by sorting the list of (in)equalities (notice that there remains only  $O(n/\log n)$  (in)equalities in  $\varphi'$ ).
- (3) For each (in)equality  $\theta = \theta'$  or  $\theta < \theta'$  of  $\varphi'$  guess the values of terms  $\theta$  and  $\theta'$  and of their subterms in the interval of integers  $[0, n^2[$  (for instance, if  $\theta = f_1 f_2 \cdots f_s(e)$ , guess values  $f_s(e), f_{s-1} f_s(e), \cdots, f_1 f_2 \cdots f_s(e)$ , successively so that the (in)equality between  $\theta$  and  $\theta'$  holds).
- (4) Verify that these guesses do not contradict each other (sort them).

Clearly,  $\varphi$  is satisfiable if and only if this nondeterministic algorithm leads to acceptance. Procedures (1) and (2) require time  $O(n \log n)$ . For analyzing the time required by procedure (3), note the following:

- Each guess of a value of a (sub) term has time cost  $O(\log n)$ ;
- The formula  $\varphi'$  contains no more than  $n$  occurrences of distinct or not distinct (sub)terms. Therefore procedure (3) guesses no more than  $n$  values in the general case;
- For each instance  $\varphi$  of  $\text{SAT}_{\geq}^{2,2}(\mathbb{N})$ , there are  $O(n/\log n)$  occurrences of distinct or not distinct (sub)terms in  $\varphi'$  (after the execution of procedure (2)) and then procedure (3) guesses  $O(n/\log n)$  values (for  $\text{SAT}_{\geq}^{2,2}(\mathbb{N})$ ).

Therefore, procedure (3) requires nondeterministic time  $O(n \log n)$  for  $\text{SAT}_{<}(\mathbb{N})$  (respectively,  $O(n)$  for  $\text{SAT}_{\geq}^{2,2}(\mathbb{N})$ ). Procedure (4) requires deterministic time  $O(n(\log n)^2)$  for  $\text{SAT}_{<}(\mathbb{N})$  (respectively,  $O(n \log n)$  for  $\text{SAT}_{\geq}^{2,2}(\mathbb{N})$ ): this is the time needed to sort a list of expressions of total length  $O(n \log n)$  (respectively,  $O(n)$ ).  $\square$

The following proposition improves in some way the nondeterministic time upper bound of  $\text{SAT}_{<}(\mathbb{N})$ .

**PROPOSITION 3.3.** Problem  $\text{SAT}_{<}(\mathbb{N})$  is solved by a Turing machine using  $O(n)$  nondeterministic steps and a polynomial number of deterministic steps.



*Proof.* The algorithm is the following. Let  $\varphi$  be an instance of  $\text{SAT}_{<}(\mathbb{N})$ , of length  $n$ . Execute procedures (1) and (2) of the proof of Proposition 3.2 and the following procedures (3)–(6):

(3) For each (in)equality  $\theta = \theta'$  or  $\theta < \theta'$  of  $\varphi'$  guess the values of the terms  $\theta, \theta'$  in the interval  $[0, n^2[$  so that the (in)equality holds (of course, if term  $\theta$  or  $\theta'$  is only an integer, do not guess its value; also examine the (in)equalities  $e = e'$  or  $e < e'$  between terms  $e, e'$  which are only integers: if any of them is (trivially) false then the algorithm rejects because  $\varphi'$  is not satisfiable; if they are (trivially) true, delete them).

Let  $\varphi''$  be the conjunction  $\bigwedge_i \theta_i = v_i$  of the equalities which are guessed in procedure (3) ( $\theta_i$  is a term,  $v_i$  is an integer). Clearly,  $\varphi'$  is satisfiable if and only if at least one of the conjunctions  $\varphi''$  is satisfiable. The following procedures (4) and (5) gradually simplify formula  $\varphi''$  by successive substitutions justified by the following equivalence where  $\mathcal{F}, \mathcal{G}$  are sequences of composed functions and  $e, e', e''$  are integers:

$$(*) \quad (\mathcal{F}e = e' \wedge \mathcal{G}\mathcal{F}e = e'') \leftrightarrow (\mathcal{F}e = e' \wedge \mathcal{G}e' = e'').$$

(4) Sort the list of conjuncts  $\theta_i = v_i$  of  $\varphi''$  according to the lexicographical order of terms  $\theta_i$  (which are read *from right to left*) and then in case a term  $\theta_i$  is a proper subterm of another term  $\theta_j$ , i.e.,  $\theta_j = \mathcal{G}\theta_i$  (regard this equality as an identity of *words*), replace the equality  $\mathcal{G}\theta_i = v_j$  by  $\mathcal{G}v_i = v_j$  (this is justified by (\*)).

In case two terms  $\theta_i$  and  $\theta_j$  are identical: if  $v_i = v_j$  then delete the repeated equality  $\theta_j = v_j$ ; if  $v_i \neq v_j$  then we obtain a contradiction.

(5) Iterate procedure (4) as long as it can be executed (i.e., some terms  $\theta_i$  are subterms of some other  $\theta_j$ ) and no contradiction is reached.

(6) If we get a contradiction then reject. Otherwise accept.

An execution of procedure (4) decreases the number of occurrences of function symbols in  $\varphi''$ . So, it is repeated only  $O(n)$  times. Notice that this does not increase the number of occurrences of integers (each of length  $O(\log n)$ ) in  $\varphi''$ : this number remains  $O(n/\log n)$ . Therefore, the length of  $\varphi''$  is always  $O(n)$ .

Clearly this algorithm needs  $O(n)$  nondeterministic steps which are only used in procedure (3) to guess the values of  $O(n/\log n)$  terms in  $[0, n^2[$  and needs a polynomial number of deterministic steps.

It remains to prove the correctness of this nondeterministic algorithm. Clearly, if it rejects then each formula  $\varphi''$  is contradictory. An acceptance means that procedure (5) leads to a conjunction  $\varphi'' = (\bigwedge_i \theta_i = v_i)$ , where no term  $\theta_i$  is a subterm of another term  $\theta_j$ . To conclude the proof, it suffices to prove the following lemma.

**LEMMA 3.4.** *If no term  $\theta_i$  is a subterm of another term  $\theta_j$  then the conjunction  $\varphi'' = (\bigwedge_i \theta_i = v_i)$  (where each  $v_i$  denotes an integer) is satisfiable.*

*Proof.* Let  $E$  denote the set of integers which occur in  $\varphi''$ . For all terms  $\theta_i = f_1 f_2 \cdots f_r(e)$  that occur in  $\varphi''$ , define the values of subterms  $f_i(e)$ ,  $f_{r-1} f_r(e), \dots, f_2 f_3 \cdots f_r(e)$  out of  $E$  so that distinct subterms have distinct values. Last, define the values of  $\theta_i = f_1 f_2 \cdots f_r(e)$  to be  $v_i$  (so,  $\varphi''$  is satisfied by the constructed structure). There is no double definition of a same value because no  $\theta_i$  is a subterm of another term  $\theta_j$ . So, Lemma 3.4 and Proposition 3.3 are proved.  $\square$

For the purpose of proving a linear time upper bound for problem  $\text{SAT}_{\leq}^{2,2}(\mathbb{N})$  on an *alternating* TM, let us present a useful language and an algorithm to recognize it.

*Notation.* Let  $\mathcal{L}$  denote the following language on an alphabet  $\Sigma \cup \{\#, \$\}$  (where  $\Sigma \cap \{\#, \$\} = \emptyset$ ):

$$\mathcal{L} = \{u_1 \# u_2 \cdots \# u_m \$ u'_1 \# u'_2 \cdots \# u'_p : u_i, u'_j \in \Sigma^*, \text{ each } u_i \text{ is a word } u'_j, \text{ each } u'_j \text{ is a word } u_i \text{ and the sequence } u'_1, u'_2 \cdots u'_p \text{ is lexicographically ordered}\}.$$

LEMMA 3.5. *Language  $\mathcal{L}$  belongs to  $\Pi_1(n)$ .*

*Proof.*  $\mathcal{L}$  is decided by the following algorithm.

- (1) Verify that words  $u'_1, u'_2 \cdots u'_p$  are lexicographically ordered.
- (2) Universally choose an index  $i \in [1, m]$  and verify that  $u_i$  is a word  $u'_j$ .
- (3) Universally choose an index  $j \in [1, p]$  and verify that  $u'_j$  is a word  $u_i$ .

This clearly requires  $O(n)$  universal (or deterministic) steps.  $\square$

THEOREM 3.6. *Problem  $\text{SAT}_{\leq}^{2,2}(\mathbb{N})$  belongs to  $\Sigma_2(n)$ .*

*Proof.* The required algorithm is obtained by transformations of the nondeterministic algorithm, denoted  $\mathfrak{A}$ , of Proposition 3.2. Let us analyse algorithm  $\mathfrak{A}$ . It has three kinds of procedures:

- (i) A nondeterministic procedure (procedure (3)) of time cost  $O(n)$ .
- (ii) Some procedures which sort lists of integers (respectively, lists of (in)equalities) of total length  $O(n)$  (examples: sort the integers which occur in  $\varphi$ ; in Procedure (4) sort a list of equalities).
- (iii) Other deterministic procedures (example: construct the sequence  $(e_0, e'_0) \cdots (e_{m-1}, e'_{m-1})$  from the sorted sequence  $e_0, \cdots, e_{m-1}$ ): they only require time  $O(n)$ .

Let us (informally) present our  $\Sigma_2$  algorithm. Let  $\varphi$  be an input of length  $n$  and  $\mathcal{C}(\varphi)$  denote a computation of  $\mathfrak{A}$  on input  $\varphi$ .

(1') Guess (existentially) all the sorted sequences obtained in  $\mathcal{C}(\varphi)$ . Examples: guess the sorted sequence of integers  $e_0, e_1 \cdots e_{m-1}$  which occur in  $\varphi$ ; guess formula  $\varphi'$  and all the transformed forms of formula  $\varphi'$  and all the transformed forms of formula  $\varphi$  constructed before (cf. Proof of Lemma 3.1(v)).

(2') Execute the nondeterministic procedure (3).

(3') Execute the deterministic procedures of  $\mathfrak{A}$ .

(4') Verify that each (guessed) sorted list is a "good" sorted list by comparing it with the corresponding (nonsorted) sequence: execute an appropriate version of the algorithm of Lemma 3.5 (it uses  $O(n)$  universal steps).

(5') Reject if one of the above procedures (3') or (4') leads to a contradiction. Otherwise accept.

This is clearly a  $\Sigma_2$  algorithm (existential steps precede universal steps) which solves problem

$\text{SAT}_{\leq}^{2,2}(\mathbb{N})$  in time  $O(n)$  on an alternating TM.  $\square$

*Remarks.* The universal steps of our  $\Sigma_2$ -algorithm are only needed in order to sort some inputs (cf. procedure (4')). Note that the proofs of upper bounds on the complexity of  $\text{SAT}_{\leq}^{2,2}(\mathbb{N})$  do not depend upon the fact that the number of distinct function symbols is bounded (by 2) but only depend upon the fact that  $s$ , the number of composed functions of a term is bounded by a fixed number (for instance 2). In fact we have even the following theorem.

THEOREM 3.7. *Problem  $\text{SAT}_{<}(\mathbb{N})$  belongs to  $\Sigma_2(n)$ .*

*Idea of the proof.* The complete proof is long and needs much care. We present it in the Appendix. The idea is as follows. A difficulty (for proving a linear time upper bound for  $\text{SAT}_{<}(\mathbb{N})$ ) originates from the *unboundedness* of the number of composed function symbols in a term of an instance of  $\text{SAT}_{<}(\mathbb{N})$ : therefore first we reduce  $\text{SAT}_{<}(\mathbb{N})$  (in linear time) to a similar problem where an instance  $\varphi$  has no term with more than  $\log n$  composed functions ( $n = \text{length}(\varphi)$ ). Second, we solve the simplified problem by associating to each satisfiable instance  $\varphi$  a conjunction  $\varphi'' = (\bigwedge_i \theta_i = v_i)$  such that:

(i)  $\varphi''$  “derives” from  $\varphi$ , i.e., is a consequence of  $\varphi$  (“derivation” will be precisely defined);

(ii) No term  $\theta_i$  is a subterm of another term  $\theta_j$  ( $\varphi''$  is satisfiable by Lemma 3.4).

This is exactly the conjunction  $\varphi''$  of length  $O(n)$  produced by the deterministic procedures (4), (5) in the proof of Proposition 3.3. However, we no longer produce the formula  $\varphi''$  deterministically, but guess it.

**4. Simulation of NTMs by NRAMs.** In order to prove our main result we need a technical version of a result of [Mo2]. The idea of its proof (given in the Introduction) is simple but the precise proof is rather long and tedious.

Let  $S \subset \{1, 2\}^*$  be a language accepted by an NTM in time  $cn$  for a constant  $c$ . We are going to exhibit an NRAM which simulates the NTM in time  $O(n/\log n)$ . An input  $w$  of the NRAM is not read one bit at a time (it would require time  $n = \text{length}(w)$ ), but is read by blocks. More precisely, let  $k$  be a (sufficiently large) fixed integer, only depending upon set  $S$  (the exact value of  $k$  will be given below); each word  $w \in \{1, 2\}^n$  is divided into subwords  $w_0, w_1, \dots, w_{m-1}$  so that:

(i)  $w = w_0 \wedge w_1 \cdots \wedge w_{m-1}$ ;

(ii) For each  $i < m - 1$   $\text{length}(w_i) = h(n)$  where  $h(n) = \lfloor 1/k \log n \rfloor$ ;

(iii)  $0 < \text{length}(w_{m-1}) \leq h(n)$ .

Consequently,  $m = \lceil n/h(n) \rceil$  (we assume  $n \geq 2^k$  so that  $h(n) \geq 1$ ).

For convenience we identify a word  $w$  with its corresponding  $m$ -tuple  $(w_0, w_1, \dots, w_{m-1})$ . Each word  $w_i$  is also identified with the integer it represents in dyadic notation: so,  $0 < w_i < 2^{h(n)+1} \leq 2n^{1/k}$ . Each instruction  $\text{Read}(R_j)$  causes the NRAM to transfer the first integer  $w_i$  which has not been read in up to this time into register  $R_j$  (convention:  $w_m = 0$ ).

Let  $\text{NRAM}^*(T(n))$  denote the class of languages  $S \subset \{1, 2\}^*$  accepted by such an NRAM for some  $k$  in time  $T(n)$  ( $n$  is the length of the input word  $w$ ).

LEMMA 4.1. (see [Mo2]). *If a language  $S \subset \{1, 2\}^*$  belongs to  $\text{NTIME}(n)$  then  $S \in \cup_c \text{NRAM}^*(cn/\log n)$ .*

*Proof.* We can assume without loss of generality that an NTM accepts  $S$  in time  $cn$  ( $c$  is a fixed integer) and has only one input tape, numbered 0, and two worktapes, numbered 1 and 2.

The time of a computation of the NTM (on an input of length  $n$ ) is divided into intervals of  $h(n)$  instants:  $\Delta_0, \Delta_1, \dots$  ( $h(n) = \lfloor 1/k \log n \rfloor$ ). Similarly, each tape of the NTM is divided into blocks of  $h(n)$  adjacent cells. If  $\Sigma = \{1, \dots, d\}$  is the tape alphabet, each description  $\delta \in \Sigma^{h(n)}$  of a block is identified with the integer  $\delta$  represents in  $d$ -adic notation: so;  $\delta < d^{h(n)+1}$ . Tape  $i$  ( $i = 0, 1, 2$ ) of the NTM is represented by a sequence of registers  $R_0^i, R_1^i \cdots$  of the NRAM so that the content of  $R_j^i$  is the description of block  $j$ .

It is clear that if a head of the NTM scans block  $j$  at a given instant, then this head can only visit blocks  $j, j-1$ , and  $j+1$  during the next  $h(n)$  instants. In order to simulate  $h(n)$  steps of the NTM in  $O(1)$  steps (cf. Part 5 below), the NRAM precomputes a binary relation  $\mathcal{R}$ : roughly,  $\tau \mathcal{R} \tau'$  will mean that the NTM transforms a “partial description”  $\tau$  into another one  $\tau'$ , in  $h(n)$  moves. More formally, a (partial) *description*  $\tau$  of type 1 (of the NTM) is defined to be a tuple of integers which consists of:

(i) A control state  $\sigma$  (encoded by an integer);

(ii) For each tape  $i = 0, 1, 2$ :

- The head position  $\pi_i$  on the visited block;

- Descriptions  $\delta_i^0, \delta_i^+, \delta_i^-$  of the visited block and of the right and left adjacent blocks, respectively. We adopt the convention  $\delta_i^- = 0$  (respectively,  $\delta_i^+ = 0$ )

in case  $\delta_i^0$  describes the leftmost (respectively, rightmost) block of tape  $i$ . Recall that all blocks have length  $h(n)$  exactly, except blocks described by 0 and the block of the input tape which contains  $w_{m-1}$ .

A description  $\tau'$  of type 2 is similar to a description of type 1 with the following change: the visited block of tape  $i$  is not necessarily the block described by  $\delta_i^0$  but must be any of the blocks  $\delta_i^0$ ,  $\delta_i^{-1}$  or  $\delta_i^{+1}$ ; the tuple  $\tau'$  includes a number 0, -1, or 1 (i.e., center, left, or right) indicating the visited block.

Clearly, the number of possible descriptions (of type 1 or 2) is  $\beta(n) = O([h(n) \cdot d^{3h(n)}]^3)$  (for each 3 blocks on a tape  $i=0, 1, 2$  there are at most  $3h(n)$  possible head positions and  $d^{3h(n)}$  possible block descriptions ( $\delta_i^0, \delta_i^{+1}, \delta_i^{-1}$ )). We write  $\tau\mathcal{R}\tau'$  if the NTM transforms description  $\tau$  (of type 1) into description  $\tau'$  (of type 2) in  $h(n)$  moves. Let us describe the work of the NRAM.

*Part 1.* Copy the input integers  $w_0, w_1, \dots, w_{m-1}, 0$  into registers  $R_0^0, R_1^0 \dots R_m^0$ , respectively. Guess the integer  $n$  and compute  $h(n)$ . Verify that  $m = \lceil n/h(n) \rceil$  and that the input is suitable, i.e., length  $(w_i) = h(n)$  for each  $i < m-1$  (verify  $2^{h(n)} - 1 \leq w_i \leq 2(2^{h(n)} - 1)$ ) and  $0 < \text{length}(w_{m-1}) \leq h(n)$ . Part 1 requires time  $O(m) = O(n/\log n)$ .

*Part 2.* Compute the addition table of integers  $< d^{h(n)+1}$  (in time  $O(d^{2h(n)})$ ) and a table  $T_{\text{div}}$  which gives the quotient and the remainder of the division of each integer  $< d^{h(n)+1}$  by  $d$  (it requires time  $O(d^{h(n)})$ ). Construct also the table  $T_{\text{div}}$  for base 2 (instead of base  $d$ ).

*Part 3.* Construct the table  $T_1$  of the possible transitions from a description  $\tau$  of type 1 or 2 to a description  $\tau'$  of type 2 by only one move of the NTM (this will be denoted as  $\tau\mathcal{R}_1\tau'$ ). For convenience, we assume that our NRAM is multidimensional. We use the fact that an integer which encodes a word of length  $h(n)$  on the alphabet  $\Sigma = \{1, \dots, d\}$  (for instance, the description  $\delta$  of a block) is decoded, i.e., analyzed by bits, in time  $O(h(n))$  (by using table  $T_{\text{div}}$ ) and that conversely, an encoding subroutine requires a similar time. Let us exhibit it:

SUBROUTINE ENCODING {Comment: it reads a sequence of  $h(n)$  symbols of  $\Sigma$   
and returns the integer  $e$  it represents}

```
begin e := 0;
  for i := 1 to h(n) do
    begin Read (u); e := de + u end
end.
```

For each description  $\tau$  the set  $\{\tau' | \tau\mathcal{R}_1\tau'\}$  can be constructed in time  $O(h(n))$ . Therefore, the NRAM computes table  $T_1$  in time  $O(\beta(n)h(n))$  since there are  $O(\beta(n))$  descriptions (of type 1 or 2).

*Part 4.* Construct the table  $T_{\mathcal{R}}$  of relation  $\mathcal{R}$ , using table  $T_1$ . For each description  $\tau$  of type 1 do as follows:

*Stage 4(i).* ( $1 \leq i \leq h(n)$ ). Construct the set  $D_i$  of descriptions  $\tau_i$  such that  $\tau_{i-1}\mathcal{R}_1\tau_i$  for a description  $\tau_{i-1} \in D_{i-1}$  (convention:  $D_0 = \{\tau\}$ ).

It is clear that  $D_{h(n)} = \{\tau' | \tau\mathcal{R}\tau'\}$ . Stage 4(i) ( $1 \leq i \leq h(n)$ ) requires time  $O(\beta(n))$ . Therefore, the construction of the set  $\{\tau' | \tau\mathcal{R}\tau'\}$ , for a fixed  $\tau$ , requires time  $O(\beta(n)h(n))$  and table  $T_{\mathcal{R}}$  is computed in time  $O(\beta(n)^2h(n))$ .

Parts 2-4 of the precomputation require time  $O(\beta(n)^2h(n)) = O(h(n)^7d^{18h(n)})$  where  $h(n) = \lfloor 1/k \log n \rfloor$ .

For a sufficiently large integer  $k$  ( $k \geq 20 \log d$  is sufficient) we obtain  $O(h(n)^7d^{18h(n)}) = O(n/\log n)$ .

*Part 5.* This is the main part of the simulation. Each interval  $\Delta_e$  (of length  $h(n) = \lfloor 1/k \log n \rfloor$ ) of a computation of the NTM is simulated in  $O(1)$  steps as follows:

exhibit the description  $\tau$  (of type 1) of the NTM at the first instant of  $\Delta_e$ ; guess a tuple  $\tau'$  (of type 2) such that  $\tau \mathcal{R} \tau'$ ; for each tape  $i = 0, 1, 2$ , transform the contents of the three visited registers  $R_{j-1}^i, R_j^i, R_{j+1}^i$  according to the new description  $\tau'$ .  $\square$

**5. NRAMs and first-order generalized spectra.** In the present section, we give a key lemma (Lemma 5.2) for the proof of our main result. It uses a logical notion: the (*generalized*) *spectrum* of a *first-order sentence*. Our idea is roughly the following. We encode  $m$  moves of an NRAM (which simulates  $\Omega(m \log m)$  of an NTM by Lemma 4.1) into a structure of domain  $m = \{0, 1, \dots, m-1\}$ . More precisely our input is encoded by a function  $f: m \rightarrow m$  (note that such a function encodes a word  $w \in \{1, 2\}^*$  of length about  $m \log m$ ); the  $m$  moves of our NRAM are encoded by a “computation structure”  $\langle m, f, <, \dots \rangle$  expanding the input structure  $\langle m, f \rangle$  (with the natural order  $<$  of the domain  $m$  and with some functions from  $m$  to  $m$  and some constants); lastly we define the “computation structures” to be exactly the finite models of some first-order sentence with only one (universally quantified) variable; the variable intuitively represents each of the  $m$  instants of the computation of the NRAM. The precise definitions and proofs are given below.

In § 4, we have identified each word  $w \in \{1, 2\}^n$  with an  $m$ -tuple  $(w_0, \dots, w_{m-1})$  of integers where  $m = \lceil n/h(n) \rceil$  ( $h(n) = \lfloor 1/k \log n \rfloor$ ) and each  $w_i < 2n^{1/k}$ . Now let us identify each sufficiently long word  $w \in \{1, 2\}^n$  with the structure  $\langle m, f \rangle$ , where domain  $m$  is  $\{0, 1, \dots, m-1\}$  and  $f$  is the function  $m \rightarrow m - \{0\}$  such that for all  $i < m, f(i) = w_i$  (this is possible since  $w_i < 2n^{1/k} \leq m$  except for finitely many  $n$ ). So, a set of words  $S \subset \{1, 2\}^*$  is identified with a set of structures  $\langle m, f \rangle$  (except for finitely many words  $w$  of  $S$ : we can eliminate these words without changing the complexity of  $S$ ).

*Notation.* Let  $S$  be a set of structures  $\langle m, f \rangle$ , where  $m$  is a positive integer and  $f$  is a function:  $m \rightarrow m - \{0\}$ . We write  $S \in \text{NRAM}(T(m))$  to mean that  $S$  is accepted by an NRAM in time  $T(m)$ .

*Remarks.*  $m$  is the cardinality of the input structure ( $m$  is not the length of the encoding of this structure!). We still use the above convention: a Read instruction reads the first value  $f(0), f(1), \dots, f(m-1), 0$  which has not been read in up to this time.

Let us reformulate Lemma 4.1.

**LEMMA 5.1.** *If a language  $S \subset \{1, 2\}^*$  belongs to  $\text{NTIME}(n)$ , then there is an integer  $c$  such that  $S \in \text{NRAM}(cm)$  (where  $S$  is regarded as a set of structures  $\langle m, f \rangle$ ).*

*Proof.* Note that  $n/\log n = O(m)$  since  $m = \lceil n/h(n) \rceil$ .  $\square$

*Remark.* The assertion  $S \in \text{NRAM}(cm)$  of Lemma 5.1 implicitly refers to the integer  $k$  used in the partition of an input  $w \in \{1, 2\}^*$  into an  $m$ -tuple  $(w_0, \dots, w_{m-1})$ . It does not matter because  $k$  only depends upon set  $S$ .

*Notation.* For an integer  $c > 1$  and a function  $f: m \rightarrow m - \{0\}$ , let  ${}^c f$  denote the function:  $cm \rightarrow cm$  such that:  ${}^c f(e) = f(e)$  for  $e < m$  and  ${}^c f(e) = 0$  for  $m \leq e < cm$ . If  $w = (w_0, \dots, w_{m-1}) = \langle m, f \rangle$  we also denote

$$c(w) = (w_0, \dots, w_{m-1}, \underbrace{0, \dots, 0}_{(c-1)m \text{ times}}) = \langle cm, {}^c f \rangle.$$

**DEFINITION.** Let  $\varphi$  be a first-order sentence whose type is  $\mathcal{T} \cup \{<, F\}$  where

- (i)  $<$  is a binary relation symbol (for linear order);
- (ii)  $F$  is a specified monadic function symbol.

The *generalized spectrum* of  $\varphi$ , denoted  $\text{GenSp}(\varphi)$ , is the set of structures  $\langle M, F \rangle$  ( $M$  is a positive integer) which have an expansion  $\langle M, <, F, g \rangle_{g \in \mathcal{T}}$  which satisfies  $\varphi$  and where:

- (i)  $<$  is the natural linear order on  $M$ ;

(ii) Symbol  $F$  is interpreted by function  $F$ . (More generally, for convenience, we confuse each symbol and its interpretation.)

The following lemma is similar to Theorem 6 of [Gr2]. We present a proof of this lemma in order to make the paper self-contained.

LEMMA 5.2. *If  $S$  is a set of structures  $\langle m, f \rangle$  such that  $S \in \text{NRAM}(cm)$  for an integer  $c \geq 1$ , then there is a sentence  $\varphi = \forall x \psi(x)$  (with only one variable  $x$ ) of type  $\mathcal{T} \cup \{<, F\}$  such that:*

- (i)  $\psi$  is quantifier-free;
- (ii)  $\mathcal{T}$  only contains monadic function symbols and constant symbols;
- (iii) For any structure  $\langle m, f \rangle$ ,  $f: m \rightarrow m = \{0\}$ , we have  $\langle m, f \rangle \in S$  if and only if  $\langle cm, {}^c f \rangle \in \text{GenSp}(\varphi)$ .

*Proof.* Let  $S \in \text{NRAM}(cm)$ . Without loss of generality we assume that a simple NRAM,  $\mathcal{M}$ , accepts  $S$  in time  $cm$ ,  $c \geq 1$  an integer. Let  $\text{inst } 0, \text{inst } 1 \cdots \text{inst } l$  be the sequence of instructions of the program of  $\mathcal{M}$ . We want to encode an accepting computation of length  $M = cm$  in a structure of domain  $M$ . We can require that all integers encountered in it be smaller than  $M$ . The type  $\mathcal{T} \cup \{<, F\}$  of our structure includes the monadic function symbols denoted  $I, A, \langle A \rangle, \langle A' \rangle, B$ , and  $P$  with the following (intuitive) meaning (note that the argument  $t$  of the functions intuitively means: "at the instant  $t$ "), see Fig. 1.

$I(t) = i$	holds iff the current instruction is $\text{inst } i$
$A(t)$	is the content of accumulator $a$
$\langle A \rangle(t)$	is the integer currently stored in register $\langle a \rangle$
$\langle A' \rangle(t)$	is the integer written in register $\langle a \rangle$
$B(t)$	is the content of accumulator $b$
$P(t) = e$	iff the next input integer to be read in is $F(e)$

FIG. 1

The conjunction,  $\varphi_0$ , of sentences  $\forall x \exists y \text{ suc}(y) = x$  and  $(\forall x \neq L) [x < \text{suc}(x) \wedge \text{suc}(L) = 0]$  defines (on domain  $M$ ) the constants  $0, L = M - 1$  and the monadic function  $\text{suc}$  such that  $\text{suc}(e) = e + 1$  modulo  $M$ , for each  $e \in M$ : in case  $e < M - 1$ , we use the more suggestive notation  $e + 1$ . Define the constants  $1 = 0 + 1, 2 = 1 + 1 \cdots$  as abbreviations.

For the purpose of describing an accepting computation of  $\mathcal{M}$ , let us give some sentences of type  $\mathcal{T} \cup \{<, F\}$ :

$$\varphi_1: [A(0) = 0 \wedge \langle A \rangle(0) = 0 \wedge B(0) = 0 \wedge I(0) = 0 \wedge P(0) = 0],$$

$\varphi_1$  describes the initial conditions. The following sentence:

$$\varphi_2: \forall t \bigvee_{i \leq l} I(t) = i$$

means that exactly one instruction of the program is current at each instant.

Let (same  $a$ )( $t$ ), (same  $\langle a \rangle$ )( $t$ ), (same  $b$ )( $t$ ), and (same  $p$ )( $t$ ), respectively, denote equalities  $A(t+1) = A(t)$ ,  $\langle A' \rangle(t+1) = \langle A \rangle(t)$ ,  $B(t+1) = B(t)$ , and  $P(t+1) = P(t)$ . Define the formula (same  $\langle a \rangle, b$ )( $t$ ) for example, as the conjunction (same  $\langle a \rangle$ )( $t$ )  $\wedge$  (same  $b$ )( $t$ ).

Figure 2 associates to each instruction a formula which describes its action.

Let  $\varphi_3$  be the sentence  $(\forall t \neq L) \bigwedge_{i \leq l} [I(t) = i \rightarrow \Psi_i(t)]$ . We want to express the following facts (recall that the content of a register remains unchanged until this register is pointed out by accumulator  $a$ ):

Instruction inst $i$	Formula $\Psi_i(t)$
Read ( $a$ )	$A(t+1) = F(P(t)) \wedge P(t+1) = P(t) + 1 \wedge (\text{same } \langle a, b \rangle(t) \wedge I(t+1) = i + 1)$
$a := a + 1$ (similarly for $a := a - 1$ and $a := 0$ )	$A(t+1) = A(t) + 1 \wedge (\text{same } \langle a, b, p \rangle(t) \wedge I(t+1) = i + 1)$
Guess ( $a$ )	$\exists y A(t+1) = y \wedge (\text{same } \langle a, b, p \rangle(t) \wedge I(t+1) = i + 1)$
$a := \langle a \rangle$	$A(t+1) = \langle A \rangle(t) \wedge (\text{same } \langle a, b, p \rangle(t) \wedge I(t+1) = i + 1)$
$b := a$	$B(t+1) = A(t) \wedge (\text{same } a, \langle a, p \rangle(t) \wedge I(t+1) = i + 1)$
$\langle a \rangle := b$	$\langle A \rangle'(t+1) = B(t) \wedge (\text{same } a, b, p)(t) \wedge I(t+1) = i + 1)$
Go to inst $i_0$ or $i_1$	$(\text{same } a, \langle a, b, p \rangle(t) \wedge [I(t+1) = i_0 \vee I(t+1) = i_1])$
If $a = b$ then go to inst $i_0$ else go to inst $i_1$	$(\text{same } a, \langle a, b, p \rangle(t) \wedge [A(t) = B(t) \rightarrow I(t+1) = i_0] \wedge [A(t) \neq B(t) \rightarrow I(t+1) = i_1])$
Accept (or Reject)	$I(t+1) = i$

FIG. 2

(1) If  $t'$  is the last instant before  $t$  when  $A(t) = A(t')$ , i.e., for all  $t''$  with  $t' < t'' < t$  it holds  $A(t'') \neq A(t)$ , then the integer currently stored in the register pointed out (by accumulator  $a$ ) at instant  $t$  is the integer written in this register at instant  $t' + 1$ .

(2) If register  $A(t)$  has never been pointed out before instant  $t$ , then the integer currently stored in this register at instant  $t$  is its initial content which is 0.

For this purpose we will define a new function symbol  $N$  ( $N$  for "numbering"), which has one argument and two values. Define the "current registers" of a computation as the couples  $(A(t), t)$ , where  $A(t)$  is the address (a natural number) of the register pointed out (by accumulator  $a$ ) at instant  $t$ ,  $t < M$ . There are exactly  $M$  "current registers." Let  $N(x)$  denote the "current register" of rank  $x$  ( $x < M$ ) in lexicographical ordering.

Clearly, function  $N$  is defined by the following sentences:

$$\varphi_4: \forall t \exists x (A(t), t) = N(x),$$

$$\varphi_5: (\forall x \neq L) N(x) < N(x+1).$$

The following sentence,  $\varphi_6$  indicates which is the integer read in the register pointed out (by  $a$ ) at each noninitial instant  $t$ . It uses function  $N$  in an essential manner.

$$\varphi_6: (\forall t \neq 0) \exists t' \exists x$$

$$[N(x) = (A(t'), t') \wedge N(x+1) = (A(t), t)$$

$$(*)1 \quad \wedge (A(t) = A(t') \rightarrow \langle A \rangle(t) = \langle A \rangle'(t' + 1))$$

$$(*)2 \quad \wedge (A(t) \neq A(t') \rightarrow \langle A \rangle(t) = 0)].$$

Implications  $(*1)$  and  $(*2)$ , respectively, describe the above-mentioned cases (1) and (2).

Let  $\varphi_7$  be the sentence  $\bigvee I(L) = i$ , where the disjunction extends over all  $i \leq l$  such that inst  $i = \text{Accept}$ .

Clearly, an input  $\langle m, f \rangle$  is accepted by a computation of  $\mathcal{M}$  (of length  $M = cm$ ) if and only if the sentence  $\varphi = \bigwedge_{i \leq 7} \varphi_i$  has a model which expands the structure  $\langle M, F \rangle = \langle cm, {}^c f \rangle$  (a model of  $\varphi$  “mimics” an accepting computation). In other words,  $\langle m, f \rangle \in S$  if and only if  $\langle cm, {}^c f \rangle$  belongs to  $\text{GenSp}(\varphi)$ . Moreover, it is easy to put the conjunction  $\varphi$  in an equivalent form with only one universal quantifier and no existential one as required: we use (Skolem) function symbols to eliminate existentially quantified variables (see [Gr1], [Gr2] for more details).  $\square$

*Remark.* Moreover, using a “folding” technique as in [Ly], [Gr2], we can prove a stronger result than Lemma 5.2: replace condition (iii) by: (iii') the set  $S$  is  $\text{GenSp}(\varphi)$ .

**6. Linear time reduction to  $\text{SAT}'_{<}(\mathbb{N})$ .** In order to give a clear idea of the proof of Lemma 6.2 which follows, we first prove a weakened form of this lemma: we consider the problem  $\text{SAT}'_{<}(\mathbb{N})$ . This problem is similarly defined as  $\text{SAT}_{<}(\mathbb{N})$  but disjunctions (with, of course, conjunctions) are permitted in an instance of  $\text{SAT}'_{<}(\mathbb{N})$ .

LEMMA 6.1. *Let  $S$  be a set of structures  $\langle m, f \rangle, f: m \rightarrow m - \{0\}$ . If  $S \in \text{NRAM}(cm)$ , for an integer  $c$ , then  $S$  is reducible to problem  $\text{SAT}'_{<}(\mathbb{N})$  in time  $O(m \log m)$  on a DTM.*

*Proof.* By Lemma 5.2 there is a sentence  $\varphi = \forall x \psi(x)$  of type  $\mathcal{T} \cup \{<, F\}$  such that:

- (i)  $\psi$  is quantifier-free;
- (ii)  $\mathcal{T}$  only contains monadic function symbols, denoted  $G$ , and constant symbols, denoted  $C$ ;
- (iii)  $\langle m, f \rangle \in S$  if and only if  $\langle cm, {}^c f \rangle \in \text{GenSp}(\varphi)$ .

Moreover, we assume that  $\psi$  contains no negation because each subformula of the form  $\uparrow t_1 = t_2$  (respectively,  $\uparrow t_1 < t_2$ ) can be replaced by the disjunction  $t_1 < t_2 \vee t_2 < t_1$  (respectively,  $t_1 = t_2 \vee t_2 < t_1$ ).

Let  $\langle M, F \rangle$  be a structure identified with the  $M$ -tuple  $W = (W_0, \dots, W_{M-1})$ , where  $0 \leq W_e = F(e) < M$  for  $e < M$ . Associate to  $W$  the conjunction  $\varphi_w$  of the following formulas  $\psi_1, \psi_2, \psi_3$ :

$$\psi_1 = \bigwedge_{G \in \mathcal{T}} \bigwedge_{e < M} G(e) < M \wedge \bigwedge_{C \in \mathcal{T}} C < M,$$

$\psi_1$  expresses that each constant  $C$  is in the domain  $M$  and that the image of domain  $M$  for each (monadic) function  $G$  is also included in  $M$ ;

$$\psi_2 = \bigwedge_{e < M} \psi(e), \quad \psi_3 = \bigwedge_{e < M} F(e) = W_e,$$

$\psi_2$  is an “unrolled” version of formula  $\varphi = \forall x \psi(x)$  and then because of formula  $\psi_3$  the conjunction  $\varphi_w = \psi_1 \wedge \psi_2 \wedge \psi_3$  expresses that  $\langle M, F \rangle = (W_0, W_1, \dots, W_{M-1})$  belongs to the generalized spectrum of sentence  $\varphi$ . Hence  $\langle M, F \rangle = W$  belongs to  $\text{GenSp}(\varphi)$  if and only if  $\varphi_w \in \text{SAT}'_{<}(\mathbb{N})$ . (Note that each occurrence of a constant symbol  $C$  can be replaced by  $C(0)$ :  $C$  is now regarded as a monadic function symbol). From condition (iii) we obtain the equivalence:  $w = \langle m, f \rangle$  belongs to  $S$  if and only if  $\varphi_{c(w)} \in \text{SAT}'_{<}(\mathbb{N})$ .

Formula  $\varphi_{c(w)}$  is a Boolean combination of  $O(cm) = O(m)$  atomic formulas, each of length  $O(\log m)$ . Hence, length  $(\varphi_{c(w)}) = O(m \log m)$ . The construction of  $\varphi_{c(w)}$  essentially consists of:

- (1) Read the input  $w = (w_0, w_1, \dots, w_{m-1})$  and copy the  $cm$ -tuple

$$c(w) = (w_0, \dots, w_{m-1}, 0, \dots, 0).$$

- (2) Write the list  $0, 1, \dots, cm - 1$  (a fixed number of times).

Clearly, the reduction  $w = (w_0, \dots, w_{m-1}) \mapsto \varphi_{c(w)}$  (from  $S$  to  $\text{SAT}'_{<}(\mathbb{N})$ ) is computable in time  $O(m \log m)$  on a DTM.  $\square$



*Remark.* We have used the same notation for symbols of the type  $\mathcal{T}$  of  $\varphi$  and for the corresponding function symbols of  $\varphi_w$ . Note that the function symbols of  $\varphi_w$  are interpreted by functions:  $\mathbb{N} \rightarrow \mathbb{N}$  whose values out of the interval of integers  $[0, M[$  can be defined arbitrarily.

*Notation.* Let  $D_q^M$  denote the set  $\{0, q, 2q, \dots, (M-1)q\}$ .

LEMMA 6.2. *Let  $S$  be a set of structures  $\langle m, f \rangle, f: m \rightarrow m - \{0\}$ . If  $S \in \text{NRAM}(cm)$  for an integer  $c$ , then  $S$  is reducible to problem  $\text{SAT}_{<}(\mathbb{N})$  in time  $O(m \log m)$  on a DTM.*

*Proof.* By Lemma 5.2, there is a sentence  $\varphi$  of type  $\mathcal{T} \cup \{<, F\}$  of the following form:

$$\varphi = \forall x \bigvee_{i < q} \bigwedge_{j < r} \sigma_j^i(x) *^i_j \tau_j^i(x) \quad \text{such that:}$$

- (i)  $\mathcal{T}$  only contains monadic function symbols, denoted  $G$  and constant symbols, denoted  $C$ ;
- (ii)  $*^i_j$  is written for “=” or “<”;
- (iii)  $\sigma_j^i$  and  $\tau_j^i$  are terms (with only one variable  $x$ );
- (iv)  $\langle m, f \rangle \in S$  if and only if  $\langle cm, {}^c f \rangle \in \text{GenSp}(\varphi)$ .

Sentence  $\varphi$  is the disjunctive normal form of the sentence  $\varphi$  of Lemma 5.2 where negation has been eliminated.

Let  $\langle M, F \rangle$  be a structure identified with the  $M$ -tuple  $W = (W_0, \dots, W_{M-1})$ , where  $0 \leq W_e = F(e) < M$  for  $e < M$ . As in Lemma 6.1, the idea of the proof is to “unroll” the sentence  $\varphi$  in an ordered domain of  $M$  elements: we now choose the domain  $D_q^M = \{0, q, 2q, \dots, (M-1)q\}$  (intuitively, an integer  $e < M$  is “represented” by  $eq$ ). We will use the intermediate elements  $eq + i$  ( $e < M, 1 \leq i < q$ ) to suppress the disjunction  $\bigvee_{i < q}$ .

Let  $P_0, P_1, Q$  be new monadic function symbols. Define  $\varphi_w$  to be the conjunction of the following formulas  $\psi_1 \cdot \dots \cdot \psi_6$ :

$$\psi_1 = \bigwedge_{e < M} Q(eq) = 1 \wedge \bigwedge_{e < M} \bigwedge_{i=1}^{q-1} Q(eq+i) = 0,$$

$\psi_1$  means  $Q$  is the characteristic function of the subset  $D_q^M$  of  $q$ -multiples in the interval of integers  $[0, Mq[$ ;

$$\begin{aligned} \psi_2 = & \bigwedge_{G \in \mathcal{T}} \bigwedge_{e < M} [G(eq) < Mq \wedge Q(G(eq)) = 1], \\ & \wedge \bigwedge_{C \in \mathcal{T}} [C(0) < Mq \wedge Q(C(0)) = 1], \end{aligned}$$

$\psi_2$  means the set  $D_q^M$  is invariant for each function  $G$  and contains the constants  $C(0)$  ( $C$  is now regarded as a function symbol);

$$\psi_3 = \bigwedge_{e < M} \bigwedge_{i < q} \left[ \bigwedge_{s=0,1} eq - 1 < P_s(eq+i) < eq + q \wedge P_0 P_1(eq+i) = eq + i \right],$$

$\psi_3$  means  $P_0$  and  $P_1$  permute each interval  $[eq, (e+1)q[$ ,  $e < M$ , and are inverses each other.

Let  $F_j^i$  and  $G_j^i$  ( $i < q, j < r$ ) be new monadic function symbols.

$$\begin{aligned} \psi_4 = & \bigwedge_{e < M} \bigwedge_{i < q} \bigwedge_{j < r} F_j^i(P_0(eq+i)) *^i_j G_j^i(P_0(eq+i)), \\ \psi_5 = & \bigwedge_{e < M} \bigwedge_{i < q} \bigwedge_{j < r} [F_j^i(eq) = \sigma_j^i(eq) \wedge G_j^i(eq) = \tau_j^i(eq)]. \end{aligned}$$

Because of the property  $\psi_3$  of permutation  $P_0$ , it is clear that  $\psi_4 \wedge \psi_5$  implies the following formula  $\varphi'$  which paraphrases sentence  $\varphi$ :

$$\varphi' = \bigwedge_{e < M} \bigvee_{i < q} \bigwedge_{j < r} \sigma_j^i(eq) *^i_j \tau_j^i(eq)$$

$(\psi_4 \wedge \psi_5$  implies for each  $e < M$  the conjunction  $\bigwedge_{j < r} \sigma_j^i(eq) *^i_j \tau_j^i(eq)$  where  $i$  is the index such that  $P_0(eq + i) = eq$ ).

Conversely, in case  $\varphi'$  is satisfied, let us construct permutation  $P_0$  and functions  $F_j^i, G_j^i$  as follows: for each  $e < M$ , choose an index  $i$  such that  $\bigwedge_{j < r} \sigma_j^i(eq) *^i_j \tau_j^i(eq)$  is satisfied and take  $P_0(eq + i) = eq$ ; then complete the definition of function  $P_0$  so that it permutes each interval  $[eq, (e + 1)q[$ .

For all  $e < M, i < q$ , and  $j < r$ , take  $F_j^i(eq) = \sigma_j^i(eq)$  and  $G_j^i(eq) = \tau_j^i(eq)$ , and for  $1 \leq i' < q$  take

$$F_j^i(eq + i') = 0 \quad \text{and} \quad G_j^i(eq + i') = 0 \quad \text{in case } *^i_j \text{ is } "="$$

and take

$$F_j^i(eq + i') = 0 \quad \text{and} \quad G_j^i(eq + i') = 1 \quad \text{in case } *^i_j \text{ is } "<".$$

From these definitions,  $\psi_5$  is trivially satisfied and so conjuncts of  $\psi_4$  corresponding to pairs  $(e, i)$  such that  $P_0(eq + i) = eq$  are also satisfied. The other conjuncts of  $\psi_4$  are satisfied because from the definition of  $F_j^i, G_j^i$  we have  $F_j^i(eq + i') *^i_j G_j^i(eq + i')$  for all  $i' \in [1, q[$ .

Finally, let us define the formulas  $\psi_6 = \bigwedge_{e < M} F(eq) = qW_e$  (where  $qW_e$  denotes the product of the integers  $q$  and  $W_e$ ) and  $\varphi_w = \bigwedge_{u \leq 6} \psi_u$ . Let  $\mathcal{A} = \langle D_q^M, <, F, G, C \rangle_{G, C \in \mathcal{T}}$  be a structure of type  $\mathcal{T} \cup \{<, F\}$  on the domain  $D_q^M$ . It must be clear that the two following assertions are equivalent:

(1) Structure  $\mathcal{A}$  satisfies formula  $\varphi'$  (which paraphrases  $\varphi$ ) and its reduct  $\langle D_q^M, F \rangle$  is isomorphic to structure  $W = \langle M, F \rangle$  by the isomorphism  $e \mapsto eq$  (notice that this isomorphism is expressed by  $\psi_6$ ).

(2) Structure  $\mathcal{A}$  is the restriction (to  $D_q^M$ ) of a reduct of a structure

$$\mathcal{B} = \langle Mq, <, F, G, C, P_0, \dots \rangle_{G, C \in \mathcal{T}, \dots}$$

(on domain  $Mq$ ) which satisfies  $\varphi_w$ .

(Note. For convenience, our notation does not distinguish between a function symbol and its interpretation in some domain  $D_q^M, M$ , or  $Mq$ , however, we must remember that they are distinct.)

Notice that in assertion (2) the domain  $Mq$  of  $\mathcal{B}$  may be replaced by  $\mathbb{N}$ . The equivalence (1)  $\leftrightarrow$  (2) implies: a structure  $W = \langle M, F \rangle$  has an expansion (of type  $\mathcal{T} \cup \{<, F\}$ ) which satisfies  $\varphi$  if and only if  $\varphi_w$  has a model on domain  $Mq$  (respectively,  $\mathbb{N}$ ).

Thus  $\langle M, F \rangle = W$  belongs to  $\text{GenSp}(\varphi)$  if and only if  $\varphi_w \in \text{SAT}_{<}(\mathbb{N})$ . From condition (iv) we obtain the equivalence:  $w = \langle m, f \rangle$  belongs to  $S$  if and only if  $\varphi_{c(w)} \in \text{SAT}_{<}(\mathbb{N})$ . As in Lemma 6.1, we easily see that the reduction  $w = (w_0, \dots, w_{m-1}) \mapsto \varphi_{c(w)}$  (from  $S$  to  $\text{SAT}_{<}(\mathbb{N})$ ) is computable in time  $O(m \log m)$  on a DTM.  $\square$

*Remark.* In the proof of Lemma 6.2 note the following facts:

FACT 6.2.1. If formula  $\varphi_{c(w)}$  is satisfiable, then it is satisfied by a structure where all terms and subterms that occur in  $\varphi_{c(w)}$  have values  $< cqm$ .

FACT 6.2.2. There is a finite set of function symbols SYMB (only depending on set  $S$ ) such that each function symbol of  $\varphi_{c(w)}$  belongs to SYMB.

FACT 6.2.3. There is a constant  $c_0$  (only depending on set  $S$ ) such that no term of  $\varphi_{c(w)}$  contains a composition of more than  $c_0$  (distinct or not distinct) function symbols of SYMB.

It is now easy to prove the main theorem.

**THEOREM 6.3.** *If a language  $S \subset \{1, 2\}^*$  belongs to  $\text{NTIME}(n)$  then  $S$  is reducible to problem  $\text{SAT}_{<}(\mathbb{N})$  in time  $O(n)$  on a one-tape transducer.*

*Proof.* From Lemma 5.1, there is an integer  $c$  such that  $S \in \text{NRAM}(cm)$  and then from Lemma 6.2 the set  $S$  (regarded as a set of structures  $\langle m, f \rangle$ ) is reducible to problem  $\text{SAT}_{<}(\mathbb{N})$  in time  $O(m \log m)$  via the reduction  $w = (w_0, \dots, w_{m-1}) \mapsto \varphi_{c(w)}$ . It remains to prove that this reduction is computable on a one-tape transducer. The algorithm is the following:

Let  $w \in \{1, 2\}^n$  be an input word.

*Part 1.* Computation of  $h(n)$  and  $m$ : write  $n$  in binary notation (in time  $O(n)$ ); compute the integers  $\lfloor n^{1/k} \rfloor$ ,  $h(n) = \lfloor \log(n^{1/k}) \rfloor$ , and  $m = \lceil n/h(n) \rceil$  (it requires a polynomial time in the length of  $n$ ).

*Part 2.* Construction of formula  $\varphi_{c(w)} = \bigwedge_{i \leq 6} \psi_i$ : we only present the construction of the conjunct  $\psi_6$  which fully depends on input  $w$  (the other conjuncts  $\psi_1 \dots \psi_5$  only depend upon length of  $w$ ). We have  $\psi_6 = \psi'_6 \wedge \psi''_6$  where  $\psi'_6 = \bigwedge_{e < m} F(eq) = qw_e$  and  $\psi''_6 = \bigwedge_{m \leq e < cm} F(eq) = 0$ . Construct  $\psi'_6$ . That is, for each  $e < m$ :

- copy the subword  $w_e$  of length  $h(n)$  on the worktape.
- multiply  $e$  and  $w_e$  (now regarded as an integer) by  $q$ .
- write the conjunct  $F(eq) = qw_e$  on the output tape.

$\psi''_6$  is constructed in a similar (simpler) manner. Part 1 of the algorithm clearly requires time  $O(n)$ . Part 2 requires time  $O(m \log m) = O(n)$ .  $\square$

We strengthen Theorem 6.3 by the following theorem.

**THEOREM 6.4.** *If a language  $S \subset \{1, 2\}^*$  belongs to  $\text{NTIME}(n)$ , then  $S$  is reducible to problem  $\text{SAT}_{<}^2(\mathbb{N})$  (respectively,  $\text{SAT}_{<}^{2,2}(\mathbb{N})$ ) in time  $O(n)$  on a one-tape transducer.*

*Proof.* We use the reduction  $w \mapsto \varphi_{c(w)}$  (from  $S$  to  $\text{SAT}_{<}(\mathbb{N})$ ) of Lemma 6.2. Let  $\text{SYMB} = \{f_\alpha \mid \alpha \in \{1, 2\}^l\}$  denote the finite set of function symbols which occur in formulas  $\varphi_{c(w)}$  (cf Fact 6.2.2). For each  $w \in \{1, 2\}^*$  let  $\varphi'_w$  denote the formula (with function symbols  $g_1, g_2$ ) obtained from  $\varphi_{c(w)}$  by replacing each occurrence of any symbol  $f_\alpha$  ( $\alpha \in \{1, 2\}^l$ ) by the composition  $g_{\alpha_1} g_{\alpha_2} \dots g_{\alpha_l}$ , where  $\alpha = \alpha_1 \alpha_2 \dots \alpha_l$ . Let us prove the equivalence  $\varphi_{c(w)} \in \text{SAT}_{<}(\mathbb{N}) \leftrightarrow \varphi'_w \in \text{SAT}_{<}^2(\mathbb{N})$ . Clearly  $\varphi'_w \in \text{SAT}_{<}^2(\mathbb{N})$  implies  $\varphi_{c(w)} \in \text{SAT}_{<}(\mathbb{N})$ .

Conversely, assume that a structure  $\mathcal{A} = \langle \mathbb{N}, F_\alpha \rangle_{\alpha \in \{1, 2\}^l}$  satisfies formula  $\varphi_{c(w)}$ . We can assume that all terms and subterms of  $\varphi_{c(w)}$  have values  $< cqm$  in structure  $\mathcal{A}$  (Fact 6.2.1).

For each integer  $e < cqm$  and each  $f_{\alpha_1 \dots \alpha_l} \in \text{SYMB}$ , let us define the values of:

$$g_{\alpha_1}(e), g_{\alpha_1-1} g_{\alpha_1}(e), \dots, g_{\alpha_2} g_{\alpha_3} \dots g_{\alpha_l}(e) \quad \text{to be integers } \cong cqm$$

(i.e., distinct from the values of the terms and subterms of  $\varphi_{c(w)}$ ) such that distinct terms have distinct values.

Last, for each  $e < cqm$ , define  $g_{\alpha_1} g_{\alpha_2} \dots g_{\alpha_l}(e)$  to be the value of  $f_{\alpha_1 \dots \alpha_l}(e)$  in structure  $\mathcal{A}$ . (This construction is similar to that of Lemma 3.4.) It follows from these last definitions that structure  $\langle \mathbb{N}, g_1, g_2 \rangle$  (where undefined values of  $g_1, g_2$  can be fixed in any manner) satisfies  $\varphi'_w$ . Hence  $\varphi'_w \in \text{SAT}_{<}^2(\mathbb{N})$ . It is obvious that reduction  $w \mapsto \varphi'_w$  from  $S$  to  $\text{SAT}_{<}^2(\mathbb{N})$  (similar to reduction  $w \mapsto \varphi_{c(w)}$ ) is computable in time  $O(n)$  on a one-tape transducer.

*Remark.* The above argument proves also that  $\text{SAT}_{<}(\mathbb{N})$  is linear time reducible to  $\text{SAT}_{<}^2(\mathbb{N})$ .

Let us state two useful facts to prove Theorem 6.4 for  $\text{SAT}_{<}^{2,2}(\mathbb{N})$ .

**FACT 6.4.1.** There is a constant integer  $c'$  (depending only upon set  $S$ ) such that if  $\varphi'_w$  is satisfiable, then it is satisfied by a structure where all terms and subterms of  $\varphi'_w$  have values  $< c'm$ . In particular, the length of each integer  $e$  which occurs in  $\varphi'_w$  is bounded by length  $(c'm) = O(\log m)$ .

FACT 6.4.2. It follows from Fact 6.2.3 that no term of  $\varphi'_w$  contains a composition of more than  $c_0l = O(1)$  function symbols  $g_1, g_2$ .

For each  $w \in \{1, 2\}^*$ , define formula  $\varphi''_w$  to be the conjunction of the following formulas:

- (i) A formula denoted  $\tilde{\varphi}_w$  obtained from  $\varphi'_w$  by replacing each term of the form  $g_{\alpha_1}g_{\alpha_2} \cdots g_{\alpha_s}(e)$  ( $\alpha_i \in \{1, 2\}, s \geq 1$ ) by  $g_1(\alpha_1\alpha_2 \cdots \alpha_s \hat{\bar{e}})$  where  $\bar{e}$  denotes the word of  $\{1, 2\}^*$  (identified to an integer) obtained by the concatenation:  $\bar{e} = 1^t 2^t e$  (where  $1^t$  denotes symbol "1" repeated  $t$  times) with  $t$  such that  $\text{length}(\bar{e}) = 1 + \text{length}(c'm)$ . (Note. We can construct  $\bar{e}$  since  $\text{length}(e) \leq \text{length}(c'm)$  by Fact 6.4.1.)
- (ii) For each term  $\theta = g_{\alpha_1}g_{\alpha_2} \cdots g_{\alpha_s}(e)$  of  $\varphi'_w$  the formula:

$$\psi_\theta = \left[ g_1(\bar{e}) = e \wedge \bigwedge_{1 \leq i \leq s} g_i(\alpha_i \alpha_{i+1} \cdots \alpha_s \hat{\bar{e}}) = g_{\alpha_i} g_1(\alpha_{i+1} \cdots \alpha_s \hat{\bar{e}}) \right].$$

Note that  $\psi_\theta$  is equivalent to the conjunction

$$\psi'_\theta = \left[ g_1(\bar{e}) = e \wedge \bigwedge_{1 \leq i \leq s} g_i(\alpha_i \cdots \alpha_s \hat{\bar{e}}) = g_{\alpha_i} \cdots g_{\alpha_s}(e) \right].$$

Clearly  $\varphi''_w$  is an instance of problem  $\text{SAT}_{\leq}^{2,2}(\mathbb{N})$ . It follows from Fact 6.4.2 that each formula  $\psi_\theta$  contains  $O(1)$  atomic subformulas and then has length  $O(\log m)$  (cf. Fact 6.4.1), so,  $\text{length}(\varphi''_w) = O(m \log m)$ .

We easily see that the application  $w \mapsto \varphi''_w$  is computable in time  $O(m \log m) = O(n)$  on a one-tape transducer.

It remains to prove the equivalence:  $\varphi'_w \in \text{SAT}_{<}(\mathbb{N}) \leftrightarrow \varphi''_w \in \text{SAT}_{\leq}^{2,2}(\mathbb{N})$ . The conjunction of  $\tilde{\varphi}_w$  and formulas  $\psi'_\theta$  clearly imply  $\varphi'_w$ . Hence  $\varphi''_w$  implies  $\varphi'_w$ .

Conversely, assume that a structure  $\langle \mathbb{N}, g_1, g_2 \rangle$  satisfies  $\varphi'_w$ . By Fact 6.4.1 we can assume that values of terms and subterms which occur in  $\varphi'_w$  are  $< c'm$ . For each term  $\theta = g_{\alpha_1}g_{\alpha_2} \cdots g_{\alpha_s}(e)$  of  $\varphi'_w$  redefine  $g_i(\alpha_i \cdots \alpha_s \hat{\bar{e}})$  to be  $g_{\alpha_i} \cdots g_{\alpha_s}(e)$  for each  $i \in [1, s]$  and redefine  $g_1(\bar{e})$  to be  $e$ .

There is no double definition of the same value because if we have any equality  $\alpha \hat{\bar{e}} = \beta \hat{\bar{e}'}$ , where  $\alpha, \beta \in \{1, 2\}^*$ , then  $\alpha = \beta$  and  $\bar{e} = \bar{e}'$  (because of the equality  $\text{length}(\bar{e}) = \text{length}(\bar{e}') = 1 + \text{length}(c'm)$ ) and consequently  $e = e'$ . These modifications of function  $g_1$  do not affect the truth of  $\varphi'_w$  in structure  $\langle \mathbb{N}, g_1, g_2 \rangle$  since they only affect values  $g_1(u)$  for some  $u > c'm$ .

Our structure  $\langle \mathbb{N}, g_1, g_2 \rangle$  trivially satisfies formulas  $\psi'_\theta$  (and, consequently  $\psi_\theta$ ) and then satisfies  $\tilde{\varphi}_w$  (which paraphrases  $\varphi'_w$ ). Hence  $\varphi''_w \in \text{SAT}_{\leq}^{2,2}(\mathbb{N})$ . This concludes the proof of Theorem 6.4.  $\square$

COROLLARY 6.5. *Theorem 6.3 (respectively, Theorem 6.4) still holds for  $\text{SAT}_{<}(\mathbb{N})$  (respectively,  $\text{SAT}_{\leq}^2(\mathbb{N})$ ) in the absence of the equality symbol "=".*

*Sketch of proof.* Let us modify the formula  $\varphi_{c(w)}$  of the proof of Theorem 6.3: replace each equality  $\sigma = \tau$  by the conjunction  $\sigma < \text{SUC}(\tau) \wedge \tau < \text{SUC}(\sigma)$ , where SUC is a new function symbol (intuitively, SUC is the successor function); take the conjunction of the obtained formula with the following formulas  $\bigwedge_{i < cqm} (G(i) < cqm)$  for all function symbols  $G$  of  $\varphi_{c(w)}$  (cf. Fact 6.2.1), and with the formula

$$\bigwedge_{i < cqm} [i < \text{SUC}(i) \wedge \text{SUC}(i) < i + 2]$$

which clearly defines the successor function restricted to the interval of integers  $[0, cqm[$ . This proves Corollary 6.5 for  $\text{SAT}_{<}(\mathbb{N})$ . Starting from this result, we prove the last part of Corollary 6.5 exactly as we proved Theorem 6.4 for  $\text{SAT}_{\leq}^2(\mathbb{N})$ .  $\square$

From the separation result of [PPST],  $\cup_c \text{DTIME}(cn) \not\subseteq \text{NTIME}(n)$  Corollary 6.6 follows.

**COROLLARY 6.6.** (i) *Problems  $\text{SAT}_{<}(\mathbb{N})$  and  $\text{SAT}_{<}^2(\mathbb{N})$  do not belong to  $\cup_c \text{DTIME}(cn)$ , even in the absence of the equality symbol.*

(ii) *Problem  $\text{SAT}_{\leq}^{2,2}(\mathbb{N})$  does not belong to  $\cup_c \text{DTIME}(cn)$ .*

*Final remarks.* Corollary 6.6 is a modest result because on one hand it concerns an NP-complete problem (often regarded as intractable) and on the other hand the linear time lower bound it states is a deterministic lower bound. It would be interesting to exhibit a (natural) NP-complete problem  $\mathcal{E}_0$  to which each problem in  $\text{NTIME}(n \log n)$ , for example, is linear time reducible; then it would follow from the hierarchy theorem on  $\text{NTIME}$  classes [Co1], [SFM] that  $\mathcal{E}_0 \notin \text{NTIME}(n)$ .

We would like to exhibit a problem  $\mathcal{E}_1 \in \text{NTIME}(n)$  to which all problems of this class are linear time reducible (on a DTM), so that  $\mathcal{E}_1 \notin \cup_c \text{DTIME}(cn)$ . However it is hard to find a (sufficiently expressible) problem the solution of which does not involve sorting.

*Note added in proof.* We have recently proved that a problem listed in [GaJo], “Reduction of Incompletely Specified Automata” has the same complexity theoretical properties as  $\text{SAT}_{\leq}^{2,2}(\mathbb{N})$  (cf. E. Grandjean, “A Nontrivial Lower Bound for an NP-Problem on Finite Automata,” to be submitted).

**Appendix A.** A proof of Theorem 3.7. Let us introduce several definitions and lemmas.

**DEFINITION.** An *elementary conjunction* is an instance of problem  $\text{SAT}_{<}(\mathbb{N})$  of the form  $\bigwedge_i \theta_i = v_i$ , where  $\theta_i$  denotes a term which includes at least one function symbol and  $v_i$  denotes an integer. If no term  $\theta_i$  is a subterm of another term  $\theta_j$ , then we say that the elementary conjunction is *suffix-free*.

Lemma 3.4 can be restated as follows.

**LEMMA A.1.** *Each suffix-free elementary conjunction is satisfiable.*

**DEFINITION.** A *partition*  $\Delta$  of an elementary conjunction  $\Phi$  is a function which associates to each conjunct  $\mathcal{F}e = e'$  of  $\Phi$  a decomposition of  $\mathcal{F}$ , i.e., a sequence  $\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_k$  of nonempty words (on the alphabet SYMB) such that  $\mathcal{F} = \mathcal{F}_k \cdots \mathcal{F}_2 \mathcal{F}_1$ .

**DEFINITION.** Let  $\Phi, \Phi'$  be two elementary conjunctions and  $\Delta$  be a partition of  $\Phi$ . We say that  $\Phi$  *derives from  $\Phi'$  modulo  $\Delta$*  if for each conjunct  $\mathcal{F}e = e'$  of  $\Phi$ , to which  $\Delta$  associates the decomposition  $\mathcal{F} = \mathcal{F}_k \cdots \mathcal{F}_2 \mathcal{F}_1$ , there exist integers  $e_0, e_1, \dots, e_k$  such that:

- (i)  $e_0 = e$ ;
- (ii)  $\mathcal{F}_i e_{i-1} = e_i$  is a conjunct of  $\Phi'$ , for each  $i = 1, 2, \dots, k$ ;
- (iii)  $e_k = e'$ .

*Remark.* Clearly, the derivation (between elementary conjunctions) is a transitive relation.

**DEFINITION.** The *depth* of an elementary conjunction  $\Phi$ , denoted  $\text{depth}(\Phi)$ , is the maximal number of occurrences of function symbols in a term of  $\Phi$ .

Of course, an elementary conjunction which derives from a suffix-free elementary conjunction is satisfiable (by Lemma A.1). Conversely we have Lemma A.2.

**LEMMA A.2.** *If an elementary conjunction  $\Phi$  is satisfiable, then it derives from a suffix-free elementary conjunction  $\Phi'$  (modulo some partition  $\Delta$ ) such that:*

- (i)  $\text{depth}(\Phi') \leq \text{depth}(\Phi)$ ;
- (ii) *the function symbols and integers which occur in  $\Phi'$  are those of  $\Phi$ ;*
- (iii)  $\text{length}(\Phi') \leq \text{length}(\Phi) + l \cdot \text{conj}$ ;

where  $l$  is the length of the largest integer which occurs in  $\Phi$  and  $\text{conj}$  is the number of conjuncts of  $\Phi$ .

*Proof.* The construction of  $\Phi'$  is given by procedures (4), (5) of the proof of Proposition 3.3. Recall that we make successive substitutions (starting from conjunction  $\Phi = \bigwedge_r \theta_r = v_r$ ) as follows: in case a term  $\theta_i$  is a proper subterm of another  $\theta_j$ , i.e.,  $\mathcal{G}\theta_i = \theta_j$ , replace the conjunct  $\theta_j = v_j$  of  $\bigwedge_r \theta_r = v_r$  by  $\mathcal{G}v_i = v_j$  (and suppress any duplicate conjunct). We obtain a sequence of elementary conjunctions  $\Phi_0 = \Phi, \Phi_1, \dots, \Phi_m = \Phi'$  such that  $\Phi_s$  derives from  $\Phi_{s+1}$  modulo some (simple) partition  $\Delta_s$  ( $s = 0, 1, \dots, m-1$ ). Clearly there are strictly fewer occurrences of function symbols in  $\Phi_{s+1}$  than in  $\Phi_s$  (in particular  $m \leq \text{length}(\Phi)$ ) and there are no more conjuncts (respectively, occurrences of integers) in  $\Phi_{s+1}$  than in  $\Phi_s$ : this proves (iii). Assertions (i) and (ii) are obvious.  $\square$

*Notation.* Let DERIVE denote the following problem.

*Instance.* A tuple  $(\Phi, \Phi', \Delta)$ , where  $\Phi, \Phi'$  are elementary conjunctions such that:

- (i) Only two distinct function symbols  $g_1, g_2$  occur in  $\Phi, \Phi'$ ;
- (ii)  $\Phi'$  is suffix-free;
- (iii)  $\Delta$  is a partition of  $\Phi$ .

*Question.* Does  $\Phi$  derive from  $\Phi'$  modulo  $\Delta$ ?

LEMMA A.3. *There is a  $\Pi_1$  Turing machine  $\mathcal{U}$  which decides the problem DERIVE in time  $O(n + d^2(d + l))$ , where  $n, d, l$  denote the following measures of an input  $(\Phi, \Phi', \Delta)$ :  $n = \text{length}(\Phi) + \text{length}(\Phi')$ ,  $d = \max(\text{depth}(\Phi), \text{depth}(\Phi'))$  and  $l$  is the length of the largest integer occurring in  $\Phi, \Phi'$ .*

*Proof.* Let  $\mathcal{U}$  be the  $\Pi_1$  Turing machine with the following program an input of which is an instance  $(\Phi, \Phi', \Delta)$  of problem DERIVE.

- ( $\mathcal{U}_1$ ) Universally choose a conjunct  $\mathcal{F}e = e'$  of  $\Phi$ .
- ( $\mathcal{U}_2$ ) If  $\Delta$  associates to this conjunct the decomposition  $\mathcal{F} = \mathcal{F}_k \cdots \mathcal{F}_2 \mathcal{F}_1$ , universally choose a set  $E_k$  of at most  $k$  conjuncts of  $\Phi'$ .
- ( $\mathcal{U}_3$ ) The last part of the algorithm is deterministic: we want to check that one of the following conditions ( $*j$ ) is true ( $0 \leq j \leq k$ ).

For  $0 \leq j < k$ , condition ( $*j$ ) is:

There are integers  $e_0 = e, e_1, \dots, e_j, e_{j+1}$  such that  $(\mathcal{F}_i e_{i-1} = e_i) \in E_k$  for each  $i = 1, \dots, j$ , and  $(\mathcal{F}_{j+1} e_j = e_{j+1})$  is a conjunct of  $\Phi'$  which does not belong to  $E_k$ .

Condition ( $*k$ ) is as follows: There are integers  $e_0 = e, e_1, \dots, e_k$  such that  $(\mathcal{F}_i e_{i-1} = e_i) \in E_k$  for each  $i = 1, \dots, k$  and  $e_k$  is  $e'$ .

( $\mathcal{U}_3$ ) uses the following recursive PROCEDURE CHECK ( $j$ ), whose parameter belongs to  $\{1, \dots, k+1\}$ .

PROCEDURE CHECK ( $j$ ).

{We assume that  $e_0 = e, e_1, \dots, e_{j-1}$  have been found so that each conjunct  $(\mathcal{F}_i e_{i-1} = e_i)$  belongs to  $E_k$ , for  $i = 1, \dots, j-1$ }

BEGIN

IF  $j \leq k$  THEN

BEGIN

look for an equality of the form  $\mathcal{F}_j e_{j-1} = a$  in  $E_k$ ;

IF such an equality is found THEN

BEGIN  $e_j := a$ ; call CHECK ( $j+1$ ) END

ELSE

BEGIN

look for such an equality  $(\mathcal{F}_j e_{j-1} = a)$  in  $\Phi'$ ;

```

    IF such an equality is found THEN
      BEGIN  $e_j := a$ ; ACCEPT {Condition  $(*j-1)$  is true} END
    ELSE REJECT
  END
END;
IF  $j = k+1$  THEN
  IF  $e_k = e'$  THEN ACCEPT {Condition  $(*k)$  is true}
  ELSE REJECT
END;

```

$(\mathcal{U}_3)$  is precisely the following:

BEGIN  $e_0 := e$ ; call CHECK (1) END.

It is clear that (for each conjunct  $\mathcal{F}e = e'$  and each set  $E_k$ )  $(\mathcal{U}_3)$  accepts if and only if one of the conditions  $(*j)$  is true (recall that  $\Phi'$  is suffix-free and then is satisfiable).

We easily see that if  $\Phi$  derives from  $\Phi$  modulo the partition  $\Delta$ , then the  $\Pi_1$  Machine  $\mathcal{U}$  accepts the input  $(\Phi, \Phi', \Delta)$ .

Conversely, assume that  $\mathcal{U}$  accepts an instance  $(\Phi, \Phi', \Delta)$ . Let  $\mathcal{F}e = e'$  be a conjunct of  $\Phi$  and let  $\mathcal{F} = \mathcal{F}_k \cdots \mathcal{F}_2 \mathcal{F}_1$  be its associated decomposition (by  $\Delta$ ). We are going to construct by induction a sequence of integers  $e_0, e_1, \dots, e_j$  ( $0 \leq j \leq k$ ) such that the following condition  $(\mathcal{C}_j)$  holds:

$(\mathcal{C}_j)$ :  $e_0$  is  $e$  and  $(\mathcal{F}_i e_{i-1} = e_i)$  is a conjunct of  $\Phi'$  for each  $i = 1, \dots, j$ .

Assume that for some  $j < k$   $(\mathcal{C}_j)$  holds for the sequence  $e_0, e_1, \dots, e_j$ . Let  $E_k$  be a set of  $k$  conjuncts (of  $\Phi'$ ) which include the  $j$  conjuncts  $(\mathcal{F}_i e_{i-1} = e_i)$  for  $1 \leq i \leq j$ . Since  $\mathcal{U}$  accepts the input  $(\Phi, \Phi', \Delta)$  there is  $j' \leq k$  such that Condition  $(*j')$  is true for the conjunct  $\mathcal{F}e = e'$  and the set  $E_k$ . We necessarily have  $j' \geq j$  and then there is an integer  $e_{j+1}$  such that  $(\mathcal{F}_{j+1} e_j = e_{j+1})$  is a conjunct of  $\Phi'$ . This proves Condition  $(\mathcal{C}_{j+1})$ . So the induction is complete and we have constructed for the conjunct  $\mathcal{F}e = e'$  a sequence  $e_0, e_1, \dots, e_k$  such that:

$(\mathcal{C}_k)$ :  $e_0$  is  $e$  and  $(\mathcal{F}_i e_{i-1} = e_i)$  is a conjunct of  $\Phi'$  for each  $i = 1, \dots, k$ .

Now defining  $E_k$  to be the set of  $k$  conjuncts  $(\mathcal{F}_i e_{i-1} = e_i)$ , for  $i = 1, \dots, k$  we clearly see that Condition  $(*k)$  holds (for the conjunct  $(\mathcal{F}e = e')$  and the set  $E_k$ ) and then  $e_k$  is  $e'$ . It proves that  $(\mathcal{F}e = e')$  derives from  $\Phi'$  modulo  $\Delta$  and then  $(\Phi, \Phi', \Delta) \in \text{DERIVE}$ .

It remains to analyze the time used by machine  $\mathcal{U}$ . Parts  $(\mathcal{U}_1)$  and  $(\mathcal{U}_2)$  clearly need time  $O(n)$ . Let us examine the time required by  $(\mathcal{U}_3)$  for a fixed conjunct  $\mathcal{F}e = e'$  of  $\Phi$  (decomposed into  $\mathcal{F} = \mathcal{F}_k \cdots \mathcal{F}_2 \mathcal{F}_1$  by  $\Delta$ , with  $k \leq d$ ) and a fixed set  $E_k$  of  $k$  conjuncts of  $\Phi'$ . Clearly each conjunct of  $E_k$  has length  $O(d+l)$ . Therefore the search of a conjunct (respectively, of  $k$  conjuncts) of the form  $\mathcal{F}_j e_{j-1} = a$  in  $E_k$  requires  $k \cdot O(d+l)$  steps (respectively,  $k^2 \cdot O(d+l)$  steps). The end of  $(\mathcal{U}_3)$  consists in either looking for such a conjunct in  $\Phi'$  or checking that  $e_k$  is  $e'$ ; it needs  $O(n)$  steps. Thus  $(\mathcal{U}_3)$  requires a time  $O(n + d^2(d+l))$ .  $\square$

Recall that an instance  $\varphi$  of  $\text{SAT}_{<}(\mathbb{N})$  is satisfiable if and only if the corresponding conjunction,  $\text{SIMPLE}(\varphi)$  (defined at the beginning of § 3) is satisfiable. Let  $\text{SHORT}(\varphi)$  denote the conjunction of the distinct conjuncts of  $\text{SIMPLE}(\varphi)$  presented in lexicographical order.

**LEMMA A.4.** *There is a  $\Sigma_2$  Turing machine which accepts the set  $\{(\varphi, \varphi') : \varphi' = \text{SHORT}(\varphi)\}$  in linear time.*

*Proof.* It is similar to (simpler than) the proof of Theorem 3.6.  $\square$

LEMMA A.5. *There is a linear time bounded nondeterministic Turing machine  $\mathcal{N}$  which for each input  $\varphi'$  of the form  $\varphi' = \text{SHORT}(\varphi)$  with  $n = \text{length}(\varphi)$ , constructs an elementary conjunction  $\Phi$  such that:*

- (i)  $\varphi'$  is satisfiable if and only if it produces some conjunction  $\Phi$  which is satisfiable;
- (ii)  $\text{Depth}(\Phi) = \lceil \log n \rceil$  and each integer which occurs in  $\Phi$  is less than  $n^2$ ;
- (iii) Function symbols of  $\Phi$  are exactly those of  $\varphi'$  (or  $\varphi$ );
- (iv)  $\Phi$  has only  $O(n/\log n)$  conjuncts and  $\text{length}(\Phi) = O(n)$ .

*Proof.* The algorithm of  $\mathcal{N}$  is the following:

(1) For each (in)equality  $\theta = \theta'$  or  $\theta < \theta'$  of  $\varphi'$ , guess the values of the terms  $\theta, \theta'$  in the interval  $[0, n^2[$  so that the (in)equality holds: it produces an elementary conjunction  $\varphi'' = (\bigwedge_i \theta_i = v_i)$ . (By Lemma 3.1(i)  $\varphi'$  is satisfiable if and only if it produces some conjunction  $\varphi''$  which is satisfiable.)

(2) Let  $l = \lceil \log n \rceil$ . For each conjunct  $\theta_i = v_i$  of  $\varphi''$  of the form  $f_p \cdots f_2 f_1(e) = e'$ , compute the quotient  $q$  and the remainder  $r$  of the Euclidean division of  $p$  by  $l$  ( $p = ql + r$ ) and guess  $q$  "intermediate" integers values  $u_1, u_2, \dots, u_q$  in the interval  $[0, n^2[$ , i.e., replace the conjunct  $\theta_i = v_i$  of  $\varphi''$  by the "guessed" conjunction:

$$\gamma_i = \bigwedge_{1 \leq j \leq q+1} \mathcal{F}_j(u_{j-1}) = u_j,$$

where  $u_0 = e$ ,  $u_{q+1} = e'$ , each  $u_j \in [0, n^2[$ ,  $\mathcal{F}_j = f_{jl} f_{j-1} \cdots f_{(j-1)l+1}$  for  $j = 1, 2, \dots, q$  and  $\mathcal{F}_{q+1} = f_{ql+r} \cdots f_{q+1}$  (in case  $r = 0$  take  $u_q = e'$  and delete the last conjunct of  $\gamma_i$ ).

Let  $\Phi$  be the conjunction  $\bigwedge_i \gamma_i$ .

(i) is a direct consequence of Lemma 3.1(i). The assertions (ii) and (iii) are obvious. It remains to prove (iv). We have

$$\text{length}(\varphi'') = \text{length}(\varphi') + \text{conj} \cdot O(\log n)$$

where  $\text{conj}$  denotes the number of distinct conjuncts of  $\varphi'$  (or  $\varphi$ ) and then is  $O(n/\log n)$ . Therefore  $\text{length}(\varphi'') = O(n)$ . The number of conjuncts of  $\Phi$  is no more than the number of conjuncts of  $\varphi''$  (i.e.,  $O(n/\log n)$ ) plus  $O(n/l) = O(n/\log n)$  (we add a conjunct for each block of  $l$  composed functions). We have  $\text{length}(\Phi) = O(n)$  since each conjunct of  $\Phi$  has length  $O(\log n)$ .  $\square$

We can prove Theorem 3.7 as follows.

THEOREM 3.7. *Problem  $\text{SAT}_{<}(\mathbb{N})$  belongs to  $\Sigma_2(n)$ .*

*Proof.* As we have noticed in the proof of Theorem 6.4, problem  $\text{SAT}_{<}(\mathbb{N})$  is linear time reducible to problem  $\text{SAT}_{<}^2(\mathbb{N})$ . Thus it suffices to prove  $\text{SAT}_{<}^2(\mathbb{N}) \in \Sigma_2(n)$  and even to prove that the set

$$\text{SET} = \{(\varphi, \varphi') : \varphi' = \text{SHORT}(\varphi) \text{ and } \varphi' \in \text{SAT}_{<}^2(\mathbb{N})\}$$

belongs to  $\Sigma_2(n)$  (because  $\text{length}(\varphi') \leq \text{length}(\varphi)$  for  $\varphi' = \text{SHORT}(\varphi)$  and then for an input  $\varphi$  we can "guess" such a conjunction  $\varphi'$  in linear time).

The following algorithm clearly decides if  $(\varphi, \varphi') \in \text{SET}$  (cf. Lemmas A.4, A.5, A.2, A.3):

- (1) Check that  $\varphi' = \text{SHORT}(\varphi)$  (cf. Lemma A.4).
- (2) Simulate the nondeterministic Turing machine  $\mathcal{N}$  (on input  $\varphi'$ ): it produces an elementary conjunction  $\Phi$  (cf. Lemma A.5).
- (3) Guess a partition  $\Delta$  of  $\Phi$  and another elementary conjunction  $\Phi'$  (cf. Lemma A.2).
- (4) Check that  $\Phi'$  is suffix-free (in linear time on a  $\Pi_1$  Turing machine).
- (5) Check that  $(\Phi, \Phi', \Delta)$  belongs to  $\text{DERIVE}$  (using the  $\Pi_1$  Turing machine of Lemma A.3).



CLAIM. *The algorithm (1)–(5) works in linear time on an alternating Turing machine.*

*Proof of the claim.* If  $\varphi' = \text{SHORT}(\varphi)$  and belongs to  $\text{SAT}_{<}^2(\mathbb{N})$ , then (by Lemma A.5)  $\mathcal{N}$  produces (starting from input  $\varphi$ ) a satisfiable elementary conjunction  $\Phi$  of depth  $\lceil \log n \rceil$  and length  $O(n)$  (where  $n = \text{length}(\varphi)$ ). It follows from Lemma A.2 that  $\Phi$  is derivable from a suffix-free elementary conjunction  $\Phi'$  (modulo some partition  $\Delta$ ) such that:  $\text{length}(\Phi') \leq \text{length}(\Phi) + l \cdot \text{conj}$  where  $l$  is the length of the largest integer occurring in  $\Phi$  and then is  $O(\log n)$  (by Lemma A.5(ii)) and  $\text{conj}$  is the number of conjuncts of  $\Phi$  which is  $O(n/\log n)$  (by Lemma A.5(iv)): hence  $\text{length}(\Phi') = O(n)$ . It follows from Lemma A.3 that the  $\Pi_1$  Turing machine  $\mathcal{U}$  accepts the input  $(\Phi, \Phi', \Delta)$  in time  $O(n + (\log n)^2(\log n + \log n)) = O(n)$ . This proves the claim.

In order to get a  $\Sigma_2$  algorithm (i.e., existential steps precede universal steps) we do not execute the whole of Part (1) before Parts (2)–(5) but execute all the existential steps of Part (1) and of Parts (2)–(5) before we execute their universal steps. This completes the proof of Theorem 3.7.  $\square$

**Acknowledgment.** I am grateful to Pascal Michel for helpful discussions and for making me aware of references [De] and [Sr].

#### REFERENCES

- [AIK] A. ADACHI, S. IWATA, AND T. KASAI, *Combinatorial game problems require time  $\Omega(n^k)$  time*, J. Assoc. Comput. Mach., 31 (1984), pp. 361–376.
- [AHO] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [CKS] A. K. CHANDRA, D. C. KOZEN, AND L. J. STOCKMEYER, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114–133.
- [ChSt] A. K. CHANDRA AND L. J. STOCKMEYER, *Alternation*, Proc. 17th Annual IEEE Symposium on Foundations of Computer Science, New York, 1976, pp. 98–108.
- [ChKe] C. C. CHANG AND H. J. KEISLER, *Model Theory*, North-Holland, Amsterdam, 1973.
- [Co1] S. A. COOK, *A hierarchy for non-deterministic time complexity*, J. Comput. Systems Sci., 7 (1973), pp. 343–353.
- [Co2] ———, *An overview of computational complexity*, Comm. ACM, 26 (1983), pp. 400–408.
- [De] A. K. DEWDNEY, *Linear time transformations between combinatorial problems*, Internat. J. Computer Math., 11 (1982), pp. 91–110.
- [GaJo] M. R. GAREY AND D. S. JOHNSON, *Computer and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
- [Gr1] E. GRANDJEAN, *The spectra of first-order sentences and computational complexity*, SIAM J. Comput. 13 (1984), pp. 356–373.
- [Gr2] ———, *Universal quantifiers and time complexity of Random Access Machines*, in Decision Problems and Complexity, LNCS 171, Springer-Verlag, Berlin, New York, 1984; Math. Systems Theory, 18 (1985), pp. 171–187.
- [HoUl] J. E. HOPCROFT AND J. D. ULLMAN, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.
- [Le] H. R. LEWIS, *Complexity of solvable cases of the decision problem for the predicate calculus*, Proc. 19th Annual IEEE Symposium on Foundations of Computer Science, New York, 1978, pp. 35–47.
- [Ly] J. F. LYNCH, *Complexity classes and theories of finite models*, Math. System Theory, 15 (1982), pp. 127–144.
- [Mo1] B. MONIEN, *Characterizations of time-bounded computations by limited primitive recursion*, 2nd Internat. Colloquium Automata Languages Programming, Springer-Verlag, Berlin, New York, 1974, pp. 280–293.
- [Mo2] ———, *About the derivation languages of grammars and machines*, 4th Internat. Colloquium Automata Languages Programming, 1977, pp. 337–351.
- [PPST] W. J. PAUL, N. PIPPENGER, E. SZEMEREDI, AND W. T. TROTTER, *On determinism versus non-determinism and related problems*, Proc. 24th Annual IEEE Symposium on Foundations of Computer Sciences, 1983, pp. 429–438.

- [Se] A. SCHÖNHAGE, *Storage modification machines*, SIAM J. Comput., 9 (1980), pp. 490-508.
- [Sr] C. P. SCHNORR, *Satisfiability is quasilinear complete in NQL*, J. Assoc. Comput. Mach., 25 (1978), pp. 136-145.
- [SFM] J. I. SEIFERAS, M. J. FISCHER, AND A. R. MEYER, *Separating non-deterministic time complexity classes*, J. Assoc. Comput. Mach., 25 (1978), pp. 146-167.

## SCHEDULING UET SYSTEMS ON TWO UNIFORM PROCESSORS AND LENGTH TWO PIPELINES \*

HAROLD N. GABOW †

**Abstract.** A set of jobs related by precedence constraints is to be executed in minimum time on two uniform processors, the faster one using  $f$  time units per job and the slower  $s \geq f$  time units. When  $s = f + 1$  the desired schedule can be characterized as HLA, "highest-level-first with abstentions." It can be found in linear time when there is no idle time; in general the time is exponential in the number of levels of the precedence graph. The problem of scheduling jobs related by precedence constraints on a length two pipeline processor is isomorphic to a simple version of the uniform processor problem. The optimum schedule is HLF, "highest-level-first," and can be found in linear time. Approximately optimum schedules for two uniform processors can be found in linear or nearly linear time. The schedule that is optimum for two identical processors has accuracy at most a factor  $2 - (f/s)$  above optimum, for arbitrary  $f$  and  $s$ . The HLF schedule has accuracy  $5/4$  for  $f/s = 1/2$  and  $6/5$  for  $f/s = 2/3$ .

**Key words.** scheduling, uniform processors, pipeline processor, highest-level-first schedule, critical path rule, precedence constraints, directed acyclic graph

**AMS (MOS) subject classifications.** 68Q25, 68R10, 90B35

**1. Introduction.** Much work has been done on the problem of finding a shortest multiprocessor schedule for a set of unit execution time jobs subject to precedence constraints (so-called *UET systems* [C]). The most encouraging results are for two identical processors. This problem is  $P2 \mid prec, p_j = 1 \mid C_{\max}$  in the notation of [GLLRK]. Several efficient algorithms have been given [FKN], [CG], [GJ77], including one with linear running time [G82a], [GT]. The problem can be efficiently solved even in the presence of release times and deadlines [GJ76], [GJ77]. On the other hand the problem is NP-complete when the two processors are identical but the job execution times are one or two ( $P2 \mid prec, p_j \in \{1, 2\} \mid C_{\max}$ ) [U].

This paper investigates the problem for two uniform processors ( $Q2 \mid prec, p_j = 1 \mid C_{\max}$ . A "uniform" processor runs at constant speed). Suppose a job can be executed in  $f$  time units on the fast processor and  $s$  units on the slow processor,  $f \leq s$ . Write  $\Delta = s - f$ . The case  $\Delta = 1$  is most amenable to analysis. We show that the HLF, "highest-level-first," characterization of an optimum schedule for two identical processors [G82a] generalizes to this case. Here the optimum schedule is HLA, "highest-level-first with abstentions." Such a schedule differs from HLF in that it contains "unforced idle time," i.e., idle that from a local viewpoint is unnecessary [GLLRK]. An HLA schedule can be constructed in linear time if the location of the

---

\* Received by the editors August 2, 1982; accepted for publication (in revised form) June 25, 1987. This work was supported in part by National Science Foundation grants MCS-7818909 and MCS-8302648.

† Department of Computer Science, University of Colorado at Boulder, Boulder, Colorado 80309.

idle time is known. So if an optimum schedule has no idle time, it can be found in  $O(n + m)$  time, where  $n$  and  $m$  are the number of nodes and edges of the precedence graph, respectively. In general the algorithm runs in  $O(2^L(n + m))$  time, where  $L$  is the number of levels in the precedence graph.

Recent developments in computer architecture motivate the problem of scheduling a pipeline computer. The case of a length two pipeline is isomorphic to a simple version of the  $\Delta = 1$  problem. In this case an HLF schedule is optimum and can be found in  $O(n + m)$  time.

To handle general speeds  $f$  and  $s$  and also to obtain polynomial running times, we study algorithms that find a schedule that is guaranteed to be nearly optimum. One strategy is to use the optimum schedule for two identical processors. This gives a uniform processor schedule that is at most a factor  $2 - (f/s)$  above optimum. (As expected this approaches one as  $f/s$  approaches one.) Another strategy is to use an HLF schedule for the two uniform processors. This gives better accuracy for two important cases:  $f/s = 1/2$  has accuracy  $5/4$  and  $f/s = 2/3$  has accuracy  $6/5$ . (All bounds on accuracies are tight.)

Section 2 gives definitions and reviews relevant results for two identical processors. Section 3 derives the HLA characterization of an optimum schedule for the  $\Delta = 1$  case. This gives the above-mentioned algorithms for uniform processors and pipelines. It also provides a new proof that HLF schedules are optimum for two identical processors. Section 4 explores the use of the identical processor schedule to approximate the optimum uniform processor schedule. Section 5 explores the HLF approximation algorithm. Section 6 gives concluding remarks.

**2. Preliminaries.** This section gives basic terminology and reviews some results in scheduling.

A scheduling problem is defined on a *dag* (directed acyclic graph). A node of the dag represents a job that requires one unit of processing time; an edge represents a precedence constraint.  $G$  always denotes the given dag;  $n$  and  $m$  denote the number of nodes and edges, respectively. If there is an edge from node  $x$  to node  $y$ , then  $x$  is an *immediate predecessor* of  $y$  and we write  $x \rightarrow y$ ; if there is a directed path (of zero or more edges) from  $x$  to  $y$ , then  $x$  is a *predecessor* of  $y$ ,  $y$  is a *successor* of  $x$  and we write  $x \rightarrow^* y$ . A node with no predecessors is *initial*. A dag can be partitioned into *levels*  $\ell$ ,  $L \geq \ell \geq 1$ : level  $\ell$  consists of all nodes  $x$  that start paths with  $\ell$  nodes but not paths with  $\ell + 1$  nodes. We write  $level(x) = \ell$ .  $L$  always denotes the highest level of the dag. Figure 2.1 shows a dag. In all figures dags are drawn with the convention that edges are directed vertically from higher node to lower.

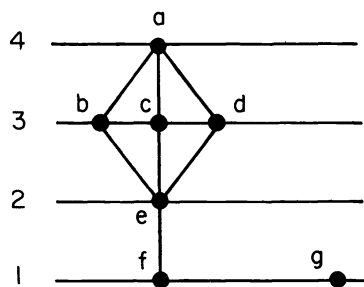


FIG. 2.1. Dag with four levels.

The nodes of the dag (jobs) are executed on two *uniform* processors, i.e., processors of constant speed. The speeds are two relatively prime positive integers  $f$  and  $s$ ,  $f \leq s$ . The *fast processor*  $P_f$  executes a node in  $f$  time units, while the *slow processor*  $P_s$  executes a node in  $s$  time units. (Strictly speaking parameters  $f$  and  $s$  are the inverses of speed. This will not cause confusion.) Define  $\Delta = s - f$ . If  $\Delta = 0$  the processors are *identical*.

Suppose  $P_f$  and  $P_s$  start simultaneously and execute nodes without interruption. Every  $fs$  time units  $P_f$  executes  $s$  nodes and  $P_s$  executes  $f$  nodes. These  $f + s$  nodes are executed in a repetitive pattern called the *execution period* (Figure 2.2). A time interval on  $P_f$  or  $P_s$  in the execution period during which one node is executed is a *slot*. The slots of a period are numbered from 1 to  $f + s$  in order of increasing right endpoint (finish time), with ties broken by left endpoint (start time); hence the last two slots are  $f + s - 1$  on  $P_s$  and  $f + s$  on  $P_f$ .

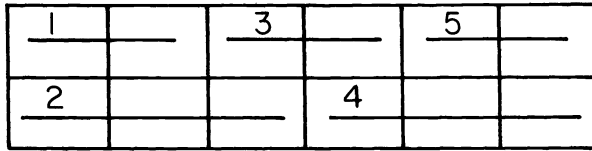


FIG. 2.2. Execution period for  $f/s = 2/3$ .

For identical processors the execution period is trivial. Assume otherwise,  $\Delta > 0$ . Then a slot on  $P_f$  overlaps at most two slots on  $P_s$ ; also a slot on  $P_s$  overlaps at least two slots on  $P_f$ . Exactly  $\Delta + 1$  slots on  $P_f$  are contained in a slot of  $P_s$ . (To show this note that each of  $f - 1$  borders of slots on  $P_s$  is in the interior of a different slot of  $P_f$ . This leaves  $s - f + 1 = \Delta + 1$  slots of  $P_f$  contained in a slot of  $P_s$ .)

When  $\Delta = 1$  every slot overlaps at most two others. This is the basic reason why the results of §3 hold. It implies the following properties, which are assumed in §3 (Figure 2.2): The slots are numbered from 1 to  $2f + 1$ . The odd-numbered slots are on  $P_f$ , the evens on  $P_{f+1}$ . The left endpoint of a slot is nondecreasing with slot number, as is the right endpoint.

A "schedule" specifies how to process each node and obey every precedence constraint. More precisely a *schedule* is an assignment of each node  $x$  to an ordered pair  $(p(x), t(x))$ , where  $p(x) \in \{f, s\}$ ,  $t(x) \geq 0$  is the *start time* for  $x$ ,  $t'(x) = t(x) + p(x)$  is the *finish time*, and for any distinct nodes  $x$  and  $y$  with either  $x \rightarrow y$  or  $p(x) = p(y)$ <sup>1</sup> and  $t(x) \leq t(y)$ , we have  $t'(x) \leq t(y)$ . Processor  $P_{p(x)}$  executes  $x$  from its start time to finish time. A processor is *idle* when it is not executing any node.  $\omega$ , the largest finish time of a node, is the *length (makespan)* of the schedule. An *optimum schedule* has length  $\omega^*$ , the minimum length possible. For further material on schedules see [C], [GLLRK].

For any slot  $\sigma$ ,  $1 \leq \sigma \leq f + s$ , a " $\sigma$ -schedule" starts in  $\sigma$ . More precisely let slot  $\sigma$  be on processor  $P$ . Let  $P'$  be the other processor and let  $\sigma'$  be the first slot on  $P'$  starting no earlier than  $\sigma$  (for  $\sigma < f + s$ ,  $\sigma' = \sigma + 1$ ; for  $\sigma = f + s$ ,  $\sigma' = 2$  of the next execution period). In a  $\sigma$ -schedule,  $P(P')$  starts at slot  $\sigma(\sigma')$  or later. So an ordinary schedule is a 1-schedule. The *length* of a  $\sigma$ -schedule is measured starting at the beginning of slot  $\sigma$ . An *optimum  $\sigma$ -schedule* is a  $\sigma$ -schedule whose length is as small

<sup>1</sup> If the processors are identical, values  $f$  and  $s$  designate distinct processors so  $f \neq s$  in the equation  $p(x) = p(y)$ .

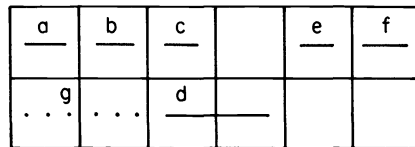
as possible (among all  $\sigma$ -schedules. In general for a fixed dag, the optimum length varies with  $\sigma$ ). Section 3 proves most results for  $\sigma$ -schedules (and a fortiori for ordinary schedules). It should be clear how to interpret a phrase like “an optimum  $\sigma$ -schedule that has property  $P$ ”—when in doubt, “optimum” modifies only “ $\sigma$ -schedule,” so this schedule is an optimum  $\sigma$ -schedule, that happens to also have property  $P$ .

The following definitions generalize HLF, highest-level-first, schedules. HLF schedules are optimum for identical processors ( $\Delta = 0$ ) [G82a]. The definitions are given for  $\Delta = 1$ , where we will show they lead to optimum schedules. However they are relevant for arbitrary  $\Delta$ .

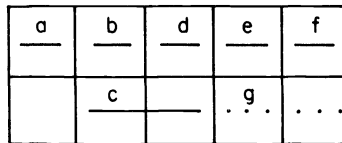
A level schedule “executes levels” in decreasing order. More precisely let  $\sigma$  be a slot,  $1 \leq \sigma \leq 2f + 1$ . Let  $H$  be the set of all nodes on the highest level  $L$ . A *level  $\sigma$ -schedule* executes the nodes of  $H$  in the first  $|H|$  slots  $\sigma, \sigma + 1, \dots, \tau$  (where  $\tau = 1 + ((|H| + \sigma - 2) \bmod (2f + 1))$ ). There are three cases for  $\tau$ :

- (i)  $\tau = 2f + 1$  is the last slot of an execution period.
- (ii)  $\tau < 2f + 1$  and no node is in slot  $\tau + 1$ .
- (iii)  $\tau < 2f + 1$  and a node  $y$  is in slot  $\tau + 1$ .

In cases (i) - (ii) the rest of the schedule is a level 1-schedule for  $G - H$ . In case (iii) the rest of the schedule is a level  $\tau'$ -schedule for  $G - H - y$ , where  $\tau' = \tau + 2$  if  $\tau < 2f$  and  $\tau' = 1$  if  $\tau = 2f$ . A *level schedule* is a level 1-schedule. Figure 2.3 shows two level schedules.



(a)



(b)

FIG. 2.3. *Level schedules for Fig. 2.1,  $f/s = 1/2$ . (a)  $\omega = 6$ . (b)  $\omega^* = 5$ .*

In a level schedule consider a level  $\ell$ ,  $L \geq \ell \geq 1$ , where case (iii) applies. Let  $x$  be the node in slot  $\tau$ . So  $level(x) = \ell$ ,  $level(y) < \ell$ , and  $x \overset{*}{\rightarrow} y$  since slots  $\tau$  and  $\tau + 1$  overlap. The ordered pair  $(x, y)$  is a *jump from  $x$  to  $y$* ; alternatively the jump goes *from level  $\ell$  to level  $\ell(y)$* , *level  $\ell$  jumps  $y$* . If case (ii) applies to level  $\ell$ ,  $\ell$  has an *idle jump*; by convention an idle jump goes to level 0 (level 0 is a fictitious level). *Level  $\ell$  has a jump* in cases (ii) and (iii) but not (i). In Figure 2.3 and all other figures nodes that are jumped are dotted. The idle times in Figure 2.3 are idle jumps.

The *execution of level  $\ell$* , for  $L \geq \ell \geq 1$ , is defined inductively. For  $\ell = L$  it consists of the above slots  $\sigma, \sigma + 1, \dots, \tau$  plus slot  $\tau + 1$  if  $L$  jumps a node. For smaller  $\ell$  it is defined in the obvious way by induction. Observe that nodes of  $\ell$  that are jumped are not included in the execution of  $\ell$  (e.g., in Figure 2.3(a),  $g$  is not in the execution of level 1).

The slots of a level schedule are linearly ordered (as in an execution period, the order is by increasing right endpoint with ties broken by left endpoint). If  $\sigma$  is a slot,  $\sigma^+$  denotes the next slot in this order. We extend this linear ordering to the nodes of the dag, e.g., if node  $x$  is in slot  $\sigma$ , the *node after  $x$*  is the node in  $\sigma^+$ .

In a level schedule let the levels with jumps be  $f_1 > \dots > f_k$ , where level  $f_i$  jumps to level  $t_i$ . The *jump sequence* of the schedule is the ordered  $k$ -tuple  $(t_1, t_2, \dots, t_k)$ . In Figure 2.3 the jump sequences are  $(1, 0, 0, 0)$  in (a) and  $(0, 1)$  in (b). Observe that the dag and the jump sequence together determine the slots where nodes are executed. In particular the length  $\omega$  can be deduced.

Jump sequences are compared using lexicographic order. That is,  $(t_1, \dots, t_k) > (s_1, \dots, s_r)$  if for some  $j$ ,  $1 \leq j \leq \min\{k, r\}$ ,  $t_i = s_i$  for  $1 \leq i < j$  and  $t_j > s_j$ . (Lexicographic order allows the possibility that  $(t_1, \dots, t_k) > (s_1, \dots, s_r)$  if  $t_i = s_i$  for  $1 \leq i \leq r$  and  $k > r$ . This cannot occur with jump sequences: If  $t_i = s_i$  for  $1 \leq i \leq r$  then  $k = r$  and  $(t_1, \dots, t_k) = (s_1, \dots, s_r)$ .)

A *highest-level-first (HLF) schedule* is a level schedule whose jump sequence is as large as possible. Such a schedule is essentially a critical-path schedule. HLF schedules are optimum for several types of identical processor scheduling: two processors [CG], [G82a], forest precedence [H61], [H82], and interval orders [PY], [G81] (the last is shown HLF in [G83]). Figure 2.3 shows HLF schedules are not optimum for uniform processors: (a) is HLF but (b) is optimum. Observe that the optimum schedule jumps to the highest level *except* when it has idle jumps. This is characteristic of an “HLA” schedule, defined as follows.

In a level schedule, a level is *abstentious* if it ends in slot  $\tau < 2f$  and has an idle jump. A level that has a jump but is not abstentious is *nonabstentious*. So a nonabstentious level either ends in slot  $\tau < 2f$  and jumps a node or ends in slot  $2f$ . (The motivation for the latter condition is that when a level ends in slot  $2f$ , the schedule includes a complete slot  $2f + 1$  regardless of whether the level jumps a node.)

Let  $A$  be a set of levels,  $A \subseteq \{1, \dots, L\}$ , and let  $\sigma$  be a slot,  $1 \leq \sigma \leq 2f + 1$ . An *HLA  $\sigma$ -schedule* for  $A$  is a level  $\sigma$ -schedule with abstentious levels  $A$ , whose jump sequence is as large as possible. An *HLA (highest-level-first with abstentions) schedule* is an HLA 1-schedule. Figure 2.3 shows HLA schedules, with abstentions 1, 2 in (a) and 4 in (b).

For identical processors an HLF schedule is an HLA schedule with no abstentions. (Any idle jump is in the slot that is the analogue of  $2f$ , the slot that never abstains.) Such a schedule can be constructed in linear time.

Similar algorithms can be used for uniform processors. In one instance we shall see the set of abstentious levels  $A$  is known in advance. (This is a nontrivial assumption. A given set of levels  $A$  may not even have an HLA schedule, since the precedence constraints may force idle jumps at levels not in  $A$ .)

**THEOREM 2.1.** *Given set of abstentious levels  $A$  and processor speeds  $f, f + 1$ , an HLA schedule, if it exists, can be found in  $O(n + m)$  time and space.*

*Proof.* The algorithm is a straightforward adaptation of the identical processor algorithm of [G82a]. The latter works in two passes: Pass I processes the levels  $t$  in descending order  $t = L, L - 1, \dots, 1$ ; it finds all levels that jump to level  $t$ . Pass II

finds the specific nodes jumped.

The same strategy works for uniform processors. The important point is that after level  $t$  is processed in Pass I it is easy to decide whether or not  $t$  jumps a node: The number of nodes on level  $t$  that are jumped is known. This determines the slot  $\tau$  that executes the last node of  $t$ . Level  $t$  jumps a node if  $t \notin A$  and  $\tau \neq 2f + 1$ . Both these conditions are easy to test.

The time and space are  $O(n + m)$ . The time bound depends on using the set merging algorithm of [GT].  $\square$

The key fact in analyzing HLF schedules is that they decompose into "blocks." The blocks are defined from a set of boundary levels  $\ell_i$ ,  $1 \leq i \leq B + 1$ , where  $\ell_1 = 1$ ,  $\ell_{B+1} = L + 1$ . The exact definition of  $\ell_i$  can be found in [G82a]. For  $1 \leq i \leq B$ , block  $X_i$  consists of all nodes scheduled after  $\ell_{i+1}$  up to and including  $\ell_i$ , except for the node (if any) jumped from  $\ell_i$ . Every node is in a block except nodes jumped from boundary levels  $\ell_i$ . The blocks partition the time units of the schedule. The following result is proved in [G82a].

**THEOREM 2.2.** *Assume  $\Delta = 0$ .*

(a) *In an HLF schedule any block  $X$  is executed in  $\lceil |X|/2 \rceil$  time units.*

(b) *For any block  $X_i$ ,  $1 < i \leq B$ ,  $X_i \xrightarrow{*} X_{i-1}$ , i.e., any node of  $X_i$  precedes any node of  $X_{i-1}$ .*  $\square$

Theorem 2.2 implies that an HLF schedule for identical processors is optimum, since its length is  $\sum_{i=1}^B \lceil |X_i|/2 \rceil$ . HLA schedules have a similar block structure, but it is not needed in this paper (although a block structure is used in §5).

**3. HLA schedules.** This section proves that when  $\Delta = 1$  there is an optimum HLA schedule. This leads to an algorithm that finds an optimum schedule in  $O(n + m)$  time when there is no idle and  $O(2^L(n + m))$  time in general. It also gives a linear-time algorithm for length two pipelines.

The proof is organized as follows. Lemmas 3.1–3.3 show that an optimum schedule that is level always exists. Lemma 3.4–3.6 establish some "highest-level-first" types of properties. Theorem 3.1 shows that an optimum schedule that is HLA always exists. All these results are for the case  $\Delta = 1$ . They simplify to show that an HLF schedule is optimum for length two pipelines (Corollary 3.2), and also  $\Delta = 0$ , i.e., two identical processors (Corollary 3.3). If  $\Delta > 1$  the results do not apply; in fact there need not even exist an optimum schedule that is level (Theorem 3.2).

The first result will allow us to consider only schedules that execute nodes in slots (Lemma 3.3).

**LEMMA 3.1.** *For any slot  $\sigma$ , there is an optimum  $\sigma$ -schedule that executes a node in the first slot  $\sigma$ .*

*Proof.* Let  $S$  be an optimum  $\sigma$ -schedule ( $S$  does not necessarily execute nodes in slots). Slot  $\sigma$  ends at the first time any schedule can complete a job (even when  $\sigma = 2f$ ). If  $S$  does not use any portion of  $\sigma$ , let  $x$  be the first node to start executing in  $S$ . Moving  $x$  to  $\sigma$  gives a valid  $\sigma$ -schedule no longer than  $S$  (recall length starts at  $\sigma$ ). Similarly if  $S$  executes a portion of some node  $x$  in  $\sigma$ , move  $x$  to  $\sigma$ .  $\square$

The next lemma is the heart of the transformation of an optimum schedule to a level schedule. First some notation. Let  $S$  be a schedule that executes nodes in slots, such as a level schedule.  $S/x$  denotes the schedule  $S$  starting at the slot after  $x$ . For example for  $S$  the schedule Figure 2.3(b),  $S/c$  is a 3-schedule for nodes  $d, e, f, g$ . (Sometimes it is convenient to use  $S/x$  to refer to the nodes after  $x$ ; no confusion will result.) An *almost-level* schedule  $S$  is a schedule such that for the first node executed



$x$ ,  $S/x$  is a level  $\sigma$ -schedule for  $G - x$  (and some  $\sigma$ ). Recall that  $L$  denotes the number of levels in the dag.

LEMMA 3.2. *If  $G$  has an optimum  $\sigma$ -schedule that is almost-level then it has an optimum  $\sigma$ -schedule with a level  $L$  node in the first slot.*

*Proof.* The proof is by induction on  $L$ . The base case  $L = 1$  is trivial. Assume the lemma holds for graphs with less than  $L$  levels and prove it for  $L$  as follows.

Consider an optimum  $\sigma$ -schedule  $S$  that is almost-level; without loss of generality assume  $x$ , the node in the first slot, has  $level(x) < L$ . Let  $y$  be the node after  $x$ ; since  $S$  is almost-level  $level(y) = L$ . Let  $S'$  be  $S$  with nodes  $x$  and  $y$  interchanged. If  $S'$  is not a valid schedule, it violates a precedence constraint  $x \xrightarrow{*} z$ , where node  $z$  is in the slot originally overlapping  $y$ . Clearly  $level(z) < level(x) \leq L - 1$ . So  $G - \{x, y\}$  is an  $L - 1$  level graph and  $S/y$  is an almost-level schedule for it. By induction it has an optimum schedule  $T$  whose first node  $w$  is on level  $L - 1$  and is in the first slot of  $S/y$ . Since  $level(x) < L$ ,  $x \not\xrightarrow{*} w$ . So a valid schedule for  $G$  results from scheduling  $y$  and  $x$  in the first two slots and then  $T$ . This is the desired schedule.  $\square$

Now we show that an optimum schedule can be transformed to a level schedule.

LEMMA 3.3. *Any slot  $\sigma$  has a level  $\sigma$ -schedule that is an optimum  $\sigma$ -schedule.*

*Proof.* The proof is by induction on  $n$ , the number of nodes in  $G$ . The base  $n = 1$  is trivial. Assume the lemma for graphs with fewer than  $n$  nodes and prove it for  $G$  as follows.

Observe that  $G$  has an optimum  $\sigma$ -schedule  $S$  that starts with a level  $L$  node in slot  $\sigma$ : There is an optimum  $\sigma$ -schedule with some node  $x$  in  $\sigma$  by Lemma 3.1. Suppose  $level(x) < L$ . By induction  $G - x$  has a level  $\sigma^+$ -schedule that is optimum. This schedule preceded by  $x$  in  $\sigma$  is an almost-level  $\sigma$ -schedule for  $G$  that is optimum. ( $x$  does not precede the node in  $\sigma^+$  since  $level(x) < L$ .) Now Lemma 3.2 shows the desired schedule  $S$  exists.

Let  $x$  be the first node of  $S$ . If  $x$  is not the only level  $L$  node then  $x$ , followed by the schedule for  $G - x$  given by induction, is the desired schedule. So suppose  $x$  is the only node of  $L$ . If in  $S$  the processor opposite  $x$  is idle during the entire duration of slot  $\sigma$ , then  $x$  is followed by a 1-schedule for  $G - x$ . By induction this schedule can be assumed level, as desired.

Otherwise the processor opposite  $x$  executes a portion of some node  $y$  during  $\sigma$ . Clearly  $y$  can be assumed to be in slot  $\sigma^+$ . By induction  $G - \{x, y\}$  has a level  $\sigma^{++}$ -schedule that is optimum. This schedule preceded by  $x$  in  $\sigma$  and  $y$  in  $\sigma^+$  is the desired schedule. ( $y$  does not precede the node in  $\sigma^{++}$  since the latter is on level  $L - 1$ .)  $\square$

The next two lemmas show initial nodes have "highest-level-first" properties.

LEMMA 3.4. *If  $x$  is an initial node then any slot  $\sigma$  has an optimum level  $\sigma$ -schedule where all nodes before  $x$  are on level( $x$ ) or higher.*

*Proof.* The proof is by contradiction. Let  $S$  be an optimum level  $\sigma$ -schedule where  $x$  is in the earliest slot possible. Suppose a node  $y$  with  $level(y) < level(x)$  is scheduled before  $x$ ; choose  $y$  as the last such node. Node  $y$  is jumped since  $x$  is scheduled at  $level(x)$  or before. We derive a contradiction by showing  $x$  can be scheduled earlier.

Let  $T$  be the result of interchanging  $x$  and  $y$  in  $S$ . The first case is when  $T$  is a valid schedule.  $T$  is a level schedule up to and including  $x$ , which is jumped. Let  $x$  be in slot  $\tau$  in  $T$ . Let  $U$  be an optimum level  $\tau^+$ -schedule for  $T/x$  (by Lemma 3.3). Let  $V$  be the schedule  $T$  with  $T/x$  replaced by  $U$ .  $V$  is a valid schedule. ( $x$  does not precede the node in  $\tau^+$ , which is on  $level(x)$  or higher.)  $V$  gives the desired contradiction.

The second case is when  $T$  is not a valid schedule.  $T$  can only violate a precedence

constraint  $y \xrightarrow{*} z$  for some node  $z$  with  $level(z) < level(y) < level(x)$ .  $S$  does not schedule  $z$  before  $x$ , by choice of  $y$ . Thus in  $S$ ,  $z$  is the node after  $x$ , and  $level(x)$  jumps from  $x$  to  $z$ .

Let  $y$  be in slot  $\kappa$  in  $T$  (so  $z$  is in  $\kappa^+$ ). Let  $U$  be an optimum level  $\kappa^+$ -schedule for  $T/y$ . Let  $V$  be  $T$  with  $T/y$  replaced by  $U$ .  $V$  is a valid schedule. ( $y$  does not precede the node in  $\kappa^+$ , which is on  $level(x) - 1$ .) If  $V$  is level it gives the desired contradiction.

If  $V$  is not level then  $\kappa = 1$  (in this case  $level(x)$  has a jump in  $S$  but not in  $T$ ). Let  $w$  be the node before  $y$  in  $V$ . Let  $W$  be  $V$  with  $V/w$  replaced by an optimum level 1-schedule.  $W$  gives the desired contradiction.  $\square$

LEMMA 3.5. *Let  $X$  and  $Y$  be dags where node  $x$  is initial in  $X$ , node  $y$  is initial in  $Y$  and  $X - x = Y - y$ . Suppose  $level(x) > level(y)$ . Then for any slot  $\sigma$  an optimum  $\sigma$ -schedule for  $Y$  is no longer than one for  $X$ .*

*Proof.* By Lemma 3.4 let  $S$  be an optimum level  $\sigma$ -schedule for  $X$  where all nodes before  $x$  are on  $level(x)$  or higher. Let  $T$  be  $S$  with  $x$  replaced by  $y$ . Suppose  $T$  is not a valid schedule for  $Y$ .  $T$  can only violate a precedence constraint  $y \xrightarrow{*} z$  for some node  $z$  with  $level(z) < level(y) < level(x)$ . This implies  $S$  jumps from  $x$  to  $z$ , so in  $T$ ,  $z$  is in the slot  $\tau$  after  $y$ . Thus  $T/y = S/x$  is a  $\tau$ -schedule for a dag with  $level(x) - 1$  levels.

Let  $U$  be an optimum level  $\tau$ -schedule for  $T/y$ . Let  $V$  be  $T$  with  $T/y$  replaced by  $U$ .  $V$  is a valid schedule for  $Y$ . ( $y$  does not precede the node in  $\tau$ , which is on  $level(x) - 1$ .) This gives the lemma.  $\square$

The next lemma investigates the situation where there is a choice of nodes to jump on the same level.

LEMMA 3.6. *Let  $X$  and  $Y$  be dags where node  $x$  is initial in  $X$ , node  $y$  is initial in  $Y$ ,  $level(x) = level(y) = \ell$  and  $X - x = Y - y$ . Let  $\sigma$  be any slot and let  $A$  be a set of abstaining levels. Let  $S$  be an HLA  $\sigma$ -schedule for  $A$  on  $X$ . Then there exists  $T$ , an HLA  $\sigma$ -schedule for  $A$  on  $Y$ , with these properties:*

- (i)  *$S$  and  $T$  have the same jump sequence before  $\ell$ ;*
- (ii) *Let  $X'$  be the subgraph of  $X$  that remains after the execution of level  $\ell$ , and similarly for  $Y'$ . Either  $X' = Y'$ , or  $\ell$  jumps a node  $y'$  in  $S$  and  $x'$  in  $T$  such that  $X' - x' = Y' - y'$ .*

*Proof.* For succinctness in this argument, “good schedule” stands for “HLA  $\sigma$ -schedule for  $A$ .” We first prove property (i). It asserts that any two good schedules for  $X$  and  $Y$  have the same jump sequence before  $\ell$ . We will show that if  $S$  is a good schedule for  $X$  its jump sequence before  $\ell$  is at most that of a good schedule for  $Y$ . Symmetry gives (i).

So let  $S$  be a good schedule for  $X$  and let  $T$  be  $S$  with  $x$  replaced by  $y$ . First observe that  $T$  is a valid schedule unless  $\ell$  jumps from  $x$  to a successor of  $y$ : Suppose  $T$  is not valid. It can only violate a precedence constraint  $y \xrightarrow{*} z$  for a node  $z$  with  $level(z) < \ell$ . Since  $S$  is HLA and  $x$  is initial,  $S$  does not jump  $z$  before  $x$  is executed. Thus in  $S$ ,  $\ell$  jumps from  $x$  to  $z$ , as desired.

The observation implies that before  $\ell$ ,  $T$  is a valid level  $\sigma$ -schedule with abstentions  $A$  ( $T$  is level since  $level(x) = level(y)$ ). So the jump sequence before  $\ell$  of a good schedule for  $Y$  is at least that of  $T$ . This is the same as that of  $S$  (again recall  $level(x) = level(y)$ ). Property (i) follows.

To show property (ii) consider two cases. First suppose that in  $S$  no level above  $\ell$  jumps below  $\ell$ . Let  $T$  be any good schedule for  $Y$ . By (i)  $T$  has the property supposed for  $S$ . Clearly (ii) holds.

The second case is when in  $S$  some level  $f > \ell$  jumps below  $\ell$ . By (i)  $f$  jumps below  $\ell$  in any good schedule for  $Y$ . Since  $S$  is HLA, it jumps  $x$  before  $f$ . Similarly any good schedule for  $Y$  jumps  $y$  before  $f$ . Now the argument for (i) shows that a good schedule has the same jump sequence in  $X$  as in  $Y$ , and that  $S$ , with  $x$  replaced by  $y$ , is a good schedule for  $Y$ . Choosing it as  $T$  gives (ii).  $\square$

Now we show the main result.

**THEOREM 3.1.** *Assume  $\Delta = 1$ . Any slot  $\sigma$  has a set of levels  $A$  such that any HLA  $\sigma$ -schedule for  $A$  is an optimum  $\sigma$ -schedule.*

*Proof.* The proof is by induction on  $L$ . If  $L = 1$  the theorem is trivial. Assume the theorem for fewer than  $L$  levels and prove it for  $L$ , as follows. Let  $S$  be an optimum level  $\sigma$ -schedule. Let  $z$  be the last node in the execution of  $L$ . By induction assume  $S/z$  is an HLA schedule. Choose  $S$  so that subject to the above conditions its jump sequence is as high as possible. Let  $A$  be the abstentious levels for  $S$ , i.e.,

$$A = \{\ell \mid \text{level } \ell \text{ ends in slot } \tau, 1 \leq \tau < 2f, \text{ but } \ell \text{ does not jump a node}\}.$$

To prove the theorem it suffices to show  $S$  is an HLA schedule for  $A$ . This is obvious if  $S$  does not jump from  $L$  (i.e., either  $L \in A$  or  $L$  ends at slot  $2f + 1$ ). So assume  $S$  does a jump from  $L$ .

We will argue inductively to find an HLA schedule  $T$  for  $A$  and levels  $\ell_j$ ,  $L = \ell_1 > \ell_2 > \dots > \ell_k = 0$ , such that:

- (i) For  $1 \leq j \leq k$ ,  $S$  and  $T$  have the same jump sequence above  $\ell_j$ .
- (ii) For  $1 \leq j < k$ ,  $\ell_j$  jumps a node  $y_j$  in  $S$  and  $x_j$  in  $T$  such that  $X_j - x_j = Y_j - y_j$ , where  $X_j$  ( $Y_j$ ) is the subgraph induced on all nodes after the execution of  $\ell_j$  in  $S$  ( $T$ ).

Property (i) for  $j = k$  gives the desired conclusion.

Start by choosing  $T$  as an arbitrary HLA schedule for  $A$ . For the base case  $j = 1$ , property (i) is vacuous. For (ii) recall  $S$  jumps a node  $y_1$  from  $L$ . So clearly  $T$  jumps a node  $x_1$  from  $L$ . Both  $X_1 - x_1$  and  $Y_1 - y_1$  are the subgraph of all nodes below level  $L$  excluding  $x_1$  and  $y_1$ .

For the inductive step assume (i)–(ii) for  $j < k$ , and prove it for  $j + 1$  as follows. Write  $\ell = \text{level}(x_j)$ . Since  $T$  is an HLA schedule for  $A$ ,  $\ell \geq \text{level}(y_j)$ . To show equality holds (as required by property (i)) suppose the contrary,  $\ell > \text{level}(y_j)$ . Let  $\tau$  be the slot after  $x_j$  (equivalently after  $y_j$ ). Let  $U$  be  $T$ , with  $T/x_j$  replaced by an optimum, HLA  $\tau$ -schedule (for  $Y_j$ ). This schedule exists by induction, although it may not abstain at the levels in  $A$ . Since  $\ell < \ell_j$ ,  $U$  is a valid level  $\sigma$ -schedule. Further Lemma 3.5 implies  $U$  is optimum for  $G$ . But  $U$  shows that  $S$  does not have the highest jump sequence possible, contradicting its definition. We conclude  $\ell = \text{level}(y_j)$  as desired.

Now Lemma 3.6 can be applied to graphs  $X_j$  and  $Y_j$  (the choice of  $S$  makes it HLA on  $X_j$ ). It shows  $S$  and  $T$  have the same jump sequence above  $\ell$ . Further  $T$  can be chosen so that (ii) of Lemma 3.6 holds. Using the notation of Lemma 3.6, either  $X' = Y'$  or  $X' - x' = Y' - y'$ .

In the first case  $X' = Y'$ ,  $S$  and  $T$  can be assumed identical after  $\ell$ . Take  $\ell_{j+1} = 0$  and  $k = j + 1$  to complete the induction.

In the second case  $X' - x' = Y' - y'$ , take  $\ell_{j+1} = \ell$ ,  $y_{j+1} = y'$  and  $x_{j+1} = x'$  (so

$X_{j+1} = X'$  and  $Y_{j+1} = Y'$ ).  $k$  will be a value  $> j + 1$ . Clearly (i) and (ii) hold for  $j + 1$ .  $\square$

The key to finding the optimum HLA schedule is finding the abstentious levels. We have not succeeded in doing this. Here are two simple applications of the theorem.

**COROLLARY 3.1.** *Assume  $\Delta = 1$ .*

(a) *If there is a schedule with no idle time, it can be found in  $O(n + m)$  time and space.*

(b) *An optimum schedule can be found in  $O(2^L(n + m))$  time and  $O(n + m)$  space.*

*Proof.* (a) Apply Theorems 3.1 and 2.1 with  $A = \emptyset$ .

(b) There are at most  $2^L$  possible sets of abstentious levels  $A$ . Find an HLA schedule for each and select the shortest.  $\square$

Note also that if the amount of idle time is  $O(1)$  the schedule can be found in polynomial time. This follows since a schedule with  $I$  units of idle abstains at most  $I$  times. This fact may be of theoretic interest: We have not eliminated the possibility that the general  $\Delta = 1$  problem is NP-complete. This fact shows that a reduction must use an unbounded amount of idle time, unlike most other scheduling problems.

We give two more applications of Theorem 3.1, for the case of no abstentions. The first is to pipeline scheduling. A *length  $k$  pipeline processor* has  $k$  stages, connected in series. A job is executed in  $k$  time units—it passes through each stage in order, spending one time unit at each stage. Thus jobs are executed in slots corresponding to the time intervals  $[t, t + k)$  for nonnegative integers  $t$ . The pattern of slots does not change even when there is idle: If a job is not started at time  $t$ , the complete slot  $[t, t + k)$  is idle.

For a length two pipeline, the pattern of slots is similar to the  $\Delta = 1$  case, since each slot (except the first and last) overlaps two others. Since the slot pattern does not change, there is no advantage in abstaining.

**COROLLARY 3.2.** *For a length two pipeline any HLF schedule is optimum. Such a schedule can be found in  $O(n + m)$  time and space.*

*Proof.* The argument of Theorem 3.1 shows any HLF schedule (i.e., an HLA schedule with  $A = \emptyset$ ) is optimum. The only change is to adopt the above notion of slot. The HLF schedule is found as in Theorem 2.1.  $\square$

The proof of Theorem 3.1 is easily adapted to show that an HLF schedule is optimum for identical processors. This derivation is independent from the one in [G82a].

**COROLLARY 3.3.** *For two identical processors any HLF schedule is optimum.*

*Proof.* For identical processors an execution period is one time unit long with slots numbered one and two. By convention a jumped node goes in slot two. The proofs of the lemmas and the theorem are simpler because slot two does not overlap the following slot.  $\square$

Returning to uniform processors, the HLA characterization holds only for speeds with  $\Delta = 1$ . When  $\Delta > 1$  some dags do not even admit an optimum level schedule.

**THEOREM 3.2.** *Assume  $\Delta > 1$ . There is a dag where no level schedule is optimum.*

*Proof.* The argument depends on the relative sizes of  $2f$  and  $s$ . Equality does not occur since it implies  $f = 1, s = 2, \Delta = 1$ .

Case  $2f > s$ . Consider Figure 3.1. It consists of components  $S$  and  $F$ , where  $S$  is a chain of  $s - 1$  nodes and  $F$  has two nodes  $a, a'$  that precede a chain of  $f - 1$  nodes.

An optimum schedule  $O$  has length  $fs$  with no idle time:  $P_f$  executes node  $a$  followed by the nodes of  $S$ .  $P_s$  executes  $a'$  followed by the remaining nodes of  $F$ .  $O$  is not a level schedule since  $a$  and  $a'$  are on level  $f < s - 1 = L$ .

To prove the theorem it suffices to show  $O$  is the only optimum schedule. First consider a dag consisting of two chains. If a schedule with no idle exists, each processor executes nodes from one chain for  $fs$  time units. At this point the processors can switch chains but not before.

Now consider a schedule with no idle for Figure 3.1. There are three cases, depending on the nodes executed at time zero. If  $a$  and  $a'$  are executed the preceding observation implies the schedule is  $O$ . If  $P_s$  executes  $b$  the observation implies  $P_f$  executes the entire  $F$  component while  $P_s$  executes  $S$ . But  $F$  has  $f + 1 < s$  nodes, giving idle time on  $P_f$ .

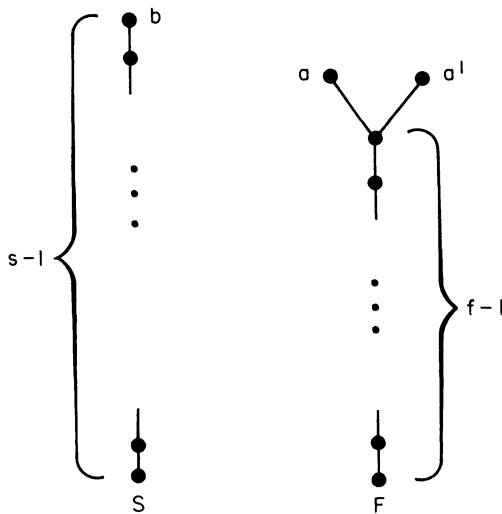


FIG. 3.1. Nonlevel dag for  $2f > s$ .

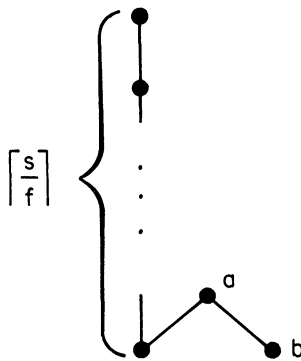


FIG. 3.2. Nonlevel dag for  $2f < s$ .

The last case is where  $P_f$  executes  $b$  and  $P_s$  executes  $a$  at time zero. If  $P_f$  executes  $a'$  after  $b$ ,  $2f > s$  implies  $P_s$  switches to component  $S$  while  $P_f$  executes  $F$ . This gives idle time as in the previous case. If  $P_f$  does not execute  $a'$  after  $b$ ,  $2f > s$  implies  $P_s$  executes  $F$  while  $P_f$  executes  $S$ . But since  $S$  has only  $s - 1$  nodes this gives idle time on  $P_f$ .

*Case  $2f < s$ .* In Figure 3.2, the dag has height  $\lceil s/f \rceil \geq 3$ . An optimum schedule  $O$  has length  $\omega^* = (\lceil s/f \rceil + 1)f$ : First  $P_f$  executes node  $a$  while  $P_s$  is idle. Then  $P_f$  executes all nodes but  $b$  while  $P_s$  executes  $b$ . ( $P_s$  can finish at time  $f + s \leq \omega^*$ .) Clearly  $O$  is not level.

The optimum is unique: Observe  $2s > s + 2f > \omega^*$ . Hence to achieve  $\omega^*$ ,  $P_s$  must execute precisely one node; further that node must be  $b$ . Now it is easy to see the schedule must be  $O$ .  $\square$

**4. The AI approximation algorithm.** This section investigates the strategy of using an optimum identical processor schedule for two uniform processors. The worst-case accuracy bound is  $2 - (f/s)$ . The time and space to find the schedule are  $O(n + m)$ .

An AI (*Approximately Identical*) schedule  $T$  is constructed as follows. Start with an HLF schedule  $S$  for two identical processors. Then convert it to  $T$ : A time unit of  $S$  where one node is executed corresponds to  $f$  time units of  $T$  where the same node is executed on  $P_f$ . A time unit of  $S$  where two nodes are executed corresponds to  $s$  time units of  $T$  where the same two nodes are executed.  $P_s$  executes one node while  $P_f$  executes the other with  $\Delta$  units of idle.

An AI schedule can be found in  $O(n + m)$  time. To derive the accuracy bound define these quantities from the AI schedule:

- $t$  = the number of nodes on  $P_f$  scheduled with a node on  $P_s$ ;
- $u$  = the number of nodes on  $P_f$  scheduled with idle on  $P_s$ .

Clearly  $\omega = st + fu$ . Since the optimum schedule executes at most  $(1/f + 1/s)\omega^*$  nodes, it is clear that

$$(1) \quad (1/f + 1/s)\omega^* \geq 2t + u.$$

Now we derive another bound on  $\omega^*$ .

LEMMA 4.1.  $\omega^* \geq tf + uf$ .

*Proof.* The lemma is proved by decomposing the schedule into blocks  $X_j$ ,  $j = 1, \dots, B$  (recall Theorem 2.2). Define these quantities for a block  $X$ .

- $\omega^*(X)$  = the length of an optimum schedule for the nodes of  $X$ ;
- $t(X)$  = the contribution to  $t$  made by  $X$  in the AI schedule;
- $u(X)$  = the contribution to  $u$  made by  $X$  in the AI schedule.

Observe that  $\omega^* \geq \sum_{j=1}^B \omega^*(X_j)$  by Theorem 2.2(b). Also  $t + u = \sum_{j=1}^B t(X_j) + u(X_j)$ . So it suffices to show  $\omega^*(X) \geq t(X)f + u(X)f$  for every block  $X$ .

Theorem 2.2(a) implies that in the AI schedule  $P_f$  executes exactly  $\lceil |X|/2 \rceil$  nodes of  $X$ . So  $\lceil |X|/2 \rceil = t(X) + u(X)$ . Clearly  $\omega^*(X) \geq \lceil |X|/2 \rceil f$ , since a schedule executes at least  $\lceil |X|/2 \rceil$  nodes on some processor. This gives the desired conclusion.  $\square$

THEOREM 4.1. (a) *On any dag the AI schedule achieves the bound  $\omega/\omega^* \leq 2 - (f/s)$ .*

(b) *There is a dag where the AI schedule has  $\omega/\omega^* = 2 - (f/s)$ .*

*Proof.* (a) Multiplying (1) by  $s - f$ , Lemma 4.1 by  $2 - (s/f)$ , and adding gives  $(2 - (f/s))\omega^* \geq st + fu$ . The right-hand side is  $\omega$ .

(b) Consider a dag consisting of a chain of  $s$  nodes plus  $f$  isolated nodes. The AI schedule executes a chain node with an isolated node until all isolated nodes are exhausted. Then it executes chain nodes with idle. So  $\omega = fs + \Delta f$ .

The optimum schedule executes the chain nodes on  $P_f$  and the isolated nodes on  $P_s$ , so  $\omega^* = fs$ .  $\square$

The obvious way to improve the AI schedule is to compress idle on  $P_f$ : When two consecutive nodes on  $P_f$  are on the same level, the second node can be scheduled after the first with no intervening idle. Similarly when a node is jumped on  $P_s$ , the node following it on  $P_f$  can be scheduled right after its predecessor. The *compressed AI (CAI) schedule* is the result of applying these rules as much as possible to remove idle. (If  $\lceil f/\Delta \rceil$  nodes are compressed a whole new slot is available on  $P_f$ . We will not concern ourselves with how this slot is used since our lower bounds do not depend on it.) The CAI schedule has an alternate description: It is a level schedule that never jumps from processor  $P_s$  but otherwise has the highest jump sequence possible.

Our results on CAI indicate that compression does not help—there are dags where the accuracy is the same as AI. The main result is stated below. The proof is a construction similar to §5. Details and other results can be found in [G82b].

**THEOREM 4.2.** *Assume  $f/s \geq 4/5$  ( $\Delta$  is arbitrary). For any  $\epsilon > 0$  there is a dag where the CAI schedule has  $\omega/\omega^* > 2 - (f/s) - \epsilon$ .  $\square$*

**5. The HLF approximation algorithm.** This section investigates the HLF scheduling rule as an approximation algorithm. When  $\Delta = 1$  an HLF schedule can be found in  $O(m + n \log \log n)$  time and  $O(n + m)$  space. Two important cases of HLF scheduling are analyzed: When  $s/f = 1/2$  the worst-case accuracy bound is  $5/4$ . When  $s/f = 2/3$  the bound is  $6/5$ . These are the main cases where HLF is preferable to the AI schedule, which is more accurate than HLF when  $s/f \geq 4/5$ .

The HLF scheduling algorithm differs from the HLA algorithm of Theorem 2.1 because the slots are not known in advance. Nonetheless a similar algorithm works for HLF scheduling: Pass I processes levels  $\ell$  in decreasing order. When processing  $\ell$  it finds the level  $t$  (if any) that  $\ell$  jumps to. (Note that since all idle jumps from levels above  $\ell$  are known, it is easy to decide if  $\ell$  has a jump or not. This organization differs from Theorem 2.1, which when processing  $\ell$  finds the levels that jump to it.) Pass II finds the specific jumps.

The main data structure of the algorithm is a priority queue that finds the highest level  $\ell$  can jump. Since levels are integers between 1 and  $n$  the queue can be efficiently implemented using the data structure of van Emde Boas [E76], [E77].

**THEOREM 5.1.** *Assume  $\Delta = 1$ . An HLF schedule can be found in  $O(m + n \log \log n)$  time and  $O(m + n)$  space.*

*Proof.* Further details are similar to the identical processor case of [G82a]. They are given explicitly in [G82b].  $\square$

HLF schedules decompose into blocks, as in Theorem 2.2:

**THEOREM 5.2.** *Assume  $\Delta = 1$ .*

(i) *In an HLF schedule any block  $X$  is executed in consecutive time slots  $\sigma, \sigma + 1, \dots, \tau$  with no intervening idle. Here  $1 \leq \sigma, \tau \leq 2f + 1$  and  $\sigma \neq \tau$ .*

(ii) *For any block  $X_i$ ,  $1 < i \leq B$ ,  $X_i \xrightarrow{*} X_{i-1}$ .*

*Proof.* In part (i),  $\sigma \neq \tau$  since a level schedule never jumps a node in slot one. The rest of the proof follows the identical processor case of [G82a]. The details can

be found in [G82b].  $\square$

The analysis of HLF schedules is done on blocks. Define the following quantities for any block  $X$ :

- $\omega(X)$  = the length of the HLF schedule for  $X$ ;
- $\omega^*(X)$  = the length of an optimum schedule for  $X$ ;
- $\delta(X) = \omega(X) - \omega^*(X)$ .

If  $\sigma$  is the first slot of  $X$  in the HLF schedule then it gives a  $\sigma$ -schedule for  $X$ .  $\omega(X)$  is the length of this  $\sigma$ -schedule, i.e., time zero is at the start of slot  $\sigma$ . So the length of the HLF schedule is  $\omega = \sum_{j=1}^B \omega(X_j)$ .  $\omega^*(X)$  is the length of an optimum 1-schedule for  $X$ . This allows both processors to start simultaneously, unlike  $\omega(X)$ . The length of an optimum schedule is  $\omega^* \geq \sum_{j=1}^B \omega^*(X_j)$ , by Theorem 5.2(b).

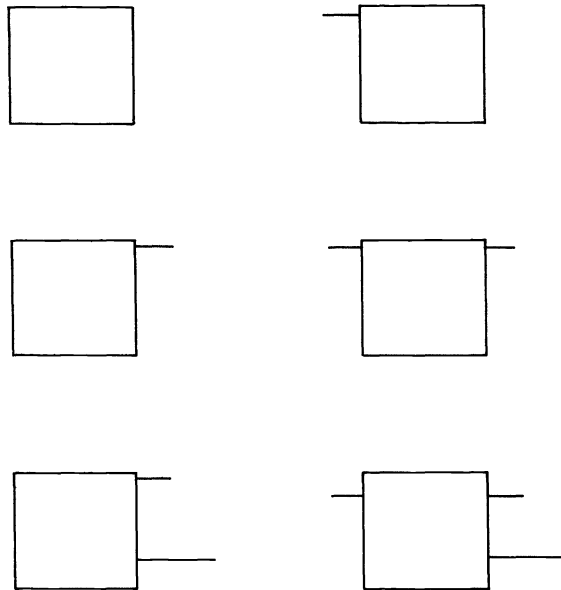


FIG. 5.1. The six block shapes for  $f/s = 1/2$ .

First we analyze the HLF schedule for  $f/s = 1/2$ . Figure 5.1 shows the six possible shapes for the slots of a block. It justifies the following result.

LEMMA 5.1. *In a nonoptimum block  $X$  the first slot is three, the last slot is two and  $\delta(X) = 1$ .*  $\square$

To analyze the accuracy of the HLF schedule let  $i$  be the number of nonoptimum blocks. The bound follows from two inequalities.

- LEMMA 5.2. (a)  $\omega^* \geq \omega - i$ .
- (b)  $(3/2)\omega^* \geq \omega + i$ .

*Proof.* (a) Lemma 5.1 shows the number of time units the schedule can decrease is at most the number of nonoptimum blocks.

(b) Observe there are at least  $\omega + i$  nodes: Figure 5.1 shows that each time unit of the schedule can be associated with a distinct node in some block. In addition there are  $i$  nodes not in any block: Every nonoptimum block starts in slot three, and the node jumped in slot two is not in any block.



This implies (b) since the optimum schedule executes at most  $(3/2)\omega^*$  nodes.  $\square$

THEOREM 5.3. Assume  $f/s = 1/2$ .

(a) On any dag the HLF schedule has  $\omega/\omega^* \leq 5/4$ .

(b) For any  $\epsilon > 0$  there is a dag where the HLF schedule has  $\omega/\omega^* > 5/4 - \epsilon$ .

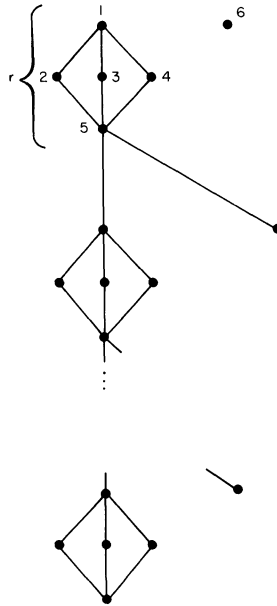


FIG. 5.2. Bad dag for the HLF schedule for  $f/s = 1/2$ .

1	2	3	5
6		4	
...			

(a)

2	3	5	1
4		6	
		.....	

(b)

FIG. 5.3. Module schedules. (a) HLF schedule. (b) Better schedule.

*Proof.* (a) Add the inequalities of Lemma 5.2.

(b) For any positive integer  $r$  consider Figure 5.2. This dag consists of  $r$  repetitions of a “module” with six nodes. (Node six is on level one but is drawn higher for clarity. Note the resemblance to Figure 2.1.)

The HLF schedule is  $r$  repetitions of the module schedule shown in Figure 5.3(a). Thus  $\omega = 5r$ . However this schedule shows  $\omega^* \leq 4r + 1$ :  $P_1$  executes node one of the first module. Then come  $r$  repetitions of the module schedule of Figure 5.3(b), where each module executes node one of the next module. (The last module has idle in the last slot three.)

Thus  $\omega/\omega^* \geq 5r/(4r + 1)$ , which approaches  $5/4$  as  $r$  approaches  $\infty$ .  $\square$

We turn to the HLF schedule for  $f/s = 2/3$ . It has a different character from previous approximate schedules. Regarding the lower bound dags, recall that in the previous algorithms the optimum schedule has  $\Theta(1)$  idle time. This is true in all other examples for level-type scheduling algorithms that we know of [L], [LS]. We have not found such examples for this algorithm— the dag that achieves the tight lower bound has  $\Theta(\omega^*)$  idle time.

This property manifests itself in the proof of the upper bound. The previous schedules are analyzed by inequalities based on the fact that the number of nodes gives a lower bound on  $\omega^*$ . Inequalities of this type cannot be tight in dags with  $\Theta(\omega^*)$  idle in the optimum schedule. Hence a different approach must be used. We use an estimate of how much  $\omega$  can decrease, derived from the precedence constraints.

We first investigate how much a block can shrink. For generality consider two processors of speeds  $f$  and  $s = f + 1$ , and a block  $X$  in some HLF schedule. Define this quantity from the HLF schedule:

$$T(X) = i/f + j/s, \text{ where } i(j) \text{ is the number of time units in } \omega(X) \text{ when } P_f(P_s) \text{ is not executing a node of } X.$$

In the HLF schedule for  $X$  both processors are always busy except possibly during the first slot (if one processor is executing the node jumped before  $X$ ) and the last slot (if one processor is idle or executing the node jumped from  $X$ ).  $T(X)$  gives the number of nodes (possibly fractional) that could have been executed in these two slots.

LEMMA 5.3.  $\delta(X) \leq T(X)/(1/f + 1/s)$ .

*Proof.* If  $n$  is the number of nodes in  $X$  then  $(1/f + 1/s)\omega(X) = n + T(X)$  and  $(1/f + 1/s)\omega^*(X) \geq n$ .  $\square$

TABLE 1  
Slot contributions to  $T(X)$  for  $f/s = 2/3$ .

		slot				
		1	2	3	4	5
first		0	—	1/3	1/2	2/3
last		2/3	1/2	1/3	1	0

Table 1 gives the contribution of each slot to  $T(X)$  when it is the first or last slot in  $X$ . For instance if the first slot of  $X$  is three then during the initial time unit of this slot  $P_3$  is idle, contributing  $1/3$  to  $T(X)$ . (If  $X$  consists of only one slot, its contribution to  $T(X)$  is the sum of the first and last contributions.)

LEMMA 5.4. *Let  $X$  be a block.*

(a)  $\delta(X) \leq 2$ .

(b) *If  $\delta(X) = 2$  then slot five is first in  $X$  and four is last.*

(c) *If  $\delta(X) > 0$  then slot three, four or five is first in  $X$ .*

*Proof.* Table 1 shows that  $T(X) \leq 5/3$ , with equality only when slot five is first and four is last. Applying Lemma 5.3 gives (a) and (b). For (c) note that any block with slot one first is optimum and slot two is never first.  $\square$

To obtain a global view of how the schedule can shrink, merge blocks into "segments." These are the portions of the schedule between idle jumps: A *segment*  $W$  consists of consecutive blocks,  $W = \cup_{i=t}^s X_i$ , where  $X_{s-1}$  and  $X_t$ , but no  $X_i$ ,  $s \geq i > t$ , end with an idle jump. (For boundary conditions allow  $s = B$  in the first segment and  $t = 1$  in the last segment.)

The segments partition the blocks. Each segment starts in slot one of a period. Both processors run uninterrupted, jumping a node between consecutive blocks of the segment, until the last slot of the segment (where an idle jump may be made). To analyze the length of the schedule for a segment  $W$  define these quantities:

$\omega(W)$  = the length of the HLF schedule for  $W$ ;

$\omega^*(W)$  = the length of the optimum schedule for  $W$ .

Note that  $W$  does not contain the nodes jumped from the last level of each of its blocks. So possibly  $\omega^*(W) < \omega(W)$ .

LEMMA 5.5. *For any segment  $W$ ,  $\omega^*(W) \geq (5/6)\omega(W)$ .*

*Proof.* Let  $W = \cup_{i=t}^s X_i$ . Set  $\delta(W) = \sum_{i=t}^s \delta(X_i)$ . Write  $\omega(W) = 6p + q$ ,  $p \geq 0$ ,  $6 > q \geq 0$ , so  $p$  is the number of complete execution periods in the segment. It suffices to show  $\delta(W) \leq p$  since this implies  $\omega^*(W) = \omega(W) - \delta(W) \geq 5p + q$ .

Consider a block  $X$  in  $W$  with  $\delta(X) > 0$ .  $X$  contains slot four or five of the execution period it starts in; also if  $\delta(X) = 2$  then  $X$  ends in another execution period, where it contains slot four. This follows from Lemma 5.4(b)-(c), and the fact that if  $X$  starts in slot three it contains slot four (otherwise  $X$  has only one node, making  $\delta(X) = 0$ ).

This implies we can assign each unit of  $\delta(W)$  to slot four or five of a distinct complete execution period of  $W$ . Hence  $\delta(W) \leq p$  as desired.  $\square$

THEOREM 5.4. *Assume  $f/s = 2/3$ .*

(a) *On any dag the HLF schedule achieves  $\omega/\omega^* \leq 6/5$ .*

(b) *For any  $\epsilon > 0$  there is a dag where the HLF schedule has  $\omega/\omega^* > 6/5 - \epsilon$ .*

*Proof.* (a) Let the segments of the HLF schedule be  $W_j$ ,  $j = S, S-1, \dots, 1$ . From Theorem 5.2(b),  $W_j \xrightarrow{*} W_{j-1}$  for  $S \geq j > 1$ . Hence  $\omega^* \geq \sum_{j=1}^S \omega^*(W_j)$ . Clearly  $\omega = \sum_{j=1}^S \omega(W_j)$ . So the theorem follows from Lemma 5.5.

(b) For any positive integer  $r$  consider Figure 5.4. It consists of  $r$  repetitions of a module with eight nodes. (As in Figure 5.2 nodes 7 and 8 are on level 1 but are drawn higher.)

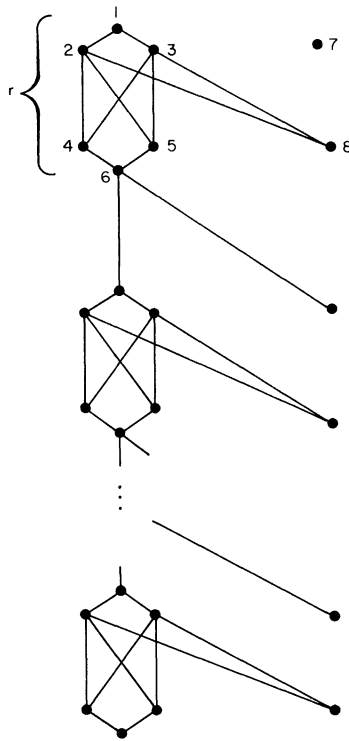


FIG. 5.4. Bad dag for the HLF schedule for  $f/s = 2/3$ .

<u>1</u> <u>2</u>	<u>4</u> <u>8</u> ..
..   .. <u>3</u>	<u>5</u> <u>6</u>

(a)

<u>2</u> .. <u>7</u> <u>5</u>	<u>6</u> <u>1</u>
<u>3</u> .. <u>4</u>	.. <u>8</u> ..

(b)

FIG. 5.5. Module schedules. (a) HLF schedule. (b) Better schedule.

The HLF schedule is  $r$  repetitions of the module schedule shown in Figure 5.5(a). Thus  $\omega = 12r$ . However this schedule shows  $\omega^* \leq 10r + 1$ :  $P_1$  executes node one of the first module. Then come  $r$  repetitions of the module schedule of Figure 5.5(b), where each module executes node one of the next module. (The last module has idle in the last slot three.)

Thus  $\omega/\omega^* \geq 12r/(10r + 1)$  which approaches  $6/5$  as  $r$  approaches  $\infty$ .  $\square$

The remaining results on HLF schedules are negative. The following bound is proved similar to the others; details are in [G82b].

**THEOREM 5.5.** *Assume  $f/s \geq 4/5$  ( $\Delta$  is arbitrary). There is a dag where the HLF schedule has  $\omega/\omega^* = 6/5$ .*  $\square$

Note that when  $f/s \geq 4/5$ , the AI schedule has accuracy  $2 - (f/s) \leq 6/5$ , so it beats the theorem. Since the AI scheduling algorithm has a slightly lower time bound, it is preferable.

Results for lower speed ratios are in [G82b].

**6. Conclusions.** We have shown that an HLF schedule is optimum for length two pipelines, and can be found in linear time. Also an optimum HLA schedule exists for two uniform processors when  $\Delta = 1$ . This gives a linear-time algorithm for an optimum schedule on dags that have no idle time. It would be interesting to extend this to a polynomial time algorithm for the general  $\Delta = 1$  case.

Before summarizing the results for approximate schedules consider one more scheme, oriented toward processors of disparate speed. An *Approximately One Processor (AO) Schedule* is any schedule with no idle time on  $P_f$ . Note that  $\omega/\omega^* \leq 1 + (f/s)$  for AO, since  $(1/f + 1/s)\omega^* \geq \omega/f$ .

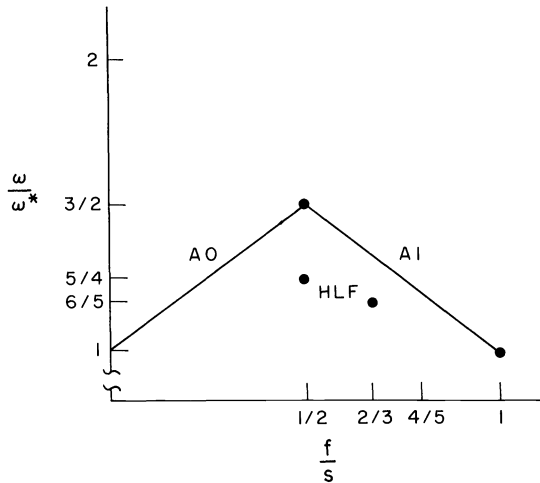


FIG. 6.1. Accuracy bounds for approximate schedules.

Figure 6.1 plots the accuracy of the approximate schedules. The AI schedule is found in linear time and has accuracy  $2 - (f/s)$ . It is more accurate than AO when  $f/s \geq 1/2$ . It is more accurate than HLF when  $f/s \geq 4/5$  and perhaps other cases. The HLF schedule is defined for  $\Delta = 1$ . It is found in  $O(m + n \log \log n)$  time. It is the most accurate for  $f/s = 1/2$  and  $2/3$ .

**Acknowledgments.** The author thanks the anonymous referees for helpful suggestions, and Ed Coffman for his encouragement.

## REFERENCES

- [C] E. G. COFFMAN JR., ed., *Computer and Job-Shop Scheduling Theory*, John Wiley, New York, 1976.
- [CG] E. G. COFFMAN JR. AND R. L. GRAHAM, *Optimal scheduling for two-processor systems*, *Acta Informatica*, 1 (1972), pp. 200-213.
- [E76] P. VAN EMDE BOAS, R. KASS, AND E. ZIJLSTRA, *Design and implementation of an efficient priority queue*, *Math. Systems Theory*, 10 (1977), pp. 99-127.
- [E77] P. VAN EMDE BOAS, *Preserving order in a forest in less than logarithmic time and linear space*, *Inform. Proc. Lett.*, 6 (1977), pp. 80-82.
- [FKN] M. FUJII, T. KASAMI, AND K. NINOMIYA, *Optimal sequencing of two equivalent processors*, *SIAM J. Appl. Math.*, 17 (1969), pp. 784-789; Erratum, *SIAM J. Appl. Math.*, 20 (1971), p.141.
- [G81] H. N. GABOW, *A linear-time recognition algorithm for interval dags*, *Inform. Proc. Lett.*, 12 (1981), pp. 20-22.
- [G82a] ———, *An almost-linear algorithm for two-processor scheduling*, *J. Assoc. Comput. Mach.*, 29 (1982), pp. 766-780.
- [G82b] ———, *Exact and approximate algorithms for scheduling UET systems on two uniform processors*, Technical Rept. CU-CS-225-82, Dept. of Comp. Sci., Univ. of Colorado, Boulder, CO, July 1982.
- [G83] ———, *On the design and analysis of efficient algorithms for deterministic scheduling*, *Proc. 2nd International Conf. on Foundations of Computer-Aided Process Designs*, 1983, pp. 473-528.
- [GJ76] M. R. GAREY AND D. S. JOHNSON, *Scheduling tasks with nonuniform deadlines on two processors*, *J. Assoc. Comput. Mach.*, 23 (1976), pp. 461-467.
- [GJ77] ———, *Two-processor scheduling with start-times and deadlines*, *SIAM J. Comput.*, 6 (1977), pp. 416-426.
- [GLLRK] R. L. GRAHAM, E. L. LAWLER, J. K. LENSTRA, AND A. H. G. RINNOOY KAN, *Optimization and approximation in deterministic sequencing and scheduling: A survey*, *Ann. Discrete Math.*, 5 (1979), pp. 287-326.
- [GT] H. N. GABOW AND R. E. TARJAN, *A linear-time algorithm for a special case of disjoint set union*, *J. Comput. System Sci.*, 30 (1985), pp. 209-221.
- [H61] T.C. HU, *Parallel sequencing and assembly line problems*, *Oper. Res.*, 9 (1961), pp. 841-848.
- [H82] ———, *Combinatorial Algorithms*, Addison-Wesley, Reading, MA, 1982.
- [L] E. L. LLOYD, *Critical path scheduling with resource and processor constraints*, *J. Assoc. Comput. Mach.*, 29 (1982), pp. 781-811.
- [LS] S. LAM AND R. SETHI, *Worst case analysis of two scheduling algorithms*, *SIAM J. Comput.*, 6 (1977), pp. 518-536.
- [PY] C. H. PAPADIMITRIOU AND M. YANNAKAKIS, *Scheduling interval-ordered tasks*, *SIAM J. Comput.*, 8 (1979), pp. 405-409.
- [U] J. D. ULLMAN, *NP-complete scheduling problems*, *J. Comput. System Sci.*, 10 (1975), pp. 384-393.

## A RANDOMIZED ALGORITHM FOR CLOSEST-POINT QUERIES\*

KENNETH L. CLARKSON†

*For Quentin Deane Clarkson, in loving memory*

**Abstract.** An algorithm for closest-point queries is given. The problem is this: given a set  $S$  of  $n$  points in  $d$ -dimensional space, build a data structure so that given an arbitrary query point  $p$ , a closest point in  $S$  to  $p$  can be found quickly. The measure of distance is the Euclidean norm. This is sometimes called the *post-office problem*. The new data structure will be termed an *RPO tree*, from Randomized Post Office. The expected time required to build an RPO tree is  $O(n^{\lceil d/2 \rceil(1+\epsilon)})$ , for any fixed  $\epsilon > 0$ , and a query can be answered in  $O(\log n)$  worst-case time. An RPO tree requires  $O(n^{\lceil d/2 \rceil(1+\epsilon)})$  space in the worst case. The constant factors in these bounds depend on  $d$  and  $\epsilon$ . The bounds are average-case due to the randomization employed by the algorithm, and hold for any set of input points. This result approaches the  $\Omega(n^{\lceil d/2 \rceil})$  worst-case time required for any algorithm that constructs the Voronoi diagram of the input points, and is a considerable improvement over previous bounds for  $d > 3$ . The main step of the construction algorithm is the determination of the Voronoi diagram of a random sample of the sites, and the triangulation of that diagram.

**Key words.** closest points, Voronoi diagrams, computational geometry

**AMS(MOS) subject classifications.** 68U05, 68P10

**1. Introduction.** The post-office problem is a fundamental problem of computational geometry, having many applications in statistics, operations research, interactive graphics, coding theory, and other areas.

Several algorithms that are asymptotically fast in the worst-case sense are known for this problem in the planar ( $d = 2$ ) case. They involve the construction of the Voronoi diagram of the sites [12], [26], and the use of fast methods for searching planar subdivisions resulting from that diagram [19], [17], [11]. By these methods, a data structure requiring  $O(n)$  space can be constructed in  $O(n \log n)$  time, so that a query can be answered in  $O(\log n)$  time.

The higher-dimensional cases are much less examined and understood. Dobkin and Lipton have described a data structure requiring  $O(n^{2^{d+1}})$  time and space to construct, giving a query time of  $O(\log n)$  [10]. Chazelle has given an algorithm for the case  $d = 3$  that requires  $O(n^2)$  preprocessing for  $O(\log^2 n)$  query time [4].

Although the time and space bounds for RPO trees are rather large for large  $d$ , they are a considerable improvement over previous general bounds. The key step in the construction of the data structure is the determination of Voronoi diagrams of small subsets of the sites. (For convenience, points in  $S$  will be called *sites*.) The bounds depend on the complexity of these Voronoi diagrams. If the diagrams have  $O(n)$  vertices, the construction requires an expected time bounded by  $O(n^C)$ , where  $C$  is a constant independent of  $d$ . Indeed, when a set of sites is uniformly distributed in a hypercube [1], or spatially Poisson-distributed [13], their Voronoi diagram has linear complexity on the average. These facts suggest that RPO trees may do considerably better in practice than the worst-case bounds would show. On the other hand, in the worst case Voronoi diagrams may require  $\Omega(n^{\lceil d/2 \rceil})$  storage [18], [24], so in a sense any algorithm using Voronoi diagrams could not perform too much better than RPO trees.

\* Received by the editors June 26, 1986; accepted for publication (in revised form) July 27, 1987.

† AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

**1.1. Overview.** The initial observation for the data structure is just this: if we want to find a closest site in  $S$  to a point  $p$ , then knowing a closest site to  $p$  in some  $R \subset S$  can help restrict the search in  $S$ . The terms *candidate sets* and *candidate sites* help to formalize this notion.

**DEFINITION.** For a given subset  $R \subset S$ , and any point  $x$ , let  $r_x$  denote the distance of  $x$  to a closest site in  $R$ . Then the *candidate region*  $C(x)$  for  $x$ , relative to  $R$ , is the ball of points whose distance to  $x$  is less than  $r_x$ . The corresponding closed ball will be denoted  $\overline{C}(x)$ . The *candidate set* for  $x$  is  $S \cap C(x)$ . The candidate region for a set of points  $A$  is  $C(A) = \cup_{x \in A} C(x)$ , with the candidate set  $S \cap C(A)$ .

Thus, for a query point  $p$  and region  $A$  with  $p \in A$ , the set  $S \cap \overline{C}(A)$  contains all the closest sites to  $p$ . If  $q$  is a closest site in  $R$  to  $p$ , then the candidate set  $S \cap C(A)$  contains all sites closer to  $p$  than  $q$ . The key idea is to find some  $R \subset S$ , and some collection of regions, such that for every region  $A$  in the collection, the candidate set of  $A$  relative to  $R$  contains few sites.

Such a collection of regions can be found using random sampling, as follows: take a random sample  $R$  of the sites, determine the Voronoi diagram  $\mathcal{V}(R)$  of that sample, and then compute  $\Delta(\mathcal{V}(R))$ , a triangulation of the Voronoi diagram. (Voronoi diagrams are defined in §2; triangulations are discussed in §3.) The result is a collection of simple regions with the following properties:

- The union of the regions covers  $E^d$ , that is,  $E^d = \cup_{A \in \Delta(\mathcal{V}(R))} A$ ;
- The number of regions is  $O(r^{\lceil d/2 \rceil})$ , for  $r \rightarrow \infty$ , where  $r$  is the size of  $R$ ;
- With high probability, the candidate sets  $S \cap C(A)$  are “small” for all regions  $A \in \Delta(\mathcal{V}(R))$ , specifically,  $|S \cap C(A)| = nO(\log r/r)$  as  $r \rightarrow \infty$ ;
- The regions in  $\Delta(\mathcal{V}(R))$  are simple, so that for point  $p$  and  $A \in \Delta(\mathcal{V}(R))$ , we can tell in  $O(1)$  time if  $p \in A$ , for fixed dimension  $d$ ;
- For each  $A \in \Delta(\mathcal{V}(R))$ , there is a site  $q \in R$  such that all points in  $A$  are as close to  $q$  as to any other site in  $R$ .

These properties suggest a two-step process for answering closest-point queries: given query point  $p$ , determine a region  $A \in \Delta(\mathcal{V}(R))$  that contains it, then determine the closest site to  $p$  in  $R \cup (S \cap C(A))$  by linear search. For a suitable sample size, with high probability this procedure is faster than directly searching  $S$ . By repeatedly taking random samples until a sample is found for which the corresponding candidate sets are all small, a data structure with an improved worst-case query time can be constructed. Since a random sample will satisfy this condition with high probability, on average only  $O(1)$  sampling repetitions need be done.

Rather than search the candidate sets in linear time, this construction can be applied recursively, using a sample size  $r$  that is independent of the number of sites. The resulting search structure is an RPO tree, in which the number of children of a node is independent of the number of sites, as is the size of the set of sites associated with each leaf node.

Each node  $t$  of an RPO tree corresponds to a collection of sites  $S'$  that contains the closest site to a set of potential query points. If  $t$  is an internal node, a suitable sample  $R' \subset S'$  is found, and for each  $A \in \Delta(\mathcal{V}(R'))$ , there is a child  $t'$  of  $t$  for which a record  $t'.region$  is  $A$ . The children of  $t$  form a list  $t.children$ . A closest-point query can be answered by tracing down from the root, moving from a current node  $t$  to a child  $t' \in t.children$ , whose associated  $t'.region$  contains the query point. If  $t$  is a leaf node, the sites  $t.sites$  associated with  $t$  are given a linear search to answer the query.

The procedures *Make\_RPO\_Tree* and *Answer\_Query* are shown in Fig. 1. The procedure *New\_RPO\_Tree* returns a new RPO tree, whose regions and subtrees are



subsequently defined. From Theorem 4.5, the sample size  $r$  should be at least about  $(d+1)^3$ . The constant  $K$  should be no smaller than  $r$ . The constant  $\alpha_{r,d} = O(\log r/r)$  is defined in Theorem 4.5.

**function** *Make\_RPO\_Tree*( $S : \text{Set\_of\_Sites}$ ) **return**  $t : \text{RPO\_Tree}$ ;

$t \leftarrow \text{New\_RPO\_Tree}$ ;

**if**  $|S| < K$  **then**  $t.\text{leaf} \leftarrow \text{true}$ ;  $t.\text{sites} \leftarrow S$ ;

**else**

$t.\text{leaf} \leftarrow \text{false}$ ;

**repeat** choose random sample  $R \subset S$  **until**  $\forall A \in \Delta(\mathcal{V}(R)), |S \cap C(A)| \leq \alpha_{r,d}|S|$ ;

$t.\text{children} \leftarrow \emptyset$ ;

**for**  $A \in \Delta(\mathcal{V}(R))$  **do**

$t' \leftarrow \text{Make\_RPO\_Tree}(S \cap C(A))$ ;

$t'.\text{region} \leftarrow A$ ;  $t'.\text{site} \leftarrow$  site  $q$  such that  $A \subseteq V_q$ ;  $t.\text{children} \leftarrow t.\text{children} \cup \{t'\}$ ;

**od**;

**fi**;

**end function** *Make\_RPO\_Tree*;

**function** *Answer\_Query*( $t : \text{RPO\_Tree}$ ;  $p : \text{query\_point}$ ) **return**  $\text{closest} : \text{site}$ ;

$\text{current\_closest} \leftarrow$  any site in  $R$ ;

**while not**  $t.\text{leaf}$  **do**

choose any  $t' \in t.\text{children}$  with  $p \in t'.\text{region}$ ;

**if**  $t'.\text{site}$  closer to  $p$  than  $\text{current\_closest}$  **then**  $\text{current\_closest} \leftarrow t'.\text{site}$ ;

$t \leftarrow t'$ ;

**od**;

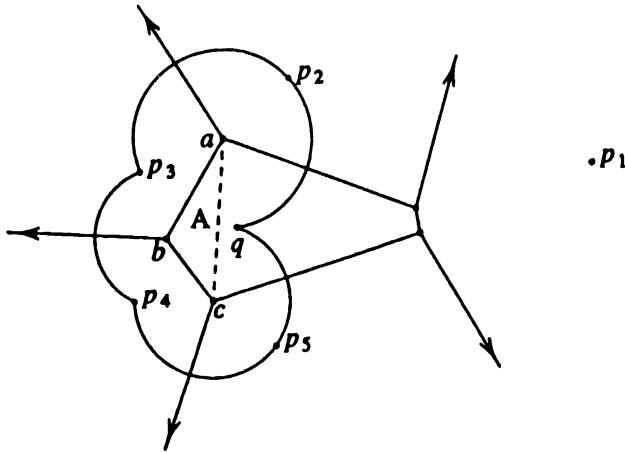
$\text{closest} \leftarrow$  site closest to  $p$  among those in  $t.\text{sites} \cup \{\text{current\_closest}\}$ ;

**end function** *Answer\_Query*;

FIG. 1. *Procedures Make\_RPO\_Tree and Answer\_Query.*

Before making a more detailed description of the algorithm, it may be helpful to consider informally the simplest interesting example of a set  $S \cap \overline{C}(A)$ , which occurs when  $A$  is a triangular region in the plane, a region in the triangulation of the Voronoi diagram of a sample  $R$ . In Fig. 2, the set  $R = \{q, p_1, \dots, p_5\}$ , and  $A$  has vertices  $a, b$ , and  $c$ , part of a triangulation of the Voronoi region  $\mathcal{V}_q$ . As will be shown in §4, the region  $\overline{C}(A)$  has a particularly simple description: it is simply  $\overline{C}(a) \cup \overline{C}(b) \cup \overline{C}(c)$ . Since  $a, b$ , and  $c$  are vertices of  $\mathcal{V}_q$ , the circles bounding  $\overline{C}(a)$ ,  $\overline{C}(b)$ , and  $\overline{C}(c)$  are *Delaunay circles* of the Voronoi diagram  $\mathcal{V}_R$ . To restate this fact, suppose  $p \in A$ . Then since  $q \in S$ , the closest site to  $p$  is contained in the disk defined by the circle centered at  $p$  that passes through  $q$ . This disk is contained in the union of the Delaunay disks at  $a, b$ , and  $c$ .

For the RPO construction to work, with high probability all these Delaunay disks should contain few sites. Why should this be? The reason is based on the fundamental fact that these Delaunay disks contain no sample sites. Intuitively, this provides some evidence that these disks contain few sites: if some arbitrary disk contains a large fraction of the sites, then with high probability, some sample site will be chosen from that disk, and it cannot be a Delaunay disk. This argument is made precise in §4.3.

FIG. 2. A triangle  $A \in \Delta(\mathcal{V}_q)$  and  $\bar{C}(A)$ .

**1.2. Outline of the paper.** To complete the description of the algorithms, it is necessary to specify the triangulation procedure  $\Delta$ , and to characterize formally the candidate regions  $C(A)$  for  $A \in \Delta(\mathcal{V}(R))$ . To analyze the algorithms, we must bound the number of children that a node can have, that is, the number of regions in  $\Delta(\mathcal{V}(R))$ , and also bound the size of the resulting subproblems, that is, the size of each  $C(A) \cap S$ . Before addressing these questions, some notation and basic lemmas will be given in §2. Many readers should be able to skim most of this section, or refer to it as needed. In §3, the triangulation procedure is given, and a bound on the size of its output is developed. In §4, it is shown that the candidate regions to be used have simple descriptions, generalizing the above example. It is also shown that with high probability, all the corresponding candidate sets have few sites. Also given in this section are the modifications to the algorithms for handling a variant of the post-office problem, in which all closest sites to a query point are desired. In §5, the complexity analysis of the algorithms is completed. Some concluding remarks are made in §6, with discussion of subsequent and related work.

The basic idea for the RPO data structure is simple, and the critical algorithmic step is the fundamental operation of computing the Voronoi diagram, followed by triangulation. Nonetheless, several factors complicate the discussion. The algorithms generalize for an arbitrary dimension, so that the descriptions and proofs of correctness are abstract. An operation of triangulation must be applied to the unbounded polyhedral sets of a Voronoi diagram, as must the determination of candidate regions. This is best done using the notion of “points at infinity,” considering an unbounded polyhedral set as the convex hull of a set of points, some of which are at infinity. This idea is made precise using *two-sided space*, described in the next section. The sample  $R$  may be degenerate, that is, not have full affine dimension. This possibility must be accounted for. These factors imply that the description must be more abstract and complicated than it otherwise would be.

**2. Notation, terminology, and background.** The notation in this paper will follow [14] in general, and use basic results from that text. The concepts of *oriented projective geometry* [27] will also play a large role. The following notation is gathered here for reference:

$E^d$  denotes  $d$ -dimensional Euclidean space;

$A + B$  is the pointwise sum  $\{x + y \mid x \in A, y \in B\}$ , for  $A, B \subset E^d$ ;

$x + A$  and  $A + x$  denote  $\{x\} + A$ , for  $x \in E^d$ ;

$\alpha A$  denotes the product  $\{\alpha x \mid x \in A\}$ , for a real number  $\alpha$  and  $A \subset E^d$ ;

$S_{x,y}$  denotes the sphere that has center  $x$  and that contains  $y$ .  $B_{x,y}$  and  $\overline{B}_{x,y}$  denote the corresponding open and closed balls;

A *flat*  $F \subset E^d$  is an *affinely closed* set, that is, if  $x, y \in F$ , then the straight line through  $x$  and  $y$  is contained in  $F$ .

$\text{aff } A$  denotes the affine closure of a point set  $A \subset E^d$ , that is, the intersection of all flats containing  $A$ ;

$\dim A$  denotes the affine dimension of  $A$ , that is, the dimension of the linear subspace  $(\text{aff } A) - p$ , for  $p \in A$ . A  $k$ -flat  $F$  has  $k = \dim F$ ;

$\text{conv } A$  denotes the convex closure of  $A$ , that is, the intersection of all convex sets containing  $A$ ;

$\text{relint } A$  is the interior of  $A$  relative to its affine closure;

$\text{relbd } A$  is the boundary of  $A$  relative to its affine closure.

**Rays and cones.** For  $x, y \in E^d$ , let  $\text{ray}_x y$  denote

$$\{x + \alpha(y - x) \mid \alpha \geq 0\}.$$

A *cone*  $C$  with apex  $a$  is a subset of  $E^d$  such that  $\text{ray}_a y \subset C$  if and only if  $y \in C$ , for  $y \in E^d$ .

**Polyhedral sets and polytopes.** A *polyhedral set* is the intersection of a finite number of closed halfspaces, and a *polytope* is a bounded polyhedral set. A polyhedral cone is a cone that is a polyhedral set. A  $d$ -polytope ( $d$ -polyhedral set)  $P$  satisfies  $d = \dim P$ .

A *supporting hyperplane*  $h$  of a polyhedral set  $P$  satisfies  $h \cap P \neq \emptyset$  and  $h^+ \cap P = \emptyset$ , where  $h^+$  is an open halfspace defined by  $h$ . A *face* of a polyhedral set  $P$  is the intersection of  $P$  with a supporting halfspace. Vertices, edges, and facets are faces of affine dimension 0, 1, and  $d - 1$ , respectively, for a  $d$ -polyhedral set. In general, a face of  $P$  with dimension  $k$  is a  $k$ -*face*, and the set of such faces is  $f_k(P)$ . The set  $f_0(P)$  of vertices (or *extreme points*) of a polyhedral set  $P$  will be denoted by  $\text{vert } P$ .

Two polyhedral sets  $A$  and  $B$  are said to be *combinatorially equivalent* if there is a bijective mapping  $\Lambda$  from the faces of  $A$  to those of  $B$  such that  $F \subset G$  if and only if  $\Lambda(F) \subset \Lambda(G)$ , for all  $F$  and  $G$  faces of  $A$ .

The set of *extreme rays* of a polyhedral set  $P$  is the set of rays  $e$  emanating from the origin such that there is some point  $q$  for which  $q + e$  is an edge (1-face) of  $P$ . The convex hull of the extreme rays of  $P$  is the *characteristic cone*  $\text{cc } P$ . From [14, 2.5.2], if  $x, y \in P$  and  $e$  is a ray from the origin, then  $x + e \subseteq P$  if and only if  $y + e \subseteq P$ . (Looking ahead, this is equivalent to the condition that  $e$  corresponds to a "point at infinity" in  $P$ , and that  $P$  is convex even when such points are included.) This fact and the convexity of  $P$  imply that  $x + \text{cc } P \subset P$  for any  $x \in P$ . It can be shown that  $\text{cc } P$  is the maximal such cone.

A set is said to be *line-free* if it contains no straight lines (no 1-flats). A line-free cone is pointed [14, p. 24], that is, it has only one apex, which is a vertex. Many basic

facts about polytopes generalize nicely to line-free polyhedral sets, using the notion of “ideal points” defined below.

**Complexes.** A *complex* is a collection of polyhedral sets such that every face of a polyhedral set in the complex is also in the complex, and the intersection of two polyhedral sets in the complex is a face of each of them. (In the complexes considered here the empty set  $\emptyset$  is a face.) A polyhedral set of dimension  $k$  in a complex is a  $k$ -*face* of that complex, and the terminology of vertices, edges, and facets carries over for complexes. The *facial lattice* of a complex is the set of faces of the complex, together with the inclusion relations between those faces.

One example of a complex is the *boundary complex*  $\mathcal{B}(P)$  of a polyhedral set  $P$ , the set of facets of  $P$  and their faces. Another example of a complex is the Voronoi diagram of a set of sites, described below.

**Two-sided space, homogeneous coordinates, and ideal points.** It will be helpful conceptually and computationally to use the notion of “points at infinity,” also known as *ideal* points, as opposed to the usual *real* points in  $E^d$ . These classes of points together make up what will be denoted  $T^d$ , or *two-sided space*. (The name will be explained below.)

To represent points in  $T^d$ , *homogeneous* coordinates will be used: a real point  $x \in E^d$  is represented by  $x_h = [x_r; x_s]$  if  $x = x_r/x_s$ , where  $x_r \in E^d$  and  $x_s \in \mathcal{R}$ ,  $x_s > 0$ . (The terminology is borrowed from projective geometry [23], although in this case, the coordinates cannot really be said to be homogeneous.) An ideal point  $x \in T^d$  is represented by the homogeneous coordinates  $[x_r; 0]$ , where  $x_r \in E^d$  and  $x_r \neq 0$ . The point  $x$  can be considered the “endpoint” of ray<sub>0</sub>  $x_r$ . If  $x$  is an ideal point and  $y$  is real, we will say that  $\text{conv}\{x, y\}$  is ray<sub>y</sub>  $(x_r + y)$ . Indeed, if  $z \in \text{conv}\{x, y\}$ , for any points  $x$  and  $y$ , then for any representations  $z_h, x_h, y_h$ , we have  $z_h = \alpha_x x_h + \alpha_y y_h$ , for some  $\alpha_x, \alpha_y \geq 0$ , and conversely. This provides a general definition of convex combination for points in  $T^d$ .

Note that homogeneous coordinate representations are not unique: if  $x_h$  is a homogeneous representation of  $x$ , then so is  $\beta x_h$ , for any  $\beta > 0$ . (This convention is different from that of projective geometry, where  $\beta$  need only be nonzero. This follows [21] and [15], and is needed to distinguish ideal points in “opposite directions.”) The two-sided nature of  $T^d$  derives from its containment of two copies of  $E^d$ , since  $[x_r; 1]$  and  $[x_r; -1]$  represent distinct points in  $T^d$  for every  $x_r \in E^d$ . There is a correspondence between points in  $T^d$  and the  $d$ -sphere

$$S^d = \{x \in E^{d+1} \mid \|x\| = 1\}.$$

A point with homogeneous coordinates  $x_h$  corresponds to  $x_h/\|x_h\|$ , where  $x_h$  is interpreted as a point in  $E^{d+1}$ . The ideal points of  $T^d$  correspond to those points of  $S^d$  on the hyperplane  $x_s = x_{d+1} = 0$ . The two halves of  $S^d$  separated by the set of ideal points correspond to the two sides of  $T^d$ .

In general, the only points in  $T^d$  considered will be those satisfying  $x_s \geq 0$ . This set of points is termed the “front range” of  $T^d$ , and will be denoted by  $F^d$ . A closed convex set  $P \subset E^d$  will be extended by including  $[b - a; 0]$  in  $P$  whenever ray<sub>a</sub>  $b \subset P$ . This implies that the set of points on  $S^d$  corresponding to  $P$  is also closed. A straight line will thus have two ideal “endpoints,” and so on for all flats. (Note that this gives a meaning to “flat” that is different from Stolfi’s [27].) This convention will extend the definition of the sum  $A + B$  for unbounded  $A$  and  $B$ . The notation for a sphere  $S_{x,y}$  can be extended to allow the center  $x$  to be an ideal point. In this case,

$S_{x,y}$  is a hyperplane normal to  $x_r$  and passing through  $y$ . The closed ball  $\overline{B}_{x,y}$  is the corresponding closed halfspace. An analytic relation describing such spheres is given below in the discussion of Voronoi diagrams.

The set  $\text{re } A$  will denote the real points of  $A$ , and  $\text{id } A$  will denote the ideal points of  $A$ . Note that  $\text{re } A = \text{re } B$  implies that  $\text{id } A = \text{id } B$ , for  $A$  and  $B$  closed and convex.

With this extension to the front range  $F^d$ , a line-free unbounded polyhedral set  $P \subset E^d$  has an implicit additional defining halfspace. That is,  $\text{id } P$  is an additional facet of  $P$ , the *ideal facet*. This facet corresponds to  $\text{cc } P$  in a natural way, and the correspondence extends to the faces of  $\text{id } P$ , so that the vertices of  $\text{id } P$  correspond to the extreme rays of  $\text{cc } P$ . Thus  $\text{vert } P$  is extended to include ideal points. The following simple lemma helps in generalizing facts about polytopes to facts about polyhedral sets.

**LEMMA 2.1.** *Let  $C$  be a polyhedral pointed cone with apex  $a$ . Let  $h$  be a supporting hyperplane of  $C$  with  $h \cap C = \{a\}$ . Let  $\nu$  be a normal vector to  $h$  contained in the same halfspace containing  $C$ . Then  $P = C \cap (h + \nu)$  is a polytope.*

*Proof.* See Appendix A.  $\square$

It is easy to show that  $C = \cup_{y \in A} \text{ray}_x y$ , and that  $x \in P$  if and only if  $x' = [x - a; 0] \in \text{id } C$ . This bijective map satisfies  $(\alpha x + \beta y)' = \alpha x' + \beta y'$ , for  $\alpha + \beta = 1$ . This implies that  $\text{id } C$  and  $P$  are combinatorially equivalent. Thus Lemma 2.1 brings  $\text{id } C$  into the “real” world of  $E^d$ .

The following lemma is a generalization of [14, 2.4.5] from polytopes to line-free polyhedral sets.

**LEMMA 2.2.** *If  $P \subset E^d$  is a line-free  $d$ -polyhedral set then  $P = \text{conv vert } P$ .*

*Proof.* See Appendix A.  $\square$

**Simplices and triangulations.** A *simplex* is a simple kind of polyhedral set: a  $d$ -simplex is a polyhedral set with  $d + 1$  vertices and affine dimension  $d$ . Note that vertices will be allowed to be ideal. For example, a triangle with two ideal vertices is a cone, and a triangle with one ideal vertex is bounded by a line segment and by two parallel rays from the endpoints of that line segment.

A *simplicial complex* is a complex composed of simplices. A *triangulation*  $\mathcal{T}$  of a complex  $\mathcal{C}$  is a simplicial complex that is a subdivision of  $\mathcal{C}$ . Every vertex of  $\mathcal{T}$  is a vertex of  $\mathcal{C}$ , every facet of  $\mathcal{T}$  is a simplex, and the union of the facets of  $\mathcal{T}$  is the union of the facets of  $\mathcal{C}$ .

In §3, a particular kind of triangulation of  $\mathcal{C}$ , denoted  $\Delta(\mathcal{C})$ , will be described. Construction of this complex is an essential procedure in the algorithm given in this paper. The complexity of  $\Delta(\mathcal{C})$  and of its construction procedure are also considered in §3.

The triangulation  $\Delta$  may involve “simplices” of even greater generality than those shown above. For example, suppose  $\dim S = 2$  but  $\dim R = 1$ , specifically,  $R$  is a set of sites on the  $x$  axis. Then each Voronoi region  $\mathcal{V}_q$  of  $R$  will be a strip bounded by two parallel lines. This can be viewed as an interval  $A$ , that is, a 1-simplex, added to the  $y$ -axis. That is,  $\mathcal{V}_q = A + l$ , where  $l$  is the  $y$ -axis line. In general, when  $d > \dim R$ , the regions of  $\Delta(\mathcal{V}(R))$  will have the form  $A + f$ , where  $A$  is a simplex and  $f$  is a flat orthogonal to  $\text{aff } A$ . This generalization is formalized by Lemma 3.4.

**Duality.** For a complex  $\mathcal{C}$ , a *dual*  $\mathcal{D}$  of  $\mathcal{C}$  is another complex for which there is an inclusion-reversing correspondence between faces of  $\mathcal{C}$  and those of  $\mathcal{D}$ . That is, there is a bijective mapping  $\Psi$  from the set of faces of  $\mathcal{C}$  to those of  $\mathcal{D}$  such that for faces  $F$  and  $G$  of  $\mathcal{C}$ , the inclusion  $F \subset G$  holds if and only if  $\Psi(F) \supset \Psi(G)$ . The facial lattice

of  $\mathcal{C}$  can be determined from the facial lattice of  $\mathcal{D}$ , and vice versa.

One particular dual relation between polyhedral sets is quite useful. For a set  $A \subset E^d$ , the polar set  $A^*$  is

$$\{y \mid y_h \cdot x_h \geq 0 \text{ for all } x \in A\}.$$

When  $P$  is a line-free  $d$ -polyhedral set in  $E^d$  with the origin in its interior, the polar set  $P^*$  of  $P$  is a polytope, even when  $P$  is unbounded. Moreover,  $P^*$  is dual to  $P$ . The ideal facet of  $P$ , if any, corresponds to the origin, which will be a vertex of  $P^*$ , for unbounded  $P$ .

**Voronoi diagrams.** The *Voronoi diagram*  $\mathcal{V}(R)$  of a set of sites (points)  $R$  in  $E^d$  is the partition of  $E^d$  into blocks, such that all points in a block have exactly the same closest sites. The Voronoi region  $\mathcal{V}_q$  associated with a site  $q \in R$  is the polyhedral set containing all points at least as close to  $q$  as they are to any other site. If  $v \in \text{revert } \mathcal{V}_q$  for some site  $q$ , then the sphere (ball) centered at  $v$  with radius  $\|v - q\|$  is termed a *Delaunay sphere* (ball) of the sites  $R$ . (A ball in the plane is also called a disk.) At least  $d + 1$  sites are on a Delaunay sphere, and none are inside it.

It is well known that  $\mathcal{V}_q$  is unbounded if and only if  $q$  is on the convex hull of  $R$ . Furthermore, each unbounded edge of  $\mathcal{V}_q$  is normal to a facet of  $\text{conv } R$  that contains  $q$ .

Brown [3] has shown that the computation of a Voronoi diagram in  $E^d$  can be reduced to the problem of computing a convex hull in  $E^{d+1}$ . The reduction is done by means of a mapping from  $E^d$  to  $E^{d+1}$ . One mapping that achieves this reduction is the function  $\Upsilon : E^d \mapsto E^{d+1}$ , which sends  $y \in E^d$  to the point  $(y_1, \dots, y_d, -y \cdot y/2) \in E^{d+1}$ . For  $x, q \in E^d$ , we have  $y \in S_{x,q}$  if and only if

$$(x_1, \dots, x_d, 1) \cdot \Upsilon(y) = (x_1, \dots, x_d, 1) \cdot \Upsilon(q).$$

This implies  $\text{aff } \Upsilon(S_{x,q})$  is a hyperplane. Furthermore, if  $\overline{B}_{x,q}$  is a Delaunay ball, then  $\Upsilon(R) \cap \Upsilon(\overline{B}_{x,q})$  is empty, so that  $\text{aff } \Upsilon(S_{x,q})$  contains a facet of  $\text{conv } \Upsilon(R)$ . It follows that  $\text{conv } \Upsilon(R)$  gives a dual complex to  $\mathcal{V}(R)$ .

Note also that the analytic condition for  $y \in S_{x,q}$  can be extended coherently to ideal  $x$  by  $x_h \cdot (\Upsilon(y) - \Upsilon(q)) = 0$ , where  $x_h$  is interpreted as a point in  $E^{d+1}$ . For ideal  $y$  and  $x$ , the appropriate condition is  $y_r \cdot x_r = 0$ .

### 3. Triangulating polytopes and Voronoi regions.

**3.1. A triangulation procedure and its correctness.** The procedure  $\Delta$  to be used for triangulating a complex  $\mathcal{C}$  is straightforward: the procedure produces a set of simplices triangulating each face of  $\mathcal{C}$ , considering these faces in increasing order of their affine dimension. Note that if  $1 \geq \dim P$ , then the face  $P$  is a simplex. If  $1 < \dim P$ , then arbitrarily pick  $v \in \text{vert } P$ , and let  $\Delta(P)$  be the collection

$$\{\text{conv}(\{v\} \cup S) \mid S \in \Delta(F), F \text{ a facet of } P, v \notin F\}.$$

For example, if  $P$  is a polygon, then for every edge  $e$  of  $P$  not containing vertex  $v$ , the triangle defined by  $e$  and  $v$  is in  $\Delta(P)$ . Note that once  $\Delta(P)$  is computed, it is “fixed,” so that the same triangulation of  $P$  is used whenever a face is triangulated for which  $P$  is a facet.

To apply  $\Delta$  to a polyhedral set, that set must have a vertex. However, not all polyhedral sets have vertices. (An example of such a polyhedral set is given in the discussion of triangulations in §2.) We will first show that  $\Delta$  can be applied to line-free polyhedral sets, and then discuss the extension of  $\Delta$  to arbitrary polyhedral sets.

LEMMA 3.1. *A line-free polyhedral set  $P$  has a vertex.*

*Proof.* If  $P$  is a real polyhedral set, the result is a special case of [14, 2.4.6]. If  $P$  is an ideal polyhedral set (say some ideal facet of an unbounded polyhedral set), then the result follows from Lemma 2.1 and [14, 2.4.6].  $\square$

This lemma implies that some choice  $v \in \text{vert } P$  can be made in  $\Delta$ . To show that  $P$  is the union of the simplices in  $\Delta(P)$ , the following lemma is useful.

LEMMA 3.2. *For a line-free polyhedral region  $P$ , if  $v \in \text{vert } P$  and  $a \in P$ , then the point  $b = \text{ray}_v a \cap \text{relbd } P$  is on a facet of  $P$  that does not contain  $v$ .*

*Proof.* See Appendix A.  $\square$

THEOREM 3.3. *Given a complex  $\mathcal{C}$  containing only line-free polyhedral sets,  $\Delta(\mathcal{C})$  is a triangulation of that complex. (Note that the simplices in  $\Delta(\mathcal{C})$  may have ideal vertices.)*

*Proof.* Induction on dimension will be applied to each polyhedral set  $P$  in  $\mathcal{C}$  to show that  $\Delta(P)$  returns a set of simplices covering  $P$ . Let  $a \in P$ . Then  $\text{ray}_v a$  intersects  $\text{relbd } P$  at  $v$  and at some point  $b$ . (If  $P$  is ideal, map the points involved to the polytope of Lemma 2.1.) By the previous lemma,  $b$  is on some facet  $F$  not containing  $v$ . Since by inductive assumption,  $b$  is in a simplex  $A$  of a triangulation of  $F$ , it follows that  $a \in \text{conv}(\{v\} \cup A)$ , and the set of regions returned by  $\Delta$  covers  $P$ .

The other properties of a triangulation follow by similar straightforward induction.  $\square$

To extend the triangulation procedure  $\Delta$  to polyhedral sets that do not have vertices, the following lemma is useful.

LEMMA 3.4. *Let  $P \subset T^d$  be a closed polyhedral set. Then there is a subspace  $L$  of maximum affine dimension for which  $x + L \subset P$  for any  $x \in P$ . Furthermore, if  $L^*$  is any flat that is orthogonally complementary to  $L$ , then  $P = (P \cap L^*) + L$ , where  $P \cap L^*$  is a line-free polyhedral set.*

*Proof.* (Recall that  $L^*$  and  $L$  orthogonal means that  $L$  and the subspace  $L^* - x$  are orthogonal, where  $x \in L^*$ . That is, every vector in  $L^* - x$  is perpendicular to every vector in  $L$ . Since  $L^*$  and  $L$  are extended to include ideal points, the sum  $L + L^*$  is the front range of  $T^d$ .)

For  $\text{re } P$ , the lemma is a restatement of [14, 2.5.4]. The extension to  $\text{id } P$  follows from this, since  $\text{re } P = \text{re}(P \cap L^*) + \text{re } L$  implies equality for the ideal parts as well.  $\square$

In the particular case where  $P$  is a Voronoi region, the flat  $L^*$  can be taken to be  $\text{aff } R$ , so that  $L$  is  $(\text{aff } R)^\perp$ , the subspace orthogonal to  $\text{aff } R$ . Observe that if  $l$  is a straight line contained in a Voronoi region  $V_q$ , then  $l = (\text{conv}\{a, c\}) \cup (\text{conv}\{a, -c\})$ , where  $a \in \text{re } l$  and  $c \in \text{id } l$ . For any other site  $q' \in R$ ,  $c \in V_q$  implies that  $q' \notin B_{c,q}$ , so that  $c_h \cdot (\Upsilon(q') - \Upsilon(q)) \geq 0$ . But  $-c \in V_q$  as well, so  $c_h \cdot (\Upsilon(q') - \Upsilon(q)) \leq 0$ . Therefore for any  $q' \in R$ , we have  $q' \in S_{c,q}$ , and so  $\text{aff } R \subseteq S_{c,q}$ . That is,  $\text{aff } R$  is perpendicular to  $c$ , and  $l \subseteq (\text{aff } R)^\perp$ .

To use this lemma to extend  $\Delta$  for Voronoi regions containing lines, simply define  $\Delta(V_q)$  to be the set of regions  $\{A + (\text{aff } R)^\perp \mid A \in \Delta(V_q \cap \text{aff } R)\}$ .

**3.2. Complexity of the complex  $\Delta(\mathcal{V}(R))$ .** In the worst case, the Voronoi diagram  $\mathcal{V}(R)$  has  $\Theta(r^{\lceil d/2 \rceil})$  faces. This follows from the correspondence discussed in §2 between  $\mathcal{V}(R)$  and  $\text{conv } \Upsilon(R) \subset E^{d+1}$ , and from the Upper Bound Theorem [20] applied to  $(d+1)$ -polytopes. As is shown below, the number of simplices in  $\Delta(\mathcal{V}(R))$  has the same  $\Theta(r^{\lceil d/2 \rceil})$  bound. First a bound will be proven for  $\Delta(P)$ , in the case where  $P$  is a *simple* polytope, defined below. Next it will be shown that for every polytope  $P$ , there is a simple polytope  $\hat{P}$  with the same number of facets, such that  $\Delta(\hat{P})$  has at least as many simplices as  $\Delta(P)$ .

LEMMA 3.5. *For a simple  $d$ -polytope  $P$  with  $n$  facets, the triangulation  $\Delta(P)$  has  $O(n^{\lfloor d/2 \rfloor})$  simplices, as  $n \rightarrow \infty$ .*

*Proof.* A simple polytope satisfies the condition that every vertex of the polytope is contained in exactly  $d$  facets of that polytope. Suppose  $P$  is a simple polytope. Then the dual  $P^*$  has the property that every facet of  $P^*$  contains exactly  $d$  vertices. (The dual and some of its properties are described in §2.) That is, the facets of  $P^*$  are all simplices, which is the definition of a *simplicial* polytope. The faces of a simplex are all simplices [14, §4.1]. That is, each  $k$ -face of  $P^*$  is a  $k$ -simplex and has  $k + 1$  facets. This means that dually, every  $(d - 1 - k)$ -face of  $P$  is a facet of  $k + 1$  faces of  $P$ . Put another way, every  $k$ -face of  $P$  is a facet of  $d - k$  faces of  $P$ .

What does this fact imply for  $\Delta(P)$ ? For a polytope  $F$ , let  $|\Delta(F)|$  denote the number of simplices in  $\Delta(F)$ . Then the definition of  $\Delta(F)$  and the above fact imply that

$$\sum_{F \in f_k(P)} |\Delta(F)| = \sum_{\substack{F \in f_k(P) \\ F' \in f_{k-1}(F)}} |\Delta(F')| \leq (d - k + 1) \sum_{F' \in f_{k-1}(P)} |\Delta(F')|.$$

The second relation holds because  $|\Delta(F')|$  appears  $d - k + 1$  times in the second sum. Putting these relations together,

$$|\Delta(P)| = \sum_{F \in f_d(P)} |\Delta(F)| \leq d! \sum_{F \in f_0(P)} |\Delta(F)| = d! |\text{vert } P|.$$

By the results of [20],  $|\text{vert } P| = O(n^{\lfloor d/2 \rfloor})$  when  $P$  is a  $d$ -polytope with  $n$  facets, so the lemma follows.  $\square$

Before proving the corresponding lemma for nonsimple polytopes, a definition is needed.

DEFINITION. Let  $P \subset E^d$  be a  $d$ -polytope and let  $h$  be a hyperplane with  $F = h \cap P$  a facet of  $P$ . Let  $\hat{h}$  be a hyperplane with nonempty intersection with  $P$ , and with all vertices of  $\text{vert } P \setminus \text{vert } F$  in the open halfspace  $\hat{h}^+$ . If  $F$  is in the open halfspace  $\hat{h}^-$ , then the polytope  $\hat{P} = P \cap \hat{h}^+$  will be said to be obtained from  $P$  by *pushing* the facet  $F$ .

This operation of pushing a facet of a polytope is the dual of the operation of *pulling* a vertex [14, §5.2]. The (polar) dual polytope of a pushed polytope  $\hat{P}$  can be obtained by pulling a vertex of  $P^*$ . In [14, §5.2] it is shown that the operation of pulling vertices, when applied successively to every vertex of a polytope, results in a simplicial polytope. Dually, the operation of pushing facets yields a simple polytope.

The following lemma is a restatement of [14, 5.2.2], together with some relevant discussion in that section.

LEMMA 3.6. *Suppose  $\hat{P} \subset E^d$  is a  $d$ -polytope obtained from the  $d$ -polytope  $P$  by pulling  $v \in \text{vert } P$ . Then the faces of  $\hat{P}$  are exactly the following:*

- *faces of  $P$  that do not contain  $v$ ;*
- *faces of the form  $\text{conv}\{v, G\}$ , where*
  - *$G$  is a face of  $P$  not containing  $v$ , and*
  - *$G$  is contained in a facet of  $P$  that contains  $v$ ;*

*Furthermore, for every face  $F$  of  $P$  containing  $v$ , there is a facet  $F'$  of  $F$  that yields a face  $\text{conv}\{v, F'\}$  of the second type.*

*Proof.* All claims except the last statement are from [14, 5.2.2]. The last statement follows by considering  $\text{conv}(\text{vert } F \setminus \{v\})$ , which does not contain  $v$ , and is either a facet of  $F$  or contains a facet of  $F$ .  $\square$



Using the inclusion-reversing correspondence between faces of a polytope and faces of its dual, the following lemma shows what pushing a facet will do:

LEMMA 3.7. *Suppose  $\hat{P} \subset E^d$  is a  $d$ -polytope obtained from the  $d$ -polytope  $P$  by pushing a facet  $F = h \cap P$  of  $P$ . Then the faces of  $\hat{P}$  correspond to the following:*

- faces of  $P$  that do not meet  $F$ ;
- faces of the form  $\hat{h} \cap G$ , where
  - $G$  is a face not contained in  $F$ , and
  - $G$  contains a vertex of  $F$ .

Furthermore, for every face  $H$  of  $F$ , there is a face  $G$  of  $P$  such that  $H$  is a facet of  $G$ , and  $G$  yields a face  $\hat{h} \cap G$  of the second type.

*Proof.* The lemma follows directly from the previous one, and duality.  $\square$

LEMMA 3.8. *For a line-free polyhedral set  $P$  of dimension  $d$  and with  $n$  facets, the triangulation  $\Delta(P)$  has  $O(n^{\lfloor d/2 \rfloor})$  simplices, as  $n \rightarrow \infty$ .*

*Proof.* If  $P$  is a line-free polyhedral set with the origin in its interior, the dual  $P^*$  is a polytope, and the operation of pushing the ideal facet of  $P$  can be defined using the dual operation of pulling the origin, which will be the vertex of  $P^*$  corresponding to  $\text{id } P$ . The analogue of Lemma 3.7 holds for the case of pushing  $\text{id } P$ , using duality.

From Lemma 3.5 and the above discussion, it suffices to show that if  $\hat{P}$  is the result of pushing facet  $F = h \cap P$  of  $P$ , then  $|\Delta(\hat{P})| \geq |\Delta(P)|$ . Lemma 3.7 implies that for every face of  $P$ , there is a face of  $\hat{P}$ ; that is, there is an injective mapping  $m : \mathcal{B}(P) \mapsto \mathcal{B}(\hat{P})$ . The lemma follows using induction on dimension.  $\square$

LEMMA 3.9. *The complex  $\Delta(\mathcal{V}(R))$  has  $O(r^{\lfloor d/2 \rfloor})$  regions, and  $O(r^{\lfloor d/2 \rfloor} \log r)$  time suffices for its construction. The constant factors are  $e^{O(d \log d)}$ .*

*Proof.* Without loss of generality, we need consider only a bound on the size of  $\Delta(\mathcal{V}(R))$  when  $d = \dim R$ .

Let  $Z_R$  denote the polytope  $\text{conv } \Upsilon(R)$ , dual to  $\mathcal{V}(R)$  from the discussion of §2. It is easy to see that  $\hat{Z}_R$  has the same facial lattice structure as  $\mathcal{V}(R)$  (is combinatorially equivalent). It follows that  $\Delta(\hat{Z}_R)$  has the same number of simplices as  $\Delta(\mathcal{V}(R))$ . Lemma 3.8 then gives the desired bound.

The time required to construct  $\Delta(\mathcal{V}(R))$  is dominated by the time necessary for determining  $\mathcal{V}(R)$ : the projection  $P_R$  can be computed in  $O(r)$  time, and  $\mathcal{V}(R)$  can be triangulated in time linear in the number of its faces.

Several algorithms are known for computing  $\mathcal{V}(R)$  in  $O(r \log r)$  time when  $2 = \dim R$  [12], [22]. As noted in §2, the computation of  $\mathcal{V}(R)$  can be reduced to the computation of the convex hull of  $\Upsilon(R)$ . This can be done in  $O(r^2 + r^{\lfloor d/2 \rfloor} \log r)$  time [25]. For  $d > 2$ , this is  $O(r^{\lfloor d/2 \rfloor} \log r)$ .  $\square$

#### 4. Candidate regions and sets for Voronoi diagram triangulations.

**4.1. Candidate regions have a simple description.** The following theorem characterizes the candidate regions of line-free simplices in  $\Delta(\mathcal{V}(R))$ . The general case is considered in Theorem 4.2 below.

THEOREM 4.1. *Let line-free  $A \in \Delta(\mathcal{V}(R))$ , with  $A \subset \mathcal{V}_q$  for a site  $q$ . Then the candidate region  $C(A)$  is*

$$C(A) = \bigcup_{a \in \text{vert } A} B_{a,q}.$$

(As noted above, the points in  $\text{vert } A$  are vertices of Voronoi regions, and the balls are either Delaunay balls or halfspaces corresponding to convex hull facets. The ideal vertices of  $A$  are ideal vertices of  $\mathcal{V}_q$ , which correspond to unbounded edges of  $\mathcal{V}_q$  that are normal to a convex hull facet containing  $q$ .)

*Proof.* It suffices to show that  $C(A) \subseteq \cup_{a \in \text{vert } A} B_{a,q}$ , as the reverse inclusion follows by definition. That is, we must show that if  $x \in A$ , then

$$C(x) = B_{x,q} \subseteq \bigcup_{a \in \text{vert } A} B_{a,q}.$$

Suppose  $y \notin B_{a,q}$  for all  $a \in \text{vert } A$ . The theorem follows if this condition implies that  $y \notin B_{x,q}$ .

From §2,  $y \notin B_{a,q}$  if and only if  $a_h \cdot (\Upsilon(y) - \Upsilon(q)) \geq 0$ . By Lemma 2.2,  $A = \text{conv vert } A$ , and so

$$x_h = \sum_{a \in \text{vert } A} \alpha_a a_h$$

for some  $\alpha_a \geq 0$ , not all zero. Therefore,

$$x_h \cdot (\Upsilon(y) - \Upsilon(q)) = \left( \sum_{a \in \text{vert } A} \alpha_a a_h \right) \cdot (\Upsilon(y) - \Upsilon(q)) = \sum_{a \in \text{vert } A} \alpha_a a_h \cdot (\Upsilon(y) - \Upsilon(q)) \geq 0.$$

This implies  $y \notin B_{x,q}$ , and the lemma follows.  $\square$

Note that the only property of the line-free polyhedral set  $A$  on which the proof depends is that  $A \subset \mathcal{V}_q$ , so that an analogous result holds for any such polyhedral set.

This characterization of  $C(A)$  must be extended to the case where  $A$  is not necessarily line-free.

**THEOREM 4.2.** *Let  $A \in \Delta(\mathcal{V}(R))$ , with  $A \subset \mathcal{V}_q$  for a site  $q$ . Then the candidate region  $C(A)$  is*

$$C(A) = C(A \cap \text{aff } R) \cup (F^d \setminus \text{aff } R).$$

*Proof.* From the discussion following Lemma 3.4,  $A$  has the form  $(A \cap \text{aff } R) + (\text{aff } R)^\perp$ . Also from that discussion,  $\text{aff } R \subseteq S_{c,q}$  for ideal  $c$  if and only if  $c \in (\text{aff } R)^\perp$ . Since  $C(A) \supseteq B_{c,q}$  for all  $c \in \text{id}(\text{aff } R)^\perp$ , we have

$$C(A) \supseteq F^d \setminus \bigcap_{c \in \text{id}(\text{aff } R)^\perp} S_{c,q},$$

that is,  $C(A) \supseteq F^d \setminus \text{aff } R$ .

The theorem now follows by showing that  $C(A) \cap \text{aff } R \subseteq C(A \cap \text{aff } R) \cap \text{aff } R$ . Let  $z \in A$ . First, suppose  $z$  is a real point. Then by the discussion following Lemma 3.4,  $z = x + y$ , for  $x \in \text{re } A \cap \text{aff } R$  and  $y \in \text{re}(\text{aff } R)^\perp$ . Since  $[x_r/x_s + y_r/y_s; 1]$  is a homogeneous representation for  $z$ , we have, for  $w \in \text{aff } R$ ,

$$z_h \cdot (\Upsilon(w) - \Upsilon(q)) = [x_r/x_s; 1] \cdot (\Upsilon(w) - \Upsilon(q)) + [y_r/y_s; 0] \cdot (\Upsilon(w) - \Upsilon(q)),$$

but  $[y_r/y_s; 0] \in \text{id}(\text{aff } R)^\perp$ , and so

$$z_h \cdot (\Upsilon(w) - \Upsilon(q)) < 0 \text{ if and only if } x_h \cdot (\Upsilon(w) - \Upsilon(q)) < 0.$$

That is,  $C(z) \cap \text{aff } R = C(x) \cap \text{aff } R$ . If  $z$  is an ideal point not in  $(\text{aff } R)^\perp$ , then  $z_h = \alpha x_h + \beta y_h$ , for some  $x \in \text{id}(A \cap \text{aff } R)$ ,  $y \in \text{id}(\text{aff } R)^\perp$ ,  $\alpha > 0$ , and  $\beta \geq 0$ . In this case, similarly,  $C(z) \cap \text{aff } R = C(x) \cap \text{aff } R$ . Finally, if  $z \in \text{id}(\text{aff } R)^\perp$ , then  $C(z) = B_{z,q}$ , and  $S_{z,q} = \text{aff } R$ , so that  $C(z) \cap \text{aff } R = \emptyset$ . Thus  $C(A)$  and  $C(A \cap \text{aff } R)$  agree on  $\text{aff } R$ , and the theorem follows.  $\square$

**4.2. Reporting all closest sites.** While an RPO tree allows a closest site to a query point to be found, sometimes it is of interest to find all the sites closest to a given point. In this case, the distinction between  $C(A)$  and  $\overline{C}(A)$  becomes important. For example, suppose the sites are all on the surface of a sphere  $S_{c,q}$ , and the query point is the center of the sphere  $c$ . Here the set  $C(c) \cap S$  will be empty, but the set  $\overline{C}(c) \cap S = S$ . To handle such situations, it will be shown below that, roughly speaking, for most points  $x \in A \in \Delta(\mathcal{V}(R))$ ,  $\overline{C}(x)$  is contained in  $C(A)$ .

**THEOREM 4.3.** *Under the conditions of Theorem 4.1, let  $x \in A$ , and  $F$  be the face of  $A$  with  $x \in \text{relint } F$ . Let  $F' = F \cap \text{aff } R$ . Then  $\overline{C}(x) \subset C(A) \cup C_{\cap}(F)$ , where*

$$C_{\cap}(F) = \bigcap_{a \in \text{vert } F'} S_{a,q} \cap \text{aff } R.$$

(The existence and uniqueness of such a face  $F$  is readily established using elementary properties of polytopes, as given in [14, §2.6]. Note that when  $x$  is in the interior of some  $\mathcal{V}_q$ , the associated region  $C_{\cap}(\mathcal{V}_q)$  is trivial: it is easy to show that  $C_{\cap}(A) = \{q\}$ . Note also that in the two-dimensional case, the region  $C_{\cap}(e)$  for some Voronoi edge  $e$  is simply the intersection of the two Delaunay circles of the endpoints of  $e$ . This intersection contains only the two sites of  $R$  that define  $e$ .)

*Proof.* It is easy to show that  $x \in \text{relint } F$  implies that  $x = z + \sum_{a \in \text{vert } F'} \alpha_a a$ , for some  $z \in (\text{aff } R)^\perp$  and some  $\alpha_a$  all strictly greater than zero. (This holds necessarily only if  $F \cap \text{aff } R$  is a simplex.) For a point  $y$ , reasoning similar to that in the proof of Theorem 4.1 implies that when  $y \notin C(A)$ , we have  $a_h \cdot (\Upsilon(y) - \Upsilon(q)) \geq 0$  for all  $a \in \text{vert } F'$ . Thus,

$$x_h \cdot (\Upsilon(y) - \Upsilon(q)) = \sum_{a \in \text{vert } F'} \alpha_a a_h \cdot (\Upsilon(y) - \Upsilon(q)) \geq 0.$$

If  $x_h \cdot (\Upsilon(y) - \Upsilon(q)) > 0$  then  $y \notin \overline{C}(x)$ , so suppose  $x_h \cdot (\Upsilon(y) - \Upsilon(q)) = 0$ . Since  $\alpha_a > 0$  and  $a_h \cdot (\Upsilon(y) - \Upsilon(q)) \geq 0$  for all  $a \in \text{vert } F'$ , we must have  $a_h \cdot (\Upsilon(y) - \Upsilon(q)) = 0$  for all  $a \in \text{vert } F'$ . The  $y$  for which this holds are precisely those in  $C_{\cap}(F)$ .  $\square$

When all sites closest to a query point are desired, the function *Make\_RPO\_Tree* is modified so that for each face  $F$  of a region  $A \in \Delta(\mathcal{V}(R))$ , the sites  $F.\text{sites} = C_{\cap}(F) \cap S$  are stored for the node  $v$  with  $v.\text{node} = A$ . When answering a query, the variable *current\_closest* represents a set of sites, the sites so far found closest to the query point  $p$ . At each step of *Answer\_Query*, the face  $F$  of  $A$  with query point  $p \in \text{relint } F$  is found, and the distance of the sites in  $F.\text{sites}$  to  $p$  is compared with the distance of those in *current\_closest*. (Note that all sites in  $F.\text{sites}$  are equidistant from  $p$ , and similarly for *current\_closest*.) If the sites in  $F.\text{sites}$  are the same distance as those in *current\_closest*, they are added to the set *current\_closest*. If they are closer, they replace that set, and if they are farther, that set is unchanged. If *current\_closest* is maintained as a list of lists of sites, this updating operation requires constant time.

**4.3. Candidate sets are likely to be small.** The theorem below implies that, with probability 1/2, the candidate sets generated by *Make\_RPO\_Tree* all contain few sites. This ensures that an RPO tree can be created that has height  $O(\log n)$ , and allows a bound on the tree's total size.

As a warm-up, here is a lemma regarding the Delaunay balls.

**LEMMA 4.4.** *For  $S \subset E^d$ , let  $R \subset S$  be a random sample (without replacement) of size  $r$ . Let  $P_\alpha$  be the probability that any open Delaunay ball  $B$  has  $|S \cap B| > \alpha n$ .*

Then  $P_\alpha \leq 1/2$ , for

$$\alpha \geq \frac{\ln(2\binom{r}{d+1})}{r-d-1}.$$

That is,  $1 - P_\alpha \geq 1/2$ , where  $1 - P_\alpha$  is the probability that for all Delaunay balls  $B$ , it holds that  $|S \cap B| \leq \alpha n$ .

*Proof.* Suppose that  $R' \subset R$  is the set of the first  $d + 1$  samples taken. If  $d = \dim R'$ , then the sphere containing  $R'$  defines an open ball  $B$ . Now suppose  $|S \cap B| > \alpha n$ . Then the probability that none of the remaining  $r - d - 1$  samples are taken from  $S \cap B$  is bounded above by  $(1 - \alpha)^{r-d-1}$ . That is, with probability at least  $1 - (1 - \alpha)^{r-d-1}$ ,  $B$  will not be a Delaunay ball of  $\mathcal{V}(R)$ . For sufficiently large  $\alpha$ , the latter probability is large.

Now let  $X$  be the set of all subsets of  $R$  of size  $d + 1$ , let  $B_{R'}$  be the open ball defined by subset  $R' \in X$ , and let  $B(X)$  be the set of open balls defined by these subsets. Let  $B_\alpha(X) \subset B(X)$  be the set of all such balls  $B$  satisfying  $|S \cap B| > \alpha n$ . If no ball  $B \in B_\alpha(X)$  satisfies  $R \cap B = \emptyset$ , then every ball in  $B(X)$  that does not contain any sample sites must not be in  $B_\alpha(X)$ . That is, every Delaunay ball of  $R$  must contain a proportion of sites smaller than  $\alpha$ .

What is an upper bound on the probability  $P_\alpha$  that at least one  $B \in B(X)$  has  $B \in B_\alpha(X)$  and  $R \cap B = \emptyset$ ? For a given ball  $B \in B(X)$ , the joint probability of these two conditions is no more than the conditional probability that  $R \cap B = \emptyset$  given  $B \in B_\alpha(X)$ . The latter probability is the same as that for the ball defined by the first  $d + 1$  sample sites. Since the probability of the union of a set of events is not more than the sum of the probabilities of the individual events, we have

$$P_\alpha = \text{Prob}\{\exists R' \in X \mid B_{R'} \in B_\alpha(X) \text{ and } R \cap B_{R'} = \emptyset\} < \binom{r}{d+1} (1 - \alpha)^{r-d-1}.$$

When  $\alpha \geq \ln(2\binom{r}{d+1})/(r - d - 1)$ , this probability is no more than  $1/2$ , using the relation  $-\ln(1 - \alpha) \geq \alpha$  for  $0 \leq \alpha < 1$ .  $\square$

This lemma is not a proof of the desired result for general  $C(A)$ , since not all regions  $C(A)$  are the union of Delaunay balls. However, the proof of the following theorem is quite similar to that of the lemma.

**THEOREM 4.5.** *For  $S \subset E^d$ , let  $R \subset S$  be a random sample of size  $r$ . Let  $P_\alpha$  be the probability that any one of the regions  $A \in \Delta(\mathcal{V}(R))$  has  $|S \cap C(A)| > \alpha n$ . Then  $P_\alpha \leq 1/2$ , for*

$$\alpha \geq \alpha_{r,d} = \frac{(d+1) \ln((d+1)^2 \binom{r}{d+1})}{r-d-1}.$$

That is,  $1 - P_\alpha \geq 1/2$ , where  $1 - P_\alpha$  is the probability that for all  $A \in \Delta(\mathcal{V}(R))$ , it holds that  $|S \cap C(A)| \leq \alpha n$ .

*Proof.* By Theorems 4.1 and 4.2, for  $A \in \Delta(\mathcal{V}(R))$  with  $A \subseteq \mathcal{V}_q$ ,

$$C(A) = \bigcup_{a \in \text{vert}(A \cap \text{aff } R)} [B_{a,q} \cup (F^d \setminus \text{aff } R)].$$

The simplex  $A \cap \text{aff } R$  has at most  $1 + \dim R$  vertices, so the number of regions making up this union is no more than  $d + 1$ . Let  $\alpha' = \alpha/(d + 1)$ . The condition  $|S \cap C(A)| > \alpha n$  thus implies that  $|S \cap C(I)| > \alpha' n$ , where  $I$  is a region  $B_{a,q} \cup (F^d \setminus \text{aff } R)$ . It suffices to prove that with probability  $1/2$ , all such regions contain no more than  $\alpha' n$  sites.

A region  $I$  may have a real or a ideal. If  $a$  is real, there is a set of  $1 + \dim R$  sites  $R' \subset R$  such that  $I$  is the union of  $F^d \setminus \text{aff } R'$  with the relatively open ball defined by the  $(\dim R)$ -sphere in  $\text{aff } R'$  that contains  $R'$ . If  $a$  is ideal, there is a set of  $\dim R$  sites  $R'$  and a site  $s \in R$  such that  $I$  is the union of  $F^d \setminus \text{aff}(R' \cup \{s\})$  with the open half-flat of  $\text{aff}(R' \cup \{s\})$  that is bounded by  $\text{aff } R'$  and that does not contain  $s$ .

Let  $X$  be the set of all nonempty subsets of  $R$  of size  $d + 1$  or less, together with the set of pairs  $(R', s)$  where  $R' \subset R$ ,  $s \in R$ , and  $1 \leq |R'| \leq d$ . Let  $I_x$  be the region corresponding to  $x \in X$  as above, and let  $I(X)$  be the set of regions corresponding to the elements of  $X$ . Let  $I_{\alpha'}(X)$  be the subset of  $I(X)$  containing regions  $I \in I(X)$  satisfying  $|S \cap I| > \alpha'n$ . We have, for any given region  $I \in I(X)$ ,

$$\text{Prob}\{R \cap I = \emptyset \text{ given } I \in I_{\alpha'}(X)\} \leq (1 - \alpha')^{r-d-1}.$$

The probability

$$\text{Prob}\{\exists x \in X \mid I_x \in I_{\alpha'}(X) \text{ and } R \cap I_x = \emptyset\}$$

is greater than  $P_\alpha$ , and is bounded above by  $(1 - \alpha')^{r-d-1}$  times the size of  $X$ . It is easy to see that  $|X| = (d + 1)\binom{r}{d+1} + d(d + 1)\binom{r}{d+1}$ , or  $|X| = (d + 1)^2\binom{r}{d+1}$ . The theorem follows, using manipulations as in the lemma above.  $\square$

**5. Time bounds for Make\_RPO\_Tree and Answer\_Query.** To bound the time needed for *Make\_RPO\_Tree*, we will first consider the work done by the procedure, aside from the recursive calls, and then bound the work for those calls.

**LEMMA 5.1.** *Let  $t(n)$  denote the expected time required by Make\_RPO\_Tree. Then  $t(n)$  satisfies*

$$t(n) \leq K_1nr^{[d/2]} \log r + K_2r^{[d/2]}t\left(K_3n\frac{\ln r}{r}\right),$$

when  $n > K$ . The constants  $K_1$  and  $K_2$  are at most exponential in  $O(d \log d)$ , and  $K_3$  is  $(d + 1)^2 + O(1/\log r)$ , as  $r \rightarrow \infty$ .

*Proof.* From Theorem 4.5, the **repeat-until** loop for determining a suitable  $\Delta(\mathcal{V}(R))$  will end after two iterations on the average, and require  $O(r^{[d/2]} \log r)$  each iteration, by Lemma 3.9.

The other operations in *Make\_RPO\_Tree* require  $O(n)$  or  $O(r)$  time, except for the recursive calls and the determination, for each  $A \in \Delta(\mathcal{V}(R))$ , of  $S \cap C(A)$ .

From the proof of Lemma 3.5 and precise bounds on the number of vertices of a  $d$ -polytope with  $r$  facets [14, §4.7], the constants  $K_1$  and  $K_2$  are dominated by  $d!$ , which is exponential in  $O(d \log d)$ .

From Theorem 4.5, the size of each subproblem is  $\alpha_{r,d}n$ . The bound for  $K_3$  follows from the value of  $\alpha_{r,d}$  and elementary approximations.  $\square$

**THEOREM 5.2.** *The expected time  $t(n)$  required by Make\_RPO\_Tree is bounded by  $t(n) = O(n^{[d/2](1+\epsilon)})$ , as  $n \rightarrow \infty$ , where*

$$\epsilon = \frac{\ln(K_3 \ln r) + (\ln K_2)/[d/2]}{\ln(r/K_3 \ln r)},$$

for fixed  $r$  and  $d$ .

*Proof.* By “unrolling” the recurrence for  $t(n)$  to depth  $m = \ln(n/K)/\ln(r/K_3 \ln r)$ , we have

$$t(n) = O(nr^{[d/2]} \log r(K_2K_3r^{[d/2]-1} \ln r)^m),$$

and the desired expression follows by algebraic manipulations.  $\square$

**THEOREM 5.3.** *The worst-case time required by procedure Answer\_Query is bounded by*

$$K + K_2 r^{\lceil d/2 \rceil} \ln(n/K) / \ln(r/K_3 \ln r).$$

*This is  $O(\log n)$  as  $n \rightarrow \infty$ , for fixed  $r$  and  $d$ .*

*Proof.* This is just the work in searching through the children of an RPO tree, times the depth of such a tree.  $\square$

**6. Conclusions.** We have seen that a simple, natural approach to the post-office problem may be used to gain great improvements in asymptotic efficiency over methods previously known for  $d > 3$ . In addition, this approach has an advantage of conceptual and programming simplicity over previous asymptotically fast methods for  $d \leq 3$ .

The approach given here may be used to yield fast algorithms for other proximity problems. For example, suppose a convex three-dimensional polytope  $P$  is given, and a data structure is to be found such that given a plane  $h$  with  $h \cap P = \emptyset$ , the vertex of  $P$  closest to  $h$  is to be determined quickly. This problem is equivalent to determining the point of vert  $P$  closest to the ideal point normal to  $h$ , and is also equivalent to linear programming in 3-D with multiple objective functions. The problem may be solved with nearly linear preprocessing and logarithmic query time using an approach analogous to that given in this paper.

After the preliminary report of these results [5], later work has shown that these ideas have applications in many other areas of discrete and computational geometry, such as arrangement searching, determining the separation of polytopes, constructing order  $k$  Voronoi diagrams [6], computing line-segment intersections, bounding ( $\leq k$ )-sets in  $E^d$  [7], computing the diameter of a point set in  $E^3$ , incremental construction of geometric structures [8], and triangulating simple polygons [9]. Independently of this work, the concept of the Vapnik-Chervonenkis (VC) dimension [28] has been applied to, for example, the problem of halfspace range queries, resulting in a randomized algorithm for the construction of a data structure for such queries [16]. This concept has also been applied to questions of learnability [2]. While apparently not equivalent, the two approaches (the VC dimension and that of this paper) are similar in spirit, and provide a useful means of applying divide-and-conquer to computational geometry.

**Appendix A.** Proofs of three technical lemmas are given below.

**LEMMA 2.1.** *Let  $C$  be a polyhedral pointed cone with apex  $a$ . Let  $h$  be a supporting hyperplane of  $C$  with  $h \cap C = \{a\}$ . Let  $\nu$  be a normal vector to  $h$  contained in the same halfspace containing  $C$ . Then  $P = C \cap (h + \nu)$  is a polytope.*

*Proof.* (Note that such a hyperplane  $h$  exists because  $a$  is a face.) Since  $C$  and  $h + \nu$  are polyhedral sets, it follows that their intersection is a polyhedral set. It remains to show that  $P$  is bounded. If not, then  $P$  contains a ray, by [14, 2.5.1]. Such a ray has the form  $\text{ray}_z y$ , where  $z, y \in P$  and  $\nu \cdot (y - z) = 0$ . Since  $C$  is a cone, the point  $(x - a) / \|x - a\| + a \in \text{ray}_a x$  is in  $C$ , for every  $x \in \text{ray}_z y$ . As  $\|x\| \rightarrow \infty$ , with  $x$  on  $\text{ray}_z y$ , the points  $(x - a) / \|x - a\| + a$  converge to  $(y - z) / \|y - z\| + a$ . Since  $C$  is closed, this point is in  $C$ . But  $(y - z) / \|y - z\| + a \in h$ , contradicting the choice of  $h$ .  $\square$

**LEMMA 2.2.** *If  $P \subset E^d$  is a line-free  $d$ -polyhedral set then  $P = \text{conv vert } P$ .*

*Proof.* The lemma is true for polytopes by [14, 2.4.5]. The unbounded case will first be considered for polyhedral cones, and then in general.

If  $C$  is a line-free polyhedral cone, then as mentioned above,  $C$  is pointed, so that Lemma 2.2 applies to  $C$ . Using the correspondence above between  $x \in P$  and  $x' \in \text{id } C$ , the fact that  $P = \text{conv vert } P$  implies that  $\text{id } C = \text{conv vert id } C$ . Since

$$C = \bigcup_{x \in \text{id } C} \text{conv}\{a, x\},$$

it follows that for any  $y \in C$ ,

$$y_h = \alpha_a a_h + \sum_{x \in \text{vert id } C} \alpha_x x_h,$$

for some  $\alpha_a, \alpha_x \geq 0$ ,  $x \in \text{vert id } C$ . That is,  $C \subset \text{conv vert } C$ . It is easy to show that  $C \supset \text{conv vert } C$ , so the lemma follows for line-free polyhedral cones.

To prove the lemma for general line-free polyhedral sets, we appeal to [14, 2.5.6], which directly implies that a line-free polyhedral set  $P$  can be expressed as  $P = \text{cc } P + \text{conv re vert } P$ . Since  $\text{cc } P$  is line-free if  $P$  is, the relation  $\text{cc } P = \text{conv vert cc } P$  holds. The lemma follows.  $\square$

**LEMMA 3.2.** *For a line-free polyhedral region  $P$ , if  $v \in \text{vert } P$  and  $a \in P$ , then the point  $b = \text{ray}_v a \cap \text{relbd } P$  is on a facet of  $P$  that does not contain  $v$ .*

*Proof.* (The relative boundary of  $P$  is generalized to include  $\text{id } P$ .) Since  $\text{relbd } P$  is the union of the facets of  $P$  [14, 2.6.3],  $b$  is on some facet of  $P$ . Suppose  $v \notin \text{id } P$ . Then the lemma follows by induction on dimension: suppose  $v$  and  $b$  are on the same facet  $F$ . Then assuming the lemma for the polytope  $F$ ,  $b$  is on some facet of  $F$  not containing  $v$ . Such a facet of  $F$  is the intersection of  $F$  with another facet  $F'$  of  $P$  [14, 2.6.4], and so  $b \in F'$  but  $v \notin F'$ . Suppose  $v \in \text{id } P$ . If  $a$  is a real point, then so is  $b$ , and the lemma follows. If  $a \in \text{id } P$ , then we map  $v$ ,  $a$ , and so  $b$  to a polytope as in Lemma 2.1, and the lemma follows by the above argument.  $\square$

**Acknowledgments.** I thank Jon Bentley, Brian Kernighan, Danny Sleator, and Chris Van Wyk for their comments on an earlier draft of this paper. I also thank the referees for their helpful comments on another draft. Finally, I thank Chris Van Wyk for comments on yet another draft.

#### REFERENCES

- [1] J. L. BENTLEY, B. WEIDE, AND A. C. YAO, *Optimal expected-time algorithms for closest-point problems*, ACM Trans. Math. Software, 6 (1982), pp. 563–579.
- [2] A. BLUMER, A. EHRENFEUCHT, D. HAUSSLER, AND M. WARMUTH, *Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension*, in Proc. 17th Annual ACM Symposium on Theory of Computing, 1986.
- [3] K. Q. BROWN, *Voronoi diagrams from convex hull*, Inform. Proc. Lett., 9 (1979), pp. 223–228.
- [4] B. CHAZELLE, *How to search in history*, Inform. and Control, 64 (1985), pp. 77–99.
- [5] K. L. CLARKSON, *A probabilistic algorithm for the post office problem*, in Proc. 17th Annual ACM Symposium on Theory of Computing, 1985, pp. 75–184.
- [6] ———, *New applications of random sampling in computational geometry*, Discrete Comput. Geometry, 2 (1987), pp. 195–222.
- [7] ———, *Random sampling in computational geometry*, II, in Proc. Fourth Symposium on Computational Geometry, 1988, pp. 1–11.
- [8] K. L. CLARKSON AND P. W. SHOR, *Algorithms for diametral pairs and convex hulls that are optimal, randomized, and incremental*, in Proc. Fourth Symposium on Computational Geometry, 1988, pp. 12–17.
- [9] K. L. CLARKSON, R. E. TARJAN, AND C. J. VAN WYK, *A fast Las Vegas algorithm for triangulating a simple polygon*, in Proc. Fourth Symposium on Computational Geometry, 1988, pp. 18–22.

- [10] D. DOBKIN AND R. J. LIPTON, *Multidimensional searching problems*, SIAM J. Comput., 5 (1976), pp. 181–186.
- [11] H. EDELSBRUNNER, L. GUIBAS, AND J. STOLFI, *Optimal point location in a planar subdivision*, SIAM J. Comput., 15 (1986), pp. 317–340.
- [12] S. J. FORTUNE, *A sweepline algorithm for Voronoi diagrams*, in Proc. Second Symposium on Computational Geometry, 1986, pp. 313–322.
- [13] E. N. GILBERT, *Random subdivisions of space into crystals*, Ann. Math. Statist., 33 (1962), pp. 958–972.
- [14] B. GRÜNBAUM, *Convex Polytopes*, John Wiley, New York, 1967.
- [15] L. J. GUIBAS, L. RAMSHAW, AND J. STOLFI, *A kinetic framework for computational geometry*, in Proc. 24th IEEE Symposium on Foundations of Computer Science, 1983, pp. 100–111.
- [16] D. HAUSSLER AND E. WELZL, *Epsilon-nets and simplex range queries*, Discrete Comput. Geometry, 2 (1987), pp. 127–151.
- [17] D. KIRKPATRICK, *Optimal search in planar subdivisions*, SIAM J. Comput., 12 (1983), pp. 28–35.
- [18] V. KLEE, *On the complexity of  $d$ -dimensional Voronoi diagrams*, Arch. Math., 34 (1980), pp. 75–80.
- [19] R. J. LIPTON AND R. E. TARJAN, *Applications of a planar separator theorem*, in Proc. 18th IEEE Symposium on Foundations of Computer Science, 1977, pp. 162–169.
- [20] P. MCMULLEN, *The maximum number of faces of a convex polytope*, Mathematika, 17 (1970), pp. 179–184.
- [21] D. E. MULLER AND F. P. PREPARATA, *Finding the intersection of two convex polyhedra*, Theoret. Comput. Sci., 7 (1978), pp. 217–236.
- [22] F. P. PREPARATA AND M. I. SHAMOS, *Computational Geometry: An Introduction*, Springer-Verlag, New York, Berlin, 1985.
- [23] E. G. REES, *Notes on Geometry*, Springer-Verlag, New York, Berlin, 1983.
- [24] R. SEIDEL, *On the size of the closest point Voronoi diagrams*, Tech. Report 94, Institut für Informationsverarbeitung, Technische Universität Graz und Österreichische Computergesellschaft, Austria, 1982.
- [25] ———, *Constructing higher-dimensional convex hulls at logarithmic cost per face*, in Proc. 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 404–413.
- [26] M. I. SHAMOS, *Computational Geometry*, Ph.D. thesis, Yale University, New Haven, CT, 1978.
- [27] J. STOLFI, *Oriented projective geometry*, in Proc. Third Symposium on Computational Geometry, 1987, pp. 76–85.
- [28] V. N. VAPNIK AND A. Y. A. CHERVONENKIS, *On the uniform convergence of relative frequencies of events to their probabilities*, Theory Probab. Appl., 16 (1971), pp. 264–280.



## EFFICIENT SOLUTIONS TO SOME TRANSPORTATION PROBLEMS WITH APPLICATIONS TO MINIMIZING ROBOT ARM TRAVEL\*

MIKHAIL J. ATALLAH† AND S. RAO KOSARAJU‡

**Abstract.** We give efficient solutions to transportation problems motivated by the following robotics problem. A robot arm has the task of rearranging  $m$  objects between  $n$  stations in the plane. Each object is initially at one of these  $n$  stations and needs to be moved to another station. The robot arm consists of a single link that rotates about a fixed pivot. The link can extend in and out (like a telescope) so that its length is a variable. At the end of this “telescoping” link lies a gripper that is capable of grasping any one of the  $m$  given objects (the gripper cannot be holding more than one object at the same time). The robot arm must transport each of the  $m$  objects to its destination and come back to where it started. Since the problem of scheduling the motion of the gripper so as to minimize the total distance traveled is NP-hard, we focus on the problem of minimizing only the total angular motion (rotation of the link about the pivot), or only the telescoping motion. We give algorithms for two different modes of operation: (i) *No-drops*. No object can be dropped before its destination is reached. (ii) *With-drops*. Any object can be dropped at any number of intermediate points. Our algorithm for case (i) runs in  $O(m + n \log n)$  time for angular motion and in  $O(m + na(n))$  time for telescoping motion. Our algorithm for case (ii) runs in  $O(m + n)$  time for angular motion and with the same time bound for telescoping motion. The most interesting problem turns out to be that of minimizing angular motion for the with-drops mode of operation.

**Key words.** transportation problems, robotics, arm motion, Euler tour, circular track, graph augmentation

**AMS(MOS) subject classifications.** 68Q25

**1. Introduction.** A robot arm has the task of rearranging  $m$  objects between  $n$  stations in the plane. Each object is initially at one of these stations and needs to be moved to another station (its destination). The robot arm consists of a single link that rotates about a fixed pivot (see Fig. 1). The link can extend in and out (like a telescope) so that its length is a variable. At the end of this “telescoping” link lies a gripper that is capable of grasping any one of the  $m$  given objects.

The gripper can pick up an object and drop it at another station, then move to another station and continue with the transfers. Many objects can be simultaneously located at the same station, but the gripper cannot be holding more than one object at a time. When the gripper is empty and is at a station, it is free to pick up any of

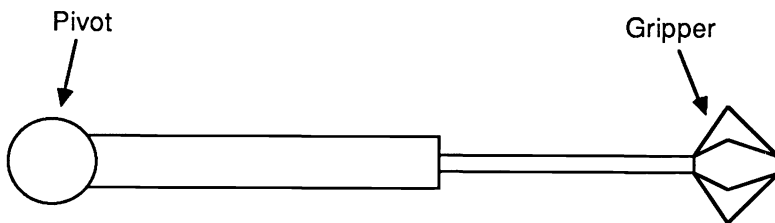


FIG. 1. The robot arm can pivot and can extend like a telescope.

\* Received by the editors June 24, 1987; accepted for publication (in revised form) October 16, 1987.

† Department of Computer Science, Purdue University, West Lafayette, Indiana 47907. The research of this author was supported by the Office of Naval Research under grants N00014-84-K-0502 and N00014-86-K-0689, and the National Science Foundation under grant DCR-8451393, with matching funds from AT&T.

‡ Department of Computer Science, Johns Hopkins University, Baltimore, Maryland 21218. The research of this author was supported by the National Science Foundation under grant DCR-856361.

the objects at that station. We also require that the gripper terminate at the station where it started. Scheduling the motion of the gripper so as to minimize the total distance it travels can be shown to be NP-hard from the NP-hardness of the Euclidean Traveling Salesperson Problem [P2]. Here we focus on the problem of minimizing only the total angular motion (rotation of the link about the pivot), or only the total telescoping motion.

For the case of minimizing angular motion we henceforth assume, without loss of generality, that

- (a) the  $n$  stations are positioned on a circular track centered at the pivot, and
- (b) the motion of the gripper is always along the circumference of this circular track.

The problem is then to minimize the total length of the circular arcs traversed by the gripper. The input specification is made by listing the destinations of the objects at each station on the circular track. The stations, in clockwise cyclic order, are denoted by the integers 1 to  $n$ , and one of them is designated as being the initial position of the gripper (we call it the *start* station). The input therefore describes a directed multigraph having  $n$  vertices and  $m$  edges (we draw a directed arc for each object—the head and the tail corresponding to the destination and the source stations, respectively). Figure 2 illustrates a four-station four-object transfer problem.

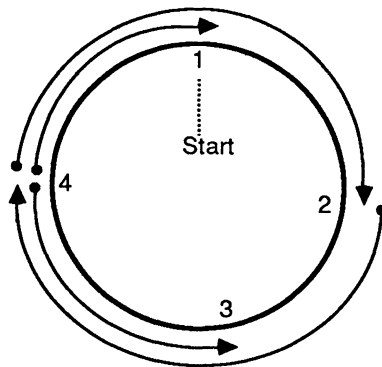


FIG. 2. A transfer problem.

The problem of minimizing the total telescoping motion rather than the angular one can be viewed as a linear track problem rather than as a circular one. The circular track case is considerably more difficult than the linear track one.

We develop fast algorithms for two different modes of operation: (i) *No-drops*. Once an object is picked up by the gripper, it cannot be dropped before its destination is reached. (ii) *With-drops*. Any object can be dropped at any number of intermediate points. The algorithm for the no-drops case runs in  $O(m + n \log n)$  time for a circular track (i.e., minimizing angular motion), in  $O(m + n\alpha(n))$  time for a linear track (i.e., minimizing telescoping motion); here  $\alpha(n)$  is the extremely slowly growing functional inverse of Ackermann's function. The no-drops problem can be cast as a graph-augmentation problem [ET], [P1]—augmentation of a graph to an Eulerian graph. The with-drops problem is more interesting and does not seem to translate into a natural graphical problem. Somewhat surprisingly, we are able to design a faster

algorithm for this problem—an  $O(m+n)$  time algorithm (for either circular or linear track). One of the difficulties in the with-drops problem for a circular track is that an optimal transportation may have to transport an object through the longer of the two circular arcs between its source and destination (such an arc is henceforth called *major*, the other arc being *minor*). In § 4 we give an example for which any optimal transportation must transport an object through the major arc. However, we prove that in a with-drops problem, an optimal transportation transports at most one object through the major arc. This nontrivial result is only one of the ingredients in our linear time solution to this problem; another ingredient is a method for quickly identifying which of the  $m$  objects (if any) should be transported through the major arc.

Throughout the paper, all graphs are actually multigraphs (i.e., can have many edges with same head and tail). A graph is directed unless we explicitly state that it is undirected. All the graphs we refer to are *embedded* on the (circular or linear) track, i.e., their vertices are the stations on the track and their edges are directed arcs drawn along the track. Therefore when we henceforth refer to an edge  $e$  of a graph  $G$ , we are really talking about a particular drawing of that edge (for a circular track, the edge can be drawn two ways). We use  $|e|$  to denote the length of the portion of the track covered by  $e$ .

For a circular track we assume, without loss of generality, that the circle's circumference equals unity. The *complement* of an edge  $e$  is the edge  $e^c$  with the same source as  $e$ , same destination as  $e$ , and such that  $e$  and  $e^c$  together cover the complete circumference (see Fig. 3). Note that  $(e^c)^c = e$ , and that  $|e| + |e^c| = 1$ . An edge  $e$  is *major* if and only if  $|e| > \frac{1}{2}$ , and is *minor* otherwise (in Fig. 3,  $e$  is minor and  $e^c$  is major). Note that if  $|e| = \frac{1}{2}$  then both  $e$  and  $e^c$  are minor. *Shortening* a major edge means replacing it with its complement.

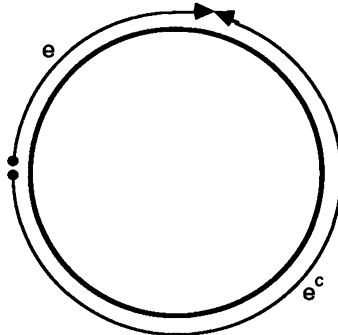


FIG. 3. An arc and its complement.

We adopt the convention that, when depicting a transportation, we draw an input source-to-destination pair as a directed (circular or linear) arc coinciding with the actual path that this transportation uses to take the object to its destination (in the with-drops case, the object transported along such an arc may be dropped many times on the way to its destination).

We assume that none of the  $n$  stations is useless, i.e., each is the source or destination of at least one object (useless stations are easily eliminated with an  $O(m+n)$  preprocessing step). This implies that  $n \leq 2m$ .

**2. No-drops problem.** In this section we prove the following result.

**THEOREM 1.** *An optimal transportation for any no-drops problem can be calculated in  $O(m + n\alpha(n))$  for a linear track, in  $O(m + n \log n)$  time for a circular track.*

The rest of this section proves the above theorem.

First, observe that in a circular no-drops problem we never need to take an object to its destination using the major arc, and therefore we always draw the input edges so that they are minor.

The no-drops problem is a graph-augmentation problem: we want to add edges to the input graph so as to make it Eulerian [E], such that the total lengths of the added edges is minimum. Any Euler tour of the resulting Eulerian graph then gives an optimal transportation. These added edges are called *augmenting edges*, and correspond to motion of the gripper when it is not holding any object. In future drawings, we distinguish such augmenting edges by drawing their arrowhead dashed, whereas that of an input edge is drawn solid.

Since the minimum Eulerian augmentation does not depend on the start vertex, the length of an optimal transportation does not depend on which vertex is the start (and hence finish) vertex. (In the with-drops case, considered in § 3, the start vertex is significant.)

Recall that a graph  $G$  is Eulerian if and only if (i) every vertex of  $G$  has its in-degree equal to its out-degree (we call this the *degree-balance* property), and (ii) the undirected version of  $G$  is connected. Condition (ii) can be replaced by “ $G$  is strongly connected,” because if (i) holds then  $G$  is strongly connected if and only if its undirected version is connected [E]. In the rest of this paper we restrict the augmenting edges to be of the form  $(i, i+1)$  or  $(i+1, i)$ , i.e., each augmenting edge covers only one of the  $n$  intervals (gaps) between adjacent stations. There is no loss of generality in doing so, since an augmenting edge that covers  $l$  intervals can always be broken into  $l$  smaller edges without increasing the total edge length, without disturbing degree balance, and without damaging undirected connectivity. Of course if there are many such augmenting edges covering an interval  $(i, i+1)$ , then we do not store each of them individually since this might take a total of  $O(mn)$  space; instead, we store a count of the number of such edges going in each direction across that interval. Thus the total storage needed for augmenting edges is  $O(n)$ .

Observe that in any optimal augmentation, if any pair of antiparallel edges  $(i, i+1)$  and  $(i+1, i)$  are augmenting edges, then between  $i$  and  $i+1$  there cannot be any other augmenting edge (otherwise removal of  $(i, i+1)$  and  $(i+1, i)$  preserves the degree-balance and undirected connectivity, contradicting the optimality of the original augmentation).

**2.1. Linear track.** We first prove the linear track part of Theorem 1, an example of which is given in Fig. 4(a), where  $n = 8$  and  $m = 5$ . We make a few trivial observations.

**OBSERVATION 1.** In any transportation, at any point  $x$  between the leftmost and the rightmost stations, the number of times the gripper moves left to right across  $x$  is the same as the number of times the gripper moves right to left across  $x$ . In addition each of these crossings is  $\geq 1$ .

Based on this observation, we add across each interval the smallest number of augmenting edges that will make the total number of edges that cross that interval from left to right equal to the number of edges that cross it from right to left. The “augmenting edges” needed for Fig. 4(a) are shown in Fig. 4(b). When the graph is augmented in this manner, every vertex will have the degree-balance property. Let this augmentation process be denoted as the *degree-balanced augmentation*.

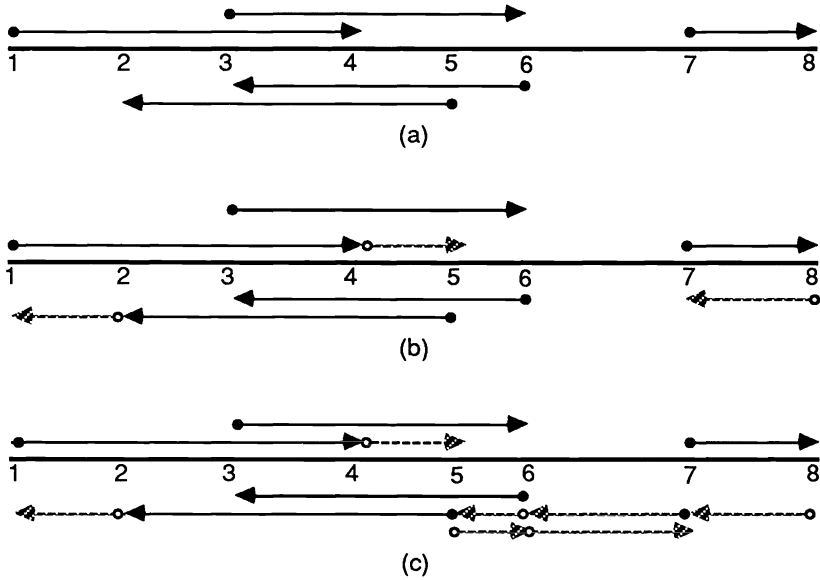


FIG. 4. (a) A linear track problem; (b) its degree-balanced augmentation; (c) the optimal augmentation.

OBSERVATION 2. If the resulting degree-balanced augmented graph is strongly connected, then it has an Euler tour, and hence it represents an optimal transportation.

However the augmented graph need not be strongly connected (sc). For example, Fig. 4(b) has three strongly connected components (scc's):  $\{1, 4, 5, 2\}$ ,  $\{3, 6\}$ ,  $\{7, 8\}$ . If the augmented graph is not sc then its scc's are disjoint in the sense that there is no edge between any two of them (because for a graph having the degree-balance property, the scc's are the connected components of the undirected version of the graph).

Now the problem reduces to adding more augmenting edges, with minimum total length, to make the graph sc without disturbing its degree-balance property. An example of this augmentation is shown in Fig. 4(c). In general, augmentation of a  $q$ -scc degree-balanced graph can be achieved by including  $q - 1$  antiparallel pairs of augmenting edges (we needed two such pairs to go from Fig. 4(b) to 4(c): one between 5 and 6, the other between 6 and 7). To find the  $q - 1$  antiparallel pairs needed to minimally make the degree-balanced graph sc, we create a  $q$ -vertex edge-weighted undirected graph, one vertex for each scc. In that undirected graph, an edge between  $i$  and  $j$  is present if and only if a station  $x$  in the  $i$ th scc is adjacent to a station  $y$  in the  $j$ th scc (i.e.,  $|x - y| = 1$ ). The weight of this edge  $\{i, j\}$  is the distance between stations  $x$  and  $y$ . If there are many such pairs  $x, y$  for a particular  $\{i, j\}$ , then the weight of  $\{i, j\}$  is the minimum over all such pairs  $x, y$ .

OBSERVATION 3. The minimum total length augmenting pairs needed to make the degree-balanced graph sc correspond to the undirected edges of a minimum spanning tree (MST) of the above-mentioned  $q$ -vertex undirected graph of scc's. (Of course, we have to map each undirected edge  $\{i, j\}$  of the MST into one antiparallel pair of augmenting edges:  $(x, y), (y, x)$  in which  $x$  and  $y$  are the stations in scc's  $i$  and  $j$ , respectively, which contributed to the weight of the  $\{i, j\}$  edge.)

The above discussion implies an  $O(m + n\alpha(n))$  time algorithm for computing the minimum Eulerian augmentation in the linear track version of the problem, using the MST algorithm of [FT]. Any Euler tour of the resulting Eulerian graph gives an optimal

transportation. Such an Euler tour can easily be found in an additional  $O(m)$  time [AHU, Exercise 5.9].

Thus the overall time for the linear track case is  $O(m + n\alpha(n))$ . We now complete the proof of Theorem 1 by considering the circular track case.

**2.2. Circular track problem.** If we know that there exists an optimal transportation in which at least one interval is not covered by any augmenting edge, then we can solve  $n$  separate straight line problems: the  $i$ th one, assuming that there is no augmenting edge between stations  $i$  and  $i+1$  (assume that station  $n+1 =$  station 1). Then the transportation corresponding to the minimum of these  $n$  solutions gives the optimal transportation. However, it is not hard to come up with an example in which any optimal solution must have augmenting edges covering the (complete) circumference.

The circular track equivalent of Observation 1 does not hold, i.e., for the circular track it is no longer true that at every point the number of clockwise crossings of the gripper is the same as the number of counterclockwise crossings. However, if we define the *flux* across an interval to be the number of clockwise crossings minus the number of counterclockwise crossings (counting both the input edges and the augmenting ones), then we have the following.

**LEMMA 1.** *For any augmentation, the degree-balance property is satisfied if and only if the flux is the same across all intervals.*

*Proof.* It suffices to show that degree balance holds if and only if, for any  $i$ , the flux across interval  $(i-1, i)$  is the same as that across interval  $(i, i+1)$ . The difference between these two fluxes equals the difference between the in-degree of  $i$  and its out-degree.  $\square$

The flux across an interval is the sum of two components. One component is the *augmenting* flux across that interval: the number of clockwise augmenting edges across that interval minus the number of counterclockwise augmenting edges across it. The other component is the *input* flux across that interval and is the number of clockwise input edges across it minus the number of counterclockwise input edges across it. Let  $\phi(i)$  denote the input flux across the interval  $(i, i+1)$ . In Fig. 2,  $\phi(1) = \phi(2) = 1$ ,  $\phi(3) = 0$ ,  $\phi(4) = 2$ . Note that  $\phi(i)$  is the number of counterclockwise augmenting edges that must be added to interval  $(i, i+1)$  in order to make its total flux equal to zero (a negative value signifies adding clockwise edges).

The next two lemmas impose constraints on the augmenting edges and possible flux values that an optimal augmentation can have.

**LEMMA 2.** *There exists an optimal augmentation in which for some  $i$  the number of augmenting edges in between  $i$  and  $i+1$  is no more than one.*

*Proof.* Let an optimal augmentation result in at least two augmenting edges between every  $i$  and  $i+1$ . Among all such optimal augmentations, select one with the fewest clockwise augmenting edges. Select any undirected circuit of  $n$  augmenting edges covering the circumference (ignoring the directions of these augmenting edges). We distinguish two cases.

*Case 1.* On this circuit, the total length of the clockwise edges is not equal to the total length of the counterclockwise edges. If it is larger (respectively, smaller), then remove the clockwise (respectively, counterclockwise) edges and duplicate the counterclockwise (respectively, clockwise) edges one more time. This preserves undirected connectivity and also the degree-balance property. In addition, this transformation decreases the total length. This contradicts the optimality of the original augmentation.

*Case 2.* On this circuit, the total length of the clockwise edges is equal to the total length of the counterclockwise edges. Remove the clockwise edges and duplicate

the counterclockwise edges one more time. This preserves undirected connectivity, the degree-balance property, and the total length. However it results in an optimal augmentation having fewer clockwise edges than the original one, a contradiction.  $\square$

Note that Lemma 1 implies that for every optimal transportation there exists a value such that the flux across every interval is that value. In addition, Lemma 2 implies that there are only  $3n$  relevant values of flux worth considering, namely  $\cup_{i=1}^n \{\phi(i) - 1, \phi(i), \phi(i) + 1\}$ .

LEMMA 3. *There exists an optimal augmentation whose flux is between  $-m - 1$  and  $m + 1$ .*

*Proof.* Lemma 2 implies that there exists an optimal augmentation in which at least one interval has at most one augmenting edge across it. The absolute value of the flux of such an augmentation is no more more than  $1 + \max_{1 \leq i \leq n} |\phi(i)| \leq 1 + m$ .  $\square$

It is easy to come up with examples in which there is a unique optimal augmentation and it has flux  $\Theta(m)$ . The range “ $-m - 1$  to  $m + 1$ ” of Lemma 3 can be narrowed to “ $-m/2$  to  $m/2$ ” but we avoid doing so for simplicity of exposition.

Observe that fixing the value of the flux (at, say,  $\psi$ ) entirely determines the cost of the minimum augmentation achieving degree balance at that flux value, because every interval  $(i, i + 1)$  needs to add across it  $\psi - \phi(i)$  clockwise augmenting edges in order for the flux across it to become  $\psi$ . The resulting graph, however, may not be sc, and additional pairs of antiparallel edges may need to be added in order to make it sc. For a given flux value, the antiparallel pairs needed to make the degree-balanced graph sc can be determined by a minimum-cost spanning tree computation similar to the one described for the linear track case. Our main problem is therefore that of determining which flux value  $\psi_0$  is such that there is an optimal Eulerian augmentation whose flux is  $\psi_0$ .

Let the *cost* of flux  $\psi$  be the total length of the minimum Eulerian augmentation whose flux is constrained to be  $\psi$ . This cost consists of two components: (i) a degree-balance component  $db(\psi)$  equal to  $\sum_{i=1}^n |\psi - \phi(i)|l_i$ , where  $l_i$  is the length of the interval  $(i, i + 1)$ , and (ii) a connectivity component  $cc(\psi)$  which accounts for the length of the antiparallel pairs of augmenting edges needed to make sc the degree-balanced graph resulting from (i). (The  $cc(\psi)$  results from the previously mentioned MST computation.)

If we knew  $db(\psi)$  and  $cc(\psi)$  values for all  $-m - 1 \leq \psi \leq m + 1$ , then the optimal  $\psi$  would be the one which minimizes  $db(\psi) + cc(\psi)$ . The next two lemmas show how to compute all the  $db(\psi)$ 's and  $cc(\psi)$ 's efficiently (the nontrivial part is computing the  $cc(\psi)$ 's).

LEMMA 4. *The  $db(\psi)$ 's ( $-m - 1 \leq \psi \leq m + 1$ ) can all be computed in  $O(m)$  time.*

*Proof.* It suffices to show that the description of the function  $db(\cdot)$  can be computed in  $O(m)$  time. Note that  $db(\psi) = \sum_{i=1}^n |\psi - \phi(i)|l_i$  is piecewise linear and has at most  $n$  angular points (at  $\psi = \phi(i)$ ,  $1 \leq i \leq n$ ). It is easy to compute  $\phi(1), \dots, \phi(n)$  in  $O(m)$  time. If we knew the slope of  $db(\psi)$  at every value of  $\psi$ ,  $-m - 1 \leq \psi \leq m + 1$ , then we could easily obtain all the  $db(\psi)$ 's with  $O(m)$  additional work. The slope at  $\psi = -m - 1$  is equal to  $-\sum_{i=1}^n l_i = -1$ , and at  $\psi = m + 1$  it equals  $\sum_{i=1}^n l_i = 1$ ; in between it changes only at values of  $\psi$  that belong to  $\{\phi(1), \dots, \phi(n)\}$ . Therefore we sort  $\{\phi(1), \dots, \phi(n)\}$  in  $O(m)$  time, and then we scan the resulting sorted sequence, updating the slope of  $db(\cdot)$  as we go along.  $\square$

LEMMA 5. *The  $cc(\psi)$ 's ( $-m - 1 \leq \psi \leq m + 1$ ) can all be computed in  $O(m + n \log n)$ .*

*Proof.* First observe that if  $\psi \notin \cup_{i=1}^n \{\phi(i)\}$ , then  $cc(\psi) = 0$  because in that case the degree-balanced graph of flux  $\psi$  is already sc (it has an augmenting edge across

every one of the  $n$  intervals). We therefore need only concern ourselves with computing the  $cc(\psi)$ 's for all  $\psi \in \cup_{i=1}^n \{\phi(i)\}$ . By its definition,  $cc(\psi)$  is equal to twice the cost of the MST of the undirected graph  $CC(\psi)$  whose vertices are the  $n$  stations, and whose edges consist of

- (i) the undirected versions of the input edges, and
- (ii) one edge  $\{i, i + 1\}$  for each interval  $(i, i + 1)$ .

The edges in (i) have zero cost in  $CC(\psi)$ , while an edge  $\{i, i + 1\}$  in (ii) has zero cost if  $\psi \neq \phi(i)$  (because in that case the minimum degree-balanced augmentation for flux value  $\psi$  already places at least one augmenting edge across the interval  $(i, i + 1)$ ), and cost equal to the interval's length  $l_i$  if  $\psi = \phi(i)$ . Note that all  $CC(\psi)$ 's have the same set of edges, the only difference being in the weights of the  $n$  edges in (ii). Since the edges in (i) have zero cost in all  $CC(\psi)$ 's, we can "collapse" each connected component of the edges in (i) into a single vertex: let  $\nu_1, \dots, \nu_q$  be the vertices resulting from this collapsing operation; if the endpoints of an edge in (ii) collapse into a single  $\nu_i$  then the edge vanishes; otherwise it survives (of course its endpoints become the collapsed vertices rather than the original stations). This collapsing operation can easily be done in  $O(m)$  time as a preprocessing step. Assume from now on that this has already been done, so that every  $CC(\psi)$  is now a  $q$ -vertex multigraph having as edges the (at most  $n$ , at least  $q$ ) edges in (ii) that survived the collapsing operation. Each of the (possibly many) edges between  $\nu_i$  and  $\nu_j$  corresponds to an interval between one of the stations that collapsed into  $\nu_i$  and one of the stations that collapsed into  $\nu_j$ . Of course  $cc(\psi)$  is still twice the cost of the MST of the new (collapsed)  $CC(\psi)$ . Let  $CC$  be identical to the (collapsed)  $CC(\psi)$ , except that in  $CC$  the costs associated with the edges of  $CC(\psi)$  are replaced by *labels*: the edge of  $CC$  that corresponds to interval  $(i, i + 1)$  is labeled by  $\phi(i)$ . Note that  $CC(\psi)$  can be obtained from  $CC$  by assigning to each edge with label  $\phi(i)$  a cost of zero if  $\phi(i) \neq \psi$ , a cost equal to the length of  $(i, i + 1)$  if  $\phi(i) = \psi$ . Let the intervals that correspond to edges of  $CC$  be denoted by  $(i_1, i_1 + 1), \dots, (i_r, i_r + 1)$ . Find the median of  $\phi(i_1), \dots, \phi(i_r)$  (call it  $\phi_0$ ), then partition the set  $\{i_1, \dots, i_r\}$  into  $A, B, C$  as follows:  $A = \{i_j: \phi(i_j) < \phi_0\}$ ,  $B = \{i_j: \phi(i_j) = \phi_0\}$ ,  $C = \{i_j: \phi(i_j) > \phi_0\}$ . (Note that each of  $A$  and  $C$  has at most  $r/2$  elements.) The important thing to notice is that, if  $\psi \cong \phi_0$  (respectively,  $\neq \phi_0, \cong \phi_0$ ) and  $i$  belongs to  $A$  (respectively,  $B, C$ ), then in  $CC(\psi)$  the edge corresponding to interval  $(i, i + 1)$  has zero cost. This suggests the following recursive procedure for computing the  $cc(\psi)$ 's for all  $\psi \in \{\phi(i): i \in A \cup B \cup C\}$ . First, create the undirected graph  $Q_A$  (respectively,  $Q_B, Q_C$ ) whose vertices are  $\nu_1, \dots, \nu_q$  and each of whose edges correspond to an interval  $(i, i + 1)$  with  $i \notin A$  (respectively,  $B, C$ ). Let  $CC_A$  (respectively,  $CC_B, CC_C$ ) be obtained from  $CC$  by collapsing each connected component of  $Q_A$  (respectively,  $Q_B, Q_C$ ) into a single vertex. Note that  $CC_A$  (respectively,  $CC_B, CC_C$ ) has no more than  $|A|$  (respectively,  $|B|, |C|$ ) edges, and no more vertices than it has edges. Recursively compute the  $cc_A(\psi)$  values for all  $\psi \in \{\phi(i): i \in A\}$ .

*Note.*  $cc_A(\psi)$  (respectively,  $cc_B(\psi), cc_C(\psi)$ ) denotes twice the cost of the MST of  $CC_A(\psi)$  (respectively,  $CC_B(\psi), CC_C(\psi)$ ). Note that if  $\psi = \phi(i)$  for some  $i$  in  $A$  (respectively,  $B, C$ ) then  $cc(\psi)$  equals  $cc_A(\psi)$  (respectively,  $cc_B(\psi), cc_C(\psi)$ ).

Next, recursively compute the  $cc_C(\psi)$  values for all  $\psi \in \{\phi(i): i \in C\}$ . Then find  $CC_B(\phi_0)$  and compute its MST in  $O(|B|\alpha(|B|))$  time [FT] ( $cc_B(\phi_0)$  is twice the cost of this MST). If  $T(r)$  denotes the overall time for this recursive procedure, then we have

$$T(r) \leq T(|A|) + T(|C|) + c_1 r + c_2 |B| \alpha(|B|),$$

where  $|A| \leq r/2, |C| \leq r/2$ , and  $|A| + |B| + |C| = r$ . This implies that  $T(r) = O(r \log r)$ .  $\square$



This completes the proof of Theorem 1.

Now we consider the more interesting with-drops mode of operation.

**3. With-drops problem.** The main result of this section is the following.

**THEOREM 2.** *When drops are allowed, an optimal transportation for the circular track problem (and hence for the linear track one as well) can be computed in  $O(m + n)$  time.*

The proof of the above theorem is developed through the end of this section, and involves several nontrivial insights into the structure of the with-drops problem. We concern ourselves with the circular track problem only, since an  $O(m + n)$  time solution to the circular track problem automatically implies an  $O(m + n)$  time solution to the linear track problem (by first embedding the linear track problem on a very small circular arc of a circular track and then using the circular track algorithm). Since  $n \leq 2m$  it suffices to give an  $O(m)$  time algorithm.

First, observe that every object can be moved to its target station by moving it in one direction only. However this observation still allows two possibilities for transporting an object: along the minor arc between its endpoints, or along the major arc. For example, Fig. 5 proves that an optimal transportation for some problems must include transporting an object by the major arc. The (1, 2), (1, 4), and (2, 4) distances are  $\frac{1}{3}$  each, and the (3, 4) distance is very small. If we transport (1, 2) and (2, 1) by the minor arcs (Fig. 5(a)) then the complete transportation length is  $\frac{4}{3}$  (a pair of antiparallel augmenting edges between 2 and 3 is then needed). However if we transport (1, 2) by the major arc, as in Fig. 5(b), we can drop it at station 4 (we henceforth call such a drop an *intermediate stop*), then complete the (4, 3) and (3, 4) transports and finally resume the transportation of the (1, 2) arc. In this case the total path length is approximately one.

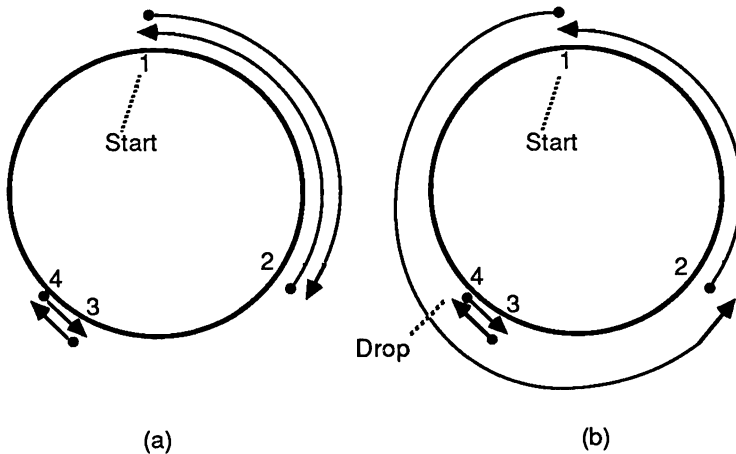


FIG. 5. (a) A with-drops problem; (b) its optimal transportation.

As in § 2, each augmenting edge corresponds to motion of the gripper when it is not holding any object, and covers only one interval (if the motion spans more than one interval, then each interval will get an augmenting edge). Also as before, if there are many augmenting edges across an interval, then we store only a count of the number of such edges going in each direction across that interval.

Lemma 1 obviously still holds. Note that the fact that the transportation may move objects to their destinations using the major arc implies that there are  $2^m$  possible

ways to draw the  $m$  input source-to-destination pairs (whereas in § 2 there was only one way to draw them). Lemma 10 will establish that we can restrict our attention to only  $m$  possibilities.

Let  $T$  be any transportation (with drops). We associate with  $T$  a graph  $G(T)$  whose vertices are the  $n$  stations, and whose edges are the  $m$  edges corresponding to the input source-to-destination pairs, plus any augmenting edges. Each input edge in the transportation might have been covered by many intermediate stops, but in the graph  $G(T)$  we simply draw the edge from its source to its destination. For example, in the transportation of Fig. 5(b), the edge  $(1, 2)$  is a single edge even though this object gets dropped at station 4 and picked up from there later on. Note that  $G(T)$  has the degree-balance property, and hence the flux corresponding to  $T$  is the same across every interval. Since  $G(T)$  is degree-balanced, its scc's are also the connected components of the undirected version of  $G(T)$ . Therefore  $G(T)$  is the union of disjoint scc's. For example, in Fig. 5(b), the graph depicted has two scc's:  $\{1, 2\}$ ,  $\{3, 4\}$ . A graph  $G$  is *transportable from vertex  $x$*  if and only if there exists a transportation  $T$  with  $x$  as its start (and hence finish) vertex, and such that  $G(T) = G$ . The graph shown in Fig. 5(b) is not transportable from vertex 3, but it is transportable from vertex 1. This example also illustrates how the length of an optimal transportation now depends on where the start vertex is.

In the following, we first establish that the graph  $G(T)$  of an optimal transportation  $T$  can be computed in  $O(m)$  time (Lemma 19). Then we show that  $T$  can be calculated from  $G(T)$  in  $O(m)$  time (Lemma 20).

For the next three lemmas, BAL is any degree-balanced graph, and the scc's of BAL are denoted by  $scc_1, \dots, scc_h$ . Observe that any  $scc_i$  can individually be transported using any vertex in  $scc_i$  as the start vertex (even without drops).

Now we define the *reachability graph* of BAL to have vertices  $scc_1, \dots, scc_h$ , and to have an edge from  $scc_i$  to  $scc_j$  if and only if there exists a vertex  $x$  of  $scc_j$  on some edge  $e$  of  $scc_i$  (i.e.,  $x$  occurs on the circular arc covered by  $e$ ). This is represented by  $scc_i \rightarrow scc_j$ , and we say that  $scc_j$  is *directly reachable* from  $scc_i$ .

The above definition of direct reachability implies that, if edges  $e_1$  and  $e_2$  of BAL overlap and neither one of them properly contains the other, then either  $e_1$  and  $e_2$  are in the same scc of BAL or the scc's containing  $e_1$  and  $e_2$  are directly reachable from each other.

We say that  $scc_j$  is *reachable* from  $scc_i$  if and only if there is a directed path from  $scc_i$  to  $scc_j$  in the reachability graph.

LEMMA 6. *In the reachability graph, if  $scc_i \rightarrow scc_j$ , then  $scc_i \cup scc_j$  can be transported using any vertex of  $scc_i$  as the start vertex.*

*Proof.* Since  $scc_i \rightarrow scc_j$ , there exists an edge  $e$  of  $scc_i$  that covers a vertex  $x$  of  $scc_j$ . Transport  $scc_i$  until point  $x$  of  $e$  is reached, drop the object, finish  $scc_j$ , and then complete  $scc_i$ .  $\square$

In fact the following generalization of Lemma 6 holds.

LEMMA 7. *If  $H$  is any directed spanning subtree of the reachability graph with  $scc_i$  as the root, then the union of all the scc's in  $H$  is transportable using any vertex of  $scc_i$  as the start vertex. In addition, the total number of intermediate stops is no more than (the number of vertices of  $H$ ) - 1.*

*Proof.* The transportation process resembles a depth-first search of  $H$ , begun at  $scc_i$ : First we mark every  $scc_j \in H$  as being "new," then we mark  $scc_i$  as being "old" and begin transporting  $scc_i$  from any start vertex in  $scc_i$ . Whenever we are transporting an edge  $e$  of the scc currently being transported, we go through every edge  $f$  that has an endpoint covered by  $e$ : if  $f$  is in a child scc of the current scc, and if the scc of  $f$

is still marked “new,” we mark the scc of  $f$  as being “old,” interrupt the transportation of the scc of  $e$ , and recursively transport the scc of  $f$  using a start (and hence finish) vertex the endpoint of  $f$  covered by  $e$ . Every  $\text{scc}_i \in H$  eventually gets transported, and every such  $\text{scc}_j$  (except the root,  $\text{scc}_i$ ) causes one intermediate stop to occur during the transportation of its parent.  $\square$

**COROLLARY 1.** *A graph is transportable from vertex  $x$  if and only if it has degree-balance and, in its reachability graph, every scc is reachable from the scc that contains  $x$ .*

*Proof.* The “only if” part of the proof is trivial; the “if” part follows from Lemma 7.  $\square$

**COROLLARY 2.** *If  $G$  is transportable from vertex  $x$ , then there is a transportation  $T$  of  $G$  (i.e.,  $G(T) = G$ ) from  $x$  such that, in  $T$ , the number of intermediate stops is no more than  $n - 1$ .*

*Proof.* The proof is an immediate consequence of Lemma 7.  $\square$

**LEMMA 8.** *Let  $S$  be a subset of the scc's of BAL such that every scc in  $S$  is reachable from  $\text{scc}_i$ . Suppose that the union of the scc's in  $S$  covers the circumference. Then for any degree-balanced graph  $G$  ( $G$  may be disconnected),  $\text{BAL} \cup G$  is transportable from any vertex in  $\text{scc}_i$ .*

*Proof.* Since the union of the scc's in  $S$  covers the circumference, every scc of BAL is reachable from at least one scc in  $S$ . This, and the fact that every scc in  $S$  is reachable from  $\text{scc}_i$ , implies that every scc of BAL is reachable from  $\text{scc}_i$ . Therefore (by Corollary 1) BAL is transportable from any vertex in  $\text{scc}_i$ . While transporting BAL, we are bound to reach a vertex in each of the scc's of  $G$ . At such vertices interrupt the main transportation of BAL and finish the scc's of  $G$ .  $\square$

**LEMMA 9.** *Let  $T$  be an optimal transportation and let  $e$  be a major edge in  $G(T)$ . Then the scc of  $G(T)$  that contains  $e$  must cover the circumference.*

*Proof.* Let  $\text{scc}(e)$  be the scc containing  $e$ , and  $\text{scc}_1$  be the scc containing the start vertex. Since  $T$  is a transportation,  $\text{scc}(e)$  is reachable in  $G(T)$  from  $\text{scc}_1$ . If  $\text{scc}(e)$  does not cover the circumference, then its individual flux (the flux due to its edges only) is zero, and hence any interval covered by  $\text{scc}(e)$  is covered by at least two edges of  $\text{scc}(e)$ . Therefore, in  $G(T) - e + e^c$ , the scc containing  $e^c$  is still reachable from  $\text{scc}_1$ , and it now covers the circumference. Therefore, by Lemma 8, all the other scc's of  $G(T) - e + e^c$  are transportable from the start vertex. Thus  $G(T) - e + e^c$  is transportable from the start vertex, which contradicts the optimality of  $T$  (since  $e$  is longer than  $e^c$ ).  $\square$

**LEMMA 10.** *In any optimal transportation, at most one object is moved to its destination along the major arc.*

*Proof.* Let  $T$  be an optimal transportation, let  $\text{scc}_1, \dots, \text{scc}_h$  be the scc's of  $G(T)$ , and let the start vertex be in  $\text{scc}_1$ . Suppose that  $G(T)$  has two major edges  $e_i$  and  $e_j$ , respectively, in  $\text{scc}_i$  and  $\text{scc}_j$ . By Lemma 9,  $\text{scc}_i$  covers the circumference, and so does  $\text{scc}_j$ .

*Case 1.*  $\text{scc}_i \neq \text{scc}_j$ . Without loss of generality, we can assume that, in the reachability graph,  $\text{scc}_i$  is reachable from  $\text{scc}_1$  without going through  $\text{scc}_j$ . Now, modify  $\text{scc}_j$  by shortening  $e_j$ . Because  $\text{scc}_i$  is still reachable from  $\text{scc}_1$  and still covers the circumference, even this modified  $\text{scc}_j$  along with all the other scc's are transportable from the start vertex (by Lemma 8). Since we made  $e_j$  shorter, the new transportation has smaller length than  $T$ , a contradiction.

*Case 2.*  $\text{scc}_i = \text{scc}_j$ . We distinguish two subcases.

*Subcase 2.1.* Every interval covered by  $e_i$  is covered by at least one other edge of  $\text{scc}_i$ . Consequently, modifying  $\text{scc}_i$  by shortening  $e_i$  leaves the circumference covered by the new  $\text{scc}_i$ , which is still reachable from  $\text{scc}_1$ . Therefore, by Lemma 8, the graph

obtained from  $G(T)$  by shortening  $e_i$  is still transportable. This contradicts the optimality of  $T$ .

*Subcase 2.2.* Some interval is covered by  $e_i$  and by no other edge of  $scc_i$ . Then the individual flux of  $scc_i$  is  $+1$  or  $-1$ , and therefore every interval in the region covered by both  $e_i$  and  $e_j$  is also covered by at least one other edge of  $scc_i$  (because the individual flux of  $scc_i$  is an odd number). In this case simultaneously shortening both  $e_i$  and  $e_j$  leaves the new  $scc_i$  still covering the circumference, leading to a contradiction.  $\square$

Lemma 10 reduces from  $2^m$  to  $m$  the number of possible drawings of the  $m$  input edges that need to be considered, but it does not yet give an  $O(m)$  time algorithm. We must still identify, in  $O(m)$  time, which edge (if any) needs to be drawn along the major arc. Even if we knew which drawing of the  $m$  input edges is best, it is not clear how to augment these into a minimum-length graph that is transportable (from the start vertex). All these nontrivial issues are addressed below.

LEMMA 11. *Let  $T$  be an optimal transportation, and let  $e$  be a major edge of  $G(T)$ . Then at least one interval covered by  $e$  is not covered by any other edge of  $G(T)$ .*

*Proof.* Suppose to the contrary that the region covered by  $e$  is also covered in  $G(T) - e$ . Then there exists in  $G(T) - e$  a sequence of edges  $f_1, \dots, f_s$  such that  $f_1 \cup \dots \cup f_s$  covers  $e$ , and every  $f_i$  contains an endpoint of  $f_{i+1}$ ,  $1 \leq i < s$ . Figure 6 illustrates this ( $s = 5$ ), ignoring edge directions as well as the distinction between input edges and augmenting ones. We assume that the sequence  $f_1, \dots, f_s$  has the smallest number of elements ( $= s$ ) among all such sequences covering  $e$ ; this implies that neither one of  $f_i$  and  $f_{i+1}$  contains the other. Note that Lemma 10 implies that  $s > 1$ . Let  $scc_1$  be the scc of  $G(T)$  that contains the start vertex, and for any edge  $x$ ,  $scc(x)$  denotes the scc that contains  $x$ . We distinguish two cases.

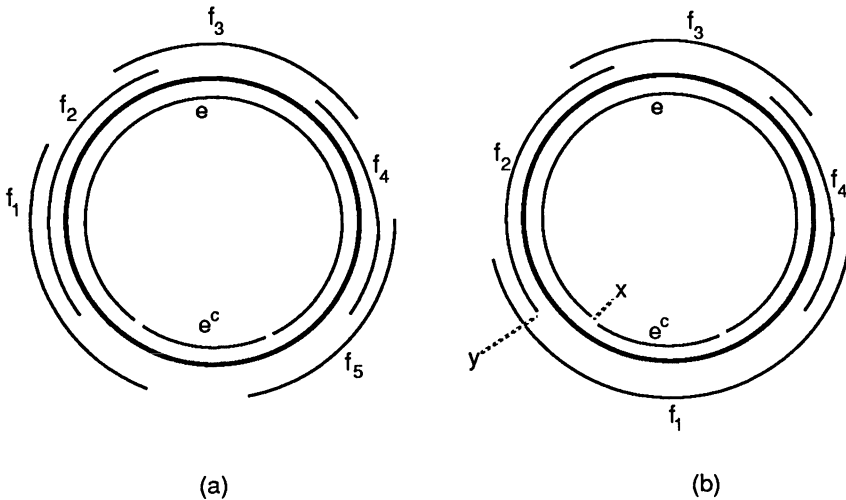


FIG. 6. Illustrating the two cases of the proof of Lemma 11.

*Case 1.*  $e^c$ , the complement of  $e$ , covers at least one endpoint of at least one of the  $f_i$ 's (see Fig. 6(a)). For every  $1 \leq i < s$ , the fact that  $f_i$  and  $f_{i+1}$  overlap without containment implies that we have either  $scc(f_i) = scc(f_{i+1})$ , or  $scc(f_i)$  and  $scc(f_{i+1})$  are directly reachable from each other. Therefore the  $scc(f_i)$ 's are reachable from one another and from  $scc(e)$ . Now, in  $G(T) - e + e^c$ ,  $scc(e^c)$  is still reachable from  $scc_1$ , and every  $scc(f_i)$  is still reachable from  $scc(e^c)$  (because  $e^c$  contains an endpoint of

some  $f_i$ ). Since the union of the  $scc(f_i)$ 's with  $scc(e^c)$  covers the circumference, it follows (by Lemma 8) that the graph  $G(T) - e + e^c$  is transportable (from the start vertex). This contradicts the optimality of  $T$ .

*Case 2.*  $e^c$  does not cover any endpoint of any  $f_i$  (see Fig. 6(b)). Note that this implies that some  $f_i$ , say  $f_1$ , contains  $e^c$ . Let  $l$  be the shortest distance between any endpoint of  $e$  (say,  $x$ ) and any endpoint of any  $f_i$  (say, endpoint  $y$  of  $f_\alpha$ ). Now, in  $G(T)$ , simultaneously shorten  $e$  and add two antiparallel augmenting arcs of length  $l$  each between  $x$  and  $y$ . In the resulting graph, the two additional augmenting edges that were added make  $scc(e^c) = scc(f_\alpha)$ , and therefore every  $scc(f_i)$  is reachable from  $scc_1$ . Since the union of the  $scc(f_i)$ 's covers the circle, Lemma 8 implies that the new graph is transportable. The decrease in cost with respect to the original  $G(T)$  is given by

$$|e| - |e^c| - 2 \cdot l = 1 - 2 \cdot (|e^c| + l) > 1 - 2|f_1| \geq 0$$

(note that  $|e^c| + l < |f_1| \leq \frac{1}{2}$ , and recall that the circle's circumference is unity). Thus the new transportable graph is shorter than  $G(T)$ , a contradiction.  $\square$

**COROLLARY 3.** *Let  $T$  be an optimal transportation, and let  $\psi(T)$  denote the flux of  $T$ . If  $G(T)$  contains a major edge, then  $|\psi(T)| = 1$ .*

*Proof.* The interval covered by  $e$  and by no other edge, in Lemma 11, has flux value of +1 or -1.  $\square$

We henceforth use  $D_0$  to refer to the graph which consists of all  $m$  input edges, drawn so that each of them is minor.

**LEMMA 12.** *If  $D_0$  covers the circumference, then no optimal transportation can contain a major edge.*

*Proof.* Let  $T$  be an optimal transportation and let  $e$  be a major edge of  $G(T)$ . Since  $e$  is major, its complement  $e^c$  is in  $D_0$ . Since  $D_0$  covers the circumference,  $D_0 - e^c$  covers  $e$ . Since  $G(T) - e$  contains  $D_0 - e^c$ ,  $G(T) - e$  covers the region covered by  $e$ . This contradicts Lemma 11.  $\square$

If two edges of  $D_0$  overlap without either of them containing the other, then in every degree-balanced augmentation of  $D_0$ , these two edges either belong to the same  $scc$  or to two  $scc$ 's that are directly reachable from each other. Based on this observation let us define a relation,  $\approx$ , between any two edges of  $D_0$ , as follows:

- (a) For any two input edges  $e_1$  and  $e_2$ ,  $e_1 \approx e_2$  if and only if either these two edges share a common endpoint, or they overlap but neither one of them contains the other.
- (b) Transitively close the relation  $\approx$ .

Note that  $\approx$  is an equivalence (eq.) relation, and, in addition, no two eq. classes of  $\approx$  have a vertex in common. Also note that in any degree-balanced augmentation of  $D_0$ , two input edges in the same eq. class of  $\approx$  belong to  $scc$ 's that are reachable from each other.

Define an ordering  $<$  among the eq. classes of  $\approx$  as follows: If  $C_i$  and  $C_j$  are any two distinct eq. classes of  $\approx$ , then  $C_i < C_j$  if and only if some edge of  $C_j$  covers all the vertices of  $C_i$  (and hence no edge of  $C_i$  covers any vertex of  $C_j$ ). Note that  $\approx$  is independent of the drawing of the edges, whereas  $<$  does depend on it. Based on this ordering we can draw a forest of trees  $F$  whose nodes are the eq. classes of  $\approx$  (the parent of  $C_i$  is the "smallest" class above it according to the  $<$  relation). An example of such a forest is shown in Fig. 7. Let  $\tau_1, \dots, \tau_k$  be the trees of  $F$ , listed in clockwise cyclic order and such that  $\tau_1$  contains the start vertex. Let  $root(\tau_i)$  denote the eq. class at the root of tree  $\tau_i$ . Let  $g_1, \dots, g_k$  be the lengths of the gaps that separate, on the circumference, the regions covered by the  $\tau_i$ 's;  $g_i$  is the gap between  $\tau_i$  and  $\tau_{i+1}$  (in

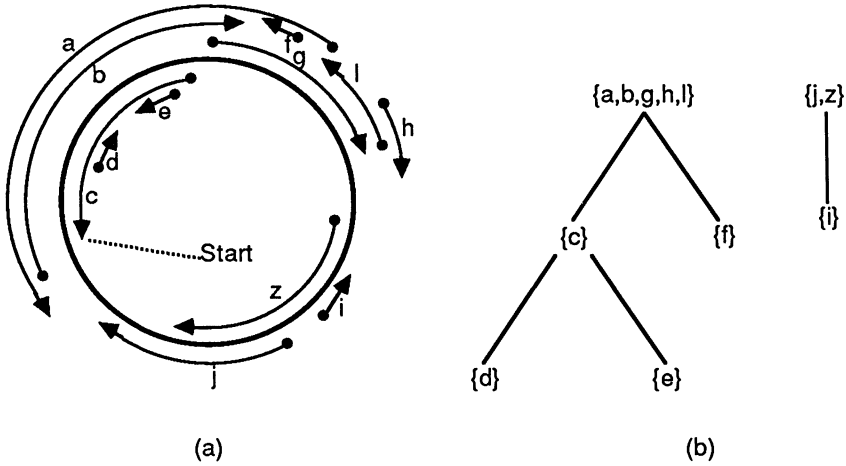


FIG. 7. The forest (b) corresponds to the input edges shown in (a).

Fig. 7,  $g_1$  is the arclength from the head of edge  $h$  to the tail of edge  $z$ , and  $g_2$  is the arclength from the head of edge  $j$  to the head of edge  $a$ .

LEMMA 13. *The eq. classes of  $\approx$  and the forest  $F$  of eq. classes can be computed in  $O(m)$  time.*

*Proof.* For the proof see Appendix A.  $\square$

Let  $G$  be any degree-balanced augmentation of  $D_0$ . (Note that  $G$  is not necessarily transportable and, since it is degree-balanced, its scc's are the same as the connected components of the undirected version of it.) We say that eq. classes  $C_i$  and  $C_j$  of  $\approx$  are *directly linked* in  $G$  if and only if at least one of the scc's of  $G$  contains vertices from both  $C_i$  and  $C_j$ ; they are *linked* if and only if there is a sequence of eq. classes beginning with  $C_i$  and ending with  $C_j$  such that every two eq. classes in the sequence are linked.

Let  $DB(\psi)$  be the graph corresponding to the minimum degree-balanced augmentation of  $D_0$  that results in flux  $\psi$ ; note that  $DB(\psi)$  is unique but need not be transportable. Figure 8(a) shows  $DB(0)$  for the  $D_0$  of Fig. 7. Let  $db(\psi)$  be the total length of the augmenting edges in  $DB(\psi)$  (i.e., edges in  $DB(\psi) - D_0$ ). By Lemma 4, it takes  $O(m)$  time to compute all the  $db(\psi)$  values,  $-m - 1 \leq \psi \leq m + 1$ .

Let  $C_{start}$  be the eq. class containing the start vertex (recall that  $\tau_1$  is the tree of  $F$  that contains  $C_{start}$ ). Let  $LR(\psi)$  be a minimum augmentation of  $DB(\psi)$  that makes  $C_{start}$  linked to root ( $\tau_1$ ) while keeping the flux equal to  $\psi$ . Hence  $LR(\psi)$  consists of  $DB(\psi)$  plus some augmenting pairs of antiparallel edges (in Fig. 8(b) there is one such pair, marked "in  $LR(0)$ "). Let  $lr(\psi)$  be the total length of the augmenting edges in  $LR(\psi)$  but not in  $DB(\psi)$ .

LEMMA 14. *The  $lr(\psi)$ 's, for  $-m - 1 \leq \psi \leq m + 1$ , can all be computed in  $O(m)$  time. For any given flux value  $\psi_0$ ,  $LR(\psi_0)$  can be computed in  $O(m)$  time.*

*Proof.* For the proof see Appendix B.  $\square$

In the next two lemmas, we show how to compute the optimum among all transportations in which no edge is major.

Let  $OPT_{minor}(\psi)$  be the graph corresponding to a transportation that is optimal among all transportations of flux  $\psi$  and that do not have any major edge. If  $LR(\psi)$  is transportable from the start vertex, then  $OPT_{minor}(\psi) = LR(\psi)$ . Otherwise, pairs of antiparallel edges are needed across some of the gaps (intervals not covered by  $D_0$ )

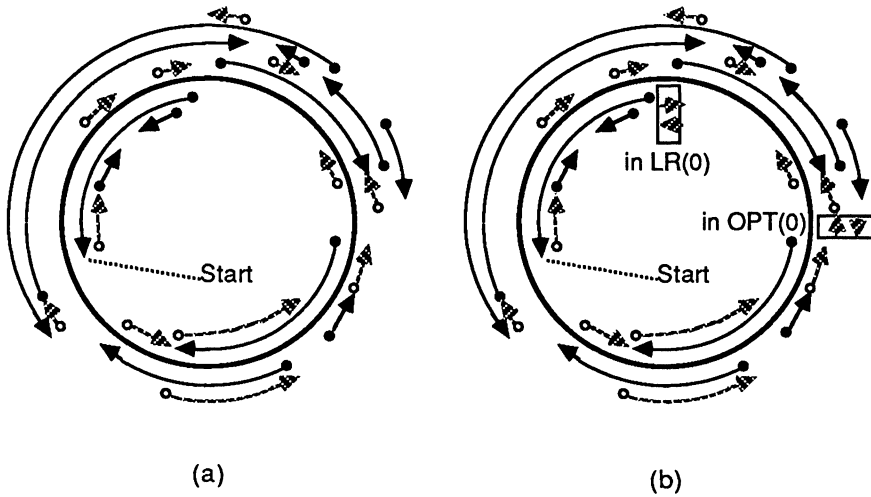


FIG. 8. Illustrating  $DB(0)$ ,  $LR(0)$ , and  $OPT(0)$  for the input edges of Fig. 7.

in order to turn  $LR(\psi)$  into  $OPT_{minor}(\psi)$ . In Fig. 8(b) there is one such antiparallel pair, and each of the two edges in it has length  $g_1$ . (Recall that  $g_1, \dots, g_k$  are the lengths of the gaps separating the  $\tau_i$ 's, listed in clockwise cyclic order, and such that  $\tau_1$  contains the start vertex.)

LEMMA 15. Let  $\hat{g}(\psi)$  be the total length of the augmenting pairs of antiparallel edges in  $OPT_{minor}(\psi) - LR(\psi)$ . If  $\psi \neq 0$  then  $\hat{g}(\psi) = 0$ ; otherwise  $\hat{g}(0) = 2 \cdot (\sum_{i=1}^k g_i - \max_i g_i)$ .

Proof.  $\hat{g}(\psi)$  is the length of the additional edges needed for linking all the root ( $\tau_i$ )'s together at flux  $\psi$ . No additional edges are needed if  $\psi \neq 0$  because then every root ( $\tau_i$ ) is already linked to root ( $\tau_{i+1}$ ) in  $LR(\psi)$ . If  $\psi = 0$  then we link the root ( $\tau_i$ )'s by adding an antiparallel pair across every gap except the longest.  $\square$

The above lemma immediately implies the following.

COROLLARY 4. The pairs of antiparallel augmenting edges in  $OPT_{minor}(\psi) - LR(\psi)$  can be computed in  $O(m)$  time.

LEMMA 16. It is possible to compute, in  $O(m)$  time, a graph  $G(T_0)$  for some transportation  $T_0$  which is optimal among all transportations none of whose edges is major.

Proof. Note that such a graph is simply an  $OPT_{minor}(\psi)$  of minimum total length. The total length of such a graph is equal to (length of edges of  $D_0$ ) +  $\min_{\psi} \{db(\psi) + lr(\psi) + \hat{g}(\psi)\}$ . Therefore Lemmas 4, 14, and 15 immediately imply that we can find a flux value  $\psi_0$  such that the length of  $OPT_{minor}(\psi)$  is minimum for  $\psi = \psi_0$ . We now show that, once we know  $\psi_0$ ,  $OPT_{minor}(\psi_0)$  itself can be computed in  $O(m)$  time. It is trivial to obtain  $DB(\psi_0)$  in  $O(m)$  time. By Lemma 14,  $LR(\psi_0)$  can be obtained from  $DB(\psi_0)$  in  $O(m)$  time. By Corollary 4,  $OPT_{minor}(\psi_0)$  can be obtained from  $LR(\psi_0)$  in  $O(m)$  time.  $\square$

We now consider transportations that have exactly one major edge.

LEMMA 17. Let  $E$  be any set of edges on the circle, exactly one of which is major (call it  $e$ ). Let  $G$  be any degree-balanced augmentation of  $E$ . If  $E$  covers the circumference, then every scc of  $G$  is reachable from the scc that contains  $e$ .

Proof. Let  $scc(e)$  denote the scc of  $G$  containing edge  $e$ . Since  $E$  covers the circumference, there exists in  $E$  a sequence of edges  $f_1, \dots, f_s$  such that  $f_1 \cup \dots \cup f_s$  covers  $e^c$ , and every  $f_i$  contains an endpoint of  $f_{i+1}$ ,  $1 \leq i < s$ . We assume that the

sequence has smallest number of elements ( $= s$ ), and hence none of  $f_i$  and  $f_{i+1}$  contains the other. See Fig. 9, and note that because  $e$  is the only major edge of  $E$ , it must contain at least one endpoint of one of the  $f_i$ 's. Therefore at least one  $\text{scc}(f_i)$  is reachable from  $\text{scc}(e)$ . For every  $1 \leq i < s$ , the fact that  $f_i$  and  $f_{i+1}$  overlap without containment implies that they belong to  $\text{scc}$ 's that are reachable from one another. Therefore every  $\text{scc}(f_i)$  is reachable from  $\text{scc}(e)$ , and hence (by Lemma 8) any  $\text{scc}$  of  $G$  is also reachable from  $\text{scc}(e)$ .  $\square$

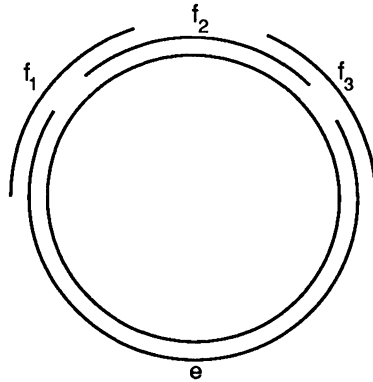


FIG. 9. Illustrating the proof of Lemma 17.

Recall that we have already stated (in Lemma 10) that in an optimal augmentation at most one edge is major. The next lemma refines this statement.

LEMMA 18. *There exists an optimal transportation  $T$  such that, if  $e \in G(T)$  is major, then its complement  $e^c \in D_0$  is in a root eq. class of  $\approx$  and is the longest edge in that class.*

*Proof.* Let there exist an optimal transportation  $T$  in which edge  $e$  is major, and hence its complement  $e^c$  is in  $D_0$ . Note that  $|e| = 1 - |e^c|$ , and  $|e^c| < \frac{1}{2}$ . Let  $e^c$  be in eq. class  $C$  of tree  $\tau_j$  (recall that the eq. classes as well as the  $\tau_i$ 's are defined using  $D_0$ , where all edges are minor).

By Lemma 12,  $D_0$  does not cover the circumference. Lemma 11 implies that  $e$  is the only input edge that covers the  $k$  gaps not covered by  $D_0$ .

Let  $G' = G(T) - e + e^c$ , and note that  $G'$  is degree-balanced and contains  $D_0$ . No  $\text{scc}$  of  $G'$  covers the circumference, because otherwise  $G'$  is transportable (from the start vertex), contradicting the optimality of  $G(T)$ . Therefore every  $\text{scc}$  of  $G'$  has an individual flux of zero, which implies the following:

- (\*) If an interval is covered by an  $\text{scc}$  of  $G'$ , then it is covered by at least two edges of that  $\text{scc}$ .

Now we prove that  $e^c$  is longest in  $C$ . Suppose to the contrary that edge  $f \in C$  is longer than  $e^c$ . Consider the  $\text{scc}$  of  $G'$  that contains  $e^c$  (call it  $\text{scc}(e^c)$ ). In  $G'$ ,  $\text{scc}(e^c)$  and  $\text{scc}(f)$  are mutually reachable from one another because  $G'$  contains  $D_0$  and, in  $D_0$ ,  $e^c$  and  $f$  are in the same eq. class. Now, since  $T$  is a transportation, the  $\text{scc}$  of  $G(T)$  that contains  $e$  is reachable from the  $\text{scc}$  of the start vertex, and therefore in  $G'$ ,  $\text{scc}(e^c)$  is reachable from the  $\text{scc}$  of the start vertex. Therefore in  $G'$ ,  $\text{scc}(f)$  is reachable from the  $\text{scc}$  of the start vertex. Consequently, we have:

- (\*\*) In  $G' - f + f^c$ ,  $\text{scc}(f^c)$  is reachable from the  $\text{scc}$  of the start vertex.

Now, (\*) implies that in  $G'$ , every interval covered by  $f$  is also covered by at least one edge of  $\text{scc}(f)$  other than  $f$ . Therefore  $\text{scc}(f) - f + f^c$  covers the circumference,



and hence by Lemma 17 every scc of  $G' - f + f^c$  is reachable from  $\text{scc}(f^c)$ . This and (\*\*\*) together imply that in  $G' - f + f^c$ , every scc is reachable from the scc of the start vertex. Thus the graph obtained from  $G(T)$  by shortening  $e$  and simultaneously lengthening  $f$  is also transportable. This contradicts the optimality of  $T$ . Thus  $e^c$  must be longest in its eq. class.

Now we prove that there is always an optimal  $T$  in which  $C$  is root in its eq. class, i.e.,  $C = \text{root}(\tau_j)$ . Suppose that  $C \neq \text{root}(\tau_j)$ . Then the parent of  $C$  in  $\tau_j$  surely contains an edge  $f$  which is longer than  $e^c$  and covers it entirely. Let  $\hat{G} = G(T) - e + e^c - f + f^c$ . Let CYCLES be the set of augmenting edges defined as follows: across every interval covered by  $f$  but not by  $e^c$ , add two antiparallel edges. Thus the length of CYCLES is  $2 \cdot (|f| - |e^c|)$ . Simple arithmetic shows that adding CYCLES to  $\hat{G}$  results in a graph of total length no more than that of  $G(T)$ . We now show that  $\hat{G} + \text{CYCLES}$  is transportable. First note that  $\hat{G} + \text{CYCLES}$  has degree-balance. Now observe that, because of the presence of CYCLES,  $e^c$  and  $f^c$  are in the same scc of  $\hat{G} + \text{CYCLES}$  (call it  $\text{scc}_x$ ). Furthermore, since  $T$  is a transportation, the scc of  $G(T)$  that contains  $e$  is reachable from the scc of the start vertex, and therefore in  $\hat{G} + \text{CYCLES}$ ,  $\text{scc}_x$  is reachable from the scc of the start vertex. This and the fact that  $\text{scc}_x$  covers the circle implies that  $\hat{G} + \text{CYCLES}$  is transportable. Since  $\hat{G} + \text{CYCLES}$  is transportable and has length no more than that of  $G(T)$ , we have obtained from  $G(T)$  another transportable graph of optimal length, one in which the complement of the major edge is in an eq. class that is one level higher in  $\tau_j$ . We can repeatedly do this until we end up with an optimal transportation whose major edge's complement belongs to  $\text{root}(\tau_j)$ .  $\square$

LEMMA 19. *It is possible to compute, in  $O(m)$  time, a graph  $G(T)$  for some optimal transportation  $T$ .*

*Proof.* Use Lemma 16 to compute  $G(T_0)$  and let  $\text{Cost}_0$  be its total length. Use Lemma 13 to compute  $F$ : if  $F$  has one tree only then return  $G(T_0)$ . Now, suppose that  $F$  has more than one tree, i.e.,  $k > 1$ . Let  $\text{Cost}_1$  be the length of a transportation  $T_1$  having exactly one major edge in it, and which is optimal among all transportations that have exactly one major edge. The optimal transportation  $T$  will have length  $\min\{\text{Cost}_0, \text{Cost}_1\}$ . If  $\text{Cost}_1 < \text{Cost}_0$  then Lemma 11 tells us that in our search for the value  $\text{Cost}_1$ , we can restrict our attention to transportations  $T_1$  such that there exists at least one interval covered by only the major edge of  $G(T_1)$ . Within this class of transportations, any  $T_1$  will have to be such that  $G(T_1)$  has the following properties where  $e$  denotes the major edge and  $e^c$  its complement:

(i)  $G(T_1) - e + e^c$  has flux equal to zero and thus contains  $\text{DB}(0)$  (which contains  $D_0$ ).

(ii) In  $G(T_1) - e + e^c$ ,  $C_{\text{start}}$  is linked to the root eq. class of its tree ( $\tau_1$ ). Therefore  $G(T_1) - e + e^c$  contains  $\text{LR}(0)$  as well.

(iii) In  $G(T_1) - e + e^c$ , the root eq. class of the tree  $\tau_j$  that contains  $e$  is linked to  $\text{root}(\tau_1)$  (simply because  $T_1$  is a transportation). The augmenting edges that cause  $\text{root}(\tau_j)$  and  $\text{root}(\tau_1)$  to be linked are pairs of antiparallel edges across either all of gaps  $\#1, \dots, \#(j-1)$ , or all of gaps  $\#j, \dots, \#k$  (depending on whichever of  $g_1 + \dots + g_{j-1}$  or  $g_j + \dots + g_k$  is smaller).

Thus  $G(T_1) - e + e^c$  equals  $\text{LR}(0)$  plus the augmenting edges referred to in (iii). The length of  $G(T_1) - e + e^c$  is therefore equal to

$$(\text{length of } D_0) + \text{db}(0) + \text{lr}(0) + 2 \cdot \min\{g_1 + \dots + g_{j-1}, g_j + \dots + g_k\}.$$

The length of  $G(T_1)$  therefore equals (using  $|e| + |e^c| = 1$ )

$$(\dagger) (\text{length of } D_0) + \text{db}(0) + \text{lr}(0) + 2 \cdot \min\{g_1 + \dots + g_{j-1}, g_j + \dots + g_k\} + 1 - 2|e^c|.$$

The first three terms of the sum ( $\dagger$ ) are the same for any such transportation  $T_1$ . Now, the edge  $e$  of  $G(T_1)$  for which the sum of the last two terms of ( $\dagger$ ) is smallest should be the one to follow the major arc. It is trivial to identify this edge in  $O(m)$  time and compute the corresponding sum ( $\dagger$ ): If this sum is smaller than  $\text{Cost}_0$  then this  $G(T_1)$  corresponds to the optimal transportation; otherwise it is  $G(T_0)$ .  $\square$

**LEMMA 20.** *Let  $G$  be transportable from a designated start vertex. A transportation  $T$  such that  $G(T) = G$  can be found in  $O(m)$  time.*

*Proof.* The proof of Lemma 7 suggests an algorithm for obtaining such a transportation. However, we cannot afford to create the graph of the reachability relation among scc's, because such a graph can be dense. Instead, we create a graph  $\text{EQUIV}'$ , defined as  $\text{EQUIV}$  in the proof of Lemma 13 (Appendix A) except that we now use the input edges as they are drawn in  $G$  rather than as they are drawn in  $D_0$  ( $G$  may include a major edge). In addition, we compute the eq. classes of a relation  $\approx'$  and forest of eq. classes  $F'$  defined as  $\approx$  and  $F$  were before, except that we now use the input edges as they are in  $G$  rather than in  $D_0$ . For each eq. class  $C$  that is parent in  $F'$  of class  $C'$ , we arbitrarily select an edge  $e$  in  $C$  and an edge  $f$  in  $C'$  such that  $e$  properly contains  $f$  (at least one such pair  $e, f$  exists); we call edge  $e$  the *parent* of  $f$ , and we call  $f$  a *child* of  $e$ . Note that the forest  $F'$  induces at most  $n - 1$  such parent-child pairs. For each edge  $e$ , let the list of edges  $\text{ADJ}(e)$  be the union of (i) the children of  $e$  induced by  $F'$ , and (ii) the adjacency list of  $e$  in the undirected graph  $\text{EQUIV}'$ . We are now ready to describe how to obtain  $T$  such that  $G(T) = G$ . The transportation process resembles a depth-first search of the scc's, begun at  $\text{scc}_1$ : First we mark all scc's of  $G$  as being "new," then we mark  $\text{scc}_1$  as being "old" and begin transporting  $\text{scc}_1$  from the start vertex of  $G$  (recall that any scc can be individually transported using any vertex in it as start and finish). Whenever we are transporting an input edge  $e$  of the scc currently being transported, we go through the list  $\text{ADJ}(e)$ : for each  $f \in \text{ADJ}(e)$  that is in a "new" scc, we mark the scc of  $f$  as being "old," interrupt the transportation of the scc of  $e$ , and recursively transport the scc of  $f$  using as start (and hence finish) vertex an endpoint of  $f$  covered by  $e$ . It is trivial to implement this procedure in  $O(m)$  time. Correctness follows from the facts that (i)  $\text{EQUIV}'$ , even though it is sparse, captures all the "overlap without containment" relationships between pairs of input edges, and (ii) the parent-child pairs induced by  $F'$  capture enough of the "proper containment" relationships between pairs of edges. More precisely, (i) guarantees that once an edge of eq. class  $C$  is reached by the transportation, eventually every edge of that class  $C$  will be transported. On the other hand, (ii) guarantees that once an edge of an eq. class  $C$  is reached by the transportation, eventually every eq. class in the subtree of  $C$  in  $F'$  will be transported.  $\square$

The last two lemmas imply Theorem 2, which is the main result of this section.

**4. Concluding remarks.** It is easy to see that our solutions to the angular motion problem (with or without drops) also work in the presence of obstacles. A preprocessing step computes the visibility polygon from the fixed pivot point of the robot arm (of course all  $n$  stations must be visible from the pivot, since an invisible station is unreachable by the robot arm). The robot arm must remain within the visibility polygon while performing the transportation. While this does not affect the rotational distance function, the telescoping distance function has to be modified appropriately because the robot arm may have to be drawn in so as to clear an obstacle.

It would be interesting to investigate the with-drops circular track problem when the gripper can simultaneously hold  $c$  objects, where  $c$  is a constant larger than one. We conjecture that Lemma 10 generalizes to that case, i.e., no optimal transportation

can transport more than  $c$  objects along the major arc. A special case of this problem for a linear track was treated in [K].

**Appendix A.** This Appendix proves Lemma 13. Computing  $F$  in linear time when we know the eq. classes is easy and this construction is omitted. We give an  $O(m)$  time algorithm for computing the eq. classes of  $\approx$ . For the purpose of this computation all the edges in  $D_0$  can be considered undirected. An edge covers the circular region going clockwise from its *beginning* to its *end*. For an edge  $e \in D_0$ , let  $CW(e)$  (respectively,  $CCW(e)$ ) be the set of edges of  $D_0$  whose beginning (respectively, end) is in the region covered by  $e$  and whose end (respectively, beginning) is in the region not covered by  $e$ . Note that  $f \in CW(e)$  if and only if  $e \in CCW(f)$ . The *clockwise* (respectively, *counterclockwise*) *successor* of  $e$  is the edge of  $CW(e)$  (respectively,  $CCW(e)$ ) whose beginning (respectively, end) is encountered first by a clockwise (respectively, counterclockwise) sweep starting at  $e$ 's beginning (respectively, end). In Fig. 10(a),  $CW(e) = \{c, d\}$ ,  $CCW(e) = \{a, b\}$ , the clockwise successor of  $e$  is  $d$  and its counterclockwise successor is  $a$ .

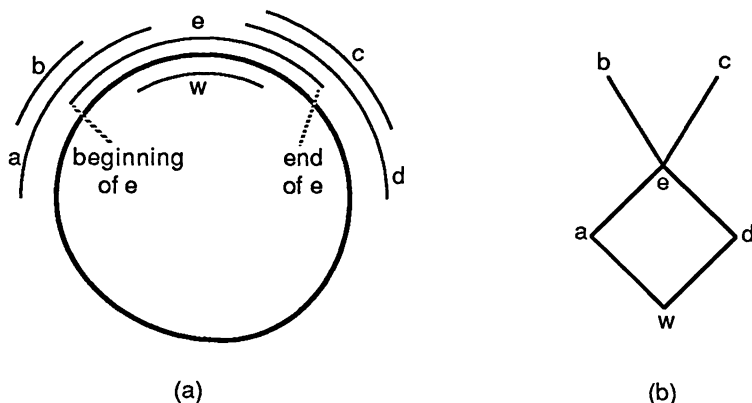


FIG. 10. Illustrating EQUIV.

The clockwise and counterclockwise successors of every edge of  $D_0$  can easily be computed in  $O(m)$  time. Assume this has already been done. Now create, in  $O(m)$  time, the undirected graph EQUIV whose vertex set is  $D_0$  and such that  $\{e, f\}$  is an edge in it if and only if one of  $e$  and  $f$  is (clockwise or counterclockwise) successor of the other. Figure 10(b) shows the graph EQUIV corresponding to Fig. 10(a). Obviously, EQUIV has at most  $2m$  edges, since every  $e \in D_0$  has at most two successors (one clockwise, one counterclockwise). Hence the connected components of EQUIV can be computed in  $O(m)$  time. Thus the lemma will follow immediately when we establish that the connected components of EQUIV are the equivalence classes of  $\approx$ . To prove this, it suffices to show that if any two edges  $e$  and  $f$  overlap without containment (i.e., without either one of them properly containing the other), then there is a path between  $e$  and  $f$  in EQUIV. If edges  $e$  and  $f$  overlap without containment, then we define the *overlap number* of the pair  $\{e, f\}$  to be the number of stations covered by both  $e$  and  $f$ , not counting the endpoints of  $e$  and  $f$ . In Fig. 10(a), the overlap number of  $\{e, d\}$  is 2, that of  $\{a, w\}$  is 0, and that of  $\{e, w\}$  is undefined because  $e$  properly contains  $w$ . We prove the desired result by contradiction: suppose there exist

pairs of edges that overlap without containment and have no path between them in EQUIV. Among all such pairs, choose  $\{e, f\}$  to have maximum overlap number. Without loss of generality, assume  $f \in CW(e)$ . Let  $g$  be the clockwise successor of  $e$ , and let  $h$  be the counterclockwise successor of  $f$  (see Fig. 11). Since  $\{e, g\}$  and  $\{f, h\}$  are edges in EQUIV, and since  $e$  and  $f$  are in different connected components of EQUIV, we must have  $g \neq f$  and  $h \neq e$ . Note that, as shown in Fig. 11, the end of  $g$  must occur after that of  $f$  in the clockwise direction, because otherwise the overlap number of  $\{g, f\}$  would exceed the overlap number of  $\{e, f\}$ , a contradiction. Similarly, the beginning of  $h$  comes before that of  $e$ , since otherwise the overlap number of  $\{e, h\}$  would exceed the overlap number of  $\{e, f\}$ , a contradiction. But then, the overlap number of  $\{g, h\}$  exceeds the overlap number of  $\{e, f\}$ , a contradiction. This completes the proof of Lemma 13.

**Appendix B.** This appendix proves Lemma 14. For every eq. class  $C \in \tau_1$ , let  $L(C)$  (respectively,  $R(C)$ ) be the vertex of  $C$  such that a clockwise sweep of the region covered by  $C$  starts at  $L(C)$  (respectively, ends at  $R(C)$ ). In Fig. 7  $L(\{c\})$  is the head of edge  $c$ ,  $L(\{j, z\})$  is the tail of edge  $z$ . If  $C$  is not root  $(\tau_1)$ , let the *left* (respectively, *right*) neighbor of  $C$  be the first vertex of  $\tau_1 - C$  that is encountered by a counterclockwise (respectively, clockwise) sweep begun at  $L(C)$  (respectively,  $R(C)$ ). In Fig. 7, the left neighbor of eq. class  $\{c\}$  is the tail of edge  $b$ , its right neighbor is the tail of edge  $g$ . We use  $LN(C)$  and  $RN(C)$  to denote the left and right neighbors of  $C$ , respectively. Note that  $LN(C) = L(C) - 1$  and  $RN(C) = R(C) + 1$ . Therefore we can talk about the *intervals*  $(LN(C), L(C))$  and  $(R(C), RN(C))$ . Let the eq. class containing any vertex  $x$  be denoted by  $Class(x)$ . Note that  $Class(LN(C))$  (respectively,  $Class(RN(C))$ ) is either the left (respectively, right) sibling of  $C$  in  $\tau_1$ , or the parent of  $C$  in  $\tau_1$ . Also note that  $Class(LN(C)) = Class(RN(C))$  is possible (if both are the parent of  $C$ ). Now, imagine starting at  $C_{start}$  and repeatedly applying the function  $Class(LN(\cdot))$  until root  $(\tau_1)$  is reached. This defines a sequence  $S_{left}$  of eq. classes from  $C_{start}$  to root  $(\tau_1)$ . Starting at  $C_{start}$  and repeatedly applying the function  $Class(RN(\cdot))$  similarly defines a sequence of eq. classes  $S_{right}$ . Let us draw a directed graph  $Q$  to depict what  $S_{left}$  and  $S_{right}$  might look like; i.e., the vertices of  $Q$  are the eq. classes of  $S_{left} \cup S_{right}$  and  $Q$  has an edge from  $C_i$  to  $C_j$  if and only if  $C_j$  immediately follows  $C_i$  in  $S_{left}$  or in  $S_{right}$ . Figure 12 shows such a graph  $Q$ . Note that the two paths corresponding to  $S_{left}$  and  $S_{right}$  are not necessarily vertex-disjoint and may

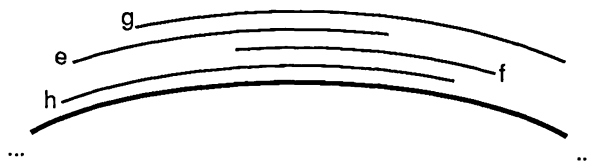


FIG. 11. Illustrating clockwise and counterclockwise successors.

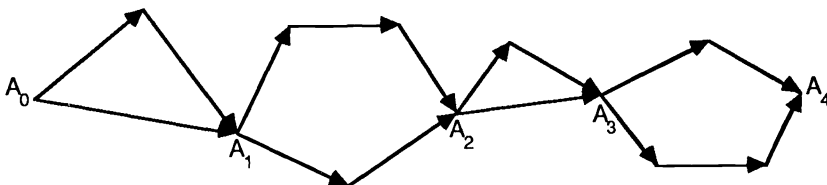


FIG. 12. A typical graph  $Q$ .

come together at *articulation points* more than once. Let these articulation points be  $A_1, A_2, \dots, A_t$ , listed by increasing distance from  $C_{start}$  in  $Q$ . Let  $A_0 = C_{start}$ ,  $A_{t+1} = \text{root}(\tau_1)$ .

Let the interval that *corresponds* to the pair  $\{C, \text{Class}(\text{LN}(C))\}$  be  $(\text{LN}(C), L(C))$ , and let the interval that corresponds to the pair  $\{C, \text{Class}(\text{RN}(C))\}$  be  $(R(C), \text{RN}(C))$ . Now, let  $Q(\psi)$  be the *undirected* graph obtained by removing from the undirected version of  $Q$  all the edges  $\{C, \text{Class}(\text{LN}(C))\}$  and  $\{C, \text{Class}(\text{RN}(C))\}$  whose corresponding intervals are not covered by any augmenting edge of  $\text{DB}(\psi)$ . Note that if  $Q(\psi)$  does not contain a path between  $A_0$  and  $A_{t+1}$ , then antiparallel pairs need to be added to  $\text{DB}(\psi)$  to turn it into  $\text{LR}(\psi)$ ; we next investigate which such antiparallel pairs should be added (the total length of these pairs is  $\text{lr}(\psi)$ ). Let  $E(\psi)$  be a subset of edges in the undirected version of  $Q$  such that adding these edges to  $Q(\psi)$  causes a path to exist between  $A_0$  to  $A_{t+1}$ ; choose  $E(\psi)$  to be such that the sum of the lengths of the intervals corresponding to its elements is minimum. It is not hard to see that  $\text{LR}(\psi)$  is obtained by adding to  $\text{DB}(\psi)$  an antiparallel pair across every interval corresponding to an element of  $E(\psi)$ . Thus  $\text{lr}(\psi)$  is twice the total length of the intervals corresponding to the elements of  $E(\psi)$ ; let us call this the *cost of  $E(\psi)$* . Now our problem is to keep track of the cost of  $E(\psi)$  as  $\psi$  changes from  $-m-1$  to  $m+1$ . Write  $E(\psi)$  as  $E_0(\psi) + E_1(\psi) + \dots + E_t(\psi)$  where  $E_i(\psi)$  is the subset of  $E(\psi)$  that is in the biconnected component of the undirected version of  $Q$  having  $A_i$  and  $A_{i+1}$  as articulation points. Now, as  $\psi$  changes,  $Q(\psi)$  changes as well, but it changes at no more than  $3n$  values of  $\psi$ . Each such change in  $Q(\psi)$  is due to the appearance or disappearance of an augmenting edge of  $\text{DB}(\psi)$  across an interval, and it is trivial to update, in constant time per interval affected, the cost of each  $E_i(\psi)$  affected (once we realize that the portion of  $Q$  between  $A_i$  and  $A_{i+1}$  consists of two disjoint paths, we can easily supply the other details). Thus it is possible to compute a description of the costs of all the  $E_i(\psi)$ 's in  $O(m)$  time. Getting the description of the cost of  $E(\psi)$  takes an additional  $O(m)$  time, which completes the proof of the first part of the lemma.

For a given flux value  $\psi_0$ , computing the actual set  $E(\psi_0)$  (and the intervals corresponding to its elements) is easily done in  $O(m)$  time, in view of the preceding discussion: first we compute  $Q$ , then  $Q(\psi_0)$ , then  $E(\psi_0)$ . This completes the proof of Lemma 14.

**Note added in proof.** Professor G. N. Frederickson pointed out that the  $\alpha(n)$  for the linear track case without drops is, in fact,  $\log(\log^* n)$ .

#### REFERENCES

- [AHU] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [ET] K. P. ESWARAN AND R. E. TARJAN, *Augmentation problems*, SIAM J. Comput., 5 (1976), pp. 653-665.
- [E] S. EVEN, *Graph Algorithms*, Computer Science Press, Potomac, MD, 1979.
- [FT] M. F. FREDMAN AND R. E. TARJAN, *Fibonacci heaps and their uses in improved network optimization algorithms*, in Proc. 25th Annual IEEE Symposium on Foundations of Computer Science, 1984, pp. 338-346.
- [K] R. M. KARP, *Two combinatorial problems associated with external sorting*, in Combinatorial Algorithms, R. Rustin, ed., Courant Computer Science Symposium 9, Algorithmics Press, New York, 1972.
- [P1] C. H. PAPADIMITRIOU, *On the complexity of edge traversing*, J. Assoc. Comput. Mach., 23 (1976), pp. 544-554.
- [P2] ———, *The Euclidean traveling salesman problem is NP-complete*, Theoret. Comput. Sci., 4 (1977), pp. 237-244.

## PROBING CONVEX POLYGONS WITH X-RAYS\*

HERBERT EDELSBRUNNER<sup>†‡</sup> AND STEVEN S. SKIENA<sup>†</sup>

**Abstract.** An X-ray probe through a polygon measures the length of intersection between a line and the polygon. This paper considers the properties of various classes of X-ray probes, and shows how they interact to give finite strategies for completely describing convex  $n$ -gons. It is shown that  $(3n/2)+6$  probes are sufficient to verify a specified  $n$ -gon, while for determining convex polygons  $(3n-1)/2$  X-ray probes are necessary and  $5n+O(1)$  sufficient, with  $3n+O(1)$  sufficient given that a lower bound on the size of the smallest edge of  $P$  is known.

**Key words.** theory of robotics, computational geometry, probing, X-rays, convexity, complexity

**AMS(MOS) subject classifications.** 52A10, 68C25

**1. Introduction.** Identifying and understanding objects from sensory data is a fundamental problem in robotics and computer vision. Such sensors can include imaging devices as well as simpler detectors. Clearly imaging devices provide a tremendous amount of information; however for reasons of economy and robustness, simple sensors are often used. There has been increasing interest [1]–[3] in analyzing the number of measurements which different types of sensors need to determine convex polytopes. This paper introduces another sensor model, the *X-ray probe*, and gives strategies for using it.

Probing polytopes can be viewed as a discrete case of sampling problems encountered in signal processing. The *Nyquist rate* [4] specifies the amount of sampling needed to reconstruct a continuous waveform. Since our objects of interest, convex polygons, have much more structure than continuous waveforms, it is clear that tighter bounds can be obtained. Continuous waveforms generated by probes analogous to our model are used in such medical instrumentation systems as tomography. The techniques used in reconstruction of these waveforms are unrelated to ours, (see [5]–[7] for surveys).

The most studied geometric probe is the *finger probe* [1]–[3], which is a directed line  $l$  and returns the first intersection of  $l$  with polygon  $P$ . Intuitively, it is like moving a finger towards an object and recording where it hits. An *X-ray probe*  $X(P, l)$  is defined to be the length of intersection between polygon  $P$  and the line  $l$ . We assume that we know a point  $O$  within  $P$ , which identifies the general location of  $P$  in the plane. Without such a point to provide a general idea of where  $P$  is, it is not clear how to find  $P$  in a finite number of probes. For convenience we shall assume  $O$  is within the interior of  $P$ .

A collection of X-ray probes through an object provides us with a great deal of information about it but not directly with the coordinates of a point on the surface. Obtaining such absolute information is the difficulty in working with X-ray probes. Figure 1 demonstrates some of these difficulties; the collection of probes provides very little constraint on the location, shape, or number of sides of  $P$ . Another polygon  $P'$  bears little resemblance to  $P$ , but gives identical results for the collection of probes. In fact, the complete set of probes  $X(P, l(O, \theta))$  through  $O$  over all angles  $\theta$  describes two different polygons,  $P$  and  $P$  reflected through  $O$ , denoted as  $-P$ . Thus it is a

\* Received by the editors December 15, 1986; accepted for publication (in revised form) October 29, 1987.

<sup>†</sup> Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801.

<sup>‡</sup> The research of this author was supported by the Amoco Foundation Facility for the Development of Computer Science 1-6-44862.

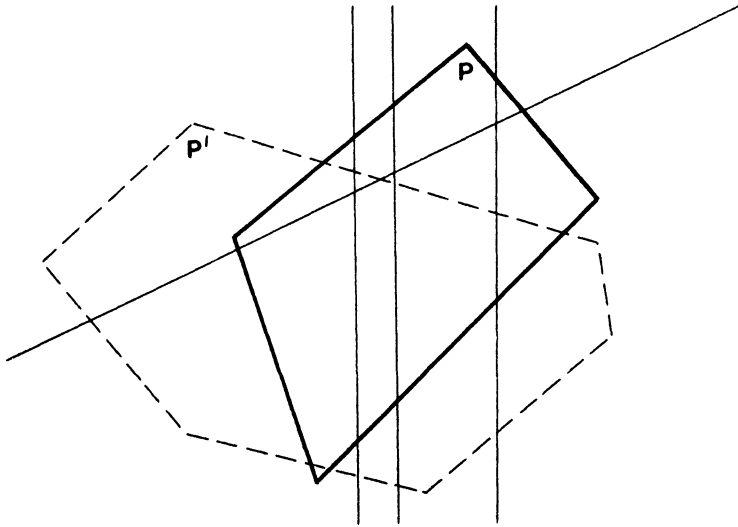


FIG. 1. Different polygons satisfying the same collection of probes.

fundamentally different type of device from the finger probe, although it is not clear which can be considered more powerful.

This paper will give upper and lower bounds on the number of X-ray probes necessary to determine convex  $n$ -gons, as well as for the problem of verifying a conjectured convex polygon. These bounds are based on the powers of different classes of probes, each of which has capabilities and limitations to be examined.

**2. A lower bound for probing.** We can prove a nontrivial lower bound for X-ray probing by a comparison to finger probing.

LEMMA 1. *An X-ray probe  $X(P, l)$  can be simulated by two finger probes.*

*Proof.* Send finger probes down each end of the line, and compute and return the Euclidean distance between these two points.  $\square$

THEOREM 2. *At least  $(3n - 1)/2$  X-ray probes are necessary to determine a convex  $n$ -gon.*

*Proof.* Cole and Yap [1] prove a lower bound of  $3n - 1$  finger probes for determining convex polygons. The result follows from Lemma 1.  $\square$

This bound is probably loose, but the concept of a lower bound on the number of probes is complicated by difficulty in determining exactly what information can be obtained in a constant number of X-ray probes. Since their power comes from collections of probes, a tighter lower bound may be difficult to obtain.

**3. Upper bounds for probing.** To obtain absolute information about  $P$  from X-ray probes, it is necessary to think in terms of groups of probes which work together to determine something about  $P$ . This section considers four different classes of probes, what powers and limitations they possess and how they interact to lead to probing strategies.

**3.1. Origin probes.** The first class of probes are *origin probes*, a set of X-ray probes all aimed through a common point  $O$  within the object. An X-ray probe which hits a convex polygon and avoids its vertices will go through exactly two edges of the object. The largest number of such edge pairs is  $n$ .

LEMMA 3. For any convex  $n$ -gon  $P$  containing  $O$ , there are at most  $n$  distinct pairs of edges  $e_1, e_2$  of  $P$  such that there exists a line which intersects  $e_1, e_2$  and  $O$ .

*Proof.* Consider a line  $l$  through  $O$  which intersects a pair of edges. If we rotate this line clockwise, only when we sweep past a vertex do we intersect a new pair of edges. After rotating past  $n$  vertices and at most  $\pi$  radians, we will return to the original pair. No other distinct pairs can be found, so there exist at most  $n$  opposing edges. Note that there are exactly  $n$  edge pairs unless  $O$  is collinear with two vertices of the  $n$ -gon.  $\square$

We can define a mapping of  $X(P, l(O, \theta))$  to two points  $p_1$  and  $p_2$  on  $l(O, \theta)$  at a distance  $X(P, l(O, \theta))$  from  $O$ , where  $l(O, \theta)$  is the line through  $O$  that encloses an angle of  $\theta$  radians with the  $x$ -axis. By the following result, these points lie on hyperbolas defined by the edges probed through.

LEMMA 4. Let  $l_1: y = m_1x + b_1$  and  $l_2: y = m_2x + b_2$  be two distinct lines and map each angle  $\theta$  to the two points on line  $l(O, \theta)$  at distance  $d$  from  $O$ , where  $d$  is the distance between  $l_1 \cap l(O, \theta)$  and  $l_2 \cap l(O, \theta)$ . Then these points define the hyperbolas

$$y^2 - xy(m_1 + m_2) \pm y(b_1 - b_2) - x^2(m_1m_2) \pm x(m_1b_2 - m_2b_1) = 0.$$

*Proof.* Consider the situation in Fig. 2. The line  $l(O, \theta)$  contains points  $(x, y)$  such that  $y = \tan(\theta)x$ . With  $t = \tan(\theta)$ , we have

$$d = \sqrt{(1 + t^2)(b_2/(t - m_2) - b_1/(t - m_1))^2}.$$

The  $x$ -coordinates of the corresponding points can be found by subtracting the  $x$ -coordinates  $x_1$  and  $x_2$  of the intersections of  $l(O, \theta)$  with  $l_1$  and with  $l_2$ , respectively:

$$x = \pm(x_2 - x_1) = \pm(b_2/(t - m_2) - b_1/(t - m_1)).$$

This can be solved for  $t$  and used with  $y = tx$  to get  $y$  as a function of  $x$ . To obtain a simpler formula, we set  $A = m_1 + m_2$ ,  $B = m_1m_2$ ,  $C = b_1 - b_2$ , and  $D = m_1b_2 - m_2b_1$ . Then we have

$$y = \frac{Ax \mp C + \sqrt{(Ax \mp C)^2 - (4Bx^2 \pm 4Dx)}}{2}$$

and

$$y = \frac{Ax \mp C - \sqrt{(Ax \mp C)^2 - (4Bx^2 \pm 4Dx)}}{2}.$$

After simplification, this reduces to the assertion.  $\square$

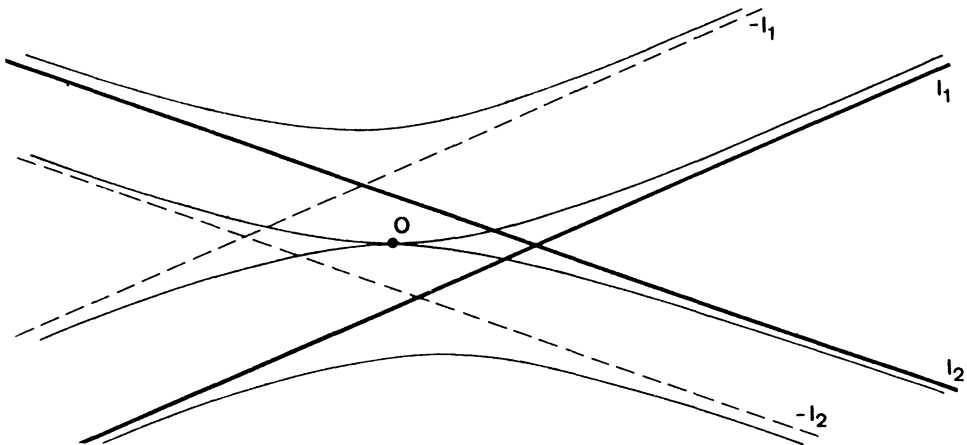


FIG. 2. Hyperbolas associated with two straight lines.



We will use Lemma 4 to determine the equations of the lines that contain edge pairs. If we have some number of origin probes through a common edge pair, then we can determine the hyperbolas through the associated points. From the equations of these hyperbolas, we then deduce the equations of the lines.

It is obvious from Fig. 2 that the hyperbolas which satisfy Lemma 4 have asymptotes  $l_1$  and  $-l_2$  and  $l_2$  and  $-l_1$ . If the angle  $\theta$  is such that  $l(O, \theta)$  intersects  $l_1$  and  $l_2$  on opposite sides of  $O$ , then the two hyperbolas that avoid  $O$  are relevant. This will be the most frequent case in our discussion below. If  $l(O, \theta)$  intersects  $l_1$  and  $l_2$  on the same side of  $O$ , then the two hyperbolas through  $O$  are relevant. This occurs when  $O$  is not between the two edges of the pair considered. In both cases, the two relevant hyperbolas are central reflections of each other.

The hyperbolas that show up in Lemma 4 are defined by four constants  $A = m_1 + m_2$ ,  $B = m_1 m_2$ ,  $C = b_1 - b_2$ , and  $D = m_1 b_2 - m_2 b_1$ . It follows that, in general, four probes through a pair of edges is enough to determine the hyperbolas, and from the hyperbolas the equations of the lines that contain the two edges. Given  $A, B, C, D$ , we have

$$m_1 = \frac{A \pm \sqrt{A^2 - 4B}}{2}, \quad m_2 = \frac{A \mp \sqrt{A^2 - 4B}}{2},$$

$$b_1 = \frac{C}{2} \pm \frac{2D + AC}{2\sqrt{A^2 - 4B}}, \quad \text{and} \quad b_2 = \frac{-C}{2} \pm \frac{2D + AC}{2\sqrt{A^2 - 4B}}.$$

From these equations two limitations on our ability to reconstruct the edges are apparent. First, there is the ambiguity between  $P$  and  $-P$ . More serious is that  $b_1$  and  $b_2$  are undefined when the square root vanishes, that is, when  $m_1 = m_2$ . Thus any probing strategy using origin probes must take special action on parallel edges. Note, however, that there is no ambiguity between  $P$  and  $-P$  for parallel edges, since reflection through  $O$  is equivalent to rotation through  $\pi$  radians.

We can now recognize the structure formed by mapping origin probes to points. Each opposing pair of edges gives rise to their own pair of hyperbolas. Two adjacent hyperbolas meet on the line through  $O$  and each vertex. Thus the probes define the extremes of a "spider web"  $S(P)$  (see Fig. 3) around the object. This permits us to interpret an origin probe for  $P$  to be a finger probe on  $S(P)$ . Both  $P$  and  $-P$  generate the same spider web,  $S(P) = S(-P)$ . By Lemma 3,  $S(P)$  consists of at most  $2n$  pieces of hyperbolas.

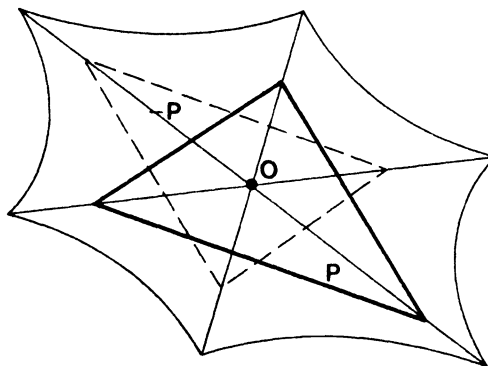


FIG. 3. The "spider web"  $S(P)$  around a convex polygon  $P$ .

However, to make use of these properties of origin probes for a probing strategy we need some means to verify that four or possibly some higher but constant number of origin probes pass through the same pair of edges. With finger probes, it was possible to confirm an edge with three probes, two to define it and one in between to verify it. Unfortunately, a constant number of extra confirmation probes lying on the same hyperbola is not sufficient to verify an edge pair.

LEMMA 5. *There is no constant  $k$  such that  $k$  probes lying on a hyperbola defined above implies that the probes pass through the same pair of edges of  $P$ .*

*Proof.* The proof is by construction. Figure 4 indicates such a polygon. A convex chain of edges can be used to insure that as many points as desired of  $S(P)$  can be made to lie on a hyperbola of the kind defined in Lemma 4. Let one edge  $e$  lie on the line  $y = x + 1$ , and let  $h$  be the line of points  $(x, y)$  that satisfy  $y = -2$ . We construct a curve  $c$  of points  $p$  with the following property: if  $l$  is a line through  $p$  and  $O$ , then  $l$  intersects edge  $e$  in point  $q$  and line  $h$  in point  $r$  such that the distance from  $p$  to  $q$  is the same as the distance from  $O$  to  $r$ .

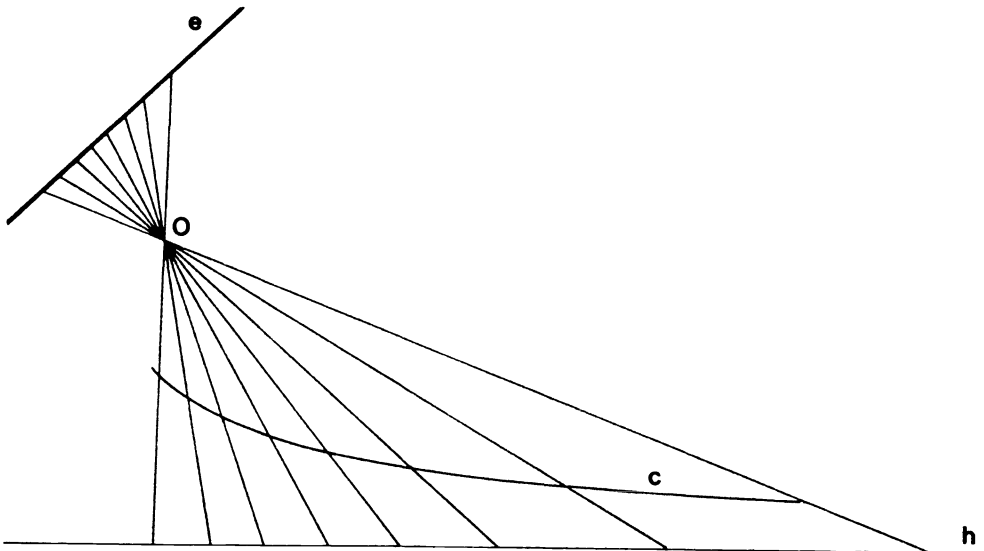


FIG. 4. Counterexample to the notion that  $k$  origin probes mapped onto the same hyperbola must intersect the same edge pair.

It is not hard to see that  $c$  is a piece of a hyperbola whose asymptotes are line  $h$  and the line that contains  $e$ . If we pick  $k$  vertices of  $P$  on curve  $c$ , then all origin probes through these vertices are mapped to points on line  $h$ . Thus, if a collection of probes were made through  $O$  and these vertices, they would all be taken as lying on a parallel pair of edges.

By construction,  $S(P)$  has  $k$  vertices on line  $h$  and the hyperbolic pieces connecting any two consecutive vertices lie all above line  $h$ . Thus, a hyperbola of the kind considered in Lemma 4 can be found that intersects  $S(P)$  at least  $2k$  times if it lies close enough and above  $h$ , where the vertices of  $S(P)$  on  $h$  lie.  $\square$

Verifying edge pairs is the motivation for parallel probes, discussed below.

**3.2. Parallel probes.** *Parallel probes* are a set of X-ray probes aimed with a common angle  $\theta$ . A complete collection of parallel probes for a given angle produce a histogram  $H(P, \theta)$  (see Fig. 5) of the thickness of the obstacle. This is the situation in a medical

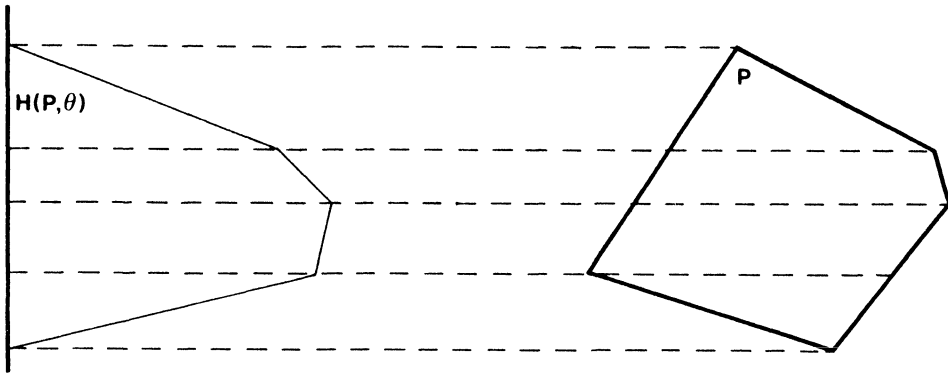


FIG. 5. The “histogram”  $H(P, \theta)$ ,  $\theta = 0$ , of a convex polygon  $P$ .

X-ray photograph. Formally,  $H(P, \theta)$  is obtained as follows. Let  $b$ , the baseline, be normal to the probing direction  $\theta$  and call one of the half-planes defined by  $b$  its *positive side*. Let  $l(b, p)$  be the line normal to  $b$  such that  $p = b \cap l(b, p)$ ; thus,  $l(b, p)$  has angle  $\theta$ . We map the probe  $X(P, l(b, p))$  into the point on  $l(b, p)$  at distance  $X(P, l(b, p))$  from  $b$  that lies in the positive side of  $b$ .  $H(P, \theta)$  is the polygon bounded by  $b$  and the images of all probes with angle  $\theta$ . For a convex  $n$ -gon this histogram will be bounded by up to  $n$  line segments, including the one on  $b$ .  $H(P, \theta)$  is convex for all convex  $P$  over all angles  $\theta$ . Note that an X-ray probe with angle  $\theta$  can be interpreted as a finger probe on  $H(P, \theta)$ . Each vertex of  $H(P, \theta)$  determines a line on which must lie a vertex of  $P$ . Thus they provide a capability for verifying edge pairs.

LEMMA 6. *Three parallel probes are sufficient to verify an edge pair.*

*Proof.* Parallel probes measure the distance between two line segments. Thus three parallel probes are mapped to three collinear points on the boundary of  $H(P, \theta)$  if they intersect the same two segments. No three parallel probes hitting different edge pairs can be mapped to collinear points without violating convexity.  $\square$

There are two apparent weaknesses of parallel probes. First, a finite number of them cannot be guaranteed to locate certain vertices, specifically the extreme vertices that correspond to the vertices of  $H(P, \theta)$  that lie on the baseline  $b$ . Without any bounding information, repeated probes may intersect the same pair of edges. Second, once a vertex is finally located, it is impossible without more information to distinguish whether there are one or two vertices on the line. Convexity restricts the number of collinear vertices to two. This phenomenon relates to the fact that  $H(P, \theta)$  neither determines  $P$  nor the number of its vertices.

It is interesting to consider using an infinite number of parallel probes, as approximated in medical X-ray photographs where all probes perpendicular to the photographic plate are recorded at once. The first weakness vanishes although the second remains. This problem was first posed for convex sets by Hammer in 1963 [8] and has generated a substantial literature [9]–[12]. The power of such a probe which returns  $H(P, \theta)$  for a specified  $\theta$  is evident in that only a constant number of *photograph probes* are needed to determine any convex polygon.

THEOREM 7. *Three X-ray photograph probes are sufficient to determine a convex polygon  $P$ .*

*Proof.* The strategy is as follows. Each photograph probe can be defined by its baseline. For the first two probes, use baselines  $b_1$  and  $b_2$  that are not parallel to each

other. Since according to the observation above, each of these probes defines up to  $n$  lines on which all vertices of  $P$  must lie, two intersecting probes will define up to  $n^2$  points at the intersection of these lines. The vertices of  $P$  must be a subset of these points. The third baseline  $b_3$  is selected such that the line through each of these points perpendicular to  $b_3$  is unique. The  $n$  points on this histogram uniquely identify each of the vertices of  $P$ .  $\square$

Because an X-ray photograph probe captures in one probe a representation of the entire polygon, they may provide a way to extend probing results to nonconvex polygons by eliminating the need for an infinite set of probes to hunt for possible concavities.

**3.3. Determining a boundary point.** Since parallel and origin probes have complementary properties of verification and identification, by combining them we can obtain our first piece of absolute information about the polygon. The idea behind this strategy is to bound a section of the polygon where we know there must be at least one edge pair and to repeatedly send parallel probes to this section until three images on the boundary of the corresponding histogram are collinear. Once we have an edge pair, origin probes can be used to determine the lines containing the two edges.

First, a section of the polygon must be bounded. Sending one horizontal probe through  $O$  gives a thickness  $\lambda$  of  $P$  along that  $x$ -axis. At least one of two vertical probes at distance  $\lambda/2$  to the left and to the right of  $O$  must intersect  $P$  and with a vertical probe  $l_O$  through  $O$  define a section which is a vertical strip such that each vertical probe in this strip intersects  $P$ . First, we discuss certain subtleties of the determined section. If both vertical probes  $l_1$  and  $l_2$  at distance  $\lambda/2$  from  $O$  intersect  $P$ , as in Fig. 6, we cannot be sure for which side of  $O$  the  $x$ -axis is within  $P$ , and thus we cannot be certain about any new interior points. In this case, we choose our section to be either to the left or to the right of  $O$ .

To find a first edge pair, we send vertical probes in the identified section until three points on the boundary of the associated histogram  $H(P, \pi/2)$  are collinear. All points lie on at most  $n-1$  edges of  $H(P, \pi/2)$  and all but one edge contains at most two of the points. It follows that we succeed after at most  $2n-3$  additional vertical probes, since we already have two vertical probes which bounded the probed section

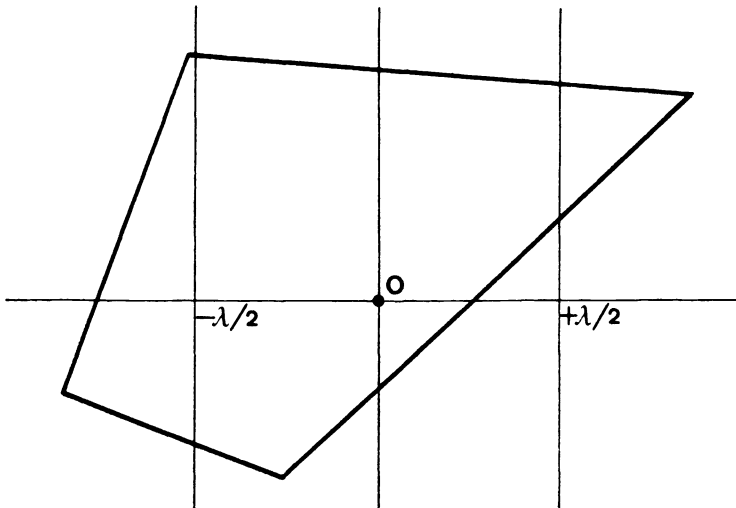


FIG. 6. Identifying a section to parallel probe.

of  $P$ . Once we have vertical probes through an edge pair, four origin probes through a common point, called the origin, can be used to determine the lines that contain the two edges. By picking the origin for this collection of probes to be where the center parallel probe intersects the  $x$ -axis, we need only make three new probes. One final probe can distinguish between the edges of  $P$  and those of  $-P$ .

In order to origin probe the edge pair between the section defined by two vertical probes, we must have an upper bound  $U$  on the distance the edges within this section are from the  $x$ -axis. Without this knowledge, we cannot design origin probes which we can be certain will intersect the boundary of  $P$  within the section, and thus whether the origin probes all hit the same pair of edges. For the situation where exactly one of the two initial vertical probes  $l_1, l_2$  intersects  $P$ , we know that the  $x$ -axis intersects  $P$  throughout the resulting section. In this case we can bound  $U$  because it is clear that the distance of a point from the  $x$ -axis is no larger than the height of the associated histogram at the vertical line through the point.

This argument fails in the case of Fig. 6, since we cannot be certain on which side  $P$  contains the  $x$ -axis. We will use a convexity argument to put a bound on  $U$ . Let us arbitrarily select the section to the right of  $O$ . If this section contains the  $x$ -axis, we know a bound on  $U$ . If not, we know that the other section contains the  $x$ -axis within  $P$  for a distance  $\lambda/2$ . The slope of the upper edge of  $P$  that intersects the  $y$ -axis is greatest if it is the only upper edge of the left section,  $l_1 = l(-\lambda/2, 0, \pi/2)$  intersects  $P$  entirely below the axis and  $l_0 = l(O, \pi/2)$  intersects  $P$  entirely above the axis. By convexity, no edge in the other section can increase faster than this line, which passes through the points  $(0, X(P, l_0))$  and  $(-\lambda/2, -X(P, l_1))$ . Reflecting this situation along the  $x$ -axis bounds the lower edges, and together provides the information we need to origin probe.

A further complication occurs when the edge pair is parallel, meaning  $b_1$  and  $b_2$  are undefined. If another potential edge pair exists in this section, that is, it required more than five parallel probes within the section to locate the parallel edge pair, this nonparallel edge pair can be uncovered by a total of  $2n$  parallel probes, since now two edge pairs can have three probes each.

If the first edge pair is parallel and another edge pair is not known, we must now repeat the sectioning process parallel to the original pair. Clearly, a line  $l_p$  through  $O$  parallel to the first edge pair intersects  $P$  between the edge pair. By a process of binary search, we can enlarge this strip of  $P$  known to lie between the edge pair as much as desired. If  $\delta_1$  is the distance between the two edges, a probe parallel to  $l_p$   $\delta_1/2$  below  $l_p$  widens this strip by  $\delta_1/2$ . If this probe intersects  $P$ , the strip is between  $l_p$  and the last probe. Otherwise, the strip is on the other side of  $l_p$ . Similarly, we can widen this known strip to  $3\delta_1/4$  with a probe parallel to  $l_p$   $\delta_1/4$  on either side of the known strip. We can continue to widen this strip by this method, although for our purposes two of these probes will suffice. This strip will serve to define a section for the next set of probes. Note that there is no reason to actually probe  $l_p$  and that at this point we do not know how long the edges of the first parallel pair are.

Since these probes are parallel with our previously encountered edge pair, they will intersect at least two edges different from the parallel edge pair. We will make five of these, one at each side of the boundary of our  $3\delta_1/4$  region, two more within this region  $\delta_1/2 + \epsilon$  apart for  $0 < \epsilon < \delta_1/4$ , and one between these final two probes. Note that it may be possible to reuse the binary search probes, but only if they intersected  $P$ . If the center three of these probes are not all of the same magnitude, they do not all intersect a parallel pair of edges. Thus with up to  $2n - 3$  more parallel probes we can locate a nonparallel edge pair, which can be origin probed to determine

the edge pair. Unfortunately, as in Fig. 7, the center three probes can instead intersect a parallel edge pair. This parallel edge pair must be greater than  $\delta_1/2$  in length.

If  $\delta_2$  is the distance between the second pair of parallel edges, we can define a strip  $3\delta_2/4$  wide within  $P$  through the binary search strategy using probes parallel to the second edge pair. Two parallel probes within this strip  $\delta_2/2 + \epsilon$  apart for  $0 < \epsilon < \delta_2/4$  can confirm that the first edge pair is greater than  $\delta_2/2$  in length. If this is not the case, we can find a nonparallel edge pair between the offending probe and the appropriate end of the  $3\delta_2/4$  region. Otherwise, the intersection of the two strips defines a rhombus  $Q$  which must lie within the interior of  $P$ . We note that the remainder of  $P$  must lie in strips less than  $\delta_1/2$  and less than  $\delta_2/2$  wide around  $Q$ . Extending these boundaries for each of the two edge pairs surrounds  $Q$  by a skewed grid of eight regions, which together contain all of the edges of  $P$ . None of these regions can contain parallel edges, since  $P$  is convex. Further, no three neighboring regions around  $Q$  including only one corner region contain any parallel edges.

Referring to Fig. 7, it is clear that a probe  $X$  from the upper left corner of the top-central region to the bottom right corner of the right-central region cannot intersect a parallel pair of edges. Because of the size and position of the central region, this probe must intersect  $Q$ , meaning it intersects a nonparallel edge pair. Along with a probe parallel to  $X$  that intersects the upper right-hand corner of  $Q$  this defines a section which can only contain nonparallel edge pairs, and thus can be parallel probed until three are collinear on the histogram. Using the earlier counting argument with  $n - 2$  edges, since the other two parallel edges cannot be within the section, shows finding an edge pair can require up to  $2n - 5$  additional probes.

Finally, one confirmation probe of the nonparallel edge pair will distinguish the edges on  $P$  from the pair on  $-P$ . Note that there is no ambiguity between  $P$  and  $-P$  for the parallel edge pairs.

LEMMA 8. *With restriction to origin and parallel probes,  $2n + 23$  X-ray probes are sufficient to identify the lines that contain the first pair of edges and a point on one of the two edges.*

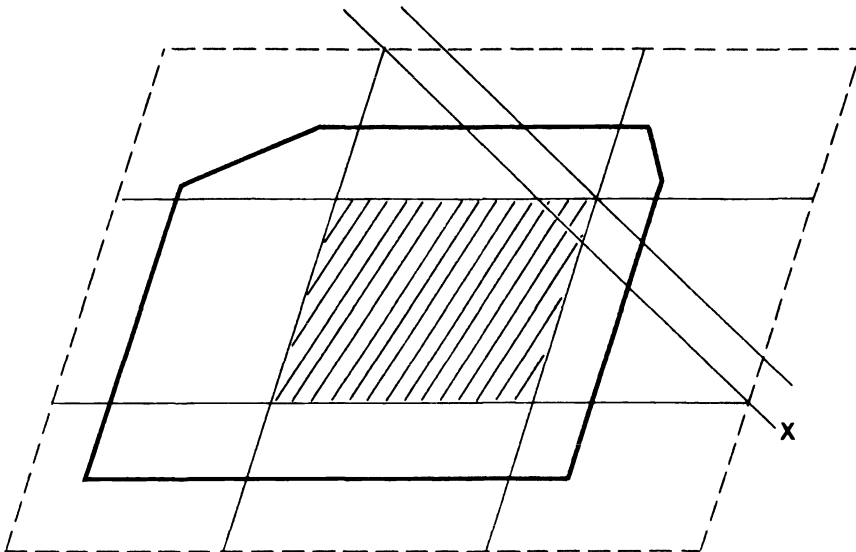


FIG. 7. Handling parallel edge pairs.

*Proof.* The above strategy will identify the lines that contain a pair of edges and a point on one of the two edges. The final accounting of probes is as follows. Four probes were used initially to define a section to probe, at least two of which can serve as parallel probes. Three more parallel probes can identify an edge pair, with the center three probes incident on a parallel pair of edges. Three origin probes found the slopes of these lines. Two parallel probes will widen the strip to  $3\delta_1/4$ . Up to five probes between the two edges will identify that this edge pair is also parallel, and three more are necessary to origin probe it. Two more probes widen the new strip, two more enlarge  $Q$ , and two diagonal probes select a nonparallel section.  $2n - 5$  parallel probes in this section will locate a nonparallel edge pair, the first two of which were the diagonal probes. Three more origin probes find the equations of these lines. Finally, there is the confirmation probe. Thus two edges can be determined in a total of

$$4 + 3 + 3 + 2 + 5 + 3 + 2 + 2 + 2 + (2n - 5 - 2) + 3 + 1 = 2n + 23$$

probes. Any point on these two edges within the appropriate section is on the boundary of  $P$ .  $\square$

A complete probing strategy for  $P$  could perhaps be constructed along these lines by repeating the process for each pair of edges. However, since  $O(n)$  edge pairs can be parallel this would lead to a quadratic number of probes. A simpler strategy can be developed once we know a point on the boundary of  $P$ .

**3.4. Boundary probes.** The power of the finger probe is that it returns a point on the boundary of the polygon. To get a similar effect, we define the *boundary probe*, which relies on the observation that sending an X-ray line probe through a known point on the boundary of a convex polygon identifies another point on the boundary of the polygon. This means that once we have identified the coordinates of any point  $p$  on the boundary of the polygon, any X-ray probe through  $p$  determines another boundary point. If we also are given a boundary point we can formulate our first probing algorithm.

**THEOREM 9.** *With restriction to origin, boundary, and parallel probes,  $5n + 19$  probes suffice to determine a convex  $n$ -gon  $P$ .*

*Proof.* By Lemma 8,  $2n + 23$  probes suffice to find a boundary point and to semi-verify two edges, in the terminology of Cole and Yap [1]. Cole and Yap give a strategy requiring  $3n$  finger probes to determine polygons which can be adapted by using boundary probes in place of finger probes.

Starting from one of the semi-verified edges, we will walk along the polygon, conjecturing vertices based on the intersection of the semi-verified edge and the line defined by the next two known points. Each of the  $n$  vertices will eventually be probed, and each of the  $n - 2$  other edges will have at most two interior points probed, for a total of  $5n + 19$ .  $\square$

Being clever about reinterpreting the parallel probes may reduce the total by  $O(n)$  more probes since once the edge one of them passed through is determined, a probe on an unverified edge can be recorded. No doubt, the additive constant of Lemma 8 can be lowered by more careful arguments.

Note that Cole and Yap's optimal  $3n - 1$  strategy cannot be adapted to X-ray probes since they probe along a semi-verified edge to obtain a vertex, which will not work with X-ray probes unless the location of the other vertex is known.

**3.5. Close probes.** If the measurements we have been using were performed on real sensing devices, there will be some uncertainty with respect to accuracy. Thus for us to completely determine an  $n$ -gon we must know that no edge has length less than

this uncertainty, or else we could never find the edge. Knowing a lower bound on the length of all the edges  $\varepsilon$  gives us extra information about the polygon. We can exploit this with a collection of *close probes*, where each probe depends on intersecting a point on the boundary within some fraction of  $\varepsilon$  of another close probe.

Close probes present a problem because they suggest strategies that are somehow “unfair” as they require additional information. Certainly in any physical implementation they would be extremely nonrobust. The virtue of close probes is that they enable us to find a boundary point in a constant number of X-ray probes, as opposed to the linear probing strategy described above.

**LEMMA 10.** *Two lines that contain a pair of edges of a convex polygon  $P$  and a point on one of the two edges can be determined in 37 X-ray probes including close probes.*

*Proof.* Our strategy is similar to that used in Lemma 8, but modified to take advantage of close probes. It should be noted that there is no fraction  $\alpha$  such that parallel probes within  $\alpha\varepsilon$  of each other are guaranteed to intersect the same edge pair. The reason is that the angle between two edges can be arbitrarily close to  $\pi$ , so even a long edge can slip between two seemingly close probes. Thus the largest angle between edges will have to be bounded to take advantage of close probes.

We will replace the linear strategy of searching a bounded section of  $P$  for an edge pair by the following constant one. Within the bounded section of  $P$ , send two more parallel probes, giving four probes intersecting  $P$  labeled from left to right  $a$ ,  $b$ ,  $c$ , and  $d$ . By the argument in the proof of Lemma 8,  $a$  and  $b$  determine the steepest increasing slope possible between  $b$  and  $c$ , and probes  $c$  and  $d$  determine the steepest decreasing slope. We define  $\theta_v$  as the greater of the two angles formed by these steepest increasing and decreasing slopes with the horizontal, so  $\theta_v$  represents the angle nearest to vertical which can occur within the section without violating convexity. An edge pair will be found within seven close probes spaced  $\varepsilon \cos(\theta_v)/8$  apart between  $b$  and  $c$ . Seven are required because up to two vertices, one each from the upper and lower vertex chains of  $P$ , may slip between the close probes.

Thus we can use the strategy of Lemma 8, substituting the two parallel and seven close probes for the linear edge pair search. Using the counting argument of Lemma 8 with this change, we can determine the first edge pair in 37 probes.  $\square$

**THEOREM 11.**  *$3n + 33$  X-ray probes (including close probes) are sufficient to determine a convex polygon  $P$  given a point  $O$  within  $P$ .*

*Proof.* Lemma 10 enables us to find an edge pair in 37 probes instead of the  $2n + 23$  of Lemma 8. Substituting the new strategy for the old improves Theorem 9 by  $2n - 14$  probes, for a total of  $3n + 33$ .  $\square$

It is certainly possible that these constants can be reduced. Other strategies involving close probes are no doubt possible.

It would be nice to find an efficient X-ray probe simulation of a finger probe. It is possible by modifying the above strategy as follows. Make one of the parallel close probes along the desired probing line and then, if it hits on a parallel edge, perform additional boundary probes through the located point to finish the description of the edge pair, and calculate what the finger probe returned. However, since this constant will be over twenty it is unlikely the simulation can prove useful in any context.

**4. Bounds for verification.** A lower bound on the number of probes required to determine an object can be based on a comparison to the *verification* problem. Suppose we are given the representation of a polygon  $P$ . How many probes will be necessary to test whether  $P$  correctly describes a particular object? It is obvious that any lower bound to verification represents a lower bound to the determination problem, since it presupposes knowledge of the polygon.



For verification, clearly each vertex and edge must be confirmed. Otherwise,  $P$  could have a triangle on any unconfirmed edge or be truncated before any unconfirmed vertex. Since an X-ray probe passes through two edges or vertices, and there are at least  $2n$  points of interest, the trivial lower bound is for  $n$  probes. It can be easily shown that three probes do not suffice to verify a triangle, since no matter how the three probes are taken the object would be indistinguishable with one of four or more sides. We conjecture that the actual bound for verification is  $(3n/2)+k$  for some constant  $k$ . This is based on the observation that although  $n/2$  probes are sufficient to verify the edges given the vertices or verify the vertices given the edges, it appears at least  $n$  probes are necessary to verify either the vertices or the edges independently. This conjectured lower bound is sufficient.

**THEOREM 12.**  $(3n/2)+6$  X-ray probes are sufficient to verify a convex  $n$ -gon  $P$ .

*Proof.* Given the polygon to verify, three parallel probes are sufficient to verify the existence of a nonparallel edge pair and three origin probes are enough to define the hyperbola of it. One final probe to verify that  $P$  is not reflected through  $O$  identifies a boundary point.

From this boundary point,  $n$  boundary probes can verify the vertices. The remaining  $n-2$  edges can be verified with  $(n-2)/2$  probes, each bisecting a different pair of edges. Since  $P$  is the convex hull of its vertices, none of these probes can have length other than expected without violating convexity, unless there exists another vertex.  $\square$

**5. Open problems and extensions.** We have given strategies for probing with X-rays. In particular, we have shown that complete information about a convex  $n$ -gon can be obtained with a linear number of carefully planned X-ray probes. Still, the power of X-ray probes is not well understood. For example, no algorithm is known that decides whether or not a given collection of X-ray probes (and answers) determines the probed object.

An obvious extension of our results would be to three or more dimensions. Since the boundary probe generalizes to a finger probe in three or more dimensions, and both parallel and origin probes can identify edge pairs from slices of polytopes, a higher-dimensional probing strategy can be based on ideas in Dobkin, Edelsbrunner, and Yap [3].

A different type of probe to consider would measure the area or volume of intersection with a half-plane or half-space instead of a line. Such an "Archemedian" probe in two dimensions would have as its derivative an X-ray probe. In three dimensions, its derivative is a cross-sectional area probe. We have proved linear upper and lower bounds for determination with half-plane probes [13]. An interesting question is whether better probing strategies will result by having access to two different devices, such as finger and X-ray probing.

It is also worth considering the computational complexity of determining the location of the probes. Finally, a further study of X-ray photograph probes may generalize some of these results to simple polygons and provide some insight into standard image-reconstruction methods.

**6. Acknowledgments.** We thank Vasant Rao for motivating the problem and for his discussion of results from the field of signal processing. We also thank Raimund Seidel for suggesting the construction used in the proof of Lemma 5.

**Note added in proof.** A  $2n$  lower bound for determination can be shown by a topological argument. See [14] for details.

## REFERENCES

- [1] R. COLE AND C. K. YAP, *Shape from probing*, J. Algorithms, 8 (1987), pp. 19–38.
- [2] H. J. BERNSTEIN, *Determining the shape of a convex  $n$ -sided polygon by using  $2n + k$  tactile probes*, Inform. Process. Lett., 22 (1986), pp. 255–260.
- [3] D. P. DOBKIN, H. EDELSBRUNNER, AND C. K. YAP, *Probing convex polytopes*, Proc. 18th Annual ACM Symposium on Theory Computing, 1986, pp. 424–432.
- [4] A. OPPENHEIM AND R. SCHAFER, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.
- [5] G. T. HERMAN, *Image Reconstruction from Projections: the Fundamentals of Computerized Tomography*, Academic Press, New York, 1980.
- [6] L. A. SHEPP AND J. B. KRUSKAL, *Computerized tomography: the new medical X-ray technology*, Amer. Math. Monthly, 85 (1978), pp. 420–439.
- [7] K. T. SMITH, D. C. SOLOMON, AND S. L. WAGNER, *Practical and mathematical aspects of the problem of reconstructing objects from radiographs*, Bull. Amer. Math. Soc., 33 (1977), pp. 1227–1270.
- [8] P. C. HAMMER, *Problem 2*, in Proc. Symposium on Pure Math Vol. VII: Convexity, American Mathematical Society, Providence, RI, 1963, pp. 498–499.
- [9] R. J. GARDNER AND P. MCMULLEN, *On Hammer's X-ray problem*, J. London Math. Soc., 21 (1980), pp. 171–175.
- [10] K. J. FALCONER, *X-ray problems for point sources*, Proc. London Math. Soc., 46 (1983), pp. 241–262.
- [11] ———, *Hammer's X-ray problem and the stable manifold theorem*, J. London Math. Soc., 28 (1983), pp. 149–160.
- [12] R. J. GARDNER, *Symmetrals and X-rays of planar convex bodies*, Arch. Math., 41 (1983), pp. 183–189.
- [13] S. S. SKIENA, *Probing convex polygons with half-planes*, Dept. of Computer Science, Report UIUCDCS-R-1380, University of Illinois at Urbana–Champaign, Urbana, IL, 1987.
- [14] S. S. SKIENA, *Geometric Probing*, Ph.D. dissertation, Department of Computer Science, University of Illinois at Urbana–Champaign, Urbana, IL, 1988.

## DEFERRED DATA STRUCTURING\*

RICHARD M. KARP†, RAJEEV MOTWANI†, AND PRABHAKAR RAGHAVAN‡

**Abstract.** We consider the problem of answering a series of on-line queries on a static data set. The conventional approach to such problems involves a preprocessing phase which constructs a data structure with good search behavior. The data structure representing the data set then remains fixed throughout the processing of the queries. Our approach involves dynamic or query-driven structuring of the data set; our algorithm processes the data set only when doing so is required for answering a query. A data structure constructed progressively in this fashion is called a *deferred data structure*.

We develop the notion of deferred data structures by solving the problem of answering membership queries on an ordered set. We obtain a randomized algorithm which achieves asymptotically optimal performance with high probability. We then present optimal deferred data structures for the following problems in the plane: testing convex-hull membership, half-plane intersection queries and fixed-constraint multi-objective linear programming. We also apply the deferred data structuring technique to multi-dimensional dominance query problems.

**Key words.** data structure, preprocessing, query processing, lower bound, randomized algorithm, computational geometry, convex hull, linear programming, half-space intersection, dominance counting

**AMS(MOS) subject classifications.** 68P05, 68P10, 68P20, 68Q20, 68U05

**1. Introduction.** We consider several search problems where we are given a set of  $n$  elements, which we call the *data set*. We are required to answer a sequence of queries about the data set.

The conventional approach to search problems consists of preprocessing the data set in time  $p(n)$ , thus building up a search structure that enables queries to be answered efficiently. Subsequently, each query can be answered in time  $q(n)$ . The time needed for answering  $r$  queries is thus  $p(n) + r \cdot q(n)$ . Very often, a single query can be answered without preprocessing in time  $o(p(n))$ . The preprocessing approach is thus uneconomical unless the number of queries  $r$  is sufficiently large.

We present here an alternative to preprocessing, in which the search structure is built up “on-the-fly” as queries are answered. Throughout this paper we assume that an adversary generates a stream of queries which can cease at any point. Each query must be answered *on-line*, before the next one is received. If the adversary generates sufficiently many queries, we will show that we build up the complete search structure in time  $O(p(n))$  so that further queries can be answered in time  $q(n)$ . If on the other hand the adversary generates few queries, we will show that the total work we expend in the process of answering them (which includes building the search structure partially) is asymptotically smaller than  $p(n) + r \cdot q(n)$ . We thus perform at least as well as the preprocessing approach, and in fact better when  $r$  is small. We do so with no a priori knowledge of  $r$ . We call our approach *deferred data structuring* since we build up the search structure gradually as queries arrive, rather than all at once. In some cases we

---

\* Received by the editors December 22, 1986, accepted for publication (in revised form) November 10, 1987.

† Computer Science Division, University of California, Berkeley, California 94720. The work of the first two authors was supported in part by the National Science Foundation under grant DCR-8411954. The results in § 4 first appeared in R. Motwani and P. Raghavan, *Deferred data structures: query-driven preprocessing for geometric search problems*, in Proc. 2nd Annual ACM Symposium on Computational Geometry, Yorktown Heights, NY, June 1986, pp. 303–312.

‡ IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598. The work of this author was supported in part by an IBM Doctoral Fellowship while he was a graduate student at the Computer Science Division, University of California, Berkeley, California 94720.

show that our deferred data structuring algorithm is of nearly optimal efficiency, even in comparison with algorithms that know  $r$ , the number of queries, in advance.

In § 2 we exemplify our approach through the *membership query* problem. We determine the complexity of answering  $r$  queries on  $n$  elements under the comparison tree model. In § 3 we present a randomized algorithm for the membership query problem whose performance matches an information-theoretic lower bound (ignoring asymptotically smaller *additive* terms). We then proceed to exhibit deferred data structure for several geometry problems. In § 4 we show that deferred data structuring is optimal for the following two-dimensional geometric problems: (1) Given  $n$  points in the plane, to determine whether a query point lies inside their convex hull. (2) Given  $n$  half-planes, to determine whether a query point lies in their common intersection. (3) Given  $n$  linear constraints in two variables, to optimize a query objective function (also linear). Our algorithms are proved optimal by means of a tight lower bound (under the algebraic computation tree model) in § 4.4. In § 5 we consider *dominance problems* in  $d$ -space. We present theorems about the deferred construction of Bentley's ECDF *search tree* [2].

In this paper all logarithms are to the base two.

**2. General principles of deferred data structuring.** In this section we develop the basic ideas involved in deferred data structuring. Let  $X = \{x_1, x_2, \dots, x_n\}$  be a set of  $n$  elements drawn from a totally ordered set  $U$ . Consider a series of queries where each query  $q_i$  is an element of  $U$ ; for each query, we must determine whether it is present in  $X$ .

If we had to answer just one query, we could simply compare the query  $q_1$  to every member of  $X$  and answer the query in  $O(n)$  comparison operations. This would be the preferred method for answering a small number of queries. On the other hand, if we knew that the number of queries  $r$  were large, we could first sort the elements of  $X$  in  $p(n) = O(n \log n)$  operations, then building up a binary search tree  $T_X$  for the elements of  $X$ . We could then do a binary search costing  $Q(n) = O(\log n)$  comparisons for each query; this takes  $O((n+r) \cdot \log n)$  comparisons.

We proceed to determine the complexity (number of comparisons) of answering  $r$  queries on the set  $X$ ; we do not know  $r$  a priori, and each query is to be answered before we know of the next one.

**2.1. The lower bound.** We first prove an information-theoretic lower bound for this problem.

**THEOREM 1.** *The number of comparisons needed to process  $r$  queries is at least  $(n+r) \cdot \log(\min\{n, r\}) - O(\min\{n, r\})$  in the worst case.*

*Remark.* Note that neither of the strategies mentioned above (linear search, or sorting followed by binary search) achieves this bound for all  $r \leq n$ .

*Proof.* If we could collect the  $r$  queries and process them *off-line*, we would have an instance of the SET INTERSECTION problem where we have to find the elements common to the sets  $X = \{x_1, x_2, \dots, x_n\}$  and  $Q = \{q_1, \dots, q_r\}$ . We will prove a lower bound of  $\Omega((n+r) \cdot \log(\min\{n, r\}))$  comparisons for determining the intersection of two sets of cardinalities  $n$  and  $r$ . This off-line lower bound will hold a fortiori for the on-line case in which we are interested. We present the argument for the case  $r \leq n$ ; the other case is symmetrical.

Since we are interested in lower bounds on this problem, we can restrict our attention to only those cases where  $X \cap Q = \emptyset$ . In this case the algorithm has to determine the relation of each element in  $X$  to each element in  $Q$ . An adversary can ensure that for any two elements in  $Q$  there will be at least one in  $X$  whose value lies

between them. In other words, the elements of  $Q$  will partition  $X$  into at least  $r-1$  nonempty classes. Each such class will consist of all those members of  $X$  which lie between two consecutive values in the total ordering of  $Q$ . We shall give an information-theoretic lower bound by counting some ways of arranging  $X$  and  $Q$  to satisfy the above constraint.

There are  $r!$  ways of ordering the elements in  $Q$ . Given a total order on  $Q$ , there are  $(r-1)!$  ways of separating the elements in  $Q$  by some arbitrary  $r-1$  elements from  $X$ . The remaining elements of  $X$  can be placed arbitrarily. There are  $r+1$  available slots as determined by the  $r$  ordered elements of  $Q$ . This can be done in  $(r+1)^{n-r+1}$  ways. Let  $I$  be the total number of interleavings (of  $X$  and  $Q$ ) possible when  $S \cap Q = \emptyset$ . Then the number of possible arrangements specified above is a lower bound on the value of  $I$ :

$$I \geq r! \cdot (r-1)! \cdot (r+1)^{n-r+1}.$$

Since the algorithm has to identify one out of (at least) this many possible arrangements the lower bound is given by  $\log I$ :

$$\log I \geq (n+r) \cdot \log r - 2r \log e.$$

Here  $e$  represents the base of the natural logarithms.  $\square$

**2.2. Upper bounds.** We now present two approaches to obtaining an upper bound which comes within a multiplicative constant factor of the lower bound. The first approach is based on merge-sort, while the second is based on recursively finding medians.

**2.2.1. An approach based on merge-sort.** The following algorithm comes within a constant factor of the lower bound. It uses a recursive merge-sort technique to totally order the elements in  $X$ . The merge-sort proceeds in  $\log n$  stages. At the end of a stage the set  $X$  is partitioned into a number of equal-sized totally ordered subsets called *runs*. Each stage pairs off all the runs resulting from the previous stage and merges them to create longer runs. These stages are interleaved with the processing of a set of queries, until a single totally ordered run remains, whereafter no more comparisons between elements of  $X$  are required. To process a query implies a binary search through each of the existing runs. The number of queries processed between consecutive merging stages or, equivalently, the minimum length of a run before the  $i$ th query, are chosen appropriately.

This algorithm ensures that the size of each run is at least  $L(i)$  before the  $i$ th query. A suitable choice for  $L(i)$  is  $\Theta(i \log i)$ . Since the length of a run must be a power of 2 we will choose

$$L(i) = 2^{\lceil \log(i \log i) \rceil}.$$

The processing cost of going from a stage with runs of length 1 to runs of length  $L(i)$  is  $O(n \log L(i))$ . Thus the total cost of processing in answering  $r$  queries is  $O(n \log r)$ . The search cost for the  $i$ th query is upper bounded by  $n \cdot \lceil \log(L(i)+1) \rceil / L(i)$ . Summing over the first  $r$  queries, the search cost is bounded by

$$\sum_{i=1}^r \frac{n}{L(i)} \cdot \lceil \log(L(i)+1) \rceil = O(n \log r).$$

**THEOREM 2.** *For  $r \leq n$ , the total cost of answering  $r$  queries is  $O(n \log r)$ .*

When  $r > n$ , we note that the set  $X$  will be completely ordered by our strategy. All queries are then answered in time  $O(\log n)$  by binary search.

*Proof.* The processing cost and the search cost are each  $O(n \log r)$ , so that the total cost of answering the first  $r$  queries is  $O(n \log r)$ .  $\square$

**2.2.2. An approach based on recursive median finding.** We now describe an alternative approach based on median finding; a specification of the algorithm in “pseudopascal” follows. The algorithm builds the binary search tree  $T_X$  in a query-driven fashion. Each internal node  $v$  of  $T_X$  is viewed as representing a subset  $X(v)$  of  $X$ —the root represents  $X$ , its left and right children represent the smallest  $(n-1)/2$  and the biggest  $(n-1)/2$  elements of  $X$ , respectively, and so on. Let  $L\text{Son}(v)$  and  $R\text{Son}(v)$  represent the left and right children of  $v$ , respectively. We can now think of building  $T_X$  as follows. For each internal node  $v$ , *expansion* consists of partitioning  $X(v)$  into two subsets of equal size—elements smaller than the median of  $X(v)$ , which will constitute  $X(L\text{son}(v))$ , and elements larger than the median, which will make up  $X(R\text{son}(v))$ . We label  $v$  by the median of  $X(v)$ . Thus a node at level  $i$  represents at most  $n/2^i$  elements of  $X$ .<sup>1</sup> Subsequently,  $L\text{Son}(v)$  and  $R\text{Son}(v)$  may be expanded. Since the median of  $X(v)$  can be found in  $3|X(v)|$  comparisons [12], the expansion of node  $v$  takes  $3|X(v)|$  comparisons. If we begin by expanding the root of  $T_X$  (which represents the entire set  $X$ ), and then expand every node created,  $T_X$  can be built up in  $3n \log n$  comparisons.

The search for a query can be thought of as tracing a root-to-leaf path in  $T_X$ . The key observation is that for any given query  $q_i$ , we need only expand those nodes visited by the search for  $q_i$ ; this is the query-driven tree construction referred to earlier. After each expansion, at most one of the resulting offspring will be visited. The first query  $q_1$  is answered in  $O(n + n/2 + \dots) = O(n)$  operations while building up one root-to-leaf path of  $T_X$ . The time taken to answer  $q_1$  is thus within a constant factor of the time for a linear search. In the process of answering  $q_1$ , we have developed some structure that will be useful in answering subsequent queries; any future search that visits a node that is already expanded will only cost us a single comparison to proceed to the next level of the search; there is no further expansion cost at this node. Nodes that remain unexpanded will be expanded when other queries visit them. When  $n$  queries that visit all  $n$  leaves have been answered,  $T_X$  will have been completely built up. In essence, we are dispensing with an explicit preprocessing phase, i.e., we are doing “preprocessing” operations only when needed. The cost of building the data structure is distributed over several queries.

**DETAILED DESCRIPTION OF THE ALGORITHM.** With every node in the tree we associate a set of values and a label, both of which may at times be undefined.

**Main body**

- Step 1. Initialize the tree,  $T_X$ , with the  $n$  data keys at the root.
- Step 2. Get a query  $q$ .
- Step 3.  $\text{Result} \leftarrow \text{SEARCH}(\text{root}, q)$ .
- Step 4. Output the result.
- Step 5. Goto Step 2

**procedure SEARCH ( $v$ : node;  $q$ :query): boolean;**

- Step 1. If ( $v$  is not labeled) then EXPAND ( $v$ ).
- Step 2. If ( $\text{label}(v) = q$ ) then return *true*.
- Step 3. If ( $v$  is a leaf node) then return *false*.

<sup>1</sup> Actually it represents slightly fewer elements, since each node picks up one element of  $X$  as its label. This does not matter, as we are deriving an upper bound.

*Step 4.* If  $(q < \text{label}(v))$  then return SEARCH ( $\text{left\_child}(v)$ ,  $q$ ).

*Step 5.* If  $(q > \text{label}(v))$  then return SEARCH ( $\text{right\_child}(v)$ ,  $q$ ).

**procedure EXPAND ( $v$ : node);**

*Step 1.*  $S \leftarrow \text{set}(v)$ .

*Step 2.*  $m \leftarrow \text{MEDIAN\_FIND}(S)$ .

*Step 3.*  $\text{label}(v) \leftarrow m$ .

*Step 4.* if  $(|S| = 1)$  then return.

*Step 5.*  $S_l \leftarrow [x \mid x \in S \text{ and } x < m]$ .

*Step 6.*  $S_r \leftarrow [x \mid x \in S \text{ and } x > m]$ .

*Step 7.*  $\text{set}(\text{leftchild}(v)) \leftarrow S_l$ .

*Step 8.*  $\text{set}(\text{rightchild}(v)) \leftarrow S_r$ .

It should be noted that the two subsets,  $S_l$  and  $S_r$ , are computed by the procedure MEDIAN\_FIND as part of the process of finding the median. There is no extra work associated with determining these two sets once the median has been found.

In order to analyze our algorithm, let us define a function on  $n$  and  $r$  as follows:

$$\Lambda(n, r) = \begin{cases} 3n \log r + r \log n, & r \leq n, \\ (3n + r) \cdot \log n, & r > n. \end{cases}$$

Note that  $\Lambda(n, r) = \Theta((n+r) \cdot \log \min(n, r))$  since  $r \cdot \log n \leq n \cdot \log r$  for  $r \leq n$ .

**THEOREM 3.** *The number of operations needed for processing  $r$  queries is no more than  $\Lambda(n, r)$ .*

*Proof.* Consider the case  $r \leq n$ . No more than  $r$  nodes will be expanded at any level of  $T_X$ , after  $r$  queries. For nodes in the top  $\log r$  levels, the total cost is thus less than  $3n \log r$ . This is because *all* nodes may be expanded at each of the first  $\log r$  levels. The expansion of a node  $v$  entails finding the median of  $X(v)$  and this requires at least  $3|X(v)|$  comparisons in the worst case [12]. For  $i > \lceil \log r \rceil$  the node-expansion cost at level  $i$  is  $O(r \cdot n/2^i)$ . This is because the cost of expanding a node at level  $i$  is at most  $3 \cdot n/2^i$ . Summing over all but the first  $\lceil \log r \rceil$  levels, the cost of node expansion at these levels is  $O(n)$ . In addition to the expansion cost, we have to consider the cost associated with search; this is at most  $\log n$  comparisons per query. The search component of the cost is thus always less than  $r \log n$ .

When  $r$  exceeds  $n$ , the expansion cost can never exceed the cost of constructing  $T_X$  completely; this cost is  $3n \log n$ . Again, note that the factor of 3 comes from the median-finding procedure.  $\square$

**2.3. A general paradigm for deferred data structuring.** We are now ready to state the general paradigm for deferred data structuring. This paradigm will isolate some features essential for a search problem to be amenable to this approach, and will simplify our description of the geometric search problems considered in §§ 4 and 5. It also enables us to identify some problems where this approach is not likely to work.

Let  $\Pi$  be a search problem with the following properties. (1) The search is on a set  $S$  of  $n$  data points (in the above example,  $S = X$ ). (2) A query  $q$  can be answered in  $O(n)$  time. (3) In time  $O(n)$ , we can partition  $S$  into two equal-sized subsets  $S_1$  and  $S_2$  such that (i) the answer to query  $q$  on set  $S$  is equal to the answer to  $q$  on either  $S_1$  or  $S_2$ ; (ii) in the course of partitioning  $S$  we can compute a function on  $S$ ,  $f(S)$ , such that there is a constant time procedure, TEST ( $f(S)$ ,  $q$ ), which will determine whether the answer to  $q$  on  $S$  is to be found in  $S_1$  or  $S_2$ . (In the above example  $f(S) = \text{MEDIAN}(S)$  and TEST is a simple comparison operation.)

Under these conditions, we can adopt the deferred data structuring approach that builds the search tree gradually. We illustrate this paradigm by several geometric examples in §§ 4 and 5.

**3. A randomized algorithm.** In the last section we saw a deterministic algorithm to answer  $r$  queries in  $O((n+r) \cdot \log \min \{n, r\})$  time using deferred data structures. The upper bound of Theorem 3 exceeds the information-theoretic lower bound by a factor of 3 if we use the median algorithm given in [12]. Finding the median of  $n$  elements takes  $3n$  comparisons and this is what leads to the gap between the upper and lower bounds. A careful implementation would reduce the constant factor to 2.5 by passing down certain partial orders generated in the median-finding algorithm from parent to children nodes. More easily implemented algorithms given in [3] would yield even higher constant factors. There is an algorithm due to Floyd [7] which computes the median in  $3n/2$  expected time. Its use would reduce our constant to  $3/2$ . Here we present a randomized algorithm in which the number of comparisons will be optimal (with high probability).

The randomized algorithm differs from the one in § 2 in just one respect. The median of the set of values stored at a node was used earlier to get a partition for the purposes of node expansion. Here we will use a *mediocre* element for the same purpose. The mediocre element will be chosen to be quite close to the median. More precisely, the rank of a mediocre element from a set of size  $t$  will lie in the range  $t/2 \pm t^{2/3}$ . We will use randomized techniques to compute a mediocre element efficiently. First, a random subset of size  $O(t^{5/6})$  is chosen from the  $t$  elements. The median of this random subset is a good candidate for being a mediocre element. It takes  $t + O(t^{5/6})$  comparisons to pick a random sample and test its median element for “mediocrity” (see Step 5 below). This sampling is repeated until a mediocre element is found. The call to the procedure MEDIAN\_FIND, in the algorithm outlined in § 2, should be replaced by a call to the procedure MEDIOCRE\_FIND outlined below.

**procedure MEDIOCRE\_FIND (T: set of values): value;**

Step 1 Let  $t = |T|$ .

Step 2 Pick a random sample  $S$  of size  $2 \cdot \lceil t^{5/6} \rceil + 1$  from  $T$ .

Step 3  $m \leftarrow \text{MEDIAN\_FIND}(S)$ .

Step 4 Compute rank( $m$ ) by comparing with each element of  $T - S$ .

Step 5 If rank( $m$ ) is not in the range  $(t/2) \pm t^{2/3}$  then goto Step 2.

Step 6 Return  $m$ .

Note that in Step 4 we need not compare  $m$  with elements of  $S$  since we assume that the procedure MEDIAN\_FIND implicitly gives us the partition of  $S$  with respect to  $m$ . At the last few levels we will revert to deterministic median finding since the node sizes will be too small to justify randomization. A good choice is to use procedure MEDIOCRE\_FIND for the first  $\log n - 5$  levels and procedure MEDIAN\_FIND thereafter. The randomized algorithm leads to the following theorem.

**THEOREM 4.** *Let  $T(n, r)$  be the total number of comparisons made by the randomized algorithm in answering  $r$  queries on  $n$  elements. Then the following holds with probability greater than  $1 - \log r / \beta \cdot n$ ,*

$$T(n, r) \leq \begin{cases} (1 + \alpha)(n \log r + r \log n), & r \leq n, \\ (1 + \alpha)(n + r) \log n, & r > n, \end{cases}$$

where  $\beta$  depends on the value of the constant  $\alpha$ .

The remainder of this section is devoted to the proof of this theorem. The proof will be organized into five lemmas.



The use of mediocre elements (instead of the median) may result in uneven splits, causing an imbalance in the binary search tree being created. Nevertheless, the following lemma shows that the higher of the new binary search tree cannot be much worse than  $\log n$ . We also show that the number of elements associated with each node at level  $i$  is close to  $n/2^i$ . Let the size of a node in the search tree be the number of elements associated with that node.

LEMMA 1. Let  $s_i$  be the size of some node at level  $i$ . Then,

$$n_i \left( 1 - \frac{20}{n_i^{1/3}} \right) \leq s_i \leq n_i \left( 1 + \frac{20}{n_i^{1/3}} \right),$$

provided  $n_i \geq 22$ , where  $n_i = n/2^i$ .

*Proof.* We will prove one side of the inequality by means of induction on the levels. The inequality is clearly true at the root ( $i = 0$ ) since  $s_0 = n$ . Suppose the inequality holds up to level  $i - 1$ , i.e.,  $s_{i-1} \leq n_{i-1} \cdot (1 + 20/n_{i-1}^{1/3})$ . We now partition the  $s_{i-1}$  elements about the mediocre element. Let  $s_i$  denote the larger of the two partition sets. By the definition of the mediocre element we have  $s_i \leq s_{i-1}/2 + s_{i-1}^{2/3}$ . Using the fact that  $(1 + x)^a \leq 1 + a \cdot x$ ,  $0 \leq a \leq 1$  we get,

$$s_i \leq n_i \left( 1 + \frac{1}{n_i^{1/3}} \left( 11 \cdot 2^{2/3} + \frac{40}{3} \cdot \frac{2^{1/3}}{n_i^{1/3}} \right) \right).$$

For  $n_i \geq 22$ , we note that,

$$\left( 11 \cdot 2^{2/3} + \frac{40}{3} \cdot \frac{2^{1/3}}{n_i^{1/3}} \right) \leq 20.$$

This implies the desired result,

$$s_i \leq n_i \left( 1 + \frac{20}{n_i^{1/3}} \right)$$

provided  $n_i \geq 22$ .  $\square$

LEMMA 2. *The height of the binary search tree in the randomized algorithm will be  $\log n + O(1)$ .*

*Proof.* At level  $k = \log n - 5$  we will no longer be using mediocre elements to expand a node. Instead, we use the median of the set of elements stored at a node to partition those elements. At this point Lemma 1 is still applicable since  $n_k = 32 \geq 22$  and we have,

$$s_k \leq n_k \left( 1 + \frac{20}{n_k^{1/3}} \right) \leq 2^8.$$

Thus, the total number of levels is no more than  $k + 8$ . The height of the tree is bounded by  $\log n + 3$ .  $\square$

From Lemma 2 it follows that the cost of searching in the randomized binary search tree will be close to optimal. Let us now consider the cost of constructing the tree, in particular the total cost of node expansions. The following result shows that the median of the random sample is a mediocre element for the entire set with very high probability.

LEMMA 3. *Let  $p(t)$  be the probability that a single iteration of the random sampling does not come up with a mediocre element for a set of size  $t$ . Then,*

$$p(t) \leq 2 \cdot t^{1/2} \cdot \exp(-4 \cdot t^{1/6}) \leq \frac{1}{4t}.$$

*Proof.* Let  $T$  be a set of size  $t$  to which a single iteration of the random sampling process has been applied. First, a random subset  $S$  of size  $s(t) = 2 \cdot f(t) + 1$  is chosen, where  $f(t) = \lceil t^{5/6} \rceil$ . The median of  $S$  is tested for being a mediocre element of  $T$ . In other words, the rank of the median of  $S$  should be in the range  $t/2 \pm t^{2/3}$  in  $T$ . Let  $P(x_r)$  be the event that the element  $x_r$  (the element of rank  $r$  in  $T$ ) is the median of  $S$ :

$$P(x_r) = \binom{r-1}{f(t)} \cdot \binom{t-r}{f(t)} / \binom{t}{s(t)}, \quad f(t) < r \leq t - f(t).$$

Let  $d(t) = t^{2/3}$ . We will refer to  $f(t)$  and  $d(t)$  as  $f$  and  $d$ , respectively, to simplify the following description. Clearly,

$$p(t) = \sum_{r=f+1}^{t/2-d} P(x_r) + \sum_{r=t/2+d}^{t-f} P(x_r) = \frac{2s}{t-2f} \sum_{r=f+1}^{t/2-d} \binom{r-f}{r} \cdot \binom{r}{f} \cdot \binom{t-r}{f} / \binom{t}{2 \cdot f}.$$

We make use of Stirling’s formula:

$$n! = (2\pi n)^{1/2} \left(\frac{n}{e}\right)^n e^{k_n}, \quad \frac{1}{12n+1} < k_n < \frac{1}{12n}$$

to derive the following inequality upon considerable simplification:

$$p(t) < 2 \cdot f^{1/2} \cdot \exp\left(\frac{-4fd^2}{t^2}\right).$$

Given the choices for  $f(t)$  and  $d(t)$  the bound on  $p(t)$  follows immediately. The second part of the inequality given below is also easy to verify:

$$p(t) < 2 \cdot t^{1/2} \cdot \exp(-4t^{1/6}) < \frac{1}{4t}. \quad \square$$

Consider now the overall cost of expanding the nodes in the randomized algorithm. First, there is the cost of finding the medians of the small random samples. Lemma 5 will show that the cost of finding the medians of the small random samples is small even when summed over the entire tree. More important is the cost of deciding whether the median for the sample is a mediocre element for the entire set. There is no cost associated with the actual partitioning since the testing for “mediocrity” implicitly determines the precise partition (see Step 5 of the procedure `MEDIOCRE_FIND`). Consider the  $i$ th level in the tree being constructed. Let  $m = 2^i$  denote the maximum number of nodes at this level. The sizes of the sets associated with the nodes at this level must lie in the range  $(n_i/2) \pm 20 \cdot n^{2/3}$ , where  $n_i = n/m$  is the average size of these sets. Suppose each application of the random sampling yielded a mediocre element. This would imply that the total cost of testing for mediocrity is  $n$ . However, there will be some bad instances where we do not generate a mediocre element. Let the number of such instances be  $s$  at the  $i$ th level. The next lemma shows that with high probability  $s$  is bounded by  $\epsilon m$ , where  $\epsilon$  is an appropriately small constant. Let  $c_i$  denote the cost of testing for mediocrity at level  $i$ . When  $s \leq \epsilon \cdot m$  we have

$$c_i \leq n + n \cdot \epsilon \left(1 + \frac{20}{n_i^{1/3}}\right) = (1 + \alpha) \cdot n.$$

Since  $n_i > 1$  at all levels it is clear the  $\alpha \leq 21 \cdot \epsilon$ .

**LEMMA 4.** *Let  $C$  denote the sum of  $c_i$  over all but the last  $O(1/\epsilon)$  levels,  $P(C \geq (1 + \alpha) \cdot n \cdot \log r) \leq \log r / k^2 \cdot n$ .*

*Proof.* Let the random variable  $\zeta_i$  denote the number of bad instances in  $l = (1 + \epsilon) \cdot m$  iterations of the random sampling at level  $i$ . We already have bounds on

$p(t)$ , the probability of a single iteration on a node of size  $t$  being bad. The  $l$  iterations at level  $i$  do not use equal-sized sets. Therefore let  $p$  denote the largest value taken by  $p(t)$  at the nodes of that level. Let  $E(\zeta)$  and  $D(\zeta)$  denote the mean and deviation of some random variable  $\zeta$ . The Chebyshev inequality states that  $P(|\zeta - E(\zeta)| \geq \lambda \cdot D(\zeta)) \leq 1/\lambda^2$ . Since  $E(\zeta_i) = l \cdot p$  and  $D(\zeta_i) = (l \cdot p \cdot (1 - p))^{1/2}$  we have the following:

$$P(\zeta_i \geq l - m) \leq \frac{1 \cdot p \cdot (1 - p)}{m \cdot \epsilon^2} \quad \text{when } p \leq \frac{\epsilon}{2 \cdot (1 + \epsilon)}.$$

Using the bounds on  $p(t)$  and the lower bound on the size of a node at level  $i$  we get,  $P(s > \epsilon m) = P(c_i \geq (1 + \alpha) \cdot n) \leq k/\epsilon^2 \cdot n$  for all but the last  $O(\log 1/\epsilon)$  levels,  $k$  is a small constant. Choosing  $\beta = \epsilon^2/k$  and summing the probability over the first  $\log r$  levels yields the required bound.  $\square$

**LEMMA 5.** *When  $r < n$ , the total cost of finding the medians of the random samples is  $O(n^{5/6} \cdot r^{1/6})$  with probability  $1 - \log r/\beta \cdot n$ .*

*Proof.* The cost of finding the median at a node of size  $t$  is  $3t$ . Let the sizes of the two children of this node be  $k \cdot t$  and  $(1 - k) \cdot t$ , where  $k$  lies between  $\frac{1}{2}$  and 1. The cost of finding the medians for the children will be proportional to  $C(k) \cdot t^{5/6}$ , where  $C(k) = (k^{5/6} + (1 - k)^{5/6})$ . Clearly,  $C(k)$  is maximized at  $k = \frac{1}{2}$ . Define  $C = C(\frac{1}{2}) = 2^{1/6}$ . Thus, the cost of finding the medians at a single level increases by at most a factor of  $C$  in going from level  $i$  to  $i + 1$ . We know that the cost of median finding at the first level is  $3 \cdot n^{5/6}$ . Hence, the total median-finding cost for the first  $\log r$  levels is

$$3 \cdot n^{5/6} \cdot (1 + C + C^2 \dots C^{\log r - 1}).$$

This sums to  $O(n^{5/6} \cdot r^{1/6})$  since  $C \leq 2^{1/6}$ . When  $r > n$  the bound on the median-finding cost becomes  $O(n)$ . In our analysis so far we have ignored the repetitions in the median finding for a given node. This will be necessary since not every median of the random sample will be a mediocre element for the entire set. However, the analysis in Lemma 4 also applies to the median-finding cost since it just bounds the number of repetitions of the mediocre finding process at a level.  $\square$

Theorem 4 follows immediately from Lemmas 2, 4, and 5.

**4. Planar convex hull and linear programming problems.**

**4.1. Point membership in a convex hull.** In this section we consider the following problem. We are given a set  $P = \{p_1, p_2, \dots, p_n\}$  of  $n$  data points in the plane. Data point  $p_i$  is specified by its two coordinates  $p_i = (p_{ix}, p_{iy})$ . The convex hull of  $P$  will be denoted by  $CH(P)$ . We are required to answer a series of queries: "Is the query point  $q_j = (q_{jx}, q_{jy})$  included in  $CH(P)$ ?"

We first present two solutions based on the preprocessing approach. Neither of these is optimal for all values of  $r$ . Let  $BCH(P)$  denote those points of  $P$  which lie on the boundary of  $CH(P)$ . A single query can be answered in  $O(n)$  time as follows. Compute the polar angles from  $q_j$  to all the data points. The query point is included in  $CH(P)$  if and only if the range of angles  $\geq 180^\circ$ . Alternatively, we can answer  $r$  queries by first constructing  $CH(P)$  in time  $O(n \log h)$ , where  $h$  is the number of points in  $BCH(P)$  [6], [11]. Now choose a point,  $O$ , in the interior of  $CH(P)$  and divide the plane into  $h$  wedges by means of  $h$  semi-infinite lines originating at  $O$  and going through each of the  $h$  vertices of  $CH(P)$ . Each wedge contains exactly one edge from the boundary of  $CH(P)$ . In any wedge, all points on the same side of this edge as  $O$  must lie inside  $CH(P)$ . To answer a query we first determine the wedge in which it lies in  $O(\log h)$  time by doing a binary search with respect to the angles subtended at  $O$ . We can now test the query point with respect to the edge of the  $CH(P)$  which

lies in that particular wedge to decide the membership in  $\text{CH}(P)$ . This requires a total of  $O((n+r) \cdot \log h)$  operations to answer  $r$  queries.

Our approach to solving the point membership problem using deferred data structuring is based on the Kirkpatrick–Seidel top-down convex-hull algorithm [6]. The edges on the boundary of  $\text{CH}(P)$  consist of an *upper chain* and a *lower chain*. Each of these is a sequence of edges going from the leftmost to the rightmost point in  $P$ . Consider a vertical line which partitions  $P$  into two nonempty subsets. Such a line will intersect with exactly one edge of each chain; these edges will be referred to as the *upper tangent* and the *lower tangent* corresponding to the line. The tangents corresponding to a vertical line which partitions  $P$  into subsets of equal size (which we call the *median line*) are called the tangents of  $P$ . Kirkpatrick and Seidel show that a tangent can be computed in  $O(|P|)$  operations.

We now describe our deferred data structure. In the following description we only refer to the upper chain and tangents; analogous reasoning applies to the lower chain and tangents. The data structure consists of a binary search tree  $T_P$  in which each internal node  $v$  represents a subset  $P(v)$  of  $P$  (where  $P(\text{root}) = P$ ). Associated with  $v$  is an  $x$ -interval  $R_v = [x_L(v), x_R(v)]$ ;  $P(v)$  consists of exactly those data points whose  $x$ -coordinates lie in  $R_v$ . We expand a node by computing the median line of  $P(v)$ . The members of  $P(v)$  are partitioned into two subsets: points lying to the left of the median line and points lying to its right. These are associated with the two children of  $v$ . The tangent for  $P(v)$  can now be computed in  $O(|P(v)|)$  operations. It is possible that the tangents corresponding to the two vertical lines demarcating  $R_v$  may be adjacent in the chain. In fact, the two tangents may be the same. In these degenerate cases we do not need to compute the tangent of  $P(v)$ . Such degeneracies can be identified from the tangents corresponding to the vertical lines bounding  $R_v$  (these tangents will have been computed by ancestors of  $v$ ). If at a node we find that both the upper and the lower tangent are degenerate, we will not expand the node; such a node is a leaf of  $T_P$ . Since at least one new tangent is discovered each time we expand a node, the number of internal nodes of  $T_P$  (and hence the number of leaves of  $T_P$ ) will never exceed  $h$ .

The search for a query traverses a root-to-leaf path in the search tree. A node is expanded when it is first visited. At any node  $v$  the search progresses to its left or right child depending on the  $x$ -coordinate of the query point. In addition, we test whether the query point lies below the upper tangent (extended to infinity in both directions) of  $P(v)$ . If this test fails at any node along the search path we know that the query point lies outside  $\text{CH}(P)$ . Similar tests apply to the lower chain/tangent.

Figure 1 shows an example in which two queries  $q_1$  and  $q_2$  have resulted in the expansion of the root and its two children. The query  $q_1$  lay to the left of the median line of  $P$ , and above the lower tangent of  $P$  (extended to the left by dotted lines). This caused  $\text{LSON}(\text{root})$  to be expanded; at this point we find that  $q_1$  lies below the lower tangent of the left child and is thus outside  $\text{CH}(P)$ . Note that the lower tangents of  $\text{root}$  and  $\text{LSON}(\text{root})$  meet at a point of  $P$ ; this means that we will never again compute a lower tangent in the right-subtree of  $\text{LSON}(\text{root})$ . Similarly,  $q_2$  expands the right child of the root node; it is found to lie between the upper and lower tangents of  $\text{RSON}(\text{root})$ , and is thus in  $\text{CH}(P)$ .

**THEOREM 5.** *The number of operations for processing  $r$  hull-membership queries is  $O(\Lambda(n, r))$ .*

*Proof.* The depth of  $T_P$  never exceeds  $\log n$ . Moreover, a node at level  $i$  can be expanded in time  $O(n/2^i)$ . This fits our paradigm. An analysis similar to the proof of Theorem 2 establishes the result.  $\square$

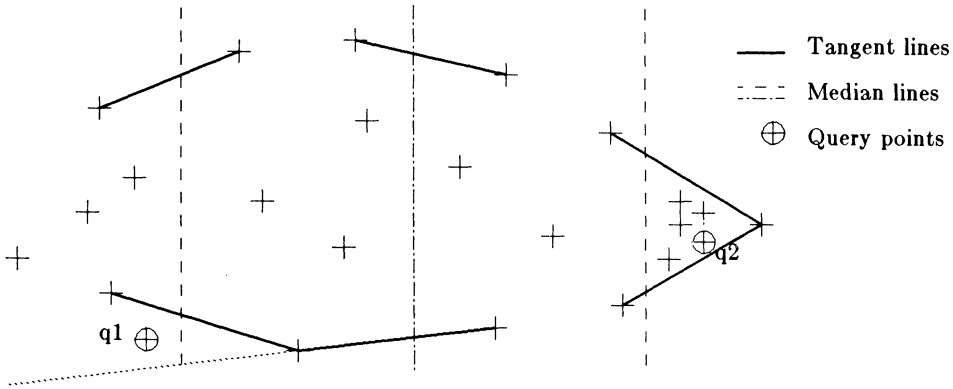


FIG. 1. Membership in a hull; two queries and the resulting development of  $T_p$ .

**4.2. Intersection of half-spaces.** We consider the problem of determining whether a query point  $q_j = (q_{jx}, q_{jy})$  lies in the intersection of  $n$  half-planes. Let  $H = \{h_1, h_2, \dots, h_n\}$  denote the set of lines which bound the half-planes. We assume that each half-plane contains the origin. If not, we can apply a suitable linear transformation in  $O(n)$  time to bring the origin into the common intersection (provided the intersection of the  $h_i$  is nonempty). This can be done by finding a point in the interior of the intersection [8] and mapping the origin onto this feasible point. We can also test in linear time whether the intersection is empty [8]. Let  $H_i$  denote the half-plane (containing the origin) which is bounded by the line  $h_i$ . We assume in this section that the intersection of the  $H_i$  is bounded—in § 4.3 we will show that the case of an unbounded intersection region is easily handled.

The notion of geometric duality (or polarity) [4], [11] will prove extremely useful in the solution of the next two problems. In the plane this reduces to a transformation between points and lines. The dual of a point  $p = (a, b)$  is the line  $l_p$  whose equation is  $ax + by + 1 = 0$ , and vice versa. A more intuitive definition is illustrated in Fig. 2.

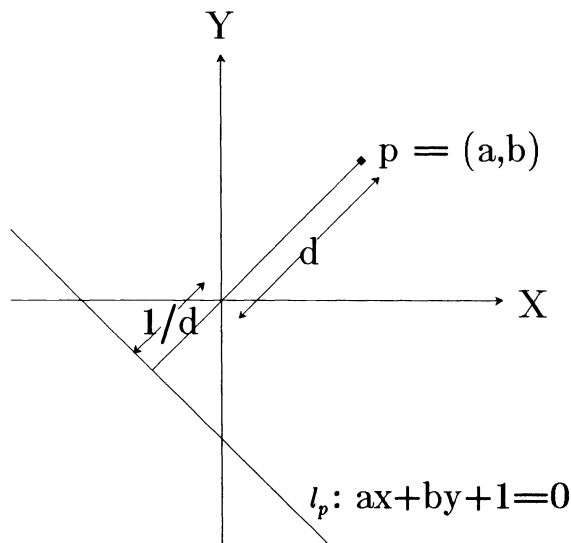


FIG. 2. Duality of points and lines.

The line  $l_p$  is perpendicular to the line joining the origin to the point  $p$ . If the distance between  $p$  and the origin is  $d$ , then the dual line  $l_p$  lies at a distance  $1/d$  from the origin in the opposite direction.

We will now apply the duality transformation to the intersection of the half-planes under consideration. The dual of the line  $h_i$  is a point, which we will denote by  $p_i$ ; we denote by  $P$  the set of these points. The dual of the intersection of the  $H_i$  is the set of all points in  $\mathbf{R}^2$  not in  $\text{CH}(P)$ . The dual of  $q_j$  is a line  $L_j$ . The query point  $q_j$  is in the intersection of the  $H_i$  if and only if  $L_j$  does not intersect  $\text{CH}(P)$ . Thus our problem reduces to determining whether each of a series of query lines intersects the convex hull of a set of points.

The search tree and the node-expansion process are exactly the same as in § 4.1. At each node  $v$ , we compute the intersection of  $L_j$  with the median line of  $P(v)$ . We know that  $L_j$  must intersect  $\text{CH}(P)$  if one of the following holds: (1) the intersection point lies between the upper and lower tangents of  $P(v)$ ; (2)  $L_j$  intersects one of the tangents of the current node. If not, we must continue the search in the left or right child of  $v$ , depending on the slopes of  $L_j$  and the tangent. These three possibilities are illustrated in Fig. 3 by lines  $L1$ ,  $L2$ , and  $L3$ , respectively. In the case of  $L3$ , we see that any intersection of  $L3$  with  $\text{CH}(P)$  must lie to the left of the median line; we therefore continue the search in  $\text{LSON}(v)$ .

The following theorem results.

**THEOREM 6.** *The number of operations for processing  $r$  half-plane intersection queries is  $O(\Lambda(n, r))$ .*

**4.3. Two-variable linear programming.** Let  $L(f)$  be a two-variable linear programming problem with  $n$  constraints and the objective function  $f$ , which is to be minimized subject to these constraints. The algorithms of Dyer [5] and Megiddo [8] can find the optimum for a single objective function in time  $O(n)$ . We consider a query version of the linear programming problem. Each query is an objective function  $f_i$ , and we are asked to solve  $L(f_i)$ .

The preprocessing approach to this problem consists of finding the intersection of the half-planes defined by the constraints. This can be done in  $O(n \log n)$  time by divide-and-conquer. The set of half-planes is partitioned into two sets of almost equal

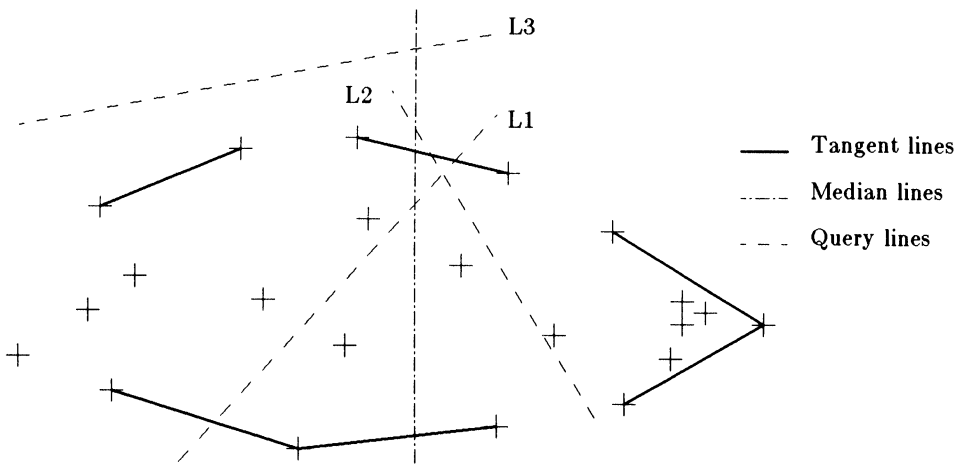


FIG. 3. Example for testing line intersection with a hull.

sizes. The intersection of half-planes in each subproblem can be found recursively; the two intersections can then be merged in linear time [11]. A binary search for the slope of the objective function then answers each query in  $O(\log n)$  time.

As before, we resort to the geometric dual to solve the problem. We may again assume without loss of generality that the feasible region  $R_L$  is nonempty and contains the origin. Each of the  $n$  constraints defines a half-plane  $H_i$ ;  $R_L$  is the intersection of these half-planes. Using the notation of § 4.1, the dual of  $R_L$  is the exterior of  $CH(P)$ .

To begin with, we will assume that  $R_L$  is bounded. This implies that the origin in the dual plane lies in  $CH(P)$ . The objective function  $f_i$  can be looked upon as a family of parallel lines in the primal. Depending on the slope of  $f_i$ , we need only consider the set of parallel lines above or below the origin. This set of lines dualizes to a semi-infinite straight line with the origin as one endpoint. We call this the objective line  $g_i$ , and note that it intersects the boundary of  $CH(P)$  at one point which corresponds to the optimum solution.

The search tree and node expansion are as in § 4.2. While searching at a node  $v$ , we compute the intersection, if any, of  $g_i$  with the median line of  $P(v)$ . If there is no intersection or if the point of intersection does not lie between the tangents, the search proceeds to the left (right) child of  $v$  if the origin lies to the left (right) of the median line. Otherwise, we proceed in the opposite direction. The search terminates if  $g_i$  intersects a tangent of  $P(v)$ .

When  $R_L$  is unbounded, the origin in the dual plane does not lie in  $CH(P)$ . If  $g_i$  does not intersect  $CH(P)$ , the solution to the problem is unbounded. This can be detected by computing in  $O(n)$  time the polar angle from the origin to all points in  $P$ ; this is done once, at the beginning. If  $g_i$  lies outside the cone defined by this range of angles, it does not intersect  $CH(P)$ . If  $g_i$  intersects  $CH(P)$ , we use the same search procedure as in the bounded case. The two points in  $BCH(P)$  which subtend the extreme angles at the origin are joined by a tangent. Intersection with this tangent is ignored for the termination criterion above.

Figure 4 shows an unbounded feasible region, and the corresponding convex hull in the dual. Two objective functions  $f_1$  and  $f_2$  and their dual objective lines are shown. The arc in the dual indicates the locus of objective lines (e.g.,  $g_2$ ) that do not intersect  $CH(P)$ , and hence have unbounded optima.

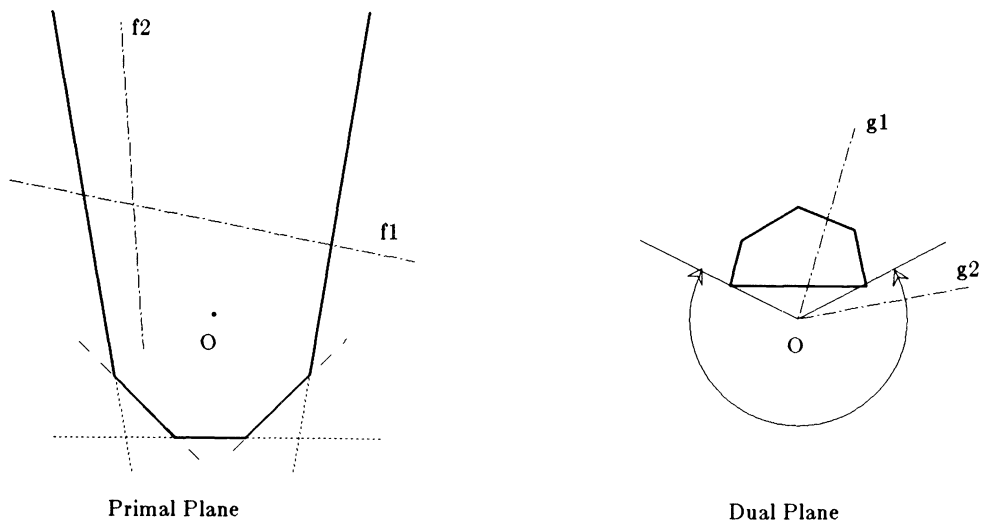


FIG. 4. Unbounded linear-programming search example.

**THEOREM 7.** *The number of operations for processing  $r$  two-variable linear programming queries is  $O(\Lambda(n, r))$ .*

**4.4. Lower bounds under the algebraic tree model.** The information-theoretic lower bound of § 2 is not valid for the geometric problems we have been considering in this section. In § 2 we were working with the comparison-tree model of computation, whereas we are allowing arithmetic operations here. We therefore use the *algebraic tree* model of computation [1].

An algebraic computation tree is an algorithm to decide whether an input vector, a point in  $\mathbf{R}^n$ , lies in a point set  $W \subseteq \mathbf{R}^n$ . The nodes in the tree are of three types: *computation* nodes, *branching* nodes, and *leaves*. A computation node has exactly one child and it can perform one of the usual arithmetic operations or compute a square root. A branching node behaves like a node in a comparison tree, i.e., it can perform comparisons with previously computed values. It has exactly two children corresponding to the possible outcomes of the comparison. A leaf is labeled either “Accept” or “Reject,” and it has no children. Each addition operation, subtraction operation or multiplication by a constant costs zero. Every other operation or comparison has a unit cost. The complexity of an algebraic computation tree is the maximum sum of costs along a root-leaf path in the tree. If  $W \subseteq \mathbf{R}^n$ , then  $C(W)$ , the complexity of  $W$ , is the minimum complexity of a tree that accepts precisely the set  $W$ . For any point set  $S \subseteq \mathbf{R}^n$ , let  $\#(S)$  denote the number of connected components of  $W$ . It was shown in [1] that  $C(W) = \Omega(\log \#(W))$ .

We now show a lower bound of  $\Omega((n+r) \cdot \log \min \{n, r\})$  algebraic operations for processing  $r$  hull-membership queries on  $n$  data points. We will in fact show that this bound holds when the  $r$  queries are processed *off-line*. The bound is obtained through a reduction from the SET DISJOINTNESS problem, defined as follows. Given two sets  $X = \{x_1, x_2, \dots, x_n\}$  and  $Q = \{q_1, q_2, \dots, q_r\}$ , determine whether their intersection is nonempty. This problem is a simpler version of the SET INTERSECTION problem mentioned in § 2. We first prove a lower bound on SET DISJOINTNESS.

**THEOREM 8.** *Any algebraic computation tree that solves SET DISJOINTNESS must have a complexity of  $\Omega((n+r) \cdot \log \min \{n, r\})$ .*

*Proof.* Assume without loss of generality that  $r \leq n$ . Every instance of SET DISJOINTNESS can be represented as a point  $\beta = (x_1, \dots, x_n, q_1, \dots, q_r)$  in  $\mathbf{R}^{n+r}$ . Let  $W \subseteq \mathbf{R}^{n+r}$  be the set of all points representing disjoint sets. The complexity of the problem is  $\Omega(\log \#(W))$ , where  $\#(W)$  is the number of connected components of  $W$  [1]. Consider instances for which the  $q_i$  are distinct. The elements of  $Q$  can be ordered as  $\{q_{(1)} < q_{(2)} < \dots < q_{(r)}\}$ , where  $(i)$  represents the index of the  $i$ th smallest value in  $\{q_1, \dots, q_r\}$ . Let  $S_\beta(i) = \{x_k : q_{(i)} < x_k < q_{(i+1)}\}$ , for  $1 \leq i \leq r-1$ . Define  $W^* = \{\beta : |S_\beta(i)| = \lfloor n/(r-1) \rfloor, 1 \leq i \leq r-1\}$ ,  $W^* \subseteq W$ . The subsets of  $W^*$  corresponding to different choices of  $S_\beta$ 's are separated by hyperplanes of the form  $x_i = q_j$ . These hyperplanes are entirely disjoint from  $W$ . This means that if two points in  $W^*$  are separated by these hyperplanes then they must also be separated in  $W$ . Hence, the number of components of  $W$  is at least as large as the number of ways of partitioning  $\{x_1, x_2, \dots, x_n\}$  into the  $S_\beta$ 's as per the definition of  $W^*$ . A counting argument shows that this is at least as large as

$$r! \frac{n!}{(\lfloor n/r \rfloor!)^{r-1}}.$$

From this it follows that the complexity is  $\Omega((n+r) \cdot \log r)$ .  $\square$

**THEOREM 9.** *The complexity of processing  $r$  hull-membership queries is  $\Omega((n+r) \cdot \log \min \{n, r\})$ .*



*Proof.* By reduction from SET DISJOINTNESS in  $O(n+r)$  time. Without loss of generality, assume that the elements of both sets lie in the interval  $[0, 2\pi)$ . Each element  $x_i$  maps onto a point  $p_i$  on the unit circle with polar coordinates  $(1, x_i)$ . This constitutes our data set  $P$ ; note that  $\text{CH}(P) = P$ . Each element  $q_j$  of  $Q$  maps onto a point  $r_j$  with polar coordinates  $(1, q_j)$ . The point  $r_j$  lies in  $\text{CH}(P)$  if and only if  $q_j \in X$ . Thus SET DISJOINTNESS  $\alpha_{n+r}$  HULL-MEMBERSHIP.  $\square$

The lower bound extends to the problems in §§ 4.2 and 4.3.

**4.5. Effect of the number of points on the convex hull.** In this section we return to the problem of determining whether a query point lies within the convex hull of  $n$  given data points. We show that a substantial improvement is possible when  $h$ , the number of data points on the boundary of the convex hull, is much smaller than  $n$ . It is clear that the guarantees of Theorem 4 are too weak in such a case, since it is possible to find  $\text{CH}(P)$  in  $O(n \log h)$  operations by the Kirkpatrick-Seidel algorithm; subsequently, queries can be answered in time  $O(\log h)$  each. This gives a time bound of  $O((n+r) \log h)$  for answering  $r$  queries. This may seem to contradict the lower bound of Theorem 9 but recall that in the lower bound reduction all  $n$  data points were on the boundary of the convex hull. When  $r$  exceeds  $h$ , the algorithm of § 4.1 achieves a time bound of  $O(n \log h + r \log n)$ , since node expansion costs add up to only  $O(n \log h)$ . The cost of searching, however, unfortunately grows as  $r \log n$  because the depth of  $T_P$  may grow as  $\log n$  even though the number of leaves is only  $h$ .

To get around this difficulty we construct, in a dovetailed fashion, two binary search trees  $T_P$  and  $T_D$ . Let  $T$  be the fully expanded version of the search tree constructed by the algorithm of § 4.1. It has  $h$  leaves and can be constructed in  $O(n \log h)$  time. The two trees  $T_P$  and  $T_D$  will be partially expanded versions of  $T$ .  $T_P$  is the version obtained by processing queries according to the algorithm of § 4.1. The other tree  $T_D$  is obtained by partially constructing  $T$  through a deferred depth-first traversal.

The depth-first traversal of a tree with  $l$  leaves can be looked upon as consisting of  $l$  phases, each of which ends when a new leaf is reached. Similarly, the depth-first construction of  $T_D$  can be broken down into  $h$  phases. These  $h$  phases are interleaves with the processing of the first  $h$  queries on the search tree  $T_P$ . Each phase can also be looked upon as the processing of a judiciously chosen query on the tree  $T_D$ . Thus the cost of the deferred construction of  $T_D$  has the same upper bound as that for  $T_P$ .

When  $r$  exceeds  $h$ , the tree  $T_D$  will be fully constructed after the first  $h$  queries have been processed on  $T_P$ . At this point  $T_P$  itself may not be fully expanded; in fact only one leaf may have been exposed in it. Since the  $\text{CH}(P)$  is now completely determined by  $T_D$  we can do away with the two search trees for further query processing. We now resort to the *wedge* method to answer each query in time  $O(\log h)$  (see § 4.1). Since the cost of constructing  $T_D$  is  $O(n \log h)$  the following theorem results.

**THEOREM 10.** *The cost of processing  $r$  hull-membership queries is  $O(\Lambda'(n, r, h))$ , where*

$$\Lambda'(n, r, h) = \begin{cases} n \log r, & r \leq h, \\ (n+r) \cdot \log h, & r > h. \end{cases}$$

Analogous results hold for the problems in §§ 4.2 and 4.3.

**5. Domination problems.** In this section we investigate a problem related to *point domination* in  $k$ -dimensional space. This problem does not fit directly into the paradigm presented at the end of § 2. However, a higher-dimensional analogue of divide-and-conquer enables us to adapt our technique to such problems.

Let  $p_i$  denote the  $i$ th coordinate of a point  $p$  in  $k$ -space. We say that  $p$  dominates  $q$  if and only if  $p_i \geq q_i$  for all  $i, 1 \leq i \leq k$ . Bentley [2] considers the *dominance counting* problem which is also called the *ECDF Searching Problem*. In this problem we are given a set  $P = \{p_1, p_2 \dots p_n\}$  of  $n$  points in  $k$ -space. For each query point  $q$ , we are asked to report the number of points of  $P$  dominated by  $q$ .

Bentley uses a *multidimensional divide-and-conquer* strategy to solve this problem. He constructs a data structure, the *ECDF tree*, which answers each query in  $O(\log^k n)$  time following a preprocessing phase requiring  $O(n \log^{k-1} n)$  time. This result holds for fixed number of dimensions ( $k$ ) and for  $n$  a power of 2. However, a more detailed analysis due to Monier [9] shows the validity of this result for arbitrary  $n$  and  $k$ . In fact, Monier shows that the constant implicit in the  $O$  result is  $1/(k-1)!$ . In the following analysis we too will assume that the number of dimensions is fixed and that  $n$  is a power of 2. Our results can be generalized to allow for arbitrary  $k$  and  $n$  by invoking the results due to Monier.

The basic paradigm of multidimensional divide-and-conquer is as follows: given a problem involving  $n$  points in  $k$ -space, first divide into (and recursively solve) two subproblems each of  $n/2$  points in  $k$ -space, and then recursively solve one problem of at most  $n$  points in  $(k-1)$ -space. When applied to the dominance counting problem, this paradigm yields the following search or counting strategy:

(1) Find a  $(k-1)$ -dimensional hyperplane  $M$  dividing  $P$  into two subsets  $P_1$  and  $P_2$ , each of cardinality  $n/2$ . We will assume that  $M$  is of the form  $x_k = c$ . Hence, all points in  $P_1$  have their  $k$ th coordinate less than  $c$ , while those in  $P_2$  have their  $k$ th coordinate greater than  $c$ .

(2) If the query point  $q$  lies on the same side of  $M$  as  $P_1$  (i.e.,  $q_k < c$ ) then recursively search in  $P_1$  only. It is clear that the query point cannot dominate any point in  $P_2$ .

(3) Otherwise,  $q$  lies on the same side of  $M$  as  $P_2$  (i.e.,  $q_k > c$ ) and we know that  $q$  dominates every point of  $P_1$  in the  $k$ th-coordinate. Now we project  $P_1$  and  $q$  onto  $M$  and recursively search in  $(k-1)$ -space. We also search  $P_2$  in  $k$ -space.

In Fig. 5 we illustrate this strategy for two-dimensional space.

In one-dimensional space the ECDF searching problem reduces to finding the rank of a query value in the given data-set. The one-dimensional ECDF search tree is an optimal binary search tree on the  $n$  points in  $P$ . The  $k$ -dimensional ECDF tree for

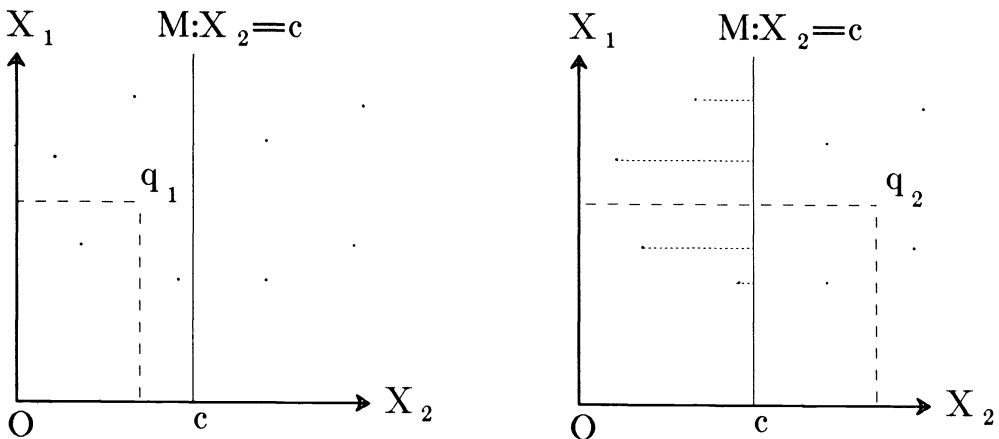


FIG. 5. The two cases for dominance counting in 2-space.

the  $n$  points in  $P$  is a recursively built data structure. The root of this tree contains  $M$ , the median hyperplane for the  $k$ th dimension. The left subtree is a  $k$ -dimensional ECDF tree for the  $n/2$  points in  $P_1$ , the points in  $P$  which lie below  $M$ . Similarly, the right subtree is a  $k$ -dimensional ECDF tree for the  $n/2$  points in  $P_2$ , the points in  $P$  which lie above  $M$ . The root also contains a  $(k - 1)$ -dimensional ECDF tree representing the points in the  $P_1$  projected onto  $M$ .

To answer a query  $q$ , the search algorithm compares  $q_k$  to  $c$ , the value defining the median plane  $M$  stored at the root. If  $q_k$  is less than  $c$  then the search is restricted to the points in  $P_1$  only. The algorithm then recursively searches in the left subtree. If, on the other hand,  $q_k$  is greater than  $c$  then the algorithm recursively searches in the right subtree as well as the  $(k - 1)$ -dimensional ECDF tree stored at the root. For the one-dimensional ECDF tree the algorithm is the standard binary tree search. For fixed  $k$ , the preprocessing time to build the  $k$ -dimensional ECDF tree is  $p(n) = O(n \log^k n)$ , and the time required to answer a single query is  $q(n) = O(\log^k n)$ .

We now apply the deferred data structuring technique to the  $k$ -dimensional ECDF tree. As before, we do not perform any preprocessing to construct the search tree. The ECDF tree is constructed on-the-fly in the process of answering the queries. Initially, all the points are stored at the root of the  $k$ -dimensional ECDF tree. In general, when a query search reaches an unexpanded node  $v$  we compute the median hyperplane,  $M_v$ , and partition the data points around  $M_v$ . The two sets are then passed down to the two descendant nodes of  $v$ . We also initialize the  $(k - 1)$ -dimensional ECDF tree which is to be created at  $v$ . Even these lower-dimensional trees are created in a deferred fashion depending upon the queries being answered. The application of deferred data structuring to the ECDF tree results in the following theorem.

**THEOREM 11.** *The cost of answering  $r$  dominance search queries in  $k$ -space is  $O(F(n, r, k))$ , where*

$$F(n, r, k) = \begin{cases} n \log^k r + r \log^k n, & r \leq n, \\ n \log^k n + r \log^k n, & r > n. \end{cases}$$

*Proof.* The proof will be by induction over both  $k$  and  $n$ . It is easy to see that the time required to answer a query remains unchanged by the process of deferring the construction of the ECDF tree. This proof will concentrate on the node-expansion component of the processing cost. Clearly, we need not consider the case where  $r > n$  since the node-expansion cost cannot exceed the total preprocessing cost of the nondeferred ECDF tree. Let  $f(n, r, k)$  denote the *worst-case* node-expansion cost for answering  $r$  queries over  $n$  data points in  $k$  dimensions using a  $k$ -dimensional ECDF tree. When  $r$  exceeds  $n$  we have  $f(n, r, k) = O(n \cdot \log^k n)$  since  $n$  queries, each leading to a different leaf, are sufficient to fully expand the ECDF tree. We will now prove that  $f(n, r, k) = O(n \cdot \log^k r)$  when  $r \leq n$ .

The basis of this induction is the case where  $k = 1$ . Consider the one-dimensional ECDF tree. It is an optimal binary search tree and we can invoke Theorem 3 to show the validity of this theorem. This establishes the base case of our induction over  $k$ , in other words,  $f(n, r, 1) = O(n \cdot \log r)$  when  $r \leq n$ . The induction hypothesis is that the above result is valid for up to  $k - 1$  dimensions, i.e.,  $f(n, r, k - 1) = O(n \cdot \log^{k-1} r)$  when  $r \leq n$ . We now prove that it must be valid for  $k$  dimensions also. At the second level of our nested induction we concentrate on the  $k$ -dimensional ECDF tree and use induction over  $n$ . It is clear that the  $k$ -dimensional ECDF tree for  $n = 1$  points will satisfy the above theorem for  $r \leq n$ . We now assume that the result is valid for up to  $n - 1$  points in  $k$  dimensions. To complete the proof we show that, under the given assumptions, the result can be extended to  $n$  points in  $k$  dimensions.

Consider the root node, say  $V$ , of the  $k$ -dimensional ECDF tree for the  $n$  points in  $P$ . It contains a median hyperplane, say  $M_V$ , which partitions the  $n$  points in  $P$  into two equal subsets,  $P_1$  and  $P_2$ . Recall that  $P_1$  is the set of all those points in  $P$  which lie below  $M_V$ ;  $P_2$  is the set of those points in  $P$  which lie above  $M_V$ . The left and right subtrees of  $V$  are the  $k$ -dimensional ECDF trees for  $P_1$  and  $P_2$ , respectively. We also store at  $V$  a  $(k-1)$ -dimensional ECDF tree, say  $T_1$ , for the projections of the points in  $P_1$  onto  $M_V$ . This lower-dimension tree creates a kind of asymmetry between  $P_1$  and  $P_2$ . This asymmetry can complicate our proof considerably. Therefore, for the purposes of this proof only, we will make a simplifying assumption about the structure of the ECDF tree. We assume that  $V$  also contains a  $(k-1)$ -dimensional ECDF tree, say  $T_2$ , for the projections of the points in  $P_2$  onto  $M_V$ .

The search procedure for the ECDF tree is also modified to introduce symmetry. Given a query  $q$ , we first test it with respect to the median hyperplane  $M_V$ . If it lies above  $M_V$  the search continues in the right subtree of  $V$  and in  $T_1$ . On the other hand, if  $q$  lies below  $M_V$  we continue the search in the left subtree of  $V$  as well as  $T_2$ . The search in  $T_2$  is redundant because  $q$ , lying below  $M_V$ , cannot dominate any point in  $P_2$ . These modifications are made not just at the root but at all nodes in an ECDF tree. It is not very hard to see that these modifications can only increase the running times of our node-expansion algorithm. Moreover, these changes entail performing redundant operations which do not change the outcome of our algorithm. It is clear, therefore, that any upper bounds on the node-expansion costs for the modified ECDF tree also apply to the original deferred data structure.

We now proceed to complete the induction proof for  $r \leq n$ . Let  $r_1$  denote the number of queries which lie below the median hyperplane  $M_V$ . These queries continue the search down the left subtree of the root. Let  $r_2 = r - r_1$  denote the remaining queries which continue the search down the right subtree as they lie above the median hyperplane  $M_V$ . Consider the node-expansion costs involved in processing these queries. Finding the median hyperplane  $M_V$  requires  $O(n)$  operations. The  $r_1$  queries lying below  $M_V$  are processed in the left subtree of  $V$  (a  $k$ -dimensional ECDF tree on  $n/2$  points) and in  $T_2$  (a  $(k-1)$ -dimensional ECDF tree on  $n/2$  points). The remaining  $r_2$  queries are processed in the right subtree of  $V$  (a  $k$ -dimensional ECDF tree on  $n/2$  points) and in  $T_1$  (a  $(k-1)$ -dimensional ECDF tree on  $n/2$  points). This gives us the following bound on the total node-expansion cost entailed by processing  $r$  queries:

$$f(n, r, k) = \max_{r_1+r_2=r} \left\{ f\left(\frac{n}{2}, r_1, k\right) + f\left(\frac{n}{2}, r_2, k\right) + f\left(\frac{n}{2}, r_1, k-1\right) + f\left(\frac{n}{2}, r_2, k-1\right) + O(n) \right\}.$$

Using the induction hypotheses we know the exact form of the functions on the right-hand side of the inequality. In particular, we know that these functions are convex. This implies that the right-hand side of the inequality is maximized when  $r_1 = r_2 = r/2$ . Putting together all this we have the desired result

$$f(n, r, k) = O(n \cdot \log^k r), \quad r \leq n.$$

Again, note that this result is valid only for fixed  $k$  and  $n$  a power of 2. The constant implicit in the  $O$  will, in general, depend on  $k$ . Monier's detailed analyses [9] of Bentley's algorithm also extends our result to arbitrary  $n$  and  $k$ .  $\square$

Bentley [2] actually has a slightly better bound on the preprocessing time for constructing ECDF trees. He makes use of a *presorting* technique to improve the bound

to  $O(n \cdot \log^{k-1} n)$  for  $k$ -dimensional ECDF trees on  $n$  points. He first sorts all  $n$  points by the first coordinate in  $O(n \cdot \log n)$  time. This ordering is maintained at every step, especially when dividing the points into two sets about a median hyperplane for some other coordinate. Consider the two-dimensional ECDF tree. Initially, all  $n$  points are stored at the root in *order by the first coordinate*. After the first query, these  $n$  points are partitioned about a median hyperplane and passed down to the children nodes. The ordering by the first coordinate is maintained during this partition. Let  $P_1$  denote the points being passed down to the left subtree,  $P_2$  denotes the points passed down to the right subtree. In the original ECDF tree we would have constructed a one-dimensional ECDF tree for the points in  $P_1$  and stored it at the root. Instead, we now just store the points of  $P_1$ , in order by the first coordinate, at the root. This process is repeated at every node in the two-dimensional ECDF tree. We now use the two-dimensional ECDF tree as the basic data structure in our recursive construction of a  $k$ -dimensional ECDF tree. In effect, we have done away with the one-dimensional ECDF tree. The preprocessing cost for constructing the presorted  $k$ -dimensional ECDF tree becomes  $O(n \cdot \log^{k-1} n) + O(n \cdot \log n)$ . The new data structure is as easily deferred as the previous one and we have the following result.

**THEOREM 12.** *The cost of answering  $r$  dominance search queries in  $k$ -space is  $O(G(n, r, k))$ , where*

$$G(n, r, k) = \begin{cases} n \log n + n \log^{k-1} r + r \log^k n, & r \leq n, \\ n \log^{k-1} n + r \log^k n, & r > n. \end{cases}$$

*Proof.* The proof follows from a straightforward modification of the proof for Theorem 11. Note that cost of presorting is subsumed by the node-expansion cost when  $r > n$ .  $\square$

**6. Conclusion.** The paradigm of deferred data structuring has been applied to some search problems. In all cases, we considered on-line queries and developed the search tree as queries were processed. For the problems studied, our method improves on existing strategies involving a preprocessing phase followed by a search phase. An interesting open problem is to design deferred data structures for dynamic data sets in which insertions and deletions are allowed concurrently with query processing.

The *nearest-neighbor problem* [13] asks for the nearest of  $n$  data points to a query point. The problem is solved using *Voronoi diagrams* in  $O(\log n)$  search time; the Voronoi diagram can be constructed in  $O(n \log n)$  time. There is no known top-down divide-and-conquer algorithm for constructing the Voronoi diagram optimally. The obvious top-down method of constructing the bisector of the left and the right  $n/2$  points (see [14] for a definition of the bisector of two sets of points) fails, since sorting reduces to computing this bisector. It remains an interesting open problem whether a deferred data structure can be devised for the nearest-neighbor search problem. Note that the techniques of § 2 can be used to solve the one-dimensional nearest-neighbor problem.

#### REFERENCES

- [1] M. BEN-OR, *Lower bounds for algebraic computation trees*, in Proc. 15th Annual ACM Symposium on Theory of Computing, May 1983, pp. 80–86.
- [2] J. L. BENTLEY, *Multidimensional divide and conquer*, Comm. ACM, 23 (1980), pp. 214–229.
- [3] M. BLUM, R. FLOYD, V. PRATT, R. RIVEST, AND R. TARJAN, *Time bounds for selection*, J. Comput. System. Sci., 7 (1973), pp. 448–461.

- [4] B. M. CHAZELLE, L. J. GUIBAS, AND D. T. LEE, *The power of geometric duality*, in Proc. 24th Annual IEEE Annual Symposium on Foundations of Computer Science, November 1983, pp. 217–225.
- [5] M. E. DYER, *Linear time algorithms for two- and three-variable linear programs*, SIAM J. Comput., 13 (1984), pp. 31–45.
- [6] D. G. KIRKPATRICK AND R. SEIDEL, *The ultimate planar convex hull algorithm?*, SIAM J. Comput., 15 (1986), pp. 287–299.
- [7] D. E. KNUTH, *The Art of Computer Programming: Sorting and Searching*, 3, Addison-Wesley, New York, 1973, pp. 217–219.
- [8] N. MEGIDDO, *Linear time algorithm for linear programming in  $R^3$  and related problems*, SIAM J. Comput., 12 (1983), pp. 759–776.
- [9] L. MONIER, *Combinatorial solutions of multidimensional divide-and-conquer recurrences*, J. Algorithms, 1 (1986), pp. 60–74.
- [10] R. MOTWANI AND P. RAGHAVAN, *Deferred data structures: query-driven preprocessing for geometric search problems*, in Proc. 2nd Annual ACM Symposium on Computational Geometry, Yorktown Heights, NY, June 1986, pp. 303–312.
- [11] F. P. PREPARATA AND M. I. SHAMOS, *Computational Geometry: An Introduction*, Springer-Verlag, Berlin, New York, 1985.
- [12] A. SCHÖNHAGE, M. PATERSON, AND N. PIPPENGER, *Finding the median*, J. Comput. System Sci., 13 (1981), pp. 184–199.
- [13] M. I. SHAMOS AND D. HOEY, *Closest-point problems*, in Proc. 16th Annual IEEE Annual Symposium on Foundations of Computer Science, October 1975, pp. 151–162.
- [14] M. I. SHAMOS, *Computational geometry*, Ph.D. thesis, Yale University, New Haven, CT, 1977.

## ON SETS TRUTH-TABLE REDUCIBLE TO SPARSE SETS\*

RONALD V. BOOK† AND KER-I KO‡

**Abstract.** We study sets that are truth-table reducible to sparse sets in polynomial time. The principal results are as follows: (1) For every integer  $k > 0$ , there is a set  $L$  and a sparse set  $S$  such that  $L \leq_{(k+1)\text{-tt}}^P S$ , but there is no sparse set  $S'$  such that  $L \leq_{k\text{-tt}}^P S'$ . (2) There exist a sparse set  $S$  and a set  $L$  such that  $L \leq_{\text{tt}}^P S$  but there is no integer  $k$  such that for some sparse  $S'$ ,  $L \leq_{k\text{-tt}}^P S'$ . (3) The class of sets that are bounded truth-table reducible to tally sets is equal to the class of sets that are many-one reducible to tally sets. (4) The class of sets having polynomial-size circuits is equal to the class of sets that are truth-table reducible to tally sets.

Similar results are developed for truth-table reducibilities that are computed nondeterministically in polynomial time or in polynomial space.

**Key words.** truth-table reducibilities, polynomial time, polynomial space, sparse sets, tally sets, hierarchies

**AMS(MOS) subject classifications.** 68Q15, 68Q05, 03D15, 03D30

**1. Introduction.** In recent years the class of sparse sets has received a great deal of attention by those researchers who study the structure of complexity classes and of complexity-bounded reducibilities. Perhaps the most prominent results are those that show how the complexity of sparse sets depends on the basic open questions of complexity theory such as the  $P = ?$  NP problem. Efforts by Berman [8], Fortune [14], and Mahaney [22] led to the following result: the class  $P$  is equal to the class NP if and only if there is a sparse set that is NP-complete [22]. Similarly, results by Karp and Lipton [16], Long and Selman [21], and Balcázar, Book, and Schöning [4] culminated in results such as the following: the class PSPACE is equal to the union (PH) of the classes in the polynomial-time hierarchy if and only if there exists a sparse set  $S$  such that PSPACE relativized to  $S$  is equal to PH relativized to  $S$  if and only if for all sparse sets  $S$ , PSPACE relativized to  $S$  is equal to PH relativized to  $S$  [4].

In the context of developing a structure theory for complexity classes and complexity-bounded reducibilities, we would like to develop intrinsic characterizations of reduction classes or degrees of a given set; i.e., for a set  $A$  characterize, in terms of properties other than reducibilities, the class of all sets  $B$  such that  $B \leq_m^P A$  or  $B \leq_T^P A$  or  $B \equiv_T^P A$ , etc. Similarly, we would like to develop intrinsic characterizations of reduction classes for a class of sets; i.e., for a class  $C$  of sets describe the class of all sets  $B$  such that for some  $C \in C$ ,  $B \leq_m^P C$ ,  $B \leq_T^P C$ , or  $B \equiv_T^P C$ . The results in the last paragraph suggest that this will be very difficult since, for example, the appropriate characterizations of the class of sets that are  $T \leq_m^P$ -reducible to sparse sets might require a solution of the  $P = ?$  NP problem. A few such characterizations are known: the class of sets that are  $\leq_T^P$ -reducible to sparse sets is the class of sets with polynomial-sized circuits (attributed to A. Meyer in [7]); the class of sets that are  $\leq_T^P$ -equivalent to tally sets is the class of sets with self-producible circuits (Balcázar and Book [2]);

---

\* Received by the editors September 29, 1986; accepted for publication (in revised form) October 8, 1987. This work was supported in part by the National Science Foundation under grants DCR-8312472, DCR-8501226, CCR-8611980, and CCR-8696135. The research reported here was performed while the second author was visiting the Department of Mathematics, University of California, Santa Barbara, California. The results presented here were announced at the Second Conference on Structure in Complexity Theory, June 1987.

† Department of Mathematics, University of California, Santa Barbara, California 93106.

‡ Department of Computer Science, State University of New York, Stony Brook, New York 11794-4400.

the class of sets that are  $p$ -isomorphic to tally sets is the class of sets with small generalized Kolmogorov complexity (Allender and Rubinfeld [1]). Other characterization theorems in terms of language-theoretic properties have been developed by Book [9], Book and Selman [12], and Balcázar, Díaz, and Gabarró [5].

The known characterizations described in the last paragraph as well as the apparent difficulty in obtaining additional characterizations led the authors to investigate properties of some of the polynomial-time reducibilities to sparse sets. In the current paper we report on the results of that investigation.

Consider the various polynomial-time reducibilities to sparse sets. Here we investigate the reduction classes of truth-table, bounded truth-table,  $k$ -truth-table for each positive integer  $k$ , and many-one reducibilities to sparse sets and also to tally sets. The main results may be summarized as follows:

(1) For every  $k > 0$ , there exist a sparse set  $S$  and a set  $L$  such that  $L \leq_{(k+1)\text{-tt}}^P S$ , but there is no sparse set  $S'$  such that  $L \leq_{k\text{-tt}}^P S'$ . Thus, the class of sets that are bounded truth-table reducible to sparse sets can be decomposed into a properly infinite hierarchy based on bounding the number of queries that are allowed.

(2) There exist a sparse set  $S$  and a set  $L$  such that  $L \leq_{\text{tt}}^P S$  but there is no integer  $k$  such that for some sparse set  $S'$ ,  $L \leq_{k\text{-tt}}^P S'$ . Thus the class of sets that are bounded truth-table reducible to sparse sets is properly included in the class of sets that are truth-table reducible to sparse sets.

(3) The class of sets that are bounded truth-table reducible to tally sets is equal to the class of sets that are many-one reducible to tally sets.

(4) The class of sets having polynomial-sized circuits is equal to the class of sets that are truth-table reducible to tally sets.

It should be noted that the sparse sets  $S$  and the sets  $L$  found in results (1) and (2) can be constructed in time  $2^{O(n)}$ .

The proofs of the main theorems are stronger than the theorems themselves. First, they allow us to extend the results of (1)–(4) to the setting of truth-table reducibilities that are computable nondeterministically in polynomial time where the functions that generate the list of queries and the truth-table condition are both computable nondeterministically but are single-valued. Second, they allow us to extend the results of (1)–(4) to the setting of truth-table reducibilities that are computable deterministically and also nondeterministically in polynomial space. In this case the polynomial space bounds and the fact that the reducibilities are nonadaptive yield the notion that only a polynomial number of queries are generated so that the reducibilities can be viewed as restrictions of the PQUERY ( ) and NPQUERY ( ) operators introduced by Book [10]. Third, they yield new proofs of results of Watanabe [27], [28] showing that no sparse set can be  $\leq_r^P$ -hard for the class of sets accepted deterministically in exponential time, where  $r$  is any of the bounded truth-table reducibilities.

The paper itself is organized in the following way. Section 2 is devoted to establishing notation and reviewing definitions. In § 3 we describe the relations between the reduction classes for the polynomial time-bounded truth-table reducibilities to sparse sets. Section 4 is devoted to the proofs of the theorems yielding (1) and (2), even though the theorems themselves are stated in § 3. In § 5 we point out how the relations described in § 3 can be extended to the cases of nondeterministic polynomial time-bounded truth-table reducibilities and of polynomial space-bounded truth-table reducibilities. In § 6 we describe the results about sparse sets and the class of sets accepted deterministically in exponential time.

## 2. In this section we review some definitions and establish notation.

Throughout this paper we will consider the alphabet  $\Sigma = \{0, 1\}$ . The length of a string  $x$  will be denoted by  $|x|$ . The cardinality of a set  $X$  will be denoted by  $\|X\|$ .



For a set  $X$  and an integer  $n$ ,  $X^n = \{x \in X \mid |x| = n\}$  and  $X^{\leq n} = \{x \in X \mid |x| \leq n\}$ . For a set  $X$ ,  $\chi_X$  denotes the characteristic function of  $X$ , and  $\bar{X} = \Sigma^* - X$ .

For an oracle machine  $M$ ,  $L(M, A)$  denotes the set of strings accepted by  $M$  relative to oracle set  $A$ , and  $L(M) = L(M, \emptyset)$  denotes the set of strings accepted by  $M$  when no oracle queries are made (or allowed). We assume that the reader is familiar with the well-studied complexity classes  $P$ ,  $NP$ , and  $PSPACE$  and with their relativizations. In particular, recall that set  $A$  is Turing-reducible to set  $B$  in polynomial time, written  $A \leq_T^P B$ , if  $A \in P(B)$ .

We are particularly concerned with truth-table reducibilities that are computed in polynomial time. Recall the following definitions (see [19] for the details and formal definitions):

(i) Set  $A$  is many-one reducible to set  $B$ , written  $A \leq_m^P B$ , if there is a function  $f$  that can be computed in polynomial time with the property that for all  $x$ ,  $x \in A$ , if and only if  $f(x) \in B$ .

(ii) For every  $k > 0$ , set  $A$  is  $k$ -truth-table reducible to set  $B$ , written  $A \leq_{k-||}^P B$ , if there exist polynomial-time computable functions  $f$  and  $g$  such that for all  $x$ ,  $f(x)$  is a list of  $k$  strings,  $g(x)$  is a truth table with  $k$  variables, and  $x \in A$  if and only if the truth-table  $g(x)$  evaluates to **true** on the  $k$ -tuple  $\langle \chi_B(y_1), \dots, \chi_B(y_k) \rangle$ , where  $f(x) = \langle y_1, \dots, y_k \rangle$ .

(iii) Set  $A$  is bounded truth-table reducible to set  $B$ , written  $A \leq_{b||}^P B$ , if there is an integer  $k$  such that  $A \leq_{k-||}^P B$ .

(iv) Set  $A$  is truth-table reducible to set  $B$ , written  $A \leq_{||}^P B$ , if there exist polynomial-time computable functions  $f$  and  $g$  such that for all  $x$ ,  $f(x)$  is a list of strings,  $g(x)$  is a truth table with the number of variables being equal to the number of strings in the list  $f(x)$ , and  $x \in A$  if and only if the truth-table  $g(x)$  evaluates to **true** on  $\langle \chi_B(y_1), \dots, \chi_B(y_k) \rangle$ , where  $f(x) = \langle y_1, \dots, y_k \rangle$ .

In the above definitions we did not specify how a truth-table  $g(x)$  with  $k$  variables is represented, because the representation does not matter as long as it evaluates each truth value in polynomial time (see [19] for several equivalent formulations). What is important to notice is that the polynomial-time truth-table reducibility is equivalent to nonadaptive polynomial-time Turing reducibility. In other words, in the above definitions (ii) and (iv), we may regard the truth-table evaluator  $g$  as a  $(k+1)$ -variable function, with the first variable  $x$  in  $\Sigma^*$  and the rest  $b_1, \dots, b_k$  in  $\{0, 1\}$ . The function  $g$  is required to run in polynomial time and satisfy the condition that  $g(x, b_1, \dots, b_k) = 1$  if and only if the intended "truth-table"  $g(x)$  evaluates to **true** on values  $\langle b_1, \dots, b_k \rangle$ .

For any truth-table reducibility  $r$  computed in polynomial time and any class  $\mathcal{C}$  of sets, let  $P_r(\mathcal{C}) = \{A \mid \text{there exists } C \in \mathcal{C} \text{ such that } A \leq_r^P C\}$ . Also, for any reducibility  $r$  computed in polynomial time,  $A \equiv_r^P B$  denotes the fact that  $A \leq_r^P B$  and  $B \leq_r^P A$  so that for any class  $\mathcal{C}$  of sets, we let  $\equiv_r^P(\mathcal{C})$  denote  $\{A \mid \text{there exists } C \in \mathcal{C} \text{ such that } A \equiv_r^P C\}$ . In addition, we write  $P_T(\mathcal{C})$  for  $P(\mathcal{C})$ . In § 5 we will also consider truth-table reducibilities computed nondeterministically in polynomial time and truth-table reducibilities computed in polynomial space; similar notation will be used.

Recall that a set  $S$  is *sparse* if there is a polynomial  $q$  such that for all  $n$ ,  $\|S^{\leq n}\| \leq q(n)$ . Let SPARSE denote the class of all sparse sets. Recall that a *tally* set is any subset of  $\{0\}^*$ . Let TALLY denote the class of all tally sets.

For any  $A$  and  $B$ , the symmetric difference  $(A - B) \cup (B - A)$  is denoted  $A \Delta B$ .

**3.** In this section we study the relationships between certain classes that contain every set in  $P$  and every sparse set. These relationships are displayed in Figs. 1 and 2.

Consider the class  $P$  and the class SPARSE. Clearly, these two classes are not comparable. It is well known that both the class  $P$  and the class SPARSE are properly

included in the class  $P_m(\text{SPARSE})$ . Also, it is clear that both  $P$  and  $\text{SPARSE}$  are properly included in the Boolean closure of their union. This latter class is of independent interest.

Schöning [25] considered sets  $A$  of the form  $A = B \triangle S$ , where  $B \in P$  and  $S \in \text{SPARSE}$ . Since  $A = B \triangle S$ ,  $S = A \triangle B$  so that  $S$  being sparse implies that for some polynomial  $h$  and all  $n$ ,  $\|(A \triangle B)^{\leq n}\| \leq h(n)$ . Such a set  $A$  is called *polynomially close to  $P$* . Schöning proved that a set is polynomially close to  $P$  if and only if it is in the

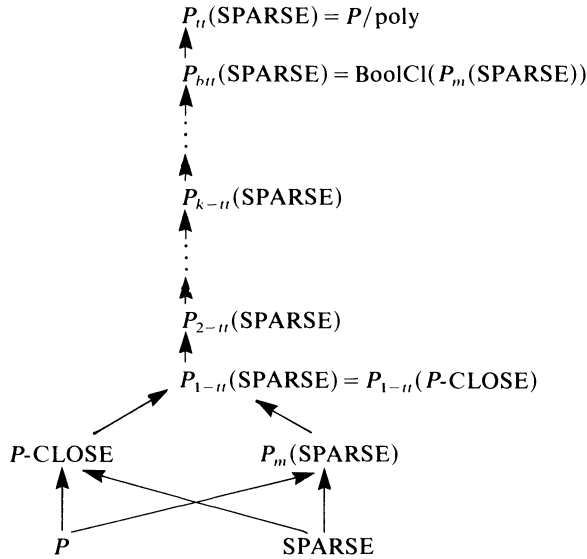


FIG. 1.  $L_1 \rightarrow L_2$  represents  $L_1 \subset L_2$ .

Boolean closure of  $P \cup \text{SPARSE}$ . Here, we refer to the class of sets that are polynomially close to  $P$  as the class  $P\text{-CLOSE}$ , so  $P\text{-CLOSE} = \{A \mid \text{for some } B \in P \text{ and } S \in \text{SPARSE}, A = B \triangle S\} = \{A \mid \text{for some } B \in P, A \triangle B \text{ is sparse}\}$ .

Since  $P \cup \text{SPARSE}$  is properly included in both  $P_m(\text{SPARSE})$  and  $P\text{-close}$ , it is appropriate to try to compare these two classes.

PROPOSITION 3.1. *The classes  $P_m(\text{SPARSE})$  and  $P\text{-CLOSE}$  are not comparable.*

*Proof.* First, we show that there is a set in  $P_m(\text{SPARSE})$  that is not in  $P\text{-CLOSE}$ . The proof is taken from that of Proposition 3.4 of Schöning [25].

Let  $T \subseteq \{0\}^*$  be a set not in  $P$  and let  $A = \{x \in \{0, 1\}^* \mid 0^{|x|} \in T\}$ . Clearly,  $T \in P\text{-CLOSE} \cap P_m(\text{SPARSE})$  and  $A \equiv_m^P T$  so that  $A \in P_m(\text{SPARSE})$ . Now  $A$  is not in  $P\text{-CLOSE}$ ; otherwise, there would be a set  $B \in P$  and a polynomial  $h$  such that for all  $n$ ,  $\|(A \triangle B)^{\leq n}\| \leq h(n)$ . Then the following algorithm would recognize  $T$  in polynomial time, contradicting the choice of  $T$ .

```

input  $0^n$ ;
determine how many of the first  $2h(n) + 1$  strings of size  $n$ 
    (in lexicographic order) are in  $B$ ;
if this number exceeds  $h(n)$  then accept else reject.
    
```

Second, we show that there is a set in  $P\text{-CLOSE}$  that is not in  $P_m(\text{SPARSE})$ . This was pointed out by Dr. José Balcázar [29]. Recall that a set  $A$  is *strongly bi-immune* for  $P$  if every  $\equiv_m^P$ -reduction from  $A$  is one-to-one almost everywhere. It is clear that

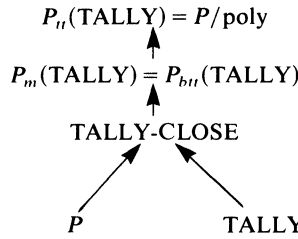


FIG. 2.  $L_1 \rightarrow L_2$  represents  $L_1 \subset L_2$ .

the class of strongly bi-immune sets is closed under complementation. Balcázar and Schöning [6] have shown that there exists a sparse set  $S$  that is strongly bi-immune; hence,  $\bar{S}$  is also strongly bi-immune. Since  $\bar{S}$  is strongly bi-immune, every  $\leq_m^P$ -reduction from  $\bar{S}$  is one-to-one almost everywhere. Hence,  $\bar{S}$  cannot be many-one reducible to a sparse set, i.e.,  $\bar{S}$  is not in  $P_m(\text{SPARSE})$ . But since  $S$  is sparse,  $\bar{S}$  is in the Boolean closure of SPARSE and, hence, in  $P\text{-CLOSE}$ .  $\square$

**COROLLARY.** *The class  $P_m(\text{SPARSE})$  is not closed under the Boolean operations.*

*Proof.* If  $P_m(\text{SPARSE})$  were closed under the Boolean operations, then  $P \cup \text{SPARSE} \subseteq P_m(\text{SPARSE})$  would imply that the Boolean closure of  $P \cup \text{SPARSE}$  would be included in  $P_m(\text{SPARSE})$ . But the Boolean closure of  $P \cup \text{SPARSE}$  is  $P\text{-CLOSE}$ , so this would contradict Proposition 3.1.  $\square$

The reducibility  $\leq_{1-tt}^P$  is weaker than the reducibility  $\leq_m^P$ . Thus, we would expect the class  $P_m(\text{SPARSE})$  to be properly included in the class  $P_{1-tt}(\text{SPARSE})$ . The next result shows that this is true and also shows that the class  $P\text{-CLOSE}$  is properly included in the class  $P_{1-tt}(\text{SPARSE})$ .

**PROPOSITION 3.2.** *Both of the classes  $P_m(\text{SPARSE})$  and  $P\text{-CLOSE}$  are properly included in  $P_{1-tt}(\text{SPARSE})$ .*

*Proof.* From the definition it follows that  $P_m(\text{SPARSE})$  is included in  $P_{1-tt}(\text{SPARSE})$ . Thus, we show first that  $P\text{-CLOSE}$  is included in  $P_{1-tt}(\text{SPARSE})$ .

Let  $A \in P\text{-CLOSE}$  so that there exist  $B \in P$  and  $S \in \text{SPARSE}$  such that  $A = B \Delta S$ . Let  $M_0$  be a polynomial-time-bounded deterministic machine that recognizes  $B$ . Consider the following algorithm.

```

input  $x$ ;
determine whether  $x$  is in  $B$  by running  $M_0$  on  $x$ ;
if  $x \in B$  then query the oracle about  $x$ 
    if the answer is "no" then accept
    else reject
else query the oracle about  $x$ 
    if the answer is "yes" then accept
    else reject
end.
    
```

It is clear that this algorithm operates in polynomial time and that, relative to the set  $S$ , it recognizes  $A = (B - S) \cup (S - B)$ . Hence, this algorithm witnesses  $A \leq_{1-tt}^P S$ .

Finally, we note that the fact that  $P\text{-CLOSE}$  and  $P_m(\text{SPARSE})$  are not comparable shows that both of the inclusions  $P\text{-CLOSE} \subseteq P_{1-tt}(\text{SPARSE})$  and  $P_m(\text{SPARSE}) \subseteq P_{1-tt}(\text{SPARSE})$  must be proper.  $\square$

In the proof of Proposition 3.2, it is shown that  $P$ -CLOSE is included in  $P_{1-ii}(\text{SPARSE})$ . This is done by arguing that if  $A = B \Delta S$ , where  $B$  is in  $P$ , then  $A \leq_{1-ii}^P S$ ; the fact that  $S \in \text{SPARSE}$  plays no role in that argument. Now  $A = B \Delta S$  if and only if  $S = B \Delta A$ , so that it is also the case that  $S \leq_{1-ii}^P A$  and, hence,  $A \equiv_{1-ii}^P S$ . We conclude that  $P$ -CLOSE is included in  $\equiv_{1-ii}^P(\text{SPARSE})$ , and the proof of Proposition 3.1 shows that this inclusion is proper. Since  $\text{SPARSE} \subset P\text{-CLOSE} \subset \equiv_{1-ii}^P(\text{SPARSE}) \subseteq P_{1-ii}(\text{SPARSE})$  and both  $\equiv_{1-ii}^P$  and  $\leq_{1-ii}^P$  are transitive, we conclude that  $\equiv_{1-ii}^P(P\text{-CLOSE}) = \equiv_{1-ii}^P(\text{SPARSE})$  and  $P_{1-ii}(P\text{-CLOSE}) = P_{1-ii}(\text{SPARSE})$ .

There is a useful result due to Köbler [17] (see also Köbler, Schöning, and Wagner [18]).

**PROPOSITION 3.3.** *For every set  $A$ , the class of sets  $B$  such that  $B \leq_{bit}^P A$  is precisely the Boolean closure of the class  $\{C \mid C \leq_m^P A\}$ .*

**COROLLARY.** *The class  $P_{bit}(\text{SPARSE})$  is equal to the Boolean closure of  $P_m(\text{SPARSE})$ .*

The reader may note that the analogue of Proposition 3.3 holds in recursive function theory, where  $\leq_{bit}$  and  $\leq_m$  are witnessed by total recursive functions.

From the Corollary we see that  $P_{bit}(\text{SPARSE})$  can also be expressed as the Boolean closure of  $P_{k-ii}(\text{SPARSE})$  for any  $k > 0$ . Thus, if  $P_{bit}(\text{SPARSE})$  can be decomposed into an infinite hierarchy based on the number of questions asked, i.e.,  $P_{k-ii}(\text{SPARSE}) \neq P_{(k+1)-ii}(\text{SPARSE})$  for all  $k$ , then there is no  $k > 0$  such that  $P_{k-ii}(\text{SPARSE})$  is closed under the Boolean operations. The fact that  $P_{bit}(\text{SPARSE})$  can be so decomposed is the main result of this paper.

**THEOREM 3.4.** *For every  $k > 0$ ,  $P_{k-ii}(\text{SPARSE}) \neq P_{(k+1)-ii}(\text{SPARSE})$ . Hence, for all  $k$ ,  $P_{k-ii}(\text{SPARSE}) \neq P_{bit}(\text{SPARSE})$  and  $P_{k-ii}(\text{SPARSE})$  is not closed under the Boolean operations.*

Theorem 3.4 will be proved in § 4.

Consider the class  $P_{ii}(\text{SPARSE})$ . From the definitions of  $\leq_{bit}^P$  and  $\leq_{ii}^P$ , it follows that  $P_{bit}(\text{SPARSE}) \subseteq P_{ii}(\text{SPARSE})$ . The fact that this inclusion is proper is the next result.

**THEOREM 3.5.** *The class  $P_{bit}(\text{SPARSE})$  is properly included in the class  $P_{ii}(\text{SPARSE})$ .*

Theorem 3.5 will be proved in § 4.

Consider the class  $P_T(\text{SPARSE})$ . Recall that  $P_T(\text{SPARSE}) = P/\text{poly}$ , where  $P/\text{poly}$  is the class of sets with polynomial-sized circuits studied by Karp and Lipton [16]. From the definitions of  $\leq_{ii}^P$  and  $\leq_T^P$ , it follows that  $P_{ii}(\text{SPARSE}) \subseteq P_T(\text{SPARSE})$ . We will see that this inclusion is not proper, that is,  $P_{ii}(\text{SPARSE}) = P_T(\text{SPARSE})$ . In this context, it is desirable to consider the class TALLY of tally sets and the classes  $P_{ii}(\text{TALLY})$  and  $P_T(\text{TALLY})$ .

The relationship between sparse sets and tally sets has been studied by many researchers. For example, Hartmanis [15] showed that every sparse set  $S$  is nondeterministically polynomial-time equivalent to some tally set, while Long [20] showed that this relationship does not hold for strong nondeterministic polynomial-time Turing reducibility. Schöning [24] proved that for every sparse set  $S$  there is a tally set  $T$  such that  $S$  is deterministically polynomial-time Turing-reducible to  $T$ . The following result relates sparse sets and tally sets by deterministic polynomial-time Turing- and truth-table reducibility. The reader should notice that it is not being claimed that for every sparse  $S$  there is a tally set  $T$  such that  $P(S) = P(T)$ ; indeed, Long [20] has shown that this is not the case.

**THEOREM 3.6.**  $P/\text{poly} = P_T(\text{SPARSE}) = P_{ii}(\text{SPARSE}) = P_{ii}(\text{TALLY}) = P_T(\text{TALLY})$ .

*Proof.* From the definitions of  $\leq_u^P$  and  $\leq_T^P$  and the fact that  $\text{TALLY} \subseteq \text{SPARSE}$ , it follows that  $P_u(\text{TALLY}) \subseteq P_u(\text{SPARSE}) \subseteq P_T(\text{SPARSE})$  and  $P_u(\text{TALLY}) \subseteq P_T(\text{TALLY}) \subseteq P_T(\text{SPARSE})$ . Recall that tally sets are self-printable in polynomial time, that is, there is a deterministic polynomial time-bounded oracle transducer that for any tally set  $T$  will compute on input  $0^n$  the list of strings in  $T$  of length at most  $n$ . Since  $\leq_u^P$  is the nonadaptive version of  $\leq_T^P$ , it follows that  $P_T(\text{TALLY}) \subseteq P_u(\text{TALLY})$ .

It is known that for every sparse set  $S$  there is a tally set  $T_S$  such that  $S \in P(T_S)$ . Since  $\leq_T^P$  is transitive, this means that  $P(S) \subseteq P(T_S)$ . Hence,  $P_T(\text{SPARSE}) \subseteq P_T(\text{TALLY})$ .  $\square$

The results in Theorem 3.6 suggest that the class of tally sets be considered with respect to truth-table reducibilities. The situation is quite different from that regarding the sparse sets.

**THEOREM 3.7.**  $P_m(\text{TALLY}) = P_{btt}(\text{TALLY})$ .

*Proof.* The proof will follow from Proposition 3.3 once it is shown that  $P_m(\text{TALLY})$  is closed under the Boolean operations. Closure under complementation follows from the fact that complementation is always taken relative to a tally set: if  $T \subseteq \{0\}^*$ , then  $\bar{T} = \{0\}^* - T$ . Closure under intersection follows from the fact that there is a polynomial-time computable pairing function  $\langle \cdot, \cdot \rangle : \{0\}^* \times \{0\}^* \rightarrow \{0\}^*$ , for if  $f$  witnesses  $A \leq_m^P T_1$  and  $g$  witnesses  $B \leq_m^P T_2$ , then  $x \in A \cap B$  if and only if  $\langle f(x); g(x) \rangle \in \{ \langle 0^m, 0^n \rangle \mid 0^m \in T_1, 0^n \in T_2 \}$ .  $\square$

**COROLLARY.** *Let  $r$  be any of the bounded truth-table reducibilities. Then there is a sparse set  $S$  such that there is no tally set  $T$  with  $S \leq_r^P T$ . Thus,  $P_r(\text{TALLY}) \neq P_r(\text{SPARSE})$ .*

*Proof.* Since  $P_m(\text{TALLY}) = P_{btt}(\text{TALLY})$ ,  $P_r(\text{TALLY}) = P_m(\text{TALLY})$ , and  $P_r(\text{TALLY})$  is closed under the Boolean operations. If  $r \neq btt$ , then  $P_r(\text{SPARSE})$  is not closed under the Boolean operations by Theorem 3.4, so  $P_r(\text{SPARSE}) \neq P_r(\text{TALLY})$ . Thus,  $P_{btt}(\text{TALLY}) = P_m(\text{TALLY}) \subset P_m(\text{SPARSE}) \subset P_{btt}(\text{SPARSE})$  and so  $P_{btt}(\text{SPARSE}) \neq P_{btt}(\text{TALLY})$ .  $\square$

Recall that a set  $A$  is in  $P$ -CLOSE if and only if there is a set  $B \in P$  and a set  $S \in \text{SPARSE}$  such that  $A = B \triangle S$ . This suggests that we consider the class of sets  $A$  such that there is a set  $B \in P$  and a set  $T \in \text{TALLY}$  such that  $A = B \triangle T$ . We refer to this class as TALLY-CLOSE. It follows from a result of Schöning [25] that TALLY-CLOSE is the Boolean closure of  $P \cup \text{TALLY}$ . From Theorem 3.7 it follows that TALLY-CLOSE is included in  $P_m(\text{TALLY})$ . That the inclusion is in fact proper follows from the proof of Proposition 3.1.

The proof of Proposition 3.1 shows that there is a set in  $P_m(\text{TALLY})$  that is not in TALLY-CLOSE. The proof of Proposition 3.2 shows that TALLY-CLOSE is included in  $P_{1-ut}(\text{TALLY})$ . Since  $P_m(\text{TALLY}) = P_{1-ut}(\text{TALLY})$ , this shows that TALLY-CLOSE is properly included in  $P_m(\text{TALLY})$ .

Notice that similar to the situation with SPARSE, we have  $\equiv_m^P(\text{TALLY-CLOSE}) = \equiv_m^P(\text{TALLY})$  and  $P_m(\text{TALLY-CLOSE}) = P_m(\text{TALLY})$ .

**4.** In this section we will prove Theorems 3.4 and 3.5. The proof of Theorem 3.4 is carried out in two steps, the first establishing that  $P_{1-ut}(\text{SPARSE}) \neq P_{2-ut}(\text{SPARSE})$  and the second that for all  $k > 1$ ,  $P_{k-ut}(\text{SPARSE}) \neq P_{(k+1)-ut}(\text{SPARSE})$ .

It will be convenient to use the following notation: for each integer  $h > 0$ ,  $\text{SPARSE}_h$  is the class of sparse sets  $S$  such that for all  $m > 0$ ,  $\|S^{\equiv m}\| \leq p_h(m)$ , where  $p_h$  is the polynomial defined for all  $m$  by  $p_h(m) = m^h + h$ . Notice that for every sparse set  $S$  there is an integer  $h > 0$  such that  $S \in \text{SPARSE}_h$ , and for all integers  $h > 0$ ,  $\text{SPARSE}_h \subseteq \text{SPARSE}_{h+1}$ .

**THEOREM 4.1.**  $P_{1-tt}(\text{SPARSE}) \neq P_{2-tt}(\text{SPARSE})$ .

*Proof.* For any set  $A$ , let  $L_1(A) = \{x \mid \text{exactly one of } 0^{|x|+1} \text{ and } x \text{ is in } A\}$  so that  $L_1(A) \leq_{2-tt}^P A$ . We will construct a sparse set  $A$  such that for all sparse sets  $S$ ,  $L_1(A) \not\leq_{1-tt}^P S$ , i.e.,  $L_1(A)$  is in  $P_{2-tt}(\text{SPARSE})$  but is not in  $P_{1-tt}(\text{SPARSE})$ . The construction will be by stages.

Assume an enumeration of  $\leq_{1-tt}^P$ -reductions:  $\{M_{i,j} \mid i, j \geq 1\}$ , where for each pair  $i, j$ ,  $M_{i,j}$  is a polynomial-time-bounded oracle machine that for each input  $x$  generates the query string  $f_i(x)$  and the  $1-tt$ -evaluator  $g_j(x)$ . (Notice that for every  $j$  and every  $x$ ,  $g_j(x)$  is a function from  $\{0, 1\}$  to  $\{0, 1\}$ .)

For each triple  $i, j, h$  let  $R_{i,j,h}$  be the predicate that is true if and only if  $L_1(A) \neq L(M_{i,j}, S)$  for all  $S \in \text{SPARSE}_h$ . Our goal is to construct  $A$  such that  $A$  is sparse and, for every triple  $i, j, h$ ,  $R_{i,j,h}$  is satisfied. This will yield  $L_1(A) \notin P_{1-tt}(\text{SPARSE})$ .

Let  $\langle i, j, h \rangle$  be a ‘‘tripling’’ function that enumerates all of the triples  $i, j, h$ .

*Stage 0.* Let  $n_0 = 1$ . Let  $A_0 = \emptyset$ .

*Stage  $\alpha = \langle i, j, h \rangle > 0$ .* Let  $q$  be a polynomial that bounds the running time of  $M_{i,j}$ . Select  $n = n_\alpha$  such that  $n \geq n_{\alpha-1} + 2$  and  $2^{n-2} > p_h(q(n))$ . For each  $1-tt$ -condition  $t$ , let  $G_t = \{x \in \Sigma^n \mid g_j(x) = t\}$ . Since each  $g_j$  can take at most four possible values, there exists some  $t$  such that  $\|G_t\| \geq 2^{n-2}$ . Choose a fixed  $t$  with this property.

Consider each possible  $1-tt$ -condition  $t$ .

*Case 1.*  $t(0) = t(1) = 1$ . In this case, for all  $x \in G_t$ ,  $x \in L(M_{i,j}, S)$  so that  $L(M_{i,j}, S) \cap G_t = G_t$  for all  $S \in \text{SPARSE}_h$ . Let  $A_\alpha := A_{\alpha-1}$ . Then  $L_1(A_\alpha) \cap G_t = \emptyset$  and so  $L_1(A_\alpha)^n \neq L(M_{i,j}, S)^n$  for all  $S \in \text{SPARSE}_h$ .

*Case 2.*  $t(0) = t(1) = 0$ . In this case, for all  $x \in G_t$ ,  $x \in L(M_{i,j}, S)$  so that  $L(M_{i,j}, S) \cap G_t = \emptyset$  for all  $S \in \text{SPARSE}_h$ . Choose an arbitrary  $x$  in  $G_t$  and let  $A_\alpha := A_{\alpha-1} \cup \{x\}$ . Then  $L_1(A_\alpha) \cap G_t = \{x\} \neq \emptyset$  and so  $L_1(A_\alpha)^n \neq L(M_{i,j}, S)^n$  for all  $S \in \text{SPARSE}_h$ .

*Case 3.*  $t(0) = 0$  and  $t(1) = 1$ .

*Subcase 3.1.* Suppose that  $f_i$  is not one-to-one on  $G_t$ , so there exist  $x, y \in G_t$ ,  $x \neq y$ , and  $f_i(x) = f_i(y)$ . Let  $A_\alpha := A_{\alpha-1} \cup \{x\}$ . Then we have that for all  $S \in \text{SPARSE}_h$ ,  $x \in L(M_{i,j}, S)^n$  if and only if  $y \in L(M_{i,j}, S)^n$ , where  $x$  and  $y$  are as in the construction. But  $x \in L_1(A_\alpha)^n$  and  $y \notin L_1(A_\alpha)^n$  so that for all  $S \in \text{SPARSE}_h$ ,  $L_1(A_\alpha)^n \neq L(M_{i,j}, S)^n$ .

*Subcase 3.2.* Suppose that  $f_i$  is one-to-one on  $G_t$ . Let  $A_\alpha := A_{\alpha-1} \cup \{0^{n+1}\}$ . Then we have that  $L_1(A_\alpha)^n \cap G_t = G_t$ . But for all  $S \in \text{SPARSE}_h$ ,  $\|L(M_{i,j}, S) \cap G_t\| \leq \|S^{\leq q(n)}\| \leq \|p_h(q(n))\| < \|G_t\|$ . Hence, for all  $S \in \text{SPARSE}_h$ ,  $L_1(A_\alpha)^n \neq L(M_{i,j}, S)^n$ .

*Case 4.*  $t(0) = 1$  and  $t(1) = 0$ . This is symmetric to Case 3.

*End of Stage  $\alpha = \langle i, j, h \rangle$ .*

We have constructed  $A_\alpha$  such that for all  $\alpha$  and all  $n = n_\alpha$ ,  $L_1(A_\alpha)^n \neq L(M_{i,j}, S)^n$  for all  $S \in \text{SPARSE}_h$ . Letting  $A := \bigcup_\alpha A_\alpha$ , the choice of  $n_\alpha$  such that  $2^{n_\alpha-2} > p_h(q(n))$  guarantees that for all  $\alpha$ , no string added to  $A$  at an earlier or a later stage can cause the relationship established at stage  $\alpha$  to change, i.e., for all  $\alpha$  and  $n = n_\alpha$ ,  $L_1(A)^n = L_1(A_\alpha)^n$ . Thus, for all  $S \in \text{SPARSE}_h$ ,  $L(M_{i,j}, S) \neq L_1(A)$ .  $\square$

The proof of Theorem 4.1 may be viewed as the simplified version of the proof of Theorem 4.2 below.

**THEOREM 4.2.** For every  $k > 1$ ,  $P_{k-tt}(\text{SPARSE}) \neq P_{(k+1)-tt}(\text{SPARSE})$ .

*Proof.* Let  $k \geq 2$ . For any set  $A$ , let  $L_k(A) = \{u_1 \cdots u_k \mid \text{for each } i, 1 \leq i \leq k, |u_i| = |u_1| = n; \text{ the number of strings in the following list that are in the set } A \text{ is odd: } 0^{k(n+1)}, u_1 0^{(k-1)(n+1)}, \dots, u_1 \cdots u_{i-1} 0^{(k-i)(n+1)}, \dots, u_1 \cdots u_k\}$ . It is clear that  $L_k(A) \leq_{(k+1)-tt}^P A$ . We will describe a specific sparse set  $A$  such that for every sparse set  $S$  it is *not* the case that  $L_k(A) \leq_{k-tt}^P S$ . This will allow us to conclude that  $P_{(k+1)-tt}(\text{SPARSE}) \neq P_{k-tt}(\text{SPARSE})$ .

Let  $k \geq 2$  be fixed. Let  $\{f_i\}$  be an enumeration of the polynomial-time computable functions that for any string  $x$  yields a list of  $k$  strings (i.e., the list of queries). Let

$\{g_j\}$  be an enumeration of all polynomial-time computable functions that for any single string  $x$  yields one of the  $2^{2^k}$   $k$ -*tt*-conditions. We can enumerate all of the  $\leq_{k-tt}^P$ -reduction machines as  $\{M_{ij}\}$  where on input  $x$ ,  $M_{i,j}$  computes the list  $f(x) = \langle x_1, \dots, x_k \rangle$  and the  $k$ -*tt*-condition  $g_j(x)$ .

We want to construct a set  $A$  such that for every integer  $n$  and every integer  $r$ ,  $0 \leq r \leq k$ , there is at most one string in  $A$  of the form  $u_1 \cdots u_{k-r} 0^{r(n+1)}$  and there is at most one string in  $A$  having length  $|u_1 \cdots u_{k-r} 0^{r(n+1)}| = kn + r$ . Thus,  $A$  will be sparse. Furthermore,  $A$  must satisfy the condition that for all  $i, j$  and every sparse set  $S$ ,  $L_k(A) \neq L(M_{i,j}, S)$ ; to accomplish this latter point it is sufficient to show that for some  $n > 0$ ,  $L_k(A)^n \neq L(M_{i,j}, S)^n$ .

The construction of  $A$  will be by stages so that for all  $i, j$ , there are infinitely many opportunities to diagonalize against  $M_{i,j}$ . At stage  $\alpha = \langle i, j, h \rangle$ , the machine  $M_{i,j}$  and the polynomial  $p_h$  are considered. At the end of stage  $\alpha = \langle i, j, h \rangle$ , the following condition will be satisfied: for every sparse set  $S \in \text{SPARSE}_h$ ,  $L(M_{i,j}, S)^{kn} \neq L_k(A)^{kn}$ .

Let  $\langle i, j, h \rangle$  be a "tripling" function that enumerates all of the triples  $i, j, h$ . Let  $\varepsilon = 1/(16k^2)$ .

*Stage 0.* Let  $n_0 = 2^{k+3}$  and  $A_1 = \emptyset$ .

*Stage  $\alpha = \langle i, j, h \rangle > 1$ .* Let  $q$  be a polynomial that bounds the running time of  $M_{i,j}$ . Choose  $n$  such that  $n > kn_{\alpha-1} + 1$  and  $2^{\varepsilon n} > p_h(q(kn))$ . Let  $n_\alpha = n$ .

For each of the  $2^{2^k}$   $k$ -*tt*-conditions  $t$ , let  $G_t = \{x \in \Sigma^{kn} \mid g_j(x) = t\}$ . Then there must exist some  $t$  such that  $\|G_t\| \geq 2^{kn} / 2^{2^k} = 2^{kn-2^k}$ . Choose any fixed  $t$  with this property.

*Case 1.*  $f_i$  is not one-to-one on  $G_t$ .

In this case, there exist  $x, y \in G_t$  with  $x \neq y$  and  $f_i(x) = f_i(y)$ . Thus, for any set  $S \in \text{SPARSE}_h$ ,  $x \in L(M_{i,j}, S)$  if and only if  $y \in L(M_{i,j}, S)$ . Let  $A_\alpha := A_{\alpha-1} \cup \{x\}$ , and notice that  $y$  is not in  $A_\alpha$ . This means that  $x \in L_k(A_\alpha)$  and  $y \notin L_k(A_\alpha)$ , and so we conclude that for every  $S \in \text{SPARSE}_h$ ,  $L_k(A_\alpha)^{kn} \neq L(M_{i,j}, S)^{kn}$ .

*Case 2.*  $f_i$  is one-to-one on  $G_t$ .

Recall that  $f_i$  always outputs a list of  $k$  strings. Let  $f_i(x) = \langle x_1, \dots, x_k \rangle$ .

*Subcase 2.1.* There exist  $r$ ,  $1 \leq r \leq k$ , and  $z$  such that  $H_r(z) =_{\text{def}} \{x \in G_t \mid x_r = z\}$  has size  $\|H_r(z)\| \geq 2^{(k-(1/4))n}$ .

This implies that there exist  $u, v \in \Sigma^n$ ,  $u \neq v$ , such that  $\|H_r(z) \cap \{u\} \Sigma^{(k-1)n}\| \geq 2^{(k-\varepsilon-(5/4))n}$  and  $\|H_r(z) \cap \{v\} \Sigma^{(k-1)n}\| \geq 2^{(k-\varepsilon-(5/4))n}$ . (Choose  $u$  such that  $H_r(z) \cap \{u\} \Sigma^{(k-1)n}$  has the maximum size. Then its size is at least  $2^{(k-(1/4))n} \cdot 2^{-n} = 2^{(k-(5/4))n}$ . Choose  $v$  such that the size of  $H_r(z) \cap \{v\} \Sigma^{(k-1)n}$  is the next largest. Then  $\|H_r(z) \cap \{v\} \Sigma^{(k-1)n}\| \geq \|H_r(z) - \{u\} \Sigma^{(k-1)n}\| \cdot 2^{-n} \geq (2^{(k-(1/4))n} - 2^{(k-1)n}) \cdot 2^{-n} \geq 2^{(k-\varepsilon-(5/4))n}$ .)

Now we claim that we can extend  $A_{\alpha-1}$  to  $A_\alpha$  to satisfy the requirement that for any  $S \in \text{SPARSE}_h$ ,  $L(M_{i,j}, S)^{kn} \neq L_k(A_\alpha)^{kn}$ . Since the proof of this claim involves a long inductive argument, we state a technical lemma and leave the proof of the lemma to the end.

**LEMMA.** *Suppose there exist a set  $G \subseteq \Sigma^{kn}$ , a function  $f(x) = \langle x_1, \dots, x_{k-1} \rangle$  on  $G$  and two strings  $u, v \in \Sigma^n$  satisfying the following conditions:*

(1)  *$f$  is one-to-one on  $G$ , and for all  $r \leq k-1$ ,  $|x_r| \leq q(kn)$ .*

(2)  *$\|G \cap U\| \geq 2^{(k-\varepsilon-(5/4))n}$  and  $\|G \cap V\| \geq 2^{(k-\varepsilon-(5/4))n}$ , where  $U = \{u\} \Sigma^{(k-1)n}$  and  $V = \{v\} \Sigma^{(k-1)n}$ .*

*Then, there exists a set  $C \subseteq \{y \mid kn \leq |y| \leq k(n+1)\}$  such that for all  $r$ ,  $0 \leq r \leq k-1$ ,  $\|C \cap \Sigma^{kn+r}\| \leq 1$ , and having the property that for every set  $S \in \text{SPARSE}_h$ , and every  $(k-1)$ -*tt*-condition  $t$ , it is not the case that for all  $x \in G$ ,  $x \in L_k(C)$  if and only if  $t(\chi_S(x_1), \dots, \chi_S(x_{k-1})) = 1$ .*

Note that by letting  $G = H_r(z)$  and  $f(x) = \langle x_1, \dots, x_{r-1}, x_{r+1}, \dots, x_k \rangle$ , the hypothesis of the lemma is satisfied. Let  $t$  be the fixed  $k$ -*tt*-condition we chose such that  $\|G_t\| \geq 2^{kn-2^k}$ , and let  $t_0$  and  $t_1$  be the two  $(k-1)$ -*tt*-conditions induced by  $t$  by

removing the  $r$ th input, i.e., for any  $k-1$  bits of inputs  $b_1, \dots, b_{r-1}, b_{r+1}, \dots, b_k$ ,  $t_0(b_1, \dots, b_{r-1}, b_{r+1}, \dots, b_k) = 1$  if and only if  $t(b_1, \dots, b_{r-1}, 0, b_{r+1}, \dots, b_k) = 1$  and  $t_1(b_1, \dots, b_{r-1}, b_{r+1}, \dots, b_k) = 1$  if and only if  $t(b_1, \dots, b_{r-1}, 1, b_{r+1}, \dots, b_k) = 1$ . Then, by the lemma, for any  $S \in \text{SPARSE}_h$ , it is not the case that for all  $x \in H_r(z)$ ,  $x \in L_k(C)$  if and only if  $t_0(\chi_S(x_1), \dots, \chi_S(x_{r-1}), \chi_S(x_{r+1}), \dots, \chi_S(x_k)) = 1$ , nor the case that for all  $x \in H_r(z)$ ,  $x \in L_k(C)$  if and only if  $t_1(\chi_S(x_1), \dots, \chi_S(x_{r-1}), \chi_S(x_{r+1}), \dots, \chi_S(x_k)) = 1$ . Since  $x_r = z$  for all  $x \in H_r(z)$ , for any  $S \in \text{SPARSE}_h$ , depending upon whether or not  $z$  is in  $S$ , we have that  $t(\chi_S(x_1), \dots, \chi_S(x_k))$  is either equal to  $t_0(\chi_S(x_1), \dots, \chi_S(x_{r-1}), \chi_S(x_{r+1}), \dots, \chi_S(x_k))$  or equal to  $t_1(\chi_S(x_1), \dots, \chi_S(x_{r-1}), \chi_S(x_{r+1}), \dots, \chi_S(x_k))$ . This implies that it is not the case that for all  $x \in H_r(z)$ ,  $x \in L_k(C)$  if and only if  $t(\chi_S(x_1), \dots, \chi_S(x_k)) = 1$ . By letting  $A_\alpha := A_{\alpha-1} \cup C$ , we have established that  $L_k(A_\alpha)^{kn} \neq L(M_{i,j}, S)^{kn}$  for every  $S \in \text{SPARSE}_h$ .

*Subcase 2.2.* The conditions for Subcase 2.1 do not hold.

There are two sub-subcases.

*Subcase 2.2.1.*  $t(0, \dots, 0) = 0$ . For every set  $S \in \text{SPARSE}_h$  and every  $x \in G_t$ ,  $x \in L(M_{i,j}, S)$  implies that there exists  $r$ ,  $1 \leq r \leq k$ , with  $x_r \in S$ . Since  $|x_r| \leq q(|x|) = q(kn)$ , there are only  $P_h(q(kn)) < 2^{\epsilon n}$  such  $x_r$ . Thus,

$$\|L(M_{i,j}, S) \cap G_t\| \leq \sum_{r=1}^k \sum_{z \in S} \|H_r(z)\| < k \cdot 2^{\epsilon n} \cdot 2^{(k-1/4)n} < 2^{(k-(1/8))n}$$

(since  $k\epsilon = 1/(16k) < 1/8$ ). However, if  $A_\alpha := A_{\alpha-1} \cup \{0^{k(n+1)}\}$ , then every element of  $\Sigma^{kn}$  enters  $L_k(A_\alpha)$ . It follows that  $G_t \subseteq \Sigma^{kn} \subseteq L_k(A_\alpha)^{kn}$  and so  $\|L_k(A_\alpha)^{kn} \cap G_t\| \geq 2^{kn-2} > 2^{(k-(1/8))n}$ . This implies that  $L_k(A_\alpha) \neq L(M_{i,j}, S)$  for all  $S \in \text{SPARSE}_h$ .

*Subcase 2.2.2.*  $t(0, \dots, 0) = 1$ . Similarly to the argument in Subcase 2.2.1, we can conclude that for every  $S \in \text{SPARSE}_h$ ,  $\|\overline{L(M_{i,j}, S)} \cap G_t\| < 2^{(k-(1/8))n}$ . Let  $A_\alpha := A_{\alpha-1}$ . Then every element of  $\Sigma^{kn}$  enters  $\overline{L_k(A_\alpha)}$ . It follows that  $G_t \subseteq \Sigma^{kn} \subseteq \overline{L_k(A_\alpha)^{kn}}$  and so  $\|\overline{L_k(A_\alpha)^{kn}} \cap G_t\| = \|G_t\| > 2^{kn-2^k} > 2^{(k-(1/8))n}$ , which implies that  $L_k(A_\alpha) \neq L(M_{i,j}, S)$  for all  $S \in \text{SPARSE}_h$ .

This concludes the proof for Subcase 2.2 and, hence, for Case 2 and Stage  $\alpha$ .

Let  $A = \bigcup_{\alpha \geq 1} A_\alpha$ .

Since the justification for the stage construction of set  $A$  was given as the construction was carried out, we can conclude that for every choice of  $i, j, h$ , and every set  $S \in \text{SPARSE}_h$ ,  $L_k(A) \neq L(M_{i,j}, S)$ .  $\square$

Now we prove the lemma. We state and prove a stronger statement.

**LEMMA** (in stronger form). *For each  $j$ ,  $1 \leq j \leq k-1$ , the following holds. Suppose there exist a set  $G \subseteq \Sigma^{kn}$ , a function  $f(x) = \langle x_1, \dots, x_j \rangle$ , a constant  $c$ , and  $k-j+1$  strings  $u_1, \dots, u_{k-j-1}, u_{k-j}, v_{k-j}$  in  $\Sigma^n$  satisfying the following conditions:*

(0)  $\frac{1}{4} \leq c \leq \frac{1}{2} - 2jk\epsilon$ ;

(1)  $f$  is one-to-one on  $G$ , and for all  $r \leq j$ ,  $|x_r| \leq q(kn)$ ;

(2) if  $U_{j+1}$  is defined to be  $\{u_1 \dots u_{k-j-1}\} \Sigma^{(j+1)n}$  and  $U_j = \{u_1 \dots u_{k-j-1} u_{k-j}\} \Sigma^{jn}$ ,  $V_j = \{u_1 \dots u_{k-j-1} v_{k-j}\} \Sigma^{jn}$ , then  $G \subseteq U_{j+1}$ ,  $\|G \cap U_j\| \geq 2^{(j-c)n}$ , and  $\|G \cap V_j\| \geq 2^{(j-c)n}$ .

Also assume that there is a set  $B$  such that

(3) for every  $i$ ,  $j+1 \leq i \leq k$ ,  $\|B \cap \Sigma^{kn+i}\| \leq 1$ , and

(4) for every  $i$ ,  $0 \leq i \leq j$ ,  $B \cap \Sigma^{kn+i} = \emptyset$ .

Then, there exists a set  $C \subseteq \{y \mid kn \leq |y| \leq kn+j\}$  such that for every  $i$ ,  $0 \leq i \leq j$ ,  $\|C \cap \Sigma^{kn+i}\| \leq 1$  and, in addition,  $C$  has the property that for every set  $S \in \text{SPARSE}_h$  and every  $j$ -tt-condition  $t$ , it is not the case that for all  $x \in G$ ,  $x \in L_k(B \cup C)$  if and only if  $t(\chi_S(x_1), \dots, \chi_S(x_j)) = 1$ .



It is easy to verify that the above statement implies the lemma used in the proof of Theorem 4.2.

*Proof of Lemma.  $j = 1$ .* Let  $C = \{u_1 \cdots u_{k-1} 0^{n+1}\}$ . This choice of  $C$  satisfies the requirements; otherwise, there exists a sparse set  $S$  and a  $j$ -tt-condition  $t$  such that for all  $x \in G$ ,  $x \in L_k(B \cup C)$  if and only if  $t(\chi_S(x_1)) = 1$ .

CLAIM 1.  $\|L_k(B \cup C) \cap G\| \geq 2^{(1-c)n}$  and  $\|\overline{L_k(B \cup C)} \cap G\| \geq 2^{(1-c)n}$ .

*Proof.* Since by hypothesis  $G \subseteq U_2$  and  $B$  contains no strings of length  $kn$  or  $kn+1$ , the set  $U_2 = \{u_1 \cdots u_{k-2}\Sigma^{2n}$  must be included in  $L_k(B)$  or in  $\overline{L_k(B)}$  depending on whether there is an odd number or an even number of strings in  $\{0^{(k+1)}, u_1 0^{(k-1)(n+1)}, \dots, u_1 \cdots u_{k-2} 0^{2n+2}\} \cap B$ . If the string  $u_1 \cdots u_{k-1} 0^{n+1}$  is added to  $C$ , then membership in  $L_k(B \cup C)$  (or in  $\overline{L_k(B \cup C)}$ ) will not change for those strings in  $U_2 - U_1$ , but membership in  $L_k(B \cup C)$  (or in  $\overline{L_k(B \cup C)}$ ) will change for strings in  $U_1$ . Thus, either  $\|L_k(B \cup C) \cap G\| = \|G - U_1\|$  and  $\|\overline{L_k(B \cup C)} \cap G\| = \|G \cap U_1\|$  with both values being no less than  $2^{(1-c)n}$ , or  $\|L_k(B \cup C) \cap G\| = \|G \cap U_1\|$  and  $\|\overline{L_k(B \cup C)} \cap G\| = \|G - U_1\|$ , with both values being no less than  $2^{(1-c)n}$ .  $\square$

CLAIM 2. If for all  $x \in G$ ,  $x \in L_k(B \cup C)$  if and only if  $t(\chi_S(x_1)) = 1$ , then  $\|L_k(B \cup C) \cap G\| < 2^{\varepsilon n}$  or  $\|\overline{L_k(B \cup C)} \cap G\| < 2^{\varepsilon n}$ .

*Proof.* There are four cases.

Case 2.1.  $t(0) = t(1) = 0$ . This implies  $L_k(B \cup C) \cap G = \emptyset$ .

Case 2.2.  $t(0) = t(1) = 1$ . This implies that  $x \in L_k(B \cup C)$  if and only if  $f(x) \in S$ . Since  $f$  is one-to-one,  $\|L_k(B \cup C) \cap G\| \leq \|S^{\leq q(kn)}\| < 2^{\varepsilon n}$ .

Case 2.3.  $t(0) = 1, t(1) = 0$ . This implies that  $x \in \overline{L_k(B \cup C)}$  if and only if  $f(x) \in S$ . Since  $f$  is one-to-one,  $\|\overline{L_k(B \cup C)} \cap G\| \leq \|S^{\leq q(kn)}\| < 2^{\varepsilon n}$ .

Case 2.4.  $t(0) = t(1) = 1$ . This implies that  $\overline{L_k(B \cap C)} \cap G = \emptyset$ .

This concludes the proof of Claim 2.  $\square$

Recall that  $2^{(1-c)n} > 2^{\varepsilon n}$ . Thus, Claims 1 and 2 yield a contradiction. Hence, the proof of the lemma for  $j = 1$  is complete.  $\square$

*Proof of Lemma. Inductive step.* Let  $j$  be such that  $1 < j \leq k-1$ . Consider two cases.

Case 1. Suppose there exist  $r, 1 \leq r \leq j$ , and  $z$  such that either  $\|H_r(z) \cap U_j\| \geq 2^{(j-c)n}$  or  $\|H_r(z) \cap V_j\| \geq 2^{(j-c)n}$ , where  $H_r(z) = \{x \in G \mid x_r = z\}$ , where  $x_r$  is the  $r$ th element in  $f(x)$  and  $c' = c + k\varepsilon$ .

Assume, without loss of generality, that  $\|H_r(z) \cap U_j\| \geq 2^{(j-c)n}$ . Then, there exist  $u, v \in \Sigma^n$ ,  $u \neq v$ , such that  $\|H_r(z) \cap U_{j-1}\| \geq 2^{(j-1-c'')n}$  and  $\|H_r(z) \cap V_{j-1}\| \geq 2^{(j-1-c'')n}$ , where  $U_{j-1} = \{u_1 \cdots u_{k-j} u\} \Sigma^{(j-1)n}$  and  $V_{j-1} = \{u_1 \cdots u_{k-j} v\} \Sigma^{(j-1)n}$  and  $c'' = c' + \varepsilon$ . Consider the induction hypothesis for  $j-1$ , set  $H_r(z) \cap U_j$ , function  $f'(x) = \langle x_1, \dots, x_{r-1}, x_{r+1}, \dots, x_j \rangle$ , constant  $c''$ , and strings  $u_1, \dots, u_{k-j}, u, v \in \Sigma^n$ . Then we can check that

(0)  $\frac{1}{4} \leq c \leq \frac{1}{2} - 2jk\varepsilon$  implies  $\frac{1}{4} \leq c'' = c + (k+1)\varepsilon \leq \frac{1}{2} - 2(j-1)k\varepsilon$ ;

(1)  $f'$  is one-to-one on  $H_r(z) \cap U_j$  (otherwise,  $f$  would not be one-to-one on  $G$ ) and for all  $i, |x_i| \leq q(kn)$ .

(2)  $H_r(z) \cap U_j \subseteq U_j$  and  $\|H_r(z) \cap U_j \cap U_{j-1}\| = \|H_r(z) \cap U_{j-1}\| \geq 2^{(j-1-c'')n}$  and  $\|H_r(z) \cap U_j \cap V_{j-1}\| = \|H_r(z) \cap V_{j-1}\| \geq 2^{(j-1-c'')n}$ .

Also,  $B$  satisfies conditions (3) and (4). By the induction hypothesis there exists a set  $C_1$  such that for all sparse sets  $S \in \text{SPARSE}_h$  and all  $(j-1)$ -tt-conditions  $t$ , it is not the case that for all  $x \in H_r(z) \cap U_j$ ,  $x \in L_k(B \cup C_1)$  if and only if  $t(\chi_S(x_1), \dots, \chi_S(x_{r-1}), \chi_S(x_{r+1}), \dots, \chi_S(x_j)) = 1$ . Let  $C = C_1$ .

CLAIM. For every sparse  $S \in \text{SPARSE}_h$  and every  $j$ -tt-condition  $t$ , it is not the case that for all  $x \in G$ ,  $x \in L_k(B \cup C)$  if and only if  $t(\chi_S(x_1), \dots, \chi_S(x_{r-1}), \chi_S(x_r), \chi_S(x_{r+1}), \dots, \chi_S(x_j)) = 1$ .

*Proof.* Suppose that this is false. So for some  $j$ - $tt$ -condition  $t$  and some sparse set  $S \in \text{SPARSE}_h$ ,  $x \in L_L(B \cup C)$  if and only if  $t(\chi_S(x_1), \dots, \chi_S(x_{r-1}), \chi_S(x_r), \chi_S(x_{r+1}), \dots, \chi_S(x_j)) = 1$ . For this  $j$ - $tt$ -condition  $t$ , there are two induced  $(j-1)$ - $tt$ -conditions  $t_0$  and  $t_1$  such that  $t_0(\chi_S(x_1), \dots, \chi_S(x_{r-1}), \chi_S(x_{r+1}), \dots, \chi_S(x_j)) = 1$  if and only if  $t(\chi_S(x_1), \dots, \chi_S(x_{r-1}), 0, \chi_S(x_{r+1}), \dots, \chi_S(x_j)) = 1$ , and  $t_1(\chi_S(x_1), \dots, \chi_S(x_{r-1}), \chi_S(x_{r+1}), \dots, \chi_S(x_j)) = 1$  if and only if  $t(\chi_S(x_1), \dots, \chi_S(x_{r-1}), 1, \chi_S(x_{r+1}), \dots, \chi_S(x_j)) = 1$ . Depending upon whether or not  $z$  is in  $S$ , we have two possibilities: either

(a) for all  $x \in H_r(z)$ ,  $x \in L_k(B \cup C)$  if and only if  $t_0(\chi_S(x_1), \dots, \chi_S(x_{r-1}), \chi_S(x_{r+1}), \dots, \chi_S(x_j)) = 1$ , or

(b) for all  $x \in H_r(z)$ ,  $x \in L_k(B \cup C)$  if and only if  $t_1(\chi_S(x_1), \dots, \chi_S(x_{r-1}), \chi_S(x_{r+1}), \dots, \chi_S(x_j)) = 1$ .

Either case contradicts the induction hypothesis.  $\square$

*Case 2.* The condition specifying Case 1 is false.

Let  $C = \{u_1 \cdots u_{k-j-1} u_{k-j} 0^{j(n+1)}\}$ . Assume, by way of contradiction, that for some sparse set  $S$ , some  $j$ - $tt$ -condition  $t$ , and all  $x \in G$ ,

(\*)  $x \in L_k(B \cup C)$  if and only if  $t(\chi_S(x_1), \dots, \chi_S(x_j)) = 1$ .

CLAIM.  $\|L_k(B \cup C) \cap G \cap U_j\| \geq 2^{(j-c)n}$  or  $\|L_k(B \cup C) \cap G \cap V_j\| \geq 2^{(j-c)n}$ .

*Proof.* Since by hypothesis  $G \subseteq U_{j+1}$  and  $B$  contains no strings of length  $\leq kn + j$ , the set  $U_{j+1} = \{u_1 \cdots u_{k-j-1}\} \Sigma^{(j+1)n}$  must be included in  $L_k(B)$  or in  $\overline{L_k(B)}$ , depending on whether there is an odd number or an even number of strings in  $\{0^{k(n+1)}, u_1 0^{(k-1)(n+1)}, \dots, u_1 \cdots u_{k-j-1} 0^{(j+1)(n+1)}\} \cap B$ . By adding  $C = \{u_1 \cdots u_{k-j-1} u_{k-j} 0^{j(n+1)}\}$  to  $B$ , the membership in  $L_k(B \cup C)$  (or in  $\overline{L_k(B \cup C)}$ ) will not change for those strings in  $U_{j+1} - U_j$ , but the membership in  $L_k(B \cup C)$  (or in  $\overline{L_k(B \cup C)}$ ) will change for those strings in  $U_j$ . Thus we have that either  $[G \cap U_j \subseteq U_j \subseteq L_k(B \cup C)$  and  $G \cap V_j \subseteq U_{j+1} - U_j \subseteq \overline{L_k(B \cup C)}$ ] or  $[G \cap U_j \subseteq U_j \subseteq \overline{L_k(B \cup C)}$  and  $G \cap V_j \subseteq U_{j+1} - U_j \subseteq L_k(B \cup C)]$ . Therefore, we have either  $\|L_k(B \cup C) \cap G \cap U_j\| = \|G \cap U_j\| \geq 2^{(j-c)n}$  or  $\|L_k(B \cup C) \cap G \cap V_j\| = \|G \cap V_j\| \geq 2^{(j-c)n}$ .  $\square$

Now consider the  $j$ - $tt$ -condition  $t$  in (\*).

*Subcase 2.1.*  $t(0, \dots, 0) = 0$ . Then, for all  $x \in G$ ,  $x \in L_k(B \cup C)$  if and only if there exists  $r$ ,  $1 \leq r \leq j$ , such that  $x_r \in S$ . But for each  $r$  and each  $z \in S$ ,  $\|H_r(z) \cap U_j\| < 2^{(j-c)n}$  and  $\|H_r(z) \cap V_j\| < 2^{(j-c)n}$ . So,

$$\|L_k(B \cup C) \cap G \cap U_j\| \leq \sum_{r=1}^j \sum_{z \in S} \|H_r(z) \cap U_j\| < j \cdot 2^{\epsilon n} \cdot 2^{(j-c)n} \leq 2^{(j-c'+k\epsilon)n} = 2^{(j-c)n},$$

and similarly,  $\|L_k(B \cup C) \cap G \cap V_j\| < 2^{(j-c)n}$ . This contradicts the above claim.

*Subcase 2.2.*  $t(0, \dots, 0) = 1$ . Then, for all  $x \in G$ ,  $x \in \overline{L_k(B \cup C)} \cap G$  if and only if there exists  $r$ ,  $1 \leq r \leq k$ , such that  $x_r \in S$ . But then

$$\|\overline{L_k(B \cup C)} \cap G \cap U_j\| \leq \sum_{r=1}^j \sum_{z \in S} \|H_r(z) \cap U_j\| < 2^{(j-c)n}$$

and  $\|\overline{L_k(B \cup C)} \cap G \cap V_j\| < 2^{(j-c)n}$ . Again, we have a contraction.

This concludes the proof of Case 2 and, hence, the proof of the inductive step.  $\square$

Now we turn to the proof of Theorem 3.5. To a very large extent the proof follows the outline of the proof of Theorem 3.4, and so only a sketch will be given here.

**THEOREM 4.3.**  $P_{bit}(\text{SPARSE}) \neq P_{it}(\text{SPARSE})$ .

*Sketch of the proof.* For any set  $A$ , let  $L_\infty(A) = \{u_1 \cdots u_n \mid |u_1| = \dots = |u_n| = n\}$ ; the number of strings in the following list that are in the set  $A$  is odd:  $0^{n(n+1)}, u_1 0^{(n-1)(n+1)}, \dots, u_1 \cdots u_i 0^{(n-i)(n+1)}, \dots, u_1 \cdots u_n$ . It is clear that  $L_\infty(A) \leq_{it}^P A$ . We will describe a specific sparse set  $A$  such that  $L_\infty(A) \not\leq_{k-it}^P S$  for every  $k$  and every sparse set  $S$ .

Similar to the proof of Theorem 4.2, we assume an enumeration of polynomial-time-bounded deterministic machines witnessing bounded truth-table reductions:  $M_{i,j,k}$  computes the  $i$ th polynomial-time function  $f_i$  that generates  $k$  queries and the  $j$ th polynomial-time function  $g_j$  that generates a  $k$ - $tt$ -condition.

The following lemma is established in the proof of Theorem 4.2.

LEMMA. *For any pair  $k, h$  of positive integers and any machine  $M$  that witnesses an instance of a  $\leq_{k- $tt$$ -reduction, there exist arbitrarily large  $n$  and a finite set  $B \subseteq \{x \mid kn \leq |x| \leq k(n+1)\}$  such that*

(1) *for any  $m, kn \leq m \leq k(n+1), \|B \cap \Sigma^m\| \leq 1$ , and*

(2)  *$L_k(B)^{kn} \neq L(M, S)^{kn}$  for all  $S \in \text{SPARSE}_h$ .*

At stage  $\alpha = \langle i, j, k, h \rangle$ , consider the machine  $M'_{i,j,k}$  which operates as follows:

on input  $x$  whose length is at least  $k^2$  and is a multiple of  $k$ , say  $|x| = kn$ , and any set  $C$ , simulate  $M_{i,j,k}$  on  $0^{(n-k)n}x$  relative to  $C$ ;

on input  $x$  whose length is less than  $k^2$  or is not a multiple of  $k$ , reject.

The machine  $M'_{i,j,k}$  witnesses an instance of a  $\leq_{k- $tt$$ -reduction so that for the polynomial  $p_h$  the lemma shows that there exist an integer  $n > n_{\alpha-1}^2 + k$  and a finite set  $B \subseteq \{x \mid kn \leq |x| \leq k(n+1)\}$  satisfying (1) and (2) of the lemma. Then,  $L_k(B)^{kn} \neq L(M'_{i,j,k}, S)^{kn}$  for all  $S \in \text{SPARSE}_h$ .

Letting  $n_\alpha = n$ , define  $A_\alpha$  by adding strings in  $\{x \mid n^2 \leq |x| \leq n(n+1)\}$  to  $A_{\alpha-1}$  as follows:  $0^{(n-k)n}u_1 \cdots u_i 0^{(k-i)(n+1)} \in A_\alpha$  if and only if  $u_1 \cdots u_i 0^{(k-i)(n+1)} \in B$ , where  $1 \leq i \leq k$  and  $|u_1| = \cdots = |u_i| = n$ . Thus, for any  $x \in \Sigma^{n^2}$ ,  $x \in L_\infty(A)$  if and only if there exists  $y \in L_k(B)^{kn}$  such that  $x = 0^{(n-k)n}y$ . This implies that  $L_\infty(A) \cap \{0^{(n-k)n}\Sigma^{kn} = \{0^{(n-k)n}\}L_k(B)^{kn} \neq \{0^{(n-k)n}\}L(M'_{i,j,k}, S)^{kn} = L(M_{i,j,k}, S) \cap \{0^{(n-k)n}\Sigma^{kn}$  so that  $L_\infty(A) \neq L(M_{i,j,k}, S)$  for all  $S \in \text{SPARSE}_h$ .  $\square$

In the proofs of Theorems 4.1, 4.2, and 4.3, it is easy to verify that the set  $A$  is computable in time  $2^{O(n)}$  and, hence, that each of the sets  $L_k(A)$ ,  $k \geq 1$ , and  $L_\infty(A)$  is in  $\text{DXT} (= \cup_{c>0} \text{DTIME}(2^{cn}))$ .

5. Consider the proofs given in § 4. If we look carefully, it becomes clear that bounding the time needed to compute the reducibilities served as a method of bounding the number and the lengths of the query strings and nothing else. Thus it is reasonable to consider other reducibilities specified in such a way that the number and the lengths of the query strings are bounded by a polynomial in the length of the input. This section is devoted to such considerations.

Consider reducibilities computed nondeterministically in polynomial time. Ladner, Lynch, and Selman [19] defined  $\leq_{tt}^{\text{NP}}$  in such a way that  $\leq_{tt}^{\text{NP}} = \leq_T^{\text{NP}}$ . Later, Book, Long, and Selman [11] considered a variation of the notion of  $\leq_{tt}^{\text{NP}}$  by requiring that both the function that generates the list of queries and the function that generates the  $tt$ -condition be functions that are computed nondeterministically but are single-valued. It is this notion of  $\leq_{tt}^{\text{NP}}$  that is considered here.

It is clear that for any set  $A$  and any of the reducibilities  $r$  considered here,  $P_r(A) \subseteq \text{NP}_r(A)$ . What we have here are the observations that simple modifications of the proofs of Theorems 4.1, 4.2, and 4.3 yield the following results.

THEOREM 5.1. (a) *For every  $k > 0$ ,  $\text{NP}_{k- $tt$ }(\text{SPARSE}) \neq \text{NP}_{(k+1)- $tt$ }(\text{SPARSE})$ .*

(b)  $\text{NP}_{bt}(\text{SPARSE}) \neq \text{NP}_{tt}(\text{SPARSE})$ .

(c) *There is no  $k$  such that  $P_{(k+1)- $tt$ }(\text{SPARSE}) \subseteq \text{NP}_{k- $tt$ }(\text{SPARSE})$ .*

(d)  $P_{tt}(\text{SPARSE}) \not\subseteq \text{NP}_{bt}(\text{SPARSE})$ , that is,  $P/\text{poly} \not\subseteq \text{NP}_{bt}(\text{SPARSE})$ .

Proof. In the proofs of Theorems 4.1 and 4.2, we enumerate, instead of  $\leq_{k- $tt$$ -reduction machines  $\{M_{i,j}\}$ ,  $\leq_{k- $tt$$ -reduction machines  $\{N_{i,j}\}$ , and construct a sparse set  $A$  in the same way. Then, for all  $i, j$ , and all sparse sets  $S$ ,  $L_k(A) \neq L(N_{i,j}, S)$ , and hence  $L_k(A) \notin \text{NP}_{k- $tt$ }(\text{SPARSE})$ . Recall that for all sparse sets  $A$ ,  $L_k(A) \in$

$P_{(k+1)-it}(\text{SPARSE})$ . This proves parts (a) and (c). A similar modification yields a proof for part (b). A similar modification of the proof of Theorem 4.3 yields a proof of part (d).  $\square$

Unlike the set  $A$  constructed in the proofs of Theorems 4.1 and 4.2, the set  $A$  constructed above is not known to be computable deterministically in exponential time. Note that in stage  $\alpha = \langle i, j, h \rangle$  of the construction, we need to simulate function  $f_i$  on all strings of length  $kn$  (where  $n = n_\alpha$ ) in order to determine their membership in  $A$  or in  $\bar{A}$ . Since the function  $f_i$  is only known to be computable nondeterministically in polynomial time, no deterministic exponential time bound is known for the computation of all of the  $f_i$ .

We can interpret parts (c) and (d) of Theorem 5.1 as asserting that the (potential) additional computational power of nondeterministic operation is not sufficient to overcome the power of a deterministic machine to make additional queries, even though these queries are made in a nonadaptive (truth-table) manner to sets that are sparse.

Analogous to Theorem 3.6, we have the following fact.

**THEOREM 5.2.**  $\text{NP/poly} = \text{NP}_T(\text{SPARSE}) = \text{NP}_it(\text{SPARSE}) = \text{NP}_it(\text{TALLY}) = \text{NP}_T(\text{TALLY})$ .

*Proof.* The proof that  $\text{NP}_it(\text{TALLY}) = \text{NP}_T(\text{TALLY})$  is just like the proof that  $P_it(\text{TALLY}) = P(\text{TALLY})$ . The proof that  $\text{NP}_T(\text{SPARSE}) = \text{NP}_T(\text{TALLY})$  is just like the proof that  $P_T(\text{SPARSE}) = P_T(\text{TALLY})$ . The result follows since  $\text{NP}_it(\text{TALLY}) \subseteq \text{NP}_it(\text{SPARSE})$ .  $\square$

Now consider reducibilities computed in polynomial space. Assume that all queries are of length bounded by a polynomial in the length of the input. The various truth-table reducibilities are nonadaptive and the list of queries must be computed and stored on a work tape before any evaluation is carried out; thus, even with polynomial space-bounded machines, only a polynomial number of queries can be made. Book [10], Book and Wrathall [13], and Balcázar, Book, and Schöning [3] have investigated the Turing reducibilities computed by deterministic and nondeterministic polynomial space-bounded oracle machines that in any computation make only a polynomial number of queries. For any set  $A$ , the class specified by deterministic machines of this type is denoted  $\text{PQUERY}_T(A)$  and the class specified by nondeterministic machines of this type is denoted  $\text{NPQUERY}_T(A)$ . It is known [3] that for any set  $A$ ,  $\text{PQUERY}_T(A) = P_T(Q \oplus A)$  and  $\text{NPQUERY}_T(A) = \text{NP}_T(Q \oplus A)$ , where  $Q$  is any set that is  $\leq_m^P$ - or  $\leq_T^P$ -complete for PSPACE. As reducibilities, let  $A \leq_T^{\text{PQ}} B$  denote  $A \in \text{PQUERY}_T(B)$  and let  $A \leq_T^{\text{NPQ}} B$  denote  $A \in \text{NPQUERY}_T(B)$ ; notice that  $\text{PQUERY}_T(\emptyset) = \text{NPQUERY}_T(\emptyset) = \text{PSPACE}$ . On the other hand, it is shown in [10] that there is a sparse set  $S$  such that  $\text{PQUERY}_T(S) \neq \text{NPQUERY}_T(S)$ .

By considering functions computable deterministically in polynomial space, we can define the various truth-table reducibilities computable deterministically in polynomial space and view them as restrictions of  $\leq_T^{\text{PQ}}$ . Again observing that the role of the polynomial time bounds in the proofs in § 4 is only to restrict the number and length of queries, we see the following facts.

**THEOREM 5.3.**

- (a) For every  $k > 0$ ,  $\text{PQUERY}_{k-it}(\text{SPARSE}) \neq \text{PQUERY}_{(k+1)-it}(\text{SPARSE})$ .
- (b)  $\text{PQUERY}_{bit}(\text{SPARSE}) \neq \text{PQUERY}_it(\text{SPARSE})$ .
- (c) There is no  $k$  such that  $P_{(k+1)-it}(\text{SPARSE}) \subseteq \text{PQUERY}_{k-it}(\text{SPARSE})$ .
- (d)  $P_it(\text{SPARSE}) \not\subseteq \text{PQUERY}_{bit}(\text{SPARSE})$ , that is,  $P/\text{poly} \not\subseteq \text{PQUERY}_{bit}(\text{SPARSE})$ .

By considering functions that are computable nondeterministically in polynomial space but are single-valued, we can define the various truth-table reducibilities compu-

table nondeterministically in polynomial space and view them as restrictions of  $\leq_T^{\text{NPQ}}$ . Combining the arguments needed for the proofs of Theorems 5.1 and 5.3, we see the following facts.

THEOREM 5.4.

- (a) For every  $k > 0$ ,  $\text{NPQUERY}_{k\text{-tt}}(\text{SPARSE}) \neq \text{NPQUERY}_{(k+1)\text{-tt}}(\text{SPARSE})$ .
- (b)  $\text{NPQUERY}_{\text{btt}}(\text{SPARSE}) \neq \text{NPQUERY}_{\text{tt}}(\text{SPARSE})$ .
- (c) There is no  $k$  such that  $P_{(k+1)\text{-tt}}(\text{SPARSE}) \subseteq \text{NPQUERY}_{k\text{-tt}}(\text{SPARSE})$ .
- (d)  $P_{\text{tt}}(\text{SPARSE}) \not\subseteq \text{NPQUERY}_{\text{btt}}(\text{SPARSE})$ , that is,  $P/\text{poly} \not\subseteq \text{NPQUERY}_{\text{btt}}(\text{SPARSE})$ .

Parts (c) and (d) of Theorems 5.3 and parts (c) and (d) of Theorem 5.4 can be interpreted as asserting that the (potential) additional computational power of space, as opposed to time, is not sufficient to overcome the ability to make additional queries, even though these queries are made in a truth-table (nonadaptive) manner to a sparse set.

Even though there is a sparse set  $S$  such that  $\text{PQUERY}_T(S) \neq \text{NPQUERY}_T(S)$ , an argument of Long [20] can be used to show that for every sparse set  $S_1$  there is a sparse set  $S_2$  such that  $\text{NPQUERY}_T(S_1) \subseteq \text{PQUERY}_T(S_2)$  and, hence,  $\text{PQUERY}_T(\text{SPARSE}) = \text{NPQUERY}_T(\text{SPARSE})$ . We will show something somewhat stronger.

THEOREM 5.5. Each of the following classes is equal to  $\text{PSPACE}/\text{poly}$ :

- (a)  $\text{PQUERY}_T(\text{SPARSE})$ ;
- (b)  $\text{PQUERY}_{\text{tt}}(\text{SPARSE})$ ;
- (c)  $\text{PQUERY}_{\text{tt}}(\text{TALLY})$ ;
- (d)  $\text{PQUERY}_T(\text{TALLY})$ ;
- (e)  $\text{NPQUERY}_T(\text{TALLY})$ ;
- (f)  $\text{NPQUERY}_{\text{tt}}(\text{TALLY})$ ;
- (g)  $\text{NPQUERY}_{\text{tt}}(\text{SPARSE})$ ;
- (h)  $\text{NPQUERY}_T(\text{SPARSE})$ .

*Proof.* First, we show that all of the classes listed in (a)–(h) are equal. Notice that each class is included in  $\text{NPQUERY}_T(\text{SPARSE})$ , and  $\text{PQUERY}_{\text{tt}}(\text{TALLY})$  is included in each of these classes. Thus, it is sufficient to show that  $\text{NPQUERY}_T(\text{SPARSE}) \subseteq \text{NPQUERY}_T(\text{TALLY}) \subseteq \text{NPQUERY}_{\text{tt}}(\text{TALLY}) \subseteq \text{PQUERY}_{\text{tt}}(\text{TALLY})$ . The first inclusion follows from the proof of Hartmanis [15] that  $\text{NP}_T(\text{SPARSE}) = \text{NP}_T(\text{TALLY})$ . The second inclusion follows from the argument that  $P_T(\text{TALLY}) = P_{\text{tt}}(\text{TALLY})$  in the proof of Theorem 3.6. The third inclusion follows from Savitch's Theorem [23].

Second, recall that  $\text{PSPACE}/\text{poly} = \text{PSPACE}_T(\text{SPARSE})$  so that each of the classes in (a)–(h) is included in  $\text{PSPACE}/\text{poly}$ . Long [20], among others, has shown that for any sparse set  $S_1$  there exist a sparse set  $S_2$  (=prefix set of  $S_1$ ) and a deterministic polynomial-time oracle machine  $M_0$  that, relative to  $S_2$ , will enumerate  $S_1$ . Let  $q$  be a polynomial that bounds the running time of  $M_0$ . Let  $L \in \text{PSPACE}(S_1)$  and let  $M_1$  be a deterministic polynomial space-bounded oracle machine that witnesses this. (Notice that Savitch's theorem allows us to take  $M_1$  as deterministic since there is no bound on the number of oracle queries.) Let  $M_2$  be a machine that on input  $x$  first simulates  $M_0$  to compute in time  $q(|x|)$ , a list containing precisely the strings in  $S_1^{\leq q(|x|)}$  and then simulates  $M_1$ 's computation on  $x$ , using the list to determine whether a query string is in  $S_1$  instead of actually querying the oracle. Clearly,  $L(M_2, S_2)$  is in  $\text{PQUERY}(S_2) \subseteq \text{PQUERY}_T(\text{SPARSE})$  and  $L = L(M_2, S_2)$ . Since  $L$  was taken arbitrarily in  $\text{PSPACE}_T(\text{SPARSE})$ , this yields the fact that  $\text{PSPACE}/\text{poly} \subseteq \text{PQUERY}_T(\text{SPARSE})$ .  $\square$

Consider reducibilities computed deterministically by machines that use work space  $\log n$ . Recall that  $\leq_T^{\log}$  is transitive. For any fixed  $k$ , the queries may be computed one at a time and the answers stored so that the list of  $k$  answers can be stored in  $\log n$  space. Thus, the reducibilities  $\leq_{k-nt}^{\log}$ ,  $k > 0$ , can be defined. For every set  $A$  and every  $k > 0$ , let  $\text{DLOG}_{k-nt}(A)$  denote the collection of sets  $B$  such that  $B \leq_{k-nt}^{\log} A$ . Also, for every  $k > 0$ , let  $\text{DLOG}_{k-nt}(\text{SPARSE})$  denote the union of all classes  $\text{DLOG}_{k-nt}(S)$  as  $S$  takes values in  $\text{SPARSE}$ . Then we have the following fact.

**THEOREM 5.6.** *For every  $k > 0$ ,  $\text{DLOG}_{k-nt}(\text{SPARSE}) \neq \text{DLOG}_{(k+1)-nt}(\text{SPARSE})$ .*

6. Consider the proofs given in § 4. The proof of Theorem 4.2 is such that for every  $k > 0$  the set that witnesses the fact that  $P_{k-nt}(\text{SPARSE}) \neq P_{(k+1)-nt}(\text{SPARSE})$  is a set that can be accepted by a deterministic Turing machine that runs in exponential time, that is, in the class  $\text{DEXT} = \bigcup_{c>0} \text{DTIME}(2^{cn})$ . Hence, there is no sparse set that can be  $\leq_{k-nt}^P$ -hard for  $\text{DEXT}$ . This yields a new proof of a result of Watanabe [27], [28].

**THEOREM 6.1.** *Let  $C$  be any class of languages that contains every language accepted deterministically in exponential time, i.e.,  $\text{DEXT} \subseteq C$ . Then for every  $k$  there is no sparse set that is  $\leq_{k-nt}^P$ -hard for  $C$ .*

Similarly, the proof of Theorem 4.3 is such that for every  $k > 0$  the set that witnesses the fact that  $P_{bnt}(\text{SPARSE}) \neq P_{nt}(\text{SPARSE})$  is a set that is in the class  $\text{DEXT}$ . Hence, there is no sparse set that can be  $\leq_{bnt}^P$ -hard for  $\text{DEXT}$ . This leads to the following result.

**THEOREM 6.2.** *Let  $C$  be any class of languages that contains every language accepted deterministically in exponential time, i.e.,  $\text{DEXT} \subseteq C$ . Then there is no sparse set that is  $\leq_{bnt}^P$ -hard for  $C$ .*

**COROLLARY.** *No sparse set is bounded truth-table hard for deterministic exponential time.*

**COROLLARY.** *No sparse set is bounded truth-table complete for deterministic exponential time.*

Results similar to Theorems 6.1 and 6.2 and the corollaries have been reported by Balcázar and Schöning [6] and by Berman and Hartmanis [7], both dealing with  $\leq_m^P$ . Watanabe [27], [28] has developed similar results for  $\leq_T^P$ -hard sets for  $\text{DEXT}$  and has shown that any such set must have density greater than  $\log \log n$ .

It is important to point out that all of the results presented here have to do with sets reducible to sparse sets. There are other results in the literature showing that the polynomial time truth-table reducibilities can be separated. Of particular interest are the results in an important new paper by Watanabe [26]. Recall that  $\text{DEXT}$  denotes the collection of all languages accepted deterministically in exponential time.

(1) For every integer  $k$  there exists a  $\leq_{(k+1)-nt}^P$ -complete set for  $\text{DEXT}$  which is not  $\leq_{k-nt}^P$ -complete for  $\text{DEXT}$ .

(2) There exists a  $\leq_{nt}^P$ -complete set for  $\text{DEXT}$  which is not  $\leq_{bnt}^P$ -complete for  $\text{DEXT}$ .

(3) There exists a  $\leq_T^P$ -complete set for  $\text{DEXT}$  which is not  $\leq_{nt}^P$ -complete for  $\text{DEXT}$ .

Thus, there is a properly infinite hierarchy of complete sets for  $\text{DEXT}$  based on the different (nonadaptive) bounded truth-table polynomial-time reducibilities.

#### REFERENCES

- [1] E. ALLENDER AND R. RUBINSTEIN, *P-printable sets*, SIAM J. Comput., 17 (1988), to appear.
- [2] J. BALCÁZAR AND R. BOOK, *Sets with small generalized Kolmogorov complexity*, Acta Inform., 23 (1986), pp. 679–688.

- [3] J. BALCÁZAR, R. BOOK, AND U. SCHÖNING, *On bounded query machines*, Theoret. Comput. Sci., 40 (1985), pp. 237–243.
- [4] ———, *The polynomial-time hierarchy and sparse oracles*, J. Assoc. Comput. Mach., 33 (1986), pp. 603–617.
- [5] J. BALCÁZAR, J. DIAZ, AND K. GABARRÓ, *On nonuniform polynomial space*, Proc. Conference on Structure in Complexity Theory, Lecture Notes in Computer Science 223, Springer-Verlag, Berlin, New York, 1986, pp. 35–50.
- [6] J. BALCÁZAR AND U. SCHÖNING, *Bi-immune sets for complexity classes*, Math. Systems Theory, 18 (1985), pp. 1–10.
- [7] L. BERMAN AND J. HARTMANIS, *On isomorphisms and density of NP and other complete sets*, SIAM J. Comput., 6 (1977), pp. 305–322.
- [8] P. BERMAN, *Relationships between density and deterministic complexity of NP-complete languages*, Proc. 5th International Colloquium on Automata, Languages, Programming, Lecture Notes in Computer Science 62, Springer-Verlag, Berlin, New York, 1978, pp. 63–71.
- [9] R. BOOK, *On languages accepted by space-bounded oracle machines*, Acta Inform., 12 (1979), pp. 177–185.
- [10] ———, *Bounded query machines: on NP and PSPACE*, Theoret. Comput. Sci., 15 (1981), pp. 27–39.
- [11] R. BOOK, T. LONG, AND A. SELMAN, *Quantitative relativizations of complexity classes*, SIAM J. Comput., 13 (1984), pp. 461–487.
- [12] R. BOOK AND A. SELMAN, *Characterizations of reduction classes modulo oracle conditions*, Math. Systems Theory, 17 (1984), pp. 263–277.
- [13] R. BOOK AND C. WRATHALL, *Bounded query machines: on NP( ) and NPQUERY( )*, Theoret. Comput. Sci., 15 (1981), pp. 41–50.
- [14] S. FORTUNE, *A note on sparse complete sets*, SIAM J. Comput., 8 (1979), pp. 431–433.
- [15] J. HARTMANIS, *On sparse sets in NP–P*, Inform. Process. Lett., 16 (1983), pp. 55–60.
- [16] R. KARP AND R. LIPTON, *Some connections between non-uniform and uniform complexity classes*, in Proc. 12th ACM Symposium on the Theory of Computing, 1980, pp. 302–309.
- [17] J. KÖBLER, *Untersuchung verschiedener polynomieller Reduktionsklassen von NP*, Diplom. thesis, Institut für Informatik, Univ. Stuttgart, Stuttgart, 1985.
- [18] J. KÖBLER, U. SCHÖNING, AND K. WAGNER, *The difference and truth-table hierarchies for NP*, RAIRO—Informatique Théorique et Applications, 21 (1987), pp. 419–435.
- [19] R. LADNER, N. LYNCH, AND A. SELMAN, *A comparison of polynomial-time reducibilities*, Theoret. Comput. Sci., 1 (1973), pp. 103–123.
- [20] T. LONG, *On restricting the size of oracles compared with restricting access to oracles*, SIAM J. Comput., 14 (1985), pp. 585–597.
- [21] T. LONG AND A. SELMAN, *Relativizing complexity classes with sparse oracles*, J. Assoc. Comput. Mach., 33 (1986), pp. 618–627.
- [22] S. MAHANEY, *Sparse complete sets for NP; solution to a conjecture by Berman and Hartmanis*, J. Comput. System Sci., 25 (1982), pp. 130–143.
- [23] W. SAVITCH, *Relationships between nondeterministic and deterministic space complexities*, J. Comput. System Sci., 4 (1970), pp. 177–192.
- [24] U. SCHÖNING, *Complexity and structure*, Lecture Notes in Computer Science 211, Springer-Verlag, Berlin, New York, 1986.
- [25] ———, *Complete sets and closeness to complexity classes*, Math. Systems Theory, 19 (1986), pp. 29–41.
- [26] O. WATANABE, *A comparison of polynomial time completeness notions*, Theoret. Comput. Sci., 54 (1987), pp. 249–265.
- [27] ———, *On the structure of intractable complexity classes*, Ph.D. dissertation, Department of Computer Science, Tokyo Institute of Technology, Tokyo, 1987.
- [28] ———, *Polynomial time reducibility to a set of small density*, Proc. 2nd Conference on Structure in Complexity Theory, 1987, pp. 138–146.
- [29] J. BALCÁZAR, personal communication, 1985.

## AN APPROXIMATION SCHEME FOR FINDING STEINER TREES WITH OBSTACLES\*

J. SCOTT PROVAN†

**Abstract.** We consider the problem of constructing a Steiner minimal tree connecting a given set  $K$  of points and lying inside a polygonally bounded, not necessarily simply connected region  $R$  in the plane. We first define the *path-convex hull* of  $K$  in  $R$ , which is a “sufficiently small” subregion of  $R$  guaranteed to contain the Steiner minimal tree. We then give an  $\epsilon$ -approximation scheme to find the Steiner minimal tree in  $R$  by reducing it to a Steiner tree problem on a “visibility graph” associated with  $K$  and the path-convex hull of  $R$ . This will be a fully polynomial approximation scheme when  $K$  is restricted to lie on a small number of interior points and boundary polygons of  $R$ . Several techniques are given which further reduce the region in which the Steiner minimal tree is known to lie, and which extend known results for the Steiner minimal tree problem without obstacles.

**Key words.** Steiner tree, approximation, polynomial algorithm, fully polynomial approximation, visibility graph

**AMS(MOS) subject classifications.** 05, 51, 90

**1. Introduction.** The *Steiner minimal tree with obstacles problem* (SMTO) studied in this paper is described as follows: Let  $R$  be a *polygonally bounded* region of the plane, that is, a connected (not necessarily simply connected) closed region of the plane whose boundary is made up of a finite number of straight-line segments, and let  $K$  be a set of *terminal points* lying in  $R$ . A *spanning graph* for  $K$  in  $R$  is a finite set of straight-line segments lying entirely in  $R$  and connecting all pairs of points in  $K$ . The *length* of  $T$ ,  $l(T)$ , is the sum of the Euclidean lengths of the line segments in  $T$ . SMTO is the problem of finding the *Steiner minimal tree* for  $K$  in  $R$ , that is, a spanning graph for  $K$  in  $R$  having minimum length, which must necessarily be a tree (see Fig. 1). The special case where  $R$  is the entire Euclidean plane is the classical

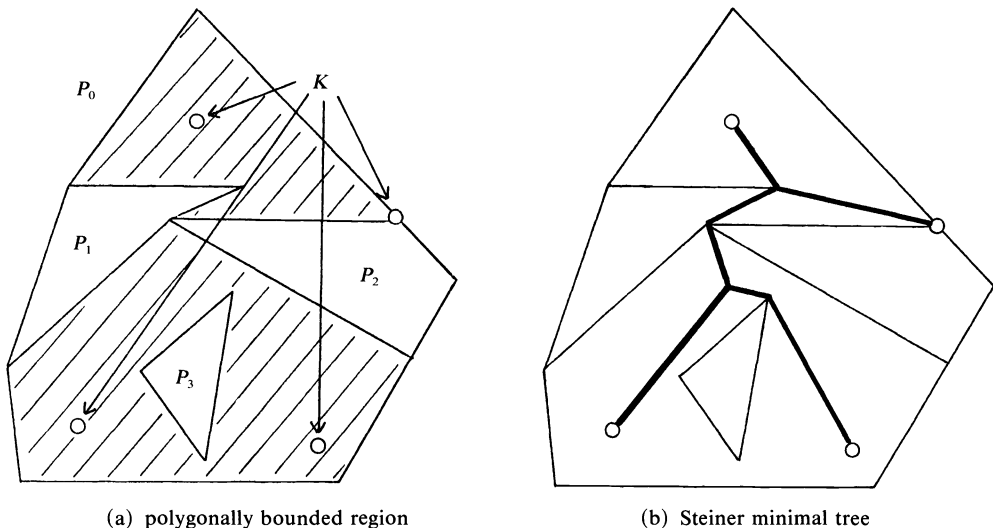


FIG. 1

\* Received by the editors August 17, 1987; accepted for publication November 13, 1987. This research was supported by the Air Force Office of Scientific Research under grant AFOSR-84-0140.

† Department of Operations Research, University of North Carolina, Chapel Hill, North Carolina 27599.



Steiner minimal tree problem, and has a long history [3], [4], [5], [8], [10], [12], [14], [18], [26]. SMTO has drawn less attention [15], [16], [25], although the related problem of shortest paths in the presence of obstacles has been fairly well studied [17], [22], [23], [28]. An important closely related problem is the *Steiner tree problem in graphs*. Here  $R$  is described as a graph  $G$  whose edges are given arbitrary lengths. The spanning graphs and Steiner minimal trees are drawn from the edges of  $G$ , with lengths defined accordingly. This problem has also been well studied [7], [13], [24], [27], [29], although little connection has been made between it and the Steiner minimal tree problem.

SMTO is known to be NP-hard [10], even in the case where  $R = \mathbb{R}^2$ . In fact even efficient approximation schemes are unlikely to exist, in the following sense. For a given instance  $(K, R)$  of SMTO, let  $T^*$  be the Steiner minimal for  $K$  in  $R$ . For  $\varepsilon > 0$ , an  $\varepsilon$ -approximation for  $T^*$  is a spanning graph  $T$  for  $K$  in  $R$  satisfying

$$l(T) \leq (1 + \varepsilon)l(T^*).$$

A *fully polynomial approximation scheme* for SMTO is an algorithm which, for any input  $(K, R)$  for SMTO and every  $\varepsilon > 0$ , gives an  $\varepsilon$ -approximation to the Steiner minimal tree for  $K$  in  $R$  in time which is bounded above by a polynomial in  $1/\varepsilon$  and the length of the input describing  $(K, R)$  (see [11, § 6.1], for a discussion of fully polynomial approximation schemes). The construction of [10] shows that a fully polynomial approximation scheme cannot exist for SMTO—even when  $R$  is restricted to be  $\mathbb{R}^2$ —unless  $P = NP$  (see [19, § 4], for details).

In a recent paper [19] the author studies the connection between the Steiner tree problem on graphs and the classical Steiner minimal tree problem. The notion of *path-convexity* is used to effectively restrict the region in which the Steiner minimal tree is known to lie. As a result, polynomial-time algorithms exist to find Steiner minimal trees, or  $\varepsilon$ -approximations thereof, when the terminals are located on the boundary of the restricted region. The purpose of this paper is to extend these results to SMTO, using some interesting recent results in the study of Steiner trees and obstacle-avoiding paths. The paper is organized as follows. In § 2 the notion of *path-convex hull* is introduced, and it is shown that a solution to SMTO always exists in any path-convex hull. An efficient algorithm is given to find a path-convex hull, which makes use of the *visibility graph* of Lozano-Perez and Wesley [17] and the results of Bienstock and Monma [1]. Section 3 gives a scheme for producing an  $\varepsilon$ -approximation to SMTO. By applying the method of Erickson, Monma, and Veinott [9] this scheme is shown to be a fully polynomial approximation scheme in the case where the set of terminals lies on a small number of boundary polygons and interior points of  $K$ . Section 4 shows how results of Bienstock and Monma [1], Gilbert and Pollak [12], Cockayne [5], and Hwang et al. [14] can be extended to shrink the size of a path-convex hull, and thus improve the complexity of the approximation scheme.

**2. The path-convex hull.** A typical input for SMTO consists of a list of terminal points, together with a description of the boundary  $\partial R$  of the enclosing region  $R$ . This latter set can be assumed to consist of a collection  $P_0, \dots, P_m$  of simple (not necessarily convex) polygons with  $P_0$  the exterior boundary of  $R$  and  $P_1, \dots, P_m$  the boundaries of the interior obstacles (see Fig. 1). If we denote by  $P_i^e$  and  $P_i^i$  the interior and exterior, respectively, of polygon  $i$ , then  $R = \mathbb{R}^2 - (P_0^e \cup P_1^i \cup \dots \cup P_m^i)$ . The  $P_1, \dots, P_m$  are assumed to have pairwise disjoint interiors and to be contained inside  $P_0$ . The  $P_i$  themselves need not be disjoint; indeed, an important special case is the Steiner tree problem on a plane graph with a given straight-line layout, where the  $P_i$  are the boundaries of the faces of the graph. It may also be that the exterior or interior boundaries are absent, as with the classical Euclidean minimal Steiner tree problem.

The most general description of a region with finite straight-line boundary also includes the case where the  $P_i$  are nonsimple, though having connected interior. We remark, however, that in this case  $R$  can be separated at the cutpoints of the  $P_i$  into regions with simple boundary polygons, and SMTO can be solved by solving smaller SMTOs separately on each subregion (see, for example, [19, § 2]).

In this paper we will actually treat the *discrete* minimal Steiner tree problem (see [10]); that is, we will assume that Euclidean distances are computed only to some predetermined number of digits' accuracy. Since the results in the paper concern  $\epsilon$ -approximations to SMTO anyway, this is not a serious restriction, although it allows intermediate complexity results to be stated without concern for the possible manipulation of irrational numbers.

Throughout the paper we will use  $V$  to denote the set of vertices in  $\cup_{i=1}^m P_i$ , and will use  $\nu$  and  $k$  to denote the cardinalities of  $V$  and  $K$ , respectively. We first describe a method for finding a "sufficiently small" subregion in which the Steiner tree is known to lie, and give properties of this region. Let  $\tilde{R}$  be a polygonally bounded, connected subregion of  $R$ . Then the exterior boundary of  $\tilde{R}$  can be traversed "clockwise" by a closed walk (a polygon with possibly repeated vertices and edges)  $\Gamma: v_0, e_1, v_1, \dots, v_{r-1}, e_r, v_r = v_0$ , where for  $i = 1, \dots, r$ ,  $e_i = [v_{i-1}, v_i]$  is the straight-line segment connecting  $v_{i-1}$  and  $v_i$ , and no point of  $\tilde{R}$  lies immediately to the left of  $e_i$  when traversed from  $v_{i-1}$  to  $v_i$ . The *perimeter* of  $\tilde{R}$  is defined to be

$$\rho(\tilde{R}) = \sum_{i=1}^r \|e_i\|,$$

where  $\|e_i\| = \|v_i - v_{i-1}\|$ . (See [19] for a more comprehensive discussion of these terms.) The region  $\tilde{R}$  is a *path-convex hull* of  $K$  in  $R$  if it contains  $K$  and has minimum perimeter over all such regions. Path-convex hulls correspond to convex hulls for the classical Steiner minimal tree problem [12, § 3.5] and to path-convex regions for the Steiner tree problem on graphs [1], [19]. Their significance in SMTO is given by the following result. Its proof is identical to that of Theorem 3 in [19].

**THEOREM 2.1.** *Let  $\tilde{R}$  be a path-convex hull of  $K$  in  $R$ . Then there exists a Steiner minimal tree for  $K$  in  $R$  which lies entirely in  $\tilde{R}$ .*

Next we turn to the problem of constructing a path-convex hull for  $K$  in  $R$ . Here we make use of a generalization of the notion of "visibility graph" introduced by Lozano-Perez and Wesley [17] (and by Shamos [22] under the name "viewability graph.") For any set  $S$  of points in  $R$ , define the *S-visibility graph* for  $R$  to be the graph whose vertices are  $V \cup S$  and whose edges consist of those pairs of vertices whose connecting straight-line segments lie entirely in  $R$  and contain no point of  $V \cup S$  in their interiors (see Fig. 2). Visibility graphs have been studied in relation to the shortest path problem [17], [23], [28], where the following theorem is implicit.

**THEOREM 2.2.** *The shortest Euclidean length path in  $R$  between any two points  $u$  and  $v$  can always be chosen to lie in the  $\{u, v\}$ -visibility graph for  $R$ .*

We also note that the  $S$ -visibility graph has  $n = |V \cup S|$  nodes and at most  $n^2$  edges, and the same methods for constructing visibility graphs (e.g., [23], § 2) can be applied to construct the  $S$ -visibility graph in time  $O(n^2 \log n)$ . We next prove a variant of Theorem 2.2 which will be useful for constructing the path-convex hull for  $K$  in  $R$ .

**LEMMA 2.3.** *Any path-convex hull  $\tilde{R}$  of  $K$  in  $R$  has its exterior boundary composed of edges of the  $K$ -visibility graph for  $R$ .*

*Proof.* Let  $\Gamma$  be the closed walk traversing the exterior boundary of  $\tilde{R}$ , specified so that the edges of  $\Gamma$  contain no points of  $V \cup K$  in their interior, and let  $v$  be a vertex of  $\Gamma$  at which two or more noncollinear edges of  $\Gamma$  meet. Suppose  $v$  is not in

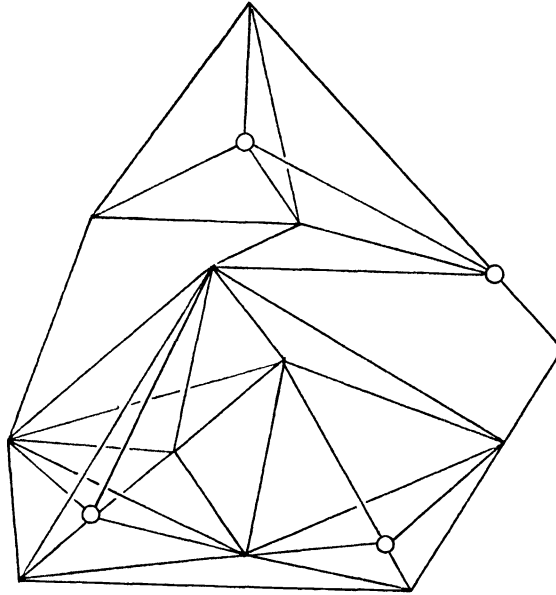


FIG. 2. The  $K$ -visibility graph.

$V \cup K$ . By the choice of  $v$  there must exist a pair of successive edges  $[u, v], [v, w]$  in  $\Gamma$  which are noncollinear and whose convex angle contains no other edges of  $\Gamma$  passing through  $v$ . Now since  $v$  is not in  $V \cup K$ , there exists a point  $v'$  on  $[u, v]$  sufficiently close to  $v$  so that  $[v', w] \subseteq R$  and the triangle  $vv'w$  contains no element of  $K$  in its interior. It follows that the closed walk  $\Gamma'$  obtained by replacing  $[u, v], [v, w]$  by  $[u, v'], [v', w]$  in  $\Gamma$  has the property that  $\Gamma'$  lies entirely in  $R$ , encloses  $K$ , and has smaller length than  $\Gamma$ , a contradiction. Thus every such point  $v$  is in  $V \cup K$ , and the theorem follows.  $\square$

The problem of finding the path-convex hull now reduces to that of finding the minimum length closed walk surrounding  $K$  in the  $K$ -visibility graph  $G_K$  for  $R$ . This is facilitated by using two results. The first is due to Bienstock and Monma [1, Thm. 1] and, although it is stated in that paper for plane graphs, it applies as well to  $G_K$ . A *shortest path tree* for  $K$  in  $G_K$  rooted at  $r \in K$  is a tree consisting of a union of shortest paths, one from  $r$  to each of the other vertices of  $K$ .

**THEOREM 2.4** (Bienstock and Monma). *Let  $T$  be any shortest path tree for  $K$  in  $R$ . Then there exists a path-convex hull of  $K$  in  $R$  which contains  $T$ .*

The second result is proved in Theorem 1 of [20]. For path  $\Gamma$ , let  $\Gamma^{-1}$  be the path obtained by traversing  $\Gamma$  in reverse order.

**THEOREM 2.5.** *Let  $G' = (V', E')$  be a graph embedded (not necessarily in planar fashion) in the plane, and let  $P$  be a simple polygon in the plane such that no edge of  $G'$  crosses  $P$ . For  $s \in V'$  let  $\tau_s$  be the shortest path tree for  $V'$  in  $G'$  rooted at  $s$ . Then a minimum length walk in  $G'$  enclosing  $P$  and containing  $s$  can always be found in the form  $\Gamma_u \cup \{(u, v)\} \cup \Gamma_v^{-1}$ , where  $(u, v)$  is an edge of  $G$ , and  $\Gamma_u$ , and  $\Gamma_v$  are the paths in  $\tau_s$  from  $s$  to  $u$  and  $v$ , respectively.*

We are now in a position to give an algorithm to find the path-convex hull of  $K$  in  $R$ . An example is given in Fig. 3.

**PATH-CONVEX HULL ALGORITHM**

**Input:** Polygonally bounded plane region  $R$ , terminal set  $K \subseteq R$ .

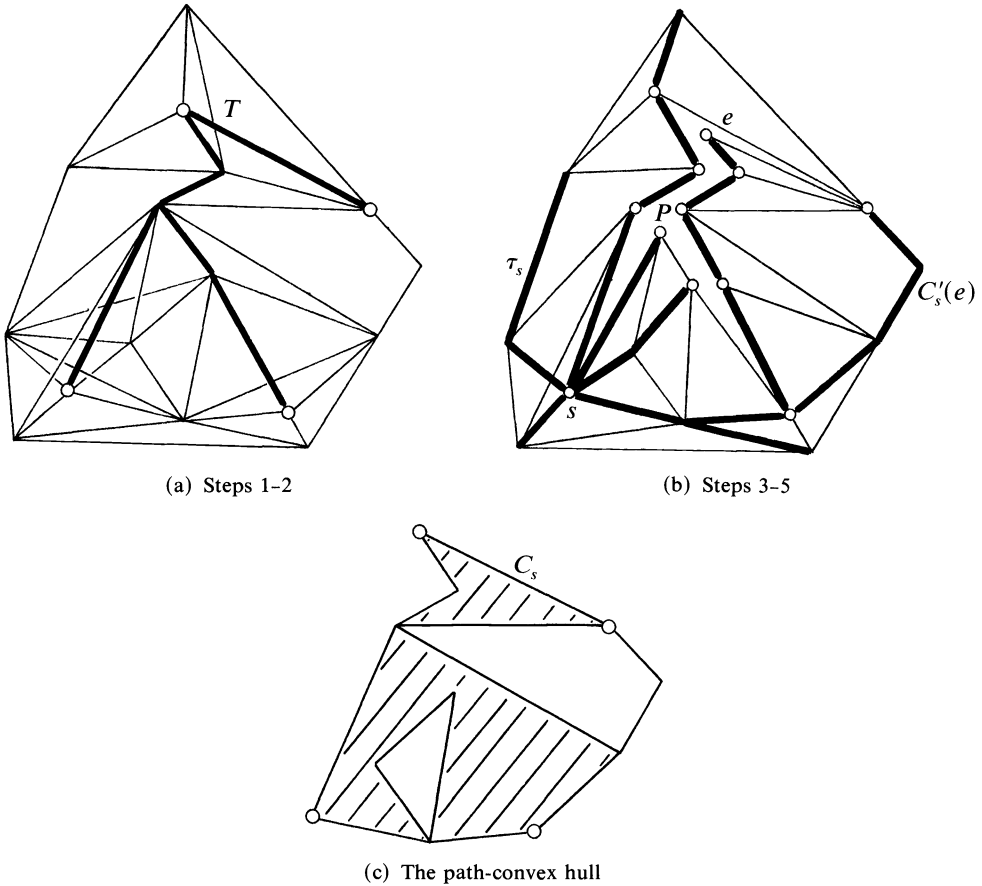


FIG. 3. Path-convex hull algorithm.

**Output:** A path-convex hull of  $K$  in  $R$ .

**The algorithm:**

1. Construct the  $K$ -visibility graph  $G_K$  for  $R$ .
2. Find a shortest path tree  $T$  for  $K$  in  $G_K$ . Let  $P$  be a clockwise traversal of  $T$  (taken as a region). “Thicken”  $P$  by making each of its repeated vertices and edges distinct, and delete all edges of  $G_K$  which cross an edge of  $P$ . Call the new graph  $G' = (V', E')$ .
3. For each  $s \in V'$  construct a shortest path tree  $\tau_s$  for  $V'$  in  $G'$  rooted at  $s$ . Let  $C'_s = C'_s(e)$  be a minimum-length circuit enclosing  $P$  formed by adding a single edge  $e$  to  $\tau_s$ , and let  $C_s$  be the associated closed walk in  $G$ .
4. A path-convex hull of  $K$  in  $R$  is that region  $\tilde{R}$  enclosed by a minimum-length walk  $C_s$  among those found in step 3.

From Lemma 2.3, Theorems 2.4 and 2.5, and the fact that the shortest  $P$ -enclosing walk in  $G'$  will always be a circuit, it follows that the above algorithm finds a path-convex hull. For the complexity of the algorithm: the dominating computation is step 3, which involves finding  $|V'|$  shortest path trees in  $G'$ . Each tree can be found in time  $O(|V'|^2)$  using Dijkstra’s algorithm [6], and determining enclosure of  $P$  can be done directly using “winding angles” (see [20]). Since  $|V'| \leq 2(\nu + k)$  we have the following result.

**THEOREM 2.6.** *The path-convex hull algorithm correctly finds a path-convex hull of  $K$  in  $R$  in time  $O((\nu + k)^3)$ .*

**3. An approximation scheme for the Steiner tree problem with obstacles.** In this section we give a procedure which for any  $\epsilon > 0$  produces a graph  $G_\epsilon$ , containing  $K$  and contained in  $R$ , such that a Steiner tree for  $K$  in  $G_\epsilon$  is an  $\epsilon$ -approximation to the Steiner minimal tree for  $K$  in  $R$ . The following simple lemma is central to such a construction.

LEMMA 3.1. *Let  $\tilde{R}$  be a path-convex hull for  $K$  in  $R$ . Then any Steiner minimal tree  $T^*$  for  $K$  in  $R$  has  $l(T^*) \geq \frac{1}{2}\rho(\tilde{R})$ .*

*Proof.* Simply note that  $T^*$  itself is a connected subregion of  $R$  containing  $K$ , with  $\rho(T^*) = 2l(T^*)$ . Since  $\tilde{R}$  is a path-convex hull for  $K$  in  $R$ , the inequality follows.  $\square$

We next state three properties of a Steiner minimal tree for  $K$  in  $R$  which were noted by Gilbert and Pollak [12, §§ 3.3, 3.4, 8.2] for the classical Steiner tree problem, and whose generalization to SMTO is immediate. A *free Steiner point* in  $T^*$  is a vertex of  $T^*$  which is not in  $V \cup K$ .

*Free Steiner Point Property.* A free Steiner point in  $T^*$  has exactly three adjacent edges in  $T^*$ ; each pair of edges forms a  $120^\circ$  angle.

*Wedge Property.* Let  $\Delta$  be a triangle, one side of which is in  $\partial R$  and the opposite angle is at least  $120^\circ$ . If no point of  $V \cup K$  is in  $\Delta$ , then neither is any free Steiner point.

*Number of Steiner Points Property.* The maximum number of free Steiner points in  $T^*$  is at most  $k - 2$ .

The first two properties are used to prove the following important technical lemma. For any  $\delta > 0$ , define  $S_\delta$  to be that set of points in  $R$  of the form  $(i\delta, j\delta)$ , where  $i$  and  $j$  are integers.

LEMMA 3.2. *Let  $T^*$  be a Steiner minimal tree of  $K$  in  $R$ , and let  $u$  be any free Steiner point in  $T^*$ . Then for any  $\delta > 0$ , there is at least one point of  $V \cup K \cup S_\delta$  which is visible from  $u$ , and whose distance to  $u$  is at most  $2\delta$ .*

*Proof.* Suppose there is no point of  $S_\delta$  which satisfies the requirements of the theorem. Let  $v$  be a point of the form  $(i\delta, j\delta)$  which lies nearest to  $u$ . Then  $\|u - v\| < \delta < 2\delta$ , and so  $v$  is not visible from  $u$ . This means that the line segment  $[u, v]$  must cross some line segment  $[w, x]$  of  $\partial R$ , with  $w, x \in V$ , at a point  $v'$  visible from  $u$ . Sweep the line  $[u, y]$  along  $y \in [w, x]$  from  $y = v'$  toward  $x$  and  $w$ , respectively, until either a point of  $V \cup K$  is intercepted or else we arrive at points  $y = w'$  and  $y = x'$  with  $\|u - w'\| = \|u - x'\| = 2\delta$ . In the first case we have a point of  $V \cup K$  satisfying the requirements of the theorem. In the second case, since  $\|u - v'\| < \delta$  it must be that  $\sphericalangle w'ux' > 120^\circ$ . By the Free Steiner Point Property for  $u$  the triangle  $\Delta = w'ux'$  contains at least one of the edges of  $T^*$  adjacent to  $u$ , and hence at least one vertex of  $T^*$ . But since  $\Delta$  contains no element of  $V \cup K$ , then by the wedge property it can contain no free Steiner point either, a contradiction. Thus the theorem is proved.  $\square$

We can now give an approximation algorithm for SMTO which is similar to the procedure given in [19] for the Euclidean minimal Steiner tree problem.

APPROXIMATION ALGORITHM FOR SMTO

**Input:** Polygonally bounded plane region  $R$ , terminal set  $K \subseteq R$ ,  $\epsilon > 0$ .

**Output:** Spanning graph  $T$  in  $R$  with  $l(T) \leq (1 + \epsilon)l(T^*)$ .

**The algorithm:**

1. Find a path-convex hull  $\tilde{R}$  of  $K$  in  $R$ .
2. For  $\delta = (\rho(\tilde{R}) / (12k - 24))\epsilon$ , construct the  $S_\delta \cup K$ -visibility graph  $G_\epsilon = (V_\epsilon, E_\epsilon)$  for  $\tilde{R}$ .
3. Return the Steiner minimal tree  $T$  of  $K$  in  $G_\epsilon$ .

THEOREM 3.3. *The tree  $T$  given by the approximation algorithm for SMTO is an  $\epsilon$ -approximation to the Steiner minimal tree for  $K$  in  $R$ .*

*Proof.* Let  $T^*$  be the Steiner minimal tree for  $K$  in  $R$ , chosen by Theorem 2.1 to lie entirely in  $\tilde{R}$ . Choose, for each vertex  $v$  of  $T^*$ , the nearest vertex  $v'$  in  $V_\varepsilon = V \cup K \cup S_\delta$  which is visible from  $v$ , so that by Lemma 3.2  $\|v - v'\| \leq 2\delta$ . For each edge  $[u, v]$  of  $T^*$  let  $u'$  and  $v'$  be the associated vertices as chosen above and let  $\Gamma_{uv}$  be a shortest path from  $u'$  to  $v'$  in  $\tilde{R}$ , so that by Theorem 2.2  $\Gamma_{uv}$  can be found in the  $K \cup \{u', v'\}$ -visibility graph for  $\tilde{R}$  and hence it is in  $G_\varepsilon$ . Let  $T'$  be the union of the edges of  $\Gamma_{uv}$ , taken over all edges  $[u, v]$  in  $T^*$ . We claim that  $T'$  is an  $\varepsilon$ -approximation for  $T^*$ , and therefore so is the Steiner minimal tree for  $K$  in  $G_\varepsilon$ . First observe that for each edge  $[u, v]$  in  $T^*$ , the path  $\Gamma'_{uv}: u', [u', u], u, [u, v], v, [v, v'], v'$  is a path from  $u'$  to  $v'$  in  $\tilde{R}$ , and thus,

$$\begin{aligned} l(\Gamma_{uv}) &\leq l(\Gamma'_{uv}) \\ &= \|u' - u\| + \|u - v\| + \|v - v'\| \\ &\leq \|u - v\| + 2r\delta, \end{aligned}$$

where  $r = 0, 1,$  or  $2$ , depending on whether  $0, 1,$  or  $2$  of  $u$  and  $v$  are free Steiner points in  $T^*$ . Summing over all edges of  $T^*$ , and using the Number of Steiner Points Property and the  $120^\circ$  property for  $T^*$ , we get

$$\begin{aligned} l(T') &\leq l(T^*) + 6(k - 2)\delta \\ &= l(T^*) + \frac{\varepsilon}{2}\rho(\tilde{R}). \end{aligned}$$

But by Lemma 3.1 we have  $l(T^*) \geq \frac{1}{2}\rho(\tilde{R})$ . Thus  $l(T') \leq (1 + \varepsilon)l(T^*)$ , and this completes the proof of the theorem.  $\square$

Theorem 3.3 now allows us to solve SMTO—at least to within an  $\varepsilon$ -approximation—by reducing it to a Steiner tree problem on graphs. This reduction, moreover, has the following important property.

**COROLLARY 3.4.** *Let  $\mathcal{C}$  be any class of instances of SMTO having the property that there exists an algorithm to find the Steiner minimal tree for any of the associated graphs  $G_\varepsilon$  whose running time is polynomial in the size of  $G_\varepsilon$ . Then there exists a fully polynomial approximation scheme for SMTO on the class  $\mathcal{C}$ .*

*Proof.* Simply note that  $\tilde{R}$  can be enclosed by a square with sides of length at most  $\frac{1}{2}\rho(\tilde{R})$ , and so the number  $n_\varepsilon$  of vertices in  $G_\varepsilon$  is at most

$$\nu + k + \left(\frac{\frac{1}{2}\rho(\tilde{R})}{\delta}\right)^2 = \nu + k + \left(\frac{6k - 12}{\varepsilon}\right)^2$$

which is a polynomial in  $\nu, k,$  and  $1/\varepsilon$ . Since  $G_\varepsilon$  can be constructed in time  $O(n_\varepsilon^2 \log n_\varepsilon)$ , the theorem follows.  $\square$

Thus any algorithm for finding Steiner minimal trees in graphs that performs well on the graphs  $G_\varepsilon$  can be used to find efficient  $\varepsilon$ -approximations, or even fully polynomial approximation schemes, for SMTO. Several possibilities can be found in the literature. The most promising in this context, however, appears to be that of Erickson, Monma, and Veinott [9], since it can take advantage of the essentially planar nature of SMTO. In order to use their method, we first prove a preliminary result related to planarity and SMTO. Let  $G$  be a graph laid out in the plane such that the edges of  $G$  are represented by straight-line segments. For vertices  $u$  and  $v$  in  $G$ , denote by  $d(u, v)$  the Euclidean length of a shortest path in  $G$  from  $u$  to  $v$ . The graph  $G$  is *plane* if it contains no crossing edges, and *weight planar* if every pair  $(u, v)$  and  $(w, x)$  of

crossing edges has the property

$$\|u - v\| + \|w - x\| > d(u, w) + d(v, x).$$

LEMMA 3.5.  $G_\epsilon$  is weight-planar.

*Proof.* Since we assume that no edge of  $G_\epsilon$  contains any vertices of  $G_\epsilon$  in its interior, if the line segments  $[u, v]$  and  $[w, x]$  cross, the intersection point  $y$  must lie in the interior of  $\tilde{R}$ . This means that there must be a point  $y'$  in the interior of, say, segment  $[y, u]$  such that  $y'$  is visible from  $w$ . By using the triangle inequality, we have

$$\begin{aligned} \|u - v\| + \|w - x\| &= \|u - y'\| + \|y' - y\| + \|y - w\| + \|v - y\| + \|y - x\| \\ &> \|u - y'\| + \|w - y'\| + \|v - y\| + \|y - x\| \\ &\cong d(u, w) + d(v, x). \end{aligned} \quad \square$$

In Theorem 5 of [19] it is shown that the Steiner minimal tree for a weight-planar graph is plane. We thus have the following theorem.

THEOREM 3.6. *The Steiner minimal tree for  $G_\epsilon$  is plane.*

The method of Erickson, Monma, and Veinott applies to finding minimum-concave-cost flows in “ $k$ -planar” networks and is a significant generalization of the method of Dreyfus and Wagner [7] for finding Steiner trees in graphs. We restate their method here in the context of SMTO. We suppose that the terminal set lies on a subset  $Q_1, \dots, Q_l$  of boundary polygons of  $\tilde{R}$  and in addition comprises points  $Q_{l+1}, \dots, Q_r$  in the interior of  $\tilde{R}$ , which for simplicity we will take to be polygons with one vertex. Each terminal  $u$  is assigned to a unique  $Q_i$  on which it lies, and we use the phrase “ $u$  is on  $Q_i$ ” to denote this. For each  $Q_i$  and each pair  $u$  and  $v$  of terminal points on  $Q_i$ , define an *interval*  $[u, v)$  between vertices  $u$  and  $v$  to be the subset of terminal points on  $Q_i$ —including  $u$  but not including  $v$ —visited by traversing  $Q_i$  clockwise from  $u$  to  $v$ . We will let  $[u, u)$  denote all of the terminals on  $Q_i$ . Define a *multi-subinterval* of  $K$  to be any nonempty subset  $I$  of  $K$  of the form  $I = [a_1, b_1) \cup \dots \cup [a_p, b_p)$ , where each  $[a_j, b_j)$  is an interval for a distinct  $Q_{i_j}$ ,  $j = 1, \dots, p$ . Note that  $Q_{i_1}, \dots, Q_{i_p}$  do not necessarily comprise all of the  $Q_i$ . Denote by  $\mathcal{I}$  the collection of multi-subintervals.

It turns out that when the Steiner minimal tree for  $K$  in  $G_\epsilon$  is known to be plane, it can be built up from Steiner minimal trees in  $G_\epsilon$  for sets  $I \in \mathcal{I}$  by recursively “patching together” trees for smaller multi-subintervals into trees for larger ones. To be precise, for any  $I \in \mathcal{I}$  and vertex  $v$  of  $G_\epsilon$ , define  $C(v, I)$  to be the minimum cost of a spanning graph  $T$  for  $I \cup \{v\}$  in  $G_\epsilon$ , and if  $|I| \geq 2$  define  $B(v, I)$  to be the minimum cost of a spanning graph  $T$  for  $I \cup \{v\}$  in  $G_\epsilon$  having the property that  $v$  is either in  $K$  or has at least two adjacent edges of  $T$ . The following recursive equations linking  $C$  and  $B$  are exactly equations (2') and (3) in [9], when applied to SMTO (cf. also equations (1) and (2) in [19]):

$$\begin{aligned} (1) \quad C(v, I) &= \min_{u \in V_\epsilon} \{B(u, I) + d(u, v)\}, \\ (2) \quad B(v, I) &= \min_{\substack{I_1, I_2 \in \mathcal{I} \\ I_1, I_2 \text{ partition } I}} \{C(v, I_1) + C(v, I_2)\}. \end{aligned}$$

By initializing  $C(v, \{a\}) = d(v, a)$  for  $a \in K$  and then solving (2) and then (1) successively for  $I$  of increasing cardinality and  $v \in V_\epsilon$ , we obtain the length of the Steiner minimal tree for  $K$  in  $G_\epsilon$  as  $C(a, K - \{a\})$  for any  $a \in K$ . The actual Steiner tree can be obtained by an appropriate labeling.

To analyze the complexity of computing  $C(a, K - \{a\})$ , let  $n_\epsilon$  be the number of vertices in  $G_\epsilon$ . Let  $q_i$  be the number of terminal points on  $Q_i$ , with  $a$  taken to be on  $Q_1$ , and let  $q = q_1 \prod_{i=2}^r (q_i + 1)$ . Then each evaluation of equation (1) takes  $O(n_\epsilon)$  steps, each evaluation of (2) takes  $O(q)$  steps, and  $|\mathcal{I}| \leq q^2$ , for total time  $O(n_\epsilon q^2 (n_\epsilon + q))$  to

compute  $C(a, K - \{a\})$ , plus an additional  $O(n_\varepsilon^3)$  time to compute all shortest path distances. The important thing to note here is that the term  $q$  does not depend on  $\varepsilon$ , but only on the location of the terminals in  $\tilde{R}$ . We thus have the following result.

**THEOREM 3.7.** *For fixed  $c$ , there exists a fully polynomial approximation scheme for any instance  $(K, R)$  of SMTO, where  $K$  lies on at most  $c$  boundary polygons and interior points of  $R$ .*

*Proof.* If  $K$  lies on at most  $c$  boundary polygons and interior points of  $R$ , then it must do so for any path-convex hull  $\tilde{R}$  of  $K$  in  $R$  as well. Further  $q \leq k^c$ , and so the complexity of the above algorithm is  $O(n_\varepsilon k^{2c}(n_\varepsilon + k^c))$ . The theorem then follows from Corollary 3.4.  $\square$

**4. Improvements on the path-convex hull.** Theorem 3.7, although sufficient to give an a priori bound on the complexity of solving an instance of SMTO, can be strengthened considerably. For one thing, the set of  $c$  polygons on which  $K$  is required to lie is allowed to include the exterior polygons of the path-convex hull as well as the actual boundary polygons of  $R$ . It turns out, in fact, that in the sense of Theorem 3.7 it is *always* more efficient to work with the smallest possible region known to contain the Steiner minimal tree for  $K$  in  $R$ . To be precise, for any polygonally bounded region  $S$  containing  $K$  define  $b(K, S)$  to be the minimum number of boundary polygons and interior points of  $S$  whose union contains all points of  $K$ . Then  $b(K, S)$  essentially determines the exponent  $c$  in the procedure given in Theorem 3.7, and hence should be minimized in order to minimize the complexity of the procedure.

**LEMMA 4.1.** *Let  $(K, R)$  be any instance of SMTO and  $\tilde{R}$  any connected polygonally bounded subregion of  $R$  containing  $K$ . Then  $b(K, \tilde{R}) \leq b(K, R)$ .*

*Proof.* Let  $Q_1, \dots, Q_r$  be any set of boundary polygons and interior points of  $R$  containing all points of  $K$ . Take any  $Q_i$  and suppose that  $Q_i$  bounds an interior forbidden region  $Q_i^0$  (the case for the exterior forbidden region is symmetric). Since  $\tilde{R} \cap Q_i^0 = \emptyset$  and  $\tilde{R}$  is connected, then  $Q_i^0$  is entirely contained in  $\tilde{Q}_j^0$  for one of the boundary polygons  $\tilde{Q}_j$  defining  $\tilde{R}$ . But since  $K$  is contained in  $\tilde{R}$  then all of the points of  $K$  on  $Q_i$  must also be contained in  $\tilde{Q}_j$ . Thus by assigning, for  $i = 1, \dots, r$  all points of  $K$  on  $Q_i$  to be on the associated polygon  $\tilde{Q}_j$  we have  $K$  contained on at most  $r$  polygons and interior points of  $\tilde{R}$ . By taking  $r = b(K, R)$  the lemma follows.  $\square$

It is known [2] that the computation of  $b(K, S)$  is NP-hard. The proof of Lemma 4.1 tells us, however, that for *any* choice of boundary polygons and interior points of  $R$  whose union contains  $K$ , we can always find at least as few of the same in any subregion  $\tilde{R}$  also containing  $K$ . Thus the search for smaller regions containing a Steiner minimal tree can only offer an improvement even if a nonoptimal choice of boundary points and polygons is made.

The discussion thus far makes it clear that any method which can decrease the size of a subregion of  $R$  known to contain a Steiner minimal tree for  $K$  will increase the efficiency with which the Steiner tree can be found. The literature suggests several important ways this can be done for both the classical Steiner minimal tree problem and the Steiner tree problem on graphs, which have interesting extensions to SMTO. They can, moreover, be applied to any combination, since they require no particular global conditions on the region on which they are applied. In what follows, therefore, we let  $R$  loosely refer to any region known to contain a Steiner minimal tree for the given instance of SMTO.

**4.1. Minimal path-convex hulls.** The path-convex hull algorithm, as given in § 2, will not necessarily give the setwise minimal path-convex hull for  $K$  in  $R$ . Bienstock and Monma [1] give valuable insight concerning minimal path-convex hulls in the



context of planar graphs. We first summarize the results in § 2 of that paper (in particular, Theorem 2 and Lemma 3) as they apply to SMTO. The proof follows essentially the discussion in that paper.

**THEOREM 4.2** (Bienstock and Monma). *Let  $T$  be any shortest path tree for  $K$  in  $R$ . Then there is a unique (setwise) minimal path-convex hull  $\tilde{R}$  of  $K$  in  $R$  which contains  $T$ , and further,  $\tilde{R}$  contains the unique maximal set of terminal points on the exterior boundary of any path-convex hull of  $K$  in  $R$ .*

This means that the best we can do as far as placing points on the exterior boundary of a path-convex hull is to find the unique minimal path-convex region containing any arbitrarily chosen shortest path tree  $T$ . One such method is given in § 2 of [1]. When treating SMTO, however, we can do this by a simple modification of the path-convex hull algorithm given in § 2 of this paper. We first modify the circuits  $C'_s(e)$  found in step 3 by pushing their sides “inward” as much as possible. Specifically, let  $\Gamma_x$  be the unique  $s$ - $x$  path in  $\tau_s$ ,  $x \in V'$ , and let  $e = (u, v)$  be labeled such that  $\Gamma_u$  is to the “left” of  $\Gamma_v$  relative to the  $P$  found in step 2. Now form the “rightmost”  $s$ - $u$  path  $\Gamma'_u$ :  $s = u_0, u_1, \dots, u_k = u$  in  $G'$  passing to the left of  $P$  by setting  $u_{k+1} = v$  and for  $i = k, k = 1, \dots, 2$  recursively choosing the first edge  $(u_{i-1}, u_i)$  in a clockwise sweep from  $(u_i, u_{i+1})$  satisfying (a)  $l(\Gamma_{u_i}) = l(\Gamma_{u_{i-1}}) + d_{u_{i-1}u_i}$  and (b) the path  $\Gamma_{u_{i-1}}, u_i, \dots, u_k$  passes to the left of  $P$  with respect to  $\Gamma_v \cup \{e\}$ . Find the analogous “leftmost”  $s$ - $v$  path  $\Gamma'_v$ :  $s = v_0, v_1, \dots, v_j = v$ , passing to the right of  $P$  by the mirror image of the procedure given above. Then  $\Gamma'_u \cup \{e\} \cup (\Gamma'_v)^{-1}$  will contain a unique  $P$ -enclosing circuit  $C''_s(e)$ , with corresponding walk  $\tilde{C}_s(e)$  enclosing  $K$  in  $R$ . We now appeal to the following result.

**LEMMA 4.3.** *The boundary of a minimal path-convex hull of  $K$  in  $R$  is among the  $\tilde{C}_s(e)$  found in the modified procedure given above.*

*Proof.* Let  $C$  be the boundary of the minimal  $T$ -enclosing path-convex hull of  $K$  in  $R$ , so that the associated walk  $C'$  in  $G'$  is the innermost  $P$ -enclosing circuit in  $G'$ . Now choose any  $s$  in  $C'$  and let  $\tau_s$  be the tree found in step 2 of the procedure, with  $\Gamma_x, x \in V'$ , the associated  $s$ - $x$  paths in  $\tau_s$ . The weight-planarity of  $G'$  ensures that the  $\Gamma_x$  do not have crossing edges. Since  $C'$  encloses  $P$ , then there must be at least one edge  $e = (u, v) \in C' - \tau_s$  whose associated circuit  $C'_s(e)$  encloses  $P$ . Let  $C'' = C''_s(e)$  be the circuit produced from  $C'_s(e)$  by the modified procedure given above. When we construct  $C''$ , we find it has the same length as  $C'$ , and hence  $C''$  completely encloses  $C'$  as well as containing both  $e$  and  $s$ . Now suppose  $C' \neq C''$  and assume by symmetry that they differ on their “left-hand” sides with respect to  $e$  and  $s$ . Let  $\Gamma'_u$ :  $s = u_0, \dots, u_k = u$  be the left-hand path of  $C''$  and let  $i$  be the largest index for which  $(u_{i-1}, u_i)$  is not on  $C'$ . If  $(x, u_i), u_{i-1} \neq x \neq u_{i+1}$ , is the corresponding edge of  $C'$  adjacent to  $u_i$ , then  $(x, u_i)$  must occur in a clockwise sweep from  $(u_i, u_{i+1})$  to  $(u_{i-1}, u_i)$ . Further, since  $C'$  has minimal length,  $\Gamma_{u_i}$  and  $\Gamma_x$  have the same length as the corresponding sections of  $C'$ , and hence  $l(\Gamma_{u_i}) = l(\Gamma_x) + d_{x, u_i}$ . Finally, since  $\Gamma_x$  lies outside  $C'$  and does not cross  $\Gamma_u$ , the path  $\Gamma_x, u_i, \dots, u_k$  must pass to the left of  $P$  with respect to  $\Gamma_v \cup \{e\}$ , and this contradicts the choice of  $u_{i-1}$ . Thus  $C' = C''$  and so the corresponding walk  $\tilde{C}_s(e)$  is the boundary of a minimal path-convex hull of  $K$  in  $R$ .  $\square$

The modified procedure now chooses, in step 4, the unique innermost of the minimum length  $C''_s(e)$  found in the modified step 3. Since the (unique) innermost  $C''_s(e)$  for each  $s$  can be found in additional time  $O(|E'|) = O((\nu + k)^2)$  (again using “winding angles”), and the innermost of the  $C''_s(e)$  overall  $s \in V'$  can be found by a single pass through the  $|E'| |V'|$  edges of the set of these cycles, then there is no resulting increase in complexity using the modified procedure. We thus have the following result.

**COROLLARY 4.4.** *A path-convex hull of  $K$  in  $R$  containing the unique maximal set of vertices on the exterior boundary can be found in time  $O((\nu + k)^3)$ .*

**4.2. Path-convex hulls with respect to interior forbidden regions.** For the purpose of bounding the size of the region containing a Steiner minimal tree it is necessary to take a path-convex hull with respect to the exterior polygon (if one exists) of  $R$ . The results of § 2 apply as well, however, when the path-convex hull is taken with respect to *any* boundary polygon (cf. [1, § 3] for planar graphs). To be precise, for interior polygon  $P_i$  of  $R$ , conceptually turn the visibility graph  $G_K$  “inside-out,” so that  $P_i$  forms the exterior boundary. Now the  $P_i$ -path-convex hull of  $K$  in  $R$  is defined to be the path-convex hull with respect to the new layout, and both the results and the path-convex hull algorithm in § 2 apply in this context. Define the  $P_i$ -boundary of any polygonally bounded region  $\tilde{R} \subseteq R$  to be the boundary polygon of  $\tilde{R}$  enclosing  $P_i$  (or enclosed by  $P_i$  if  $P_i$  is the exterior boundary). Then generalizing to Corollary 4.4 we have the following result.

**COROLLARY 4.5.** *For any boundary polygon  $P_i$  of  $R$ , a  $P_i$ -path-convex hull of  $K$  in  $R$  containing the unique maximal set of vertices on its  $P_i$ -boundary can be found in time  $O((\nu + k)^3)$ .*

The modified path-convex hull algorithm can thus be applied successively to each boundary polygon  $P_0, \dots, P_m$ , resulting in a region  $R^*$  which, in a sense, contains the unique maximal set of terminals on its boundary polygons which can be achieved by using  $P_i$ -path-convexity alone.

**4.3. Convex hulls.** As a consequence of the discussion in § 2, we obtain a property of the region  $R^*$  which is analogous to the convex hull property given in [12], § 3.5. Let  $R$  be any polygonally bounded region, and let  $v$  be a point on boundary polygon  $P_i$  of  $R$ . Then  $v$  is called a *simple boundary point* of  $R$  if it is not a terminal point and is contained in no other boundary polygon of  $R$ . By applying the procedure given in § 2, we have the following result.

**COROLLARY 4.6.** *Let  $\tilde{R}$  be a  $P_i$ -convex hull of  $K$  in  $R$ . Then every simple boundary point of the  $P_i$ -boundary of  $\tilde{R}$  has its interior angle (with respect to  $\tilde{R}$ ) greater than or equal to  $180^\circ$ .*

**4.4. Steiner hulls and extensions.** Cockayne [5] and Hwang et al. [14], in the context of the classical Steiner tree problem, provide methods for further reducing the size of a region known to contain the Steiner minimal tree. The reduced regions are often much smaller than the path-convex hull, and the methods used are significantly different from those related to the construction of the path-convex hull. To extend these methods to SMTO, we first give the basic constructions of Cockayne and of Hwang et al. as they apply to SMTO. Let  $\Gamma$  be a path in some boundary polygon  $P_i$  connecting points (not necessarily vertices)  $u$  and  $v$  of  $P_i$ , and let  $\Lambda$  be a path in  $R$  connecting  $u$  to  $v$ , which is otherwise disjoint from  $\Gamma$ . Finally let  $S$  be the region enclosed by  $\Gamma \cup \Lambda$ . Both of the proposed criteria require the following common condition:

- (1)  $S - \Lambda$  contains neither  $P_i^0$  nor any point of  $K$ .

The criteria now differ depending on the precise structure of  $\Lambda$ . In what follows, angles will always refer to interior angles with respect to  $S$ .

**3-point criterion (Cockayne).**  $\Lambda = [u, a] \cup [a, v]$  such that (1) holds and

- (2)  $a \in K$ ;
- (3)  $\sphericalangle uav \geq 120^\circ$ .

- 4-point criterion** (Hwang et al.).  $\Lambda = [u, a] \cup [a, b] \cup [b, v]$  such that (1) holds and
- (2)  $a, b, \in K$  and  $[a, b]$  is a boundary edge of  $R$ ;
  - (3)  $\sphericalangle uab$  and  $\sphericalangle abv$  are both convex and at least  $120^\circ$ ;
  - (4) the point  $O$  of the intersection of  $[u, b]$  and  $[a, v]$  satisfies  $\sphericalangle uOv \geq \sphericalangle uab + \sphericalangle abv - 150^\circ$ .

To find  $\Gamma$  and  $\Lambda$  satisfying the 3- or 4-point criterion in  $O(k(\nu + k)^2)$  time, look at each terminal (or pair of terminals connected by a boundary edge) and check each pair of adjacent edges in the  $K$ -visibility graph to see whether or not the 3- (or 4-) point criterion is satisfied. Note that since  $\partial R$  forms a planar graph, the number of boundary edges connecting pairs of terminals is  $O(k)$ . The key results of Cockayne and Hwang et al. (Theorem 1 of [5] and the main theorem of [14]) can be extended to SMT0 via the following theorem.

**THEOREM 4.7.** *Let  $\Gamma$  and  $\Lambda$ , as described above, satisfy either the 3-point or 4-point criterion. Then the region  $S - \Lambda$  contains no part of a Steiner minimal tree for  $K$  in  $R$ .*

*Proof.* First, let  $T'$  be the Steiner minimal tree for  $K$  in the region  $R'$  obtained from  $R$  by removing all obstacles lying inside  $S$ . Then  $l(T') \leq l(T^*)$ , the length of the Steiner minimal tree in  $R$ , and so if  $T'$  can be shown to be shorter than any spanning graph passing through  $S - \Lambda$ , then  $T^*$  will also contain no part in  $S - \Lambda$ . Second, let  $\Gamma_0$  be the (unique) shortest path from  $u$  to  $v$  in  $S$  (without obstacles), and let  $S_0$  be the region enclosed by  $\Lambda \cup \Gamma_0$ . Then a simple convexity argument shows that  $T'$  must be contained entirely in  $S_0$ .

*Case 1.* Suppose that all angles of  $\Lambda$  are convex. Then  $\Gamma_0 = [u, v]$ , and we can apply the results of Cockayne and of Hwang et al. as cited above. The proofs of these theorems say, in effect, that if (i)  $a$  (and  $b$ , respectively) are in  $K$ , (ii)  $S_0 - \Lambda$  contains no elements of  $K$ , and (iii)  $T$  is a spanning graph for  $K$  which passes into  $S_0$  only via the edges  $[u, a]$  and  $[a, v]$  ( $[b, v]$ , respectively), then that part of  $T$  in  $S_0 - \Lambda$  can be replaced by appropriate segments of  $\Lambda$  to yield a smaller length spanning graph for  $K$ . (The requirement in these theorems that  $u$  and  $v$  belong to  $K$  is irrelevant here.) Since that is precisely the situation in this case,  $T'$  can contain no part in  $S - \Lambda$ .

*Case 2.* Suppose that  $\Gamma$  and  $\Lambda$  satisfy the 3-point criterion, with  $\sphericalangle uav \geq 180^\circ$ . Then  $S_0 = \Lambda$ , and we are done.  $\square$

Theorem 4.7 can be used to produce a region, similar to Cockayne's "Steiner hull," that is guaranteed to contain the Steiner minimal tree for  $K$  in  $R$ . Such a region, which will be called the *path-Steiner hull* for  $K$  in  $R$ , is obtained by successively performing the following operation until it can no longer be applied.

*Wedge operation.* Let  $\Gamma \subset P_i$  and  $\Lambda$  be paths as defined above which satisfy either the 2- or 3-point criterion, and let  $S$  be the region enclosed by  $\Gamma \cup \Lambda$ . Replace  $S$  by  $\Lambda$ , and take the  $P_i$ -path-convex hull of  $K$  in the resulting region.

Figure 4 shows the application of a wedge operation. Theorem 4.7 along with Theorem 2.1 immediately imply the following corollary.

**COROLLARY 4.8.** *A path-Steiner hull of  $K$  in  $R$  always contains a Steiner minimal tree for  $K$  in  $R$ .*

The next result gives the complexity of finding a path-Steiner hull.

**THEOREM 4.9.** *Let  $m$  be the number of boundary polygons for  $R$ . Then a path-Steiner hull for  $K$  in  $R$  can be found in  $O((m + k)(\nu + k)^3)$  steps.*

*Proof.* We first prove that the wedge operation can be applied at most  $6(m + k) - 12$  times. To show this, start with the graph  $G$  whose vertices are the points of  $K$ , together with a point  $v_i$  chosen in the interior of polygon  $P_i$ ,  $i = 1, \dots, m$ , and whose edges are of the form  $(u, v_i)$  where  $u \in K$  is on  $P_i$ . Then  $G$  is bipartite and planar, with  $m + k$

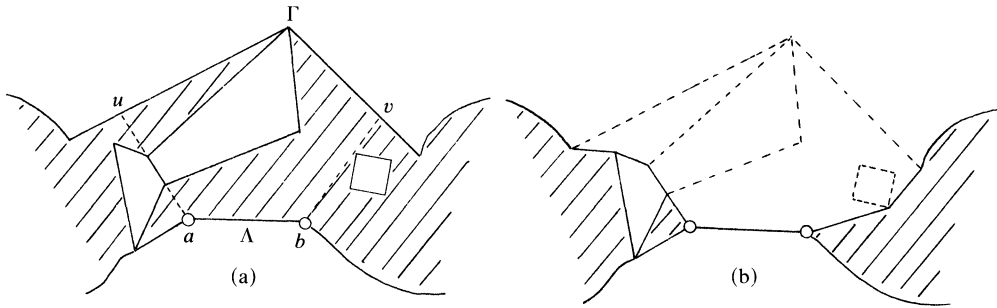


FIG. 4. Wedge operation.

vertices. Now each time a wedge operation is performed, we add an edge to  $G$  connecting the points  $a$  (and  $b$ ) in  $\Lambda$  to the vertex  $v_i$  such that  $\Gamma \subseteq P_i$ . The wedging operation may coalesce obstacles; in this case the points  $a$  (and  $b$ ) are connected to the appropriate  $v_i$  so as to preserve planarity in  $G$ . We also note that a point may be attached to the same boundary more than once, so that multiple edges are created. The 2- and 3-point criteria imply, however, that each time a point  $a$  (or  $b$ ) is attached to boundary  $P_i$ , it lies on an angle of at least  $120^\circ$  interior to  $P_i$ . Thus a point cannot be attached more than three times to the same boundary, and so edges in  $G$  can have multiplicity no greater than 3. The construction of  $G$  ensures that it always remains bipartite and planar, and has edge multiplicity at most 3. Since a bipartite planar graph with  $n$  vertices and no multiple edges can have no more than  $2n - 4$  edges,  $G$  can have at most  $6(m + k) - 12$  edges. Further, a new edge is added for each wedge operation, and so the number of wedge operations is at most  $6(m + k) - 12$ , as asserted.

For the complexity of the entire process, we note that the wedge operation will add no new vertices to  $V \cup K$ , since if either  $u$  or  $v$  in  $\Gamma$  is not a point of  $V \cup K$ , then replacing  $S$  by  $\Lambda$  will make this point a simple vertex of  $P_i$  with interior angle less than  $180^\circ$ . By Corollary 4.6, the  $P_i$ -path convex hull  $\tilde{R}$  will not contain this point, and hence by Lemma 2.3,  $\partial\tilde{R}$  will contain no new vertices. It follows that the  $P_i$ -path-convex hull can be computed in time  $O((\nu + k)^3)$ , and since this dominates the search for points satisfying the 2- and 3-point properties, then the overall complexity is  $O((m + k)(\nu + k)^3)$  as required.  $\square$

To end this section, we show how the 2- and 3-point criteria can be extended considerably by applying them in groups, so as to ignore obstacles which would prevent them from being applied individually. In particular, let  $\Gamma$  and  $\Lambda$  be paths as defined for the 2- and 3-point criteria, with  $S$  the enclosed region. Consider the following condition.

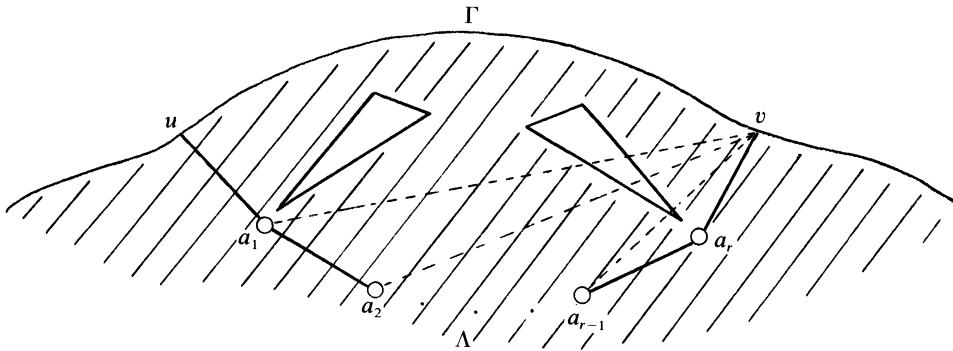
**$r$ -point criterion.**  $\Lambda = [u, a_1] \cup [a_1, a_2] \cup \dots \cup [a_r, v]$  such that (1) holds and

- (2) if all obstacles are removed from inside  $S$ , then there exists a series of wedging operations which can be performed to reduce  $S$  to  $\Lambda$ .

An example is given in Fig. 5, where the 3-point wedging operation can be applied successively with  $\Gamma_1 = \Gamma$ ,  $\Lambda = [u, a_1] \cup [a_1, v]$  and  $\Gamma_i = [u, a_{i-1}]$ ,  $\Lambda_i = [a_{i-1}, a_i] \cup [a_i, v]$   $i = 2, \dots, r$ . As an extension to Theorem 4.7 we have the following result.

**COROLLARY 4.10.** Let  $\Gamma$  and  $\Lambda$  as described above satisfy the  $r$ -point criterion. Then the region  $S - \Lambda$  contains no part of a Steiner minimal tree for  $K$  in  $R$ .

*Proof.* The proof follows from repeated applications of Theorem 4.7, when we use the argument in the first paragraph of the proof of that theorem.  $\square$

FIG. 5.  $r$ -point criterion.

As a simple application of Corollary 4.10, we can relax the requirement in the 4-point criterion that  $\sphericalangle uab$  and  $\sphericalangle abv$  be convex, for if either angle fails to be convex, then the  $r$ -point criterion applies by twice using the appropriate 3-point criterion. The wedging operation can now be extended to  $\Gamma$  and  $\Lambda$ , thus satisfying the  $r$ -point criterion to add to the repertoire of operations which reduce the region known to contain the Steiner minimal tree. It is immediate from the proof of Theorem 4.9 that the wedging operation continues to be limited to  $6(m+k)-12$  applications, with complexity  $O((\nu+k)^2)$  time per application. Unfortunately the complexity of finding  $\Gamma$  and  $\Lambda$  satisfying the  $r$ -point criterion now dominates the wedge operation itself. It is not even clear that the  $r$ -point criterion can be recognized in time which grows polynomial in  $r$ , although it is polynomial for fixed  $r$ .

**5. Summary and extensions.** As shown in this paper, SMTO provides a rich class of important Steiner tree problems whose solution methods draw from both graph theory and computational geometry. The results in § 4, moreover, show the areas where the greatest improvement in computational complexity can be achieved. Particularly intriguing are results extending the material in § 4, since this is where the specific geometry of the Steiner minimal tree pays a critical role. An interesting extension to the work in this paper is suggested by the work of Schäffer and Van Wyk [21]. Suppose the region  $R$  has boundaries made up not of straight-line segments, but rather of smooth non-self-intersecting curves. There exist straightforward analogues to most of the results in this paper, although the description and complexity of the corresponding procedures are significant unresolved issues. Little has been done even for the associated problem of shortest paths with obstacles, and so we leave this whole area as an interesting topic for future research.

**Acknowledgment.** The author would like to thank Dan Bienstock for pointing out a substantial error in the original Path-Convex Hull Algorithm of § 2.

## REFERENCES

- [1] D. BIENSTOCK AND C. L. MONMA, *Optimal enclosing regions in planar graphs*, Networks, to appear.
- [2] ———, *On the complexity of covering the faces of a planar graph*, SIAM J. Comput., 17 (1988), pp. 53–76.
- [3] S. K. CHANG, *The generation of minimal trees with Steiner topology*, J. Assoc. Comput. Mach., 19 (1972), pp. 699–711.
- [4] F. R. K. CHUNG AND R. L. GRAHAM, *Steiner tree for ladders*, Ann. Discrete Math., 2 (1978), pp. 173–200.

- [5] E. J. COCKAYNE, *On the efficiency of the algorithm for Steiner minimal trees*, SIAM J. Appl. Math., 18 (1970), pp. 150-159.
- [6] E. W. DIJKSTRA, *A note on two problems in connection with graphs*, Numer. Math., 1 (1959), pp. 269-271.
- [7] S. E. DREYFUS AND R. A. WAGNER, *The Steiner problem in graphs*, Networks, 1 (1972), pp. 195-207.
- [8] D. Z. DU, F. K. HWANG, AND S. C. CHAO, *Steiner minimal tree for points on a circle*, Proc. Amer. Math. Soc., 95 (1985), pp. 613-618.
- [9] R. E. ERICKSON, C. L. MONMA, AND A. F. VEINOTT, *Send and split method for minimum-concave-cost network flows*, Math. of Oper. Res., 12 (1987), pp. 634-664.
- [10] M. R. GAREY, R. L. GRAHAM, AND D. S. JOHNSON, *The complexity of computing Steiner minimal trees*, SIAM J. Appl. Math., 32 (1977), pp. 835-859.
- [11] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.
- [12] E. N. GILBERT AND H. O. POLLAK, *Steiner minimal trees*, SIAM J. Appl. Math., 16 (1968), pp. 1-29.
- [13] S. L. HAKIMI, *Steiner's problem in graphs and its implications*, Networks, 1 (1971), pp. 113-133.
- [14] F. K. HWANG, G. D. SONG, G. Y. TING, AND D. Z. DU, *A decomposition theorem on Euclidean Steiner minimal trees*, Discrete Comput. Geom., 3 (1988), pp. 367-382.
- [15] I. N. KATZ AND L. COOPER, *Facility location in the presence of forbidden regions*, European J. Oper. Res., 6 (1981), pp. 166-173.
- [16] A. L. LIESTMAN, *Construction of Steiner trees with obstacles in the plane*, Master's thesis, University of Illinois, Urbana-Champaign, Urbana, IL, 1978.
- [17] T. LOZANO-PEREZ AND M. A. WESLEY, *An algorithm for planning collision-free paths among polyhedral obstacles*, Comm. ACM, 22 (1979), pp. 560-570.
- [18] Z. A. MELZAK, *On the problem of Steiner*, Canad. Math. Bull., 4 (1961), pp. 143-148.
- [19] J. S. PROVAN, *Convexity and the Steiner tree problem*, Networks, 18 (1988), pp. 55-72.
- [20] ———, *Shortest enclosing walks and cycles in embedding graphs*, Inform. Process. Lett., to appear.
- [21] A. A. SHÄFFER AND C. J. VAN WYK, *Convex hulls of piecewise-smooth Jordan curves*, J. Algorithms, 8 (1987), pp. 66-94.
- [22] M. I. SHAMOS, *Computational Geometry*, Ph.D. dissertation, Yale University, New Haven, CT, 1975.
- [23] M. SHAMIR AND A. SCHORR, *On shortest paths in polyhedral spaces*, SIAM J. Comput., 15 (1986), pp. 193-215.
- [24] M. L. SHORE, L. R. FOULDS, AND P. B. GIBBONS, *An algorithm for the Steiner problem in graphs*, Networks, 12 (1982), pp. 323-333.
- [25] J. M. SMITH, *Steiner minimal trees with obstacles*, Tech. Report of the Department of Industrial Engineering and Operations Research, University of Massachusetts, Amherst, MA, 1982.
- [26] J. M. SMITH, D. T. LEE, AND J. S. LIEBMAN, *An  $O(N \log N)$  heuristic algorithm for Steiner minimal tree problems on the Euclidean metric*, Networks, 11 (1981), pp. 23-39.
- [27] J. A. WALD AND C. J. COLBOURN, *Steiner trees, partial 2-trees, and minimum IFI networks*, Networks, 13 (1983), pp. 159-167.
- [28] G. E. WANGDAHL, S. M. POLLOCK, AND J. B. WOODWARD, *Minimum trajectory pipe routing*, J. Ship Research, 18 (1974), pp. 46-49.
- [29] P. WINTER, *Steiner problem in networks: a survey*, Networks, 17 (1987), pp. 129-168.

## NONDETERMINISTIC SPACE IS CLOSED UNDER COMPLEMENTATION\*

NEIL IMMERMANT†

**Abstract.** In this paper we show that nondeterministic space  $s(n)$  is closed under complementation for  $s(n)$  greater than or equal to  $\log n$ . It immediately follows that the context-sensitive languages are closed under complementation, thus settling a question raised by Kuroda [*Inform. and Control*, 7 (1964), pp. 207–233].

**Key words.** nondeterministic space, context-sensitive language, complementation, first-order expressibility, computational complexity

**AMS(MOS) subject classification.** 68O25

**1. Introduction.** In this paper we show that nondeterministic space  $s(n)$  is closed under complementation for  $s(n)$  greater than or equal to  $\log n$ . It immediately follows that the context-sensitive languages are closed under complementation, thus settling a question raised by Kuroda in 1964 [9]. See Hartmanis and Hunt [4] for a discussion of the history and importance of this problem, and Hopcroft and Ullman [5] for all relevant background material and definitions.

The history behind the proof is as follows. In 1981 we showed that the set of first-order inductive definitions over finite structures is closed under complementation [6]. This holds with or without an ordering relation on the structure. If an ordering is present the resulting class is  $P$ . Many people expected that the result was false in the absence of an ordering. In 1983 we studied first-order logic, with ordering, with a transitive closure operator. We showed that NSPACE  $[\log n]$  is equal to (FO + pos TC), i.e., first-order logic with ordering, plus a transitive closure operation, in which the transitive closure operator does not appear within any negation symbols [7]. Now we have returned to the issue of complementation in light of the recent results on the collapse of the log space hierarchies [10], [2], [14]. We have shown that the class (FO + pos TC) is closed under complementation. Our main result follows. In this paper we give the proof in terms of machines and then state the result for transitive closure as Corollary 3. The question of whether (FO + pos TC) *without ordering* is closed under complementation remains open.

Our work in first-order expressibility led to our proof that nondeterministic space is closed under complementation. However, because first-order expressibility classes are not directly relevant to the proofs in this paper, we omit those definitions here. The interested reader should refer to [7] for all these definitions. Note that the proof of Theorem 3.3 in [7] is more complicated than the proof of Theorem 1, though it is quite similar to it. The same is true of the proof in [6] that the first-order inductive formulas are closed under complementation.

### 2. Results.

**THEOREM 1.** For any  $s(n) \geq \log n$ ,

$$\text{NSPACE}[s(n)] = \text{co-NSPACE}[s(n)].$$

*Proof.* We do this by two lemmas. We will show that counting the exact number of reachable configurations<sup>1</sup> of an NSPACE  $[s(n)]$  machine can be done in

\* Received by the editors August 6, 1987; accepted for publication (in revised form) November 2, 1987. This research was supported by National Science Foundation grant DCR-8603346.

† Computer Science Department, Yale University, New Haven, Connecticut 06520.

<sup>1</sup> The *configuration* of a Turing machine is the contents of its work tapes, the positions of its heads, and its state. Note that for  $s(n) \geq \log n$ , the number of possible configurations is less than  $c^{s(n)}$  for some constant  $c$ , and thus can be written in  $O[s(n)]$  space.

NSPACE  $[s(n)]$  (Lemma 2). Lemma 1 says that once this number has been calculated we can detect rejection as well as acceptance. Note the similarity between Lemma 1 and a similar result about census functions in [12].

**LEMMA 1.** *Suppose we are given an NSPACE  $[s(n)]$  machine  $M$ , a size  $s(n)$  initial configuration, START, and the exact number  $N$  of configurations of size  $s(n)$  reachable by  $M$  from START. Then we can test in NSPACE  $[s(n)]$  if  $M$  rejects.*

*Proof.* Our NSPACE  $[s(n)]$  tester does the following. It initializes a counter to 0, and a target configuration to the lexicographically first string of length  $s(n)$ . For each such target either we guess a computation path of  $M$  from START to target and increment both counter and target, or we simply increment target. For each target that we have found a path to, if it is an accept configuration of  $M$  then we reject. Finally, if when we are done with the last target the counter is equal to  $N$ , we accept; otherwise we reject. Note that we accept if and only if we have found  $N$  reachable configurations, none of which is accepting. (Suppose that  $M$  accepts. In this case there can be at most  $N - 1$  reachable configurations that are not accepting, and our machine will reject. On the other hand, if  $M$  rejects then there are  $N$  nonaccepting reachable configurations. Thus our nondeterministic machine can guess paths to each of them in turn and accept.) That is, we accept if and only if  $M$  rejects.  $\square$

**LEMMA 2.** *Given START, as in Lemma 1, we can calculate  $N$ —the total number of configurations of size  $s(n)$  reachable by  $M$  from START—in NSPACE  $[s(n)]$ .*

*Proof.* Let  $N_d$  be the number of configurations reachable from START in at most  $d$  steps. The computation proceeds by calculating  $N_0, N_1$ , and so on. By induction on  $d$  we show that each  $N_d$  may be calculated in NSPACE  $[s(n)]$ . The base case  $d = 0$  is obvious.

*Inductive step.* Given  $N_d$  we show how to calculate  $N_{d+1}$ . As in Lemma 1 we keep a counter of the number of  $d + 1$  reachable configurations, and we cycle through all the target configurations in lexicographical order. For each target we do the following: Cycle through all  $N_d$  configurations reachable in at most  $d$  steps; again we find a path of length at most  $d$  for each reachable one, and if we do not find all  $N_d$  of them then we will reject. For each of these  $N_d$  configurations check if it is equal to target, or if target is reachable from it in one step. If so then increment the counter, and start on target +1. If we finish visiting all  $N_d$  configurations without reaching target, then just start again on target +1 without incrementing the counter. When we have completed this algorithm for all targets, our counter contains  $N_{d+1}$ . Since  $N$  is bounded above by  $c^{s(n)}$  for some constant  $c$ , the space needed is  $O[s(n)]$ .

To complete the proof of the lemma and the theorem note that  $N$  is equal to the first  $N_d$  such that  $N_d = N_{d+1}$ .  $\square$

*Remark.* In our original statement of Theorem 1 we made the assumption that  $s(n)$  is space-constructible. However, the standard definition of a nondeterministic Turing machine having space complexity  $s(n)$  is that, "...no sequence of choices enables it to scan more than  $s(n)$  cells ...," [5]. Thus, the above proof works even if  $s(n)$  is not space-constructible. We just let  $s(n)$  increase as needed.

The following corollary is immediate.

**COROLLARY 1.** *The class of context-sensitive languages is closed under complementation.*

*Proof.* Kuroda showed in 1964 that CSL = NSPACE  $[n]$  [9].  $\square$

The  $k$ th level of the log space alternating hierarchy ( $\Sigma_k$  ALOG) is defined to be the set of problems accepted by alternating log space Turing machines that make at most  $k - 1$  alternations and begin in an existential state. Recently Lange, Jenner, and Kirsig [10] showed that this hierarchy collapsed to the second level,  $\Sigma_2$  ALOG. This



result was then extended by several authors [2], [14] who showed that the log space oracle hierarchy collapses to  $L^{NL}$ . Here  $L = DSPACE[\log n]$ , and  $NL = NSPACE[\log n]$ . The logspace oracle hierarchy is given by  $\Sigma_1 OLOG = NL$ , and  $\Sigma_{k+1} OLOG = NL^{\Sigma_k OLOG}$ . In the case of the polynomial time hierarchy, the oracle and alternating hierarchies are identical, but they appeared to be different in the log space case. We knew that the log space oracle hierarchy is equal to  $(FO + TC)$ . This, together with the above results, led us to expect Theorem 1. The following is again immediate.

**COROLLARY 2.** *The log space alternating hierarchy and the log space oracle hierarchy both collapse to  $NSPACE[\log n]$ .*

In [7] we showed that  $NL$  is equal to  $(FO + \text{pos } TC)$ . In Theorem 3.3 of [7] we also showed that any problem in  $NL$  may be expressed in the form  $TC[\varphi](\bar{0}, \overline{\text{max}})$ , where  $\varphi$  is a quantifier-free first-order formula, and  $0$  and  $\text{max}$  are constant symbols. It now follows that the same is true for the class  $(FO + TC)$ .

**COROLLARY 3.** (1)  $NSPACE[\log n] = (FO + \text{pos } TC) = (FO + TC)$ .

(2) *Any formula in  $(FO + TC)$  may be expressed in the form  $TC[\varphi](\bar{0}, \overline{\text{max}})$ , where  $\varphi$  is a quantifier-free first-order formula.*

Michael Fischer has observed that we can now diagonalize nondeterministic space and thus easily prove a tight hierarchy theorem for nondeterministic space. Although Corollary 4 is not new, our techniques give a much simpler proof than was previously known. (See Chapter 12 in [5] for the old proof.)

**COROLLARY 4.** *For any tape-constructible  $s(n) \geq \log n$ ,*

$$\lim_{n \rightarrow \infty} \frac{t(n)}{s(n)} = 0$$

*implies*

$$NSPACE[t(n)] \neq NSPACE[s(n)].$$

**3. Conclusions and directions for future work.** Most of the interesting questions concerning the power of nondeterminism remain open. We still do not know whether nondeterministic space is equal to deterministic space, or whether Savitch's Theorem [15] is optimal. It is interesting to consider whether our proof method can be extended to answer these questions, or to tell us anything new about nondeterministic time.

Soon after we proved Theorem 1, Tompa et al. [1] gave two extensions: they proved that  $LOG(CFL)$ —the set of problems log space-reducible to a context-free language—is closed under complementation, and they showed that symmetric log space (cf. [11], [13]) is contained in  $ZPLP$ , "... the class of errorless probabilistic Turing machines running in  $O[\log n]$  space and polynomial expected time." We suggest the following open problems.

- (1) Is  $(FO \text{ without } \leq + \text{pos } TC)$  closed under complementation?
- (2) Is symmetric log space, equivalently  $(FO + \text{pos } STC)$ , closed under complementation?
- (3) Is  $NL$  equal to a complexity class that was previously known to be closed under complementation, e.g.,  $L$ ,  $AC^1$ , or  $DSPACE[\log^2 n]$ ?
- (4) In the proof of Theorem 1 we made use of the linear space compression theorem, Theorem 12.1 in [5]. Our actual construction multiplies the space bound by a factor of about eight. It is interesting to ask how much this can be reduced. Note in particular that if we could complement  $\log n$  times, while only increasing the space bound by a constant factor, then it would follow that  $NL = AC^1$ .

**Acknowledgments.** I thank Sam Buss, Mike Fischer, Lane Hemachandra, Steve Mahaney, and Joel Seiferas, all of whom contributed comments and corrections to this paper.

**Note added in proof.** See Róbert Szelepcsényi, *The method of forcing for nondeterministic automata*, Bull. Europ. Assoc. Theor. Comp. Sci., 33 (1987), pp. 96–100, for an independent proof of Theorem 1.

## REFERENCES

- [1] A. BORODIN, S. A. COOK, P. W. DYMOND, W. L. RUZZO, AND M. TOMPA, *Two applications of complementation via inductive counting*, Tech. Report, IBM T. J. Watson Research Center, Hampton, VA, 1987.
- [2] S. R. BUSS, S. A. COOK, P. W. DYMOND, AND L. HAY, *The log space oracle hierarchy collapses*, in preparation.
- [3] S. A. COOK, *A taxonomy of problems with fast parallel algorithms*, Inform. and Control, 64 (1985), pp. 2–22.
- [4] J. HARTMANIS AND H. B. HUNT, III, *The LBA problem*, in Complexity of Computation, R. Karp, ed, SIAM-AMS Proc. 7, 1974, pp. 1–26.
- [5] J. E. HOPCROFT AND J. D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.
- [6] N. IMMERMAN, *Relational queries computable in polynomial time*, Inform. and Control, 68 (1986), pp. 86–104. (A preliminary version of this paper appeared in Proc. 14th Annual ACM Symposium on Theory of Computing, 1982, pp. 147–152.)
- [7] ———, *Languages that capture complexity classes*, SIAM J. Comput. 16 (1987), pp. 760–778. (A preliminary version of this paper appeared in Proc. 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 347–354.)
- [8] ———, *Expressibility as a complexity measure: results and directions*, Second Structure in Complexity Theory Conference, 1987, pp. 194–202.
- [9] S. Y. KURODA, *Classes of languages and linear-bounded automata*, Inform. and Control, 7 (1964), pp. 207–233.
- [10] K. J. LANGE, B. JENNER, AND B. KIRSIG, *The logarithmic hierarchy collapses:  $A\Sigma_2^L = A\Pi_2^L$* , 14th International Colloquium on Automata Languages, and Programming, 1987.
- [11] H. LEWIS AND C. H. PAPADIMITRIOU, *Symmetric space bounded computation*, ICALP (1980). (A revised version appeared in Theoret. Comput. Sci., 19 (1982), pp. 161–187.)
- [12] S. R. MAHANEY, *Sparse complete sets for NP: solution of a conjecture of Berman and Hartmanis*, J. Comput. Systems Sci., 25 (1982), pp. 130–143.
- [13] J. REIF, *Symmetric complementation*, J. Assoc. Comput. Math., 31 (1984), pp. 401–421.
- [14] U. SCHÖNING AND K. W. WAGNER, *Collapsing oracles, census functions, and logarithmically many queries*, Report 140, Mathematics Institute, University of Augsburg, 1987.
- [15] W. J. SAVITCH, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. System Sci., 4 (1970), pp. 177–192.

## VARIETIES OF ITERATION THEORIES\*

STEPHEN L. BLOOM† AND ZOLTAN ESIK‡

**Abstract.** The equational properties of iteration, when combined with composition and pairing, are captured by the notion of “iteration theory,” which was introduced in [*SIAM J. Comput.*, 9 (1980), pp. 26–45; 9 (1980), pp. 525–540]. We believe, although all of our evidence will not be exhibited here, that every iterative construction satisfies at least the properties of iteration theories. In this paper, axiomatizations are given for several varieties of iteration theories which occur naturally in the semantics of programming languages, i.e., those generated by theories of trees [*J. Comput. Sys. Sci.*, 16 (1978), pp. 362–399], theories of sequacious functions [Proc. 1973 Colloquium, Vol. 80, *Studies in Logic*, North Holland, Amsterdam, 1975, pp. 175–230], theories of partial functions, and theories of both sequacious and partial functions with distinguished predicates. We show which additional equations must be added to the axioms for iteration theories [*Comput. Ling. Comput. Lang.*, 14 (1980), pp. 183–207] in order to obtain a set of axioms for these subvarieties. Concrete descriptions of the free theories in each variety are given.

**Key words.** iteration, algebraic theories, varieties

**AMS(MOS) subject classifications.** 08B99, 18C05, 18C10

**Introduction.** Iteration plays a fundamental role in the theory of computation: for example, in the theory of automata, languages, formal power series, the semantics of flowchart algorithms, and in the solution of domain equations. Since iteration almost never occurs in isolation, but in conjunction with other basic operations such as function composition, it makes sense to study this operation in the context of algebraic theories, or clone of operations. The equational properties of iteration, when combined with composition and pairing, are captured by the notion of iteration theory, which was introduced in [BEW]. We believe, although all of our evidence will not be exhibited here, that every iterative construction satisfies at least the properties of iteration theories.

In this paper, axioms are given for several varieties of iteration theories that occur naturally in the semantics of programming languages, i.e., those generated by theories of trees [EBT], theories of sequacious functions [CCE], theories of partial functions, and theories of both sequacious and partial functions with distinguished predicates. Since a set of equational axioms for all iteration theories has already been found [Es80], we show which additional equations must be added to obtain a set of axioms for these subvarieties. If one is convinced that these varieties have some intrinsic interest, as we are, then the simple axioms for these varieties yield some important information as to their essential properties. It should be noted that the framework of iteration theories is purely algebraic; i.e., these theories make no use of order, metric, or other additional structure. Nevertheless, this framework is adequate for the treatment of the semantics of flowchart algorithms.

**I. Preliminaries.** Although we do not assume any acquaintance with iteration theories, we will assume the reader has some familiarity with the calculus of algebraic theories as presented in [CCE], and later in [BEW] and [Es80]. For some background information on iteration theories, see [CCE] or [BEW]. Other papers that make use of algebraic and/or iteration theories include the ADJ group [ADJ], Benson and Guessarian [BG], Courcelle [BC], Nelson [EN], Stefanescu [St] and Troeger [DT].

---

\* Received by the editors August 11, 1985; accepted for publication (in revised form) December 1, 1987.

† Department of Computer Science, Stevens Institute of Technology, Hoboken, New Jersey 07030. The research of this author was partially supported by National Science Foundation grant CCR-8620250.

‡ A. Jozsef University, Bolyai Institute, Szeged, Hungary.

**0. Notation.** We will use  $\Sigma$  to denote a ranked set, i.e., a collection of pairwise disjoint set  $\Sigma_n, n \geq 0$ . The value of a function  $f: X \rightarrow Y$  on an element  $x \in X$  will be written variously as

$$xf, f(x), \langle x, f \rangle, f_x.$$

If  $A$  is a set,  $A^*$  is the set of all finite sequences of elements of  $A$ , including the empty sequence  $\lambda$ .  $A^+$  is  $A^* - \{\lambda\}$ . We let  $A^\infty$  denote the set of all finite and infinite sequences of elements of  $A$ . The set of nonnegative integers is denoted  $N$ , and for each  $n \in N, [n] = \{1, \dots, n\}$ . We will sometimes identify the set  $X$  with the set  $X \times [1]$ .

**1. Iteration theories.** An iteration theory is, briefly, an algebra which satisfies the same identities as the unfoldings of finite flowchart schemes (see [BE] for an analysis of the equational properties of flowchart schemes). Equational axioms for these algebras were given in [Es80]. As algebras, iteration theories are many-sorted, where the set of sorts is the set  $N \times N$  of ordered pairs of nonnegative integers. The algebras have several constants and three operations which correspond to three fundamental control structures in flowchart algorithms: composition (or sequencing), tupling (a case statement), and iteration (or looping).

An equivalent and somewhat simpler description of these structures is possible when we use the terminology of elementary category theory. An **algebraic theory**  $T$  [WL], [CCE] is a category whose objects are the nonnegative integers such that for each positive integer  $n$ , there are  $n$  distinguished “injections”

$$i_n : 1 \rightarrow n$$

$i = 1, 2, \dots, n$ , which make  $n$  the coproduct of 1 with itself  $n$ -times. More precisely, for any  $n$  morphisms  $g_i : 1 \rightarrow p, i = 1, \dots, n$ , there is a unique morphism  $g : n \rightarrow p$  such that

$$i_n \cdot g = g_i$$

for each  $i$ . (We will omit the subscript on  $i$  when the target is clear from context.) The uniquely determined morphism  $g$ , written  $\langle g_1, \dots, g_n \rangle$ , is called the **source tupling** of the morphisms  $g_i$ . In particular, for each  $n$  there is a unique morphism

$$0_n : 0 \rightarrow n.$$

We will assume that when  $n = 1$ , the co-product injection  $1_1$  is the identity morphism  $1 \rightarrow 1$ . A **theory morphism**  $\varphi : T \rightarrow T'$  between algebraic theories is a functor which preserves objects and the distinguished co-product injections.

It may be shown that the source tuplings of the distinguished injections correspond to functions  $[n] \rightarrow [p]$ , for all  $n$  and  $p$  in  $N$ . This correspondence is bijective whenever  $1_2 \neq 2_2$ , a condition satisfied whenever there are at least two morphisms  $1 \rightarrow p$ , for some  $p$ . From now on, we identify any such function with the corresponding morphism. These morphisms are called the base morphisms in any algebraic theory. In particular, the identity function  $[n] \rightarrow [n]$  corresponds to the identity morphism  $1_n : n \rightarrow n$  in  $T$ .

For any pair of morphisms  $f : n \rightarrow p, g : m \rightarrow p$  with the same target, there is a unique morphism  $\langle f, g \rangle : n + m \rightarrow p$  such that  $i_{n+m} \cdot \langle f, g \rangle = i_n \cdot f$ , for  $1 \leq i \leq n$ , and  $(n + j)_{n+m} \cdot \langle f, g \rangle = j_m \cdot g$ , for  $1 \leq j \leq m$ . The morphism  $\langle f, g \rangle$  is the **source pairing** of  $f$  and  $g$ .

If  $f_i : n_i \rightarrow p_i, i = 1, 2$ , the morphism

$$f_1 + f_2 : n_1 + n_2 \rightarrow p_1 + p_2,$$

called the **separated sum** of  $f_1$  and  $f_2$ , is defined as the source pairing of  $f_1 \cdot in_1$  and  $f_2 \cdot in_2$ , where  $in_1 : p_1 \rightarrow p_1 + p_2$  and  $in_2 : p_2 \rightarrow p_1 + p_2$  are base morphisms corresponding

to the inclusion and translated inclusion functions. The separated sum operation is associative.

The following useful identities concerning source pairing and sums were proved in [CCE], where we assume that the sources and targets of the indicated morphisms are appropriate:

$$(1.1) \quad (f_1 + g_1) \cdot (f_2 + g_2) = (f_1 \cdot f_2) + (g_1 \cdot g_2),$$

$$(1.2) \quad (f_1 + g_1) \cdot \langle f_2, g_2 \rangle = \langle f_1 \cdot f_2, g_1 \cdot g_2 \rangle,$$

$$(1.3) \quad \langle 0_p, g \rangle = g = \langle g, 0_p \rangle,$$

$$(1.4) \quad \langle f, g \rangle \cdot h = \langle f \cdot h, g \cdot h \rangle.$$

In [Es80] it is proved that an **iteration theory** may be defined as an algebraic theory equipped with an operation  $\dagger$ , which takes a morphism  $f: n \rightarrow p + n$  to a morphism  $f^\dagger: n \rightarrow p$ , and which satisfies the identities (1.5)–(1.8) below:

$$(1.5) \quad \textit{Left zero identity: } (0_n + f)^\dagger = 0_n + f^\dagger, \text{ where } f: m \rightarrow p + m.$$

$$(1.6) \quad \textit{Right zero identity: } (f + 0_m)^\dagger = f, \text{ where } f: m \rightarrow p.$$

$$(1.7) \quad \textit{Pairing identity: } \langle f, g \rangle^\dagger = \langle h^\dagger, g^\dagger \cdot \langle 1_p, h^\dagger \rangle \rangle, \text{ where } h = f \cdot \langle 1_{p+n}, g^\dagger \rangle, f: n \rightarrow p + n + m \text{ and } g: m \rightarrow p + n + m.$$

$$(1.8) \quad \textit{Commutative identity:}$$

$$\langle 1 \cdot \eta \cdot f \cdot (1_p + \eta_1), \dots, m \cdot \eta \cdot f \cdot (1_p + \eta_m) \rangle^\dagger = \eta \cdot (f \cdot (1_p + \eta))^\dagger,$$

where  $f: n \rightarrow p + m$ ,  $\eta: m \rightarrow n$  is a surjective base morphism, and  $\eta_i: m \rightarrow m$  are base with  $\eta_i \cdot \eta = \eta$ .

In any algebraic theory satisfying (1.5)–(1.7), the equation (1.8) is implied by the following implication:

$$(1.9) \quad \textit{Functorial dagger: } \text{ if } g: n \rightarrow p + n \text{ and } \eta: m \rightarrow n \text{ is a surjective base morphism such that } \eta \cdot g = f \cdot (1_p + \eta), \text{ for } f: m \rightarrow p + m, \text{ then } \eta \cdot g^\dagger = f^\dagger.$$

It is shown in [Es83] that the fixed-point identity

$$(1.10) \quad f^\dagger = f \cdot \langle 1_p, f^\dagger \rangle, f: n \rightarrow p + n$$

is a consequence of the other conditions.

An **iteration theory morphism**  $\varphi: I \rightarrow I'$  between iteration theories is a theory morphism which preserves dagger; i.e.,  $(f^\dagger)\varphi = (f\varphi)^\dagger$ , for all  $f: n \rightarrow p + n$  in  $I$ .

**2. Theories as algebras.** If we so wish, we might define an algebraic theory as an  $N \times N$  sorted algebra

$$T = \{T(n, p): n, p \geq 0\},$$

having, for each  $n \geq 1$ ,  $n$  distinguished constants  $i_n \in T(1, n)$ , for  $i \in [n]$ , and the following operations:

$$\textit{composition: } T(n, p) \times T(p, q) \rightarrow T(n, q),$$

$$f, g \mapsto f \cdot g;$$

$$\textit{source tupling: } T(1, p)^n \rightarrow T(n, p),$$

$$f_1, \dots, f_n \mapsto \langle f_1, \dots, f_n \rangle$$

which satisfy the following identities:

- (2.1)  $f \cdot (g \cdot h) = (f \cdot g) \cdot h$  for all  $f \in T(m, n)$ ,  $g \in T(n, p)$ ,  $h \in T(p, q)$ ,
- (2.2)  $\langle 1_n, \dots, n_n \rangle \cdot f = f$ ,  $g \cdot \langle 1_n, \dots, n_n \rangle = g$  for all  $f \in T(n, p)$ ,  $g \in T(m, n)$ ,
- (2.3)  $i_n \cdot \langle f_1, \dots, f_n \rangle = f_i$ ,  $i \in [n]$  for all  $f_j \in T(1, p)$ ,
- (2.4)  $\langle 1_n \cdot f, \dots, n_n \cdot f \rangle = f$  for all  $f \in T(n, p)$ ,
- (2.5)  $\langle 1_1 \rangle = 1_1$ .

The first two identities ensure that  $T$  is a category with  $T(n, p)$  as the set of morphisms  $n \rightarrow p$ , and with  $\langle 1_n, \dots, n_n \rangle: n \rightarrow n$  as the identity morphism; the next two make the distinguished morphisms co-product injections. Equation (2.5) identifies the identity morphism  $1 \rightarrow 1$  with the co-product injection  $1 \rightarrow 1$ . It should be noted that (2.1)–(2.5) are not independent. As an algebra, an iteration theory is an  $N \times N$  sorted algebra equipped with an additional family of operations

$$\begin{aligned} \dagger &: (n, p + n) \rightarrow T(n, p), \\ f &\mapsto f^\dagger \end{aligned}$$

which, in addition to the theory identities (2.1)–(2.5), also satisfy the dagger identities (1.5)–(1.8) above. An iteration theory morphism is just a homomorphism between the corresponding algebras. An iteration theory morphism is determined by its values on **scalar morphisms**, i.e., morphisms with source 1. An iteration theory morphism is injective if and only if it is injective on scalar morphisms.

We will feel free to treat iteration theories as either categories or many-sorted algebras, whichever seems more appropriate.

**3. Some special base morphisms.** In any theory, we denote the identity morphism  $s \rightarrow s$  as  $1_s$ . The base morphism  $1 \rightarrow p$  with value  $i$  is denoted  $i_p$ . This notation can be ambiguous when  $i = 1$ , but the context will prevent confusion. For positive integers  $k$  and  $s$ , we denote by  $\Psi_i^{k,s}: s \rightarrow ks$  the morphism

$$(3.1) \quad \Psi_i^{k,s} = 0_{(i-1)s} + 1_s + 0_{(k-i)s}: s \rightarrow ks,$$

for  $i = 1, \dots, k$ . Here,  $1_s$  must be the identity  $s \rightarrow s$  due to the fact that the source of the morphism  $\Psi_i^{k,s}$  is  $s$ . For example, when  $i = 2$ ,  $\Psi_i^{k,s}$  corresponds to the function mapping  $i \in [s]$  to  $s + i$ . For use in § V, note that for any morphism  $\Delta: 1 \rightarrow s$ , the  $k$ -fold sum

$$\Delta + \Delta + \dots + \Delta: k \rightarrow ks$$

may be written as the tuple

$$\langle \Delta \cdot \Psi_1^{k,s}, \dots, \Delta \cdot \Psi_k^{k,s} \rangle.$$

Note also that the base identity  $s^2 \rightarrow s^2$  is the tuple

$$\langle \Psi_1^{s,s}, \dots, \Psi_s^{s,s} \rangle: s^2 \rightarrow s^2.$$

**4. Trees.** Let  $\Sigma$  be a ranked set. We let  $X_p = \{x_1, \dots, x_p\}$ .

**DEFINITION 4.1.** A  $\Sigma$ -tree  $t: 1 \rightarrow p$  is a partial function

$$t: N^* \rightarrow \Sigma \cup X_p$$

such that  $\text{dom}(t)$ , the domain of  $t$ , is a prefix closed set containing the empty word  $\lambda$ . Furthermore, for each word  $w$  in  $N^*$ , we have the following:

$$(4.1.1) \quad \text{If } wt \in \Sigma_n, n > 0, \text{ then } wi \in \text{dom}(t) \text{ if and only if } i \in [n].$$

(4.1.2) If  $wt \in X_p \cup \Sigma_0$ , then for each  $i$ ,  $w_i$  is not in  $\text{dom}(t)$ , i.e.,  $w$  is a leaf of  $t$ . A  $\Sigma$ -tree  $n \rightarrow p$  is an  $n$ -tuple  $\langle t_1, \dots, t_n \rangle$  of  $\Sigma$ -trees  $1 \rightarrow p$ . The words in  $\text{dom}(t)$  are the vertices of  $t$ , and for  $v$  in  $\text{dom}(t)$ ,  $vt$  is the label of the vertex  $v$ .

Thus, there is a unique  $\Sigma$ -tree  $0 \rightarrow p$ , for any  $p$ . The **composite**  $t \cdot g$  of the tree  $t: 1 \rightarrow p$  with the tree  $g = \langle g_1, \dots, g_p \rangle: p \rightarrow q$  is the tree  $t \cdot g$  obtained by attaching a copy of  $g_i$  to each leaf of  $t$  labeled  $x_i$ . More precisely, for each  $w, v$  in  $N^*$ , if  $wt = x_i$  then  $\langle wv, t \cdot g \rangle$ , the value of  $t \cdot g$  on  $wv$ , is  $vg_i$ . The algebraic theory identities force the definition of the composite of  $\langle t_1, \dots, t_n \rangle: n \rightarrow p$  with  $g: p \rightarrow q$  to be

$$\langle t_1 \cdot g, \dots, t_n \cdot g \rangle.$$

The operation of composition is associative. (A rigorous discussion of the algebraic theory of trees can be found in [EBT].)

If  $t: 1 \rightarrow p+1$  is a  $\Sigma$ -tree the **iteration equation for  $t$**  is the equation in the variable  $z: 1 \rightarrow p$

$$z = t \cdot \langle 1_p, z \rangle.$$

For all trees  $t: 1 \rightarrow p$  except the tree with  $\lambda t = x_{p+1}$ , the iteration equation for  $t$  has a unique solution, denoted  $t^\dagger$ . In the case that  $\lambda t = x_{p+1}$ , if we define  $t^\dagger$  to be the tree

$$\Phi \cdot 0_p,$$

where  $\Phi: 1 \rightarrow 0$ , is any fixed tree, then the operation of **iteration**  $t \mapsto t^\dagger$  is a total operation on all scalar trees. The operation is extended to all vector trees  $t: n \rightarrow p+n$  using the pairing identity (1.7). Note that  $1_1^\dagger = \Phi$ . (See [BEW] for an analysis of how  $^\dagger$  can be defined in other theories when the iteration equation does not have a unique solution.)

**DEFINITION 4.2.** If  $t$  is a  $\Sigma$ -tree  $1 \rightarrow p$ , and  $w$  is a word in  $\text{dom}(t)$ , then  ${}_w t$  is the subtree of  $t$  at  $w$ , defined by the equation

$$\langle u, {}_w t \rangle = (wu)t,$$

for all  $u$  in  $N^*$ .

**DEFINITION 4.3.** For any  $\Sigma$ -tree  $\Phi: 1 \rightarrow 0$ ,  $(\Sigma\text{TR}, \Phi)$  is the algebraic theory of *all*  $\Sigma$ -trees. In this theory, the operation  $^\dagger$  is defined as above, so that  $1_1^\dagger = \Phi$ . We let  $\Sigma_\perp$  denote the ranked set obtained from  $\Sigma$  by the addition of a new letter  $\perp$  to  $\Sigma_0$ .  $(\Sigma_\perp\text{tr}, \perp)$  is the subtheory of  $(\Sigma_\perp\text{TR}, \perp)$  consisting of all  $\Sigma_\perp$ -trees  $t$  having only finitely many subtrees  ${}_w t$ ,  $w \in N^*$ . The trees in  $\Sigma_\perp\text{tr}$  are the “unfoldings” of finite flowchart schemes.

It is shown in [BEW] that both theories  $(\Sigma\text{TR}, \Phi)$  and  $(\Sigma_\perp\text{tr}, \perp)$  are iteration theories. Moreover,  $(\Sigma_\perp\text{tr}, \perp)$  is the free iteration theory freely generated by  $\Sigma$  (see [Es80], [BEW]). *From now on we will write “ $\Sigma\text{tr}$ ” instead of “ $(\Sigma_\perp\text{tr}, \perp)$ .”* The inclusion  $\Sigma \rightarrow \Sigma\text{tr}$  will usually be denoted  $\eta$ . In the sequel, we will identify any letter  $\sigma$  in  $\Sigma$  with the corresponding tree. Note that in  $\Sigma\text{tr}$  the distinguished morphism  $i_n: 1 \rightarrow n$  is the tree  $t$  with  $\lambda t = x_i$ . This tree will henceforth be denoted  $x_i$ .

**FACT 4.4.** It is proved in [Es80] that an algebraic theory  $T$  (equipped with a dagger operation) is an iteration theory if and only if  $T$  satisfies all of the identities valid in  $\Sigma\text{tr}$ , for each  $\Sigma$ .

**5. Sequacious functions.** Theories of sequacious functions were introduced in Elgot [CCE]. These functions model the sequence of states produced by the atomic actions occurring during the operation of a machine which follows a sequential flowchart algorithm. It may be easier to follow the formal Definition 5.1 if the reader interprets the set  $A$  in Definition 5.1 as the set of all possible states of the memory,

and interprets  $f: n \rightarrow p$  as a machine operating on  $A$  which takes inputs on any one of  $n$  input lines. If the computation is successful, a finite number of states will have been produced and the final state will determine one of  $p$  output lines. When the computation is unsuccessful (either nonterminating or abortive), a finite or infinite number of states will occur during the run of  $f$ , and no output line will be determined.

DEFINITION 5.1. Let  $A$  be any set. A **sequacious function**  $f: n \rightarrow p$  on  $A$  is a function

$$f: A^+ \times [n] \cup A^\infty \rightarrow A^+ \times [p] \cup A^\infty$$

such that the following conditions hold:

(5.1.1)  $vf = v$ , any  $v \in A^\infty$ .

(5.1.2) If  $(a, j)f = (v, i) \in A^+ \times [p]$ ,  $a \in A$ , then for all  $u$  in  $A^*$ ,  $(ua, j)f = (uv, i)$ ; similarly, if  $(a, j)f = w \in A^\infty$ , then  $(ua, j)f = uw$ , all  $u$  in  $A^*$ .

(5.1.3) If  $(a, j)f = (v, i) \in A^+ \times [p]$ , then  $v = av'$ , for some  $v'$  in  $A^*$ ; similarly, if  $(a, j)f = w \in A^\infty$ , then  $w = aw'$ , for some  $w' \in A^\infty$ .

The properties (5.1.1)–(5.1.3) imply the following.

**5.2.** A sequacious function is uniquely determined by its values on arguments of the form  $(a, i) \in A \times [n]$ .

If  $u$  and  $v$  are sequences in  $A^\infty$ , we sometimes write  $u^{\wedge}v$  for the sequence obtained by concatenating  $u$  and  $v$ ;  $u^{\wedge}v = u$  if  $u$  is infinite.

The **composite**  $f \cdot g$  of the sequacious functions  $f: n \rightarrow p$  and  $g: p \rightarrow q$  is obtained by standard function composition. The **source tupling** of sequacious functions  $f_i: 1 \rightarrow p$ ,  $i \in [n]$ , is the sequacious function  $n \rightarrow p$  whose value on  $(a, i)$  is the value of  $f_i$  on  $(a, 1)$ . If  $i_n: 1 \rightarrow n$  is the sequacious function determined by the requirement

$$(a, 1)i_n = (a, i),$$

it is easy to check that with the functions  $i_n$  as co-product injections,  $i \in [n]$ , the sequacious functions on  $A$  form an algebraic theory. We now define the operation of iteration in this theory.

Let  $f: n \rightarrow p + n$  be a sequacious function on  $A$ . We will define  $f^\dagger: n \rightarrow p$  by making use of some fixed element  $\perp$  in  $A^\infty$ . Given  $(a, i)$  in  $A \times [n]$ , we consider the sequence

$$(u_0, i_0), (u_1, i_1), \dots,$$

where  $(a, i) = (u_0, i_0)$ , and where, for all  $j$ ,  $(u_{j+1}, i_{j+1})$  is defined if

$$(u_j, i_j)f = (u_{j+1}, p + i_{j+1}).$$

Either  $(u_j, i_j)$  is defined for all  $j \geq 0$ , or not. If not, let  $j$  be least such that

$$(u_j, i_j)f \in A^+ \times [p] \cup A^\infty.$$

In this case we **define**

$$(a, i)f^\dagger = (u_j, i_j)f.$$

In the case where the sequence  $(u_j, i_j)$  is infinite, the sequence of words  $(u_j)$  is increasing, by (5.1.3), and has a limit, say,  $u$ . The word  $u$  is finite if and only if the sequence  $(u_j)$  is eventually constant. In this second case we **define**

$$(a, i)f^\dagger = u^{\wedge}\perp.$$

Now  $f^\dagger$  has been defined for all arguments in  $A \times [n]$ , and its values elsewhere are determined by the requirements (5.1.1)–(5.1.3). Note that  $1_1^\dagger$  is the sequacious function



taking  $(a, 1)$  to  $a^\wedge \perp$ . The resulting algebraic theory of sequacious functions is denoted  $(\text{Seq}(A), \perp)$ .

**THEOREM 5.3.** *The theory  $(\text{Seq}(A), \perp)$  is an iteration theory.*

It can be proved that, aside from satisfying the dagger identities (1.5)–(1.7), the sequacious functions satisfy the functorial dagger implication, and hence the commutative identity (1.8).

**6. Partial functions.** Given a nonempty set  $A$ , the theory  $\text{Pfn}(A)$  is defined as follows: a morphism  $f: n \rightarrow p$  is a partial function

$$f: A \times [n] \rightarrow A \times [p].$$

The distinguished morphism

$$i_n: A \times [1] \rightarrow A \times [n]$$

is the total function taking  $(a, 1)$  to  $(a, i)$ . Composition is standard function composition and the source tupling of the partial functions  $f_i: 1 \rightarrow n$  is the function  $f: n \rightarrow p$  defined by

$$(a, i)f = (a, 1)f_i.$$

It is easy to see that  $\text{Pfn}(A)$  is an algebraic theory. Moreover, in  $\text{Pfn}(A)$ , there is a unique morphism  $\perp: 1 \rightarrow 0$ , the function which is always undefined. We turn  $\text{Pfn}(A)$  into an iteration theory by defining  $f^\dagger$  as follows: given  $f: n \rightarrow p+n$  and an element  $(a, i)$  in  $A \times [n]$ , consider the sequence

$$(u_0, i_0), (u_1, i_1), \dots$$

defined by  $(u_0, i_0) = (a, i)$ , and  $(u_{j+1}, p+i_{j+1}) = (u_j, i_j)f$ . Now, either the sequence is defined for all  $j$ , or there is a least  $j$  such that either  $f$  is undefined on  $(u_j, i_j)$  or  $(u_j, i_j)f \in A \times [p]$ . In the latter case, we **define**  $(a, i)f^\dagger = (u_j, i_j)f$ . In the former, the value of  $f^\dagger$  on  $(a, i)$  is undefined.

**THEOREM 6.1.**  *$\text{Pfn}(A)$  is an iteration theory.*

One proof of this theorem may be found in [BEW].

**7. Congruences.** A congruence  $\approx$  on an algebraic theory  $T$  is a family of equivalence relations, one on each set  $T(n, p)$ ,  $n, p \geq 0$ , which is compatible with the theory operations of composition and source tupling; i.e., if  $f_i \approx g_i$ , then  $f_1 \cdot g_1 \approx f_2 \cdot g_2$ , and  $\langle f_1, \dots, f_n \rangle \approx \langle g_1, \dots, g_n \rangle$ . An **iteration theory congruence** is a theory congruence compatible with the dagger operation: i.e.,  $f^\dagger \approx g^\dagger$  whenever  $f \approx g$ . If  $\varphi: T \rightarrow T'$  is a morphism between iteration theories, then  $\ker(\varphi)$  is an iteration theory congruence on  $T$ , where  $f \approx g$  ( $\ker \varphi$ ) if  $f\varphi = g\varphi$ . Conversely, if  $\approx$  is an iteration theory congruence on  $T$ , the collection  $T/\approx$  of  $\approx$ -equivalence classes forms an iteration theory in the usual way, and the canonical map  $\kappa: T \rightarrow T/\approx$  is an iteration theory morphism, where  $f\kappa$  is the  $\approx$ -equivalence class of  $f$ . Congruences, as well as iteration theory morphisms, are determined by their scalar components.

**II. The method.** An iteration theory identity is an equation  $f = f'$  between terms built from a ranked set, say  $\Omega$ , with  $\Omega_k$  infinite for each  $k \geq 0$ , from constants  $i_n, i \in [n]$ ,  $n > 0$ , using the operation symbols for composition, source tupling, and iteration. We let  $\Omega\text{term}$  denote the  $N \times N$ -sorted algebra of terms. If  $T$  is an iteration theory, a rank-preserving function  $h: \Omega\text{term} \rightarrow T$  is a homomorphism if  $h$  preserves the constants and the operations. Thus, for example,  $(f \cdot g)h = fh \cdot gh$ . An identity  $f = f'$  is valid in  $T$  if  $fh = f'h$  for every homomorphism  $h: \Omega\text{term} \rightarrow T$ .

If  $K$  is a class of iteration theories, we let  $\text{id}(K)$  be the set of all identities true in  $T$  for each  $T$  in  $K$ . The variety generated by a class  $K$ ,  $V(K)$ , is the class of all iteration theories  $T$  such that each identity in  $\text{id}(K)$  is true in  $T$ .

In this paper, we are concerned with the problem of showing that for each of several classes  $K$  of iteration theories, some subset  $Ax$  of identities has the property that  $T \in V(K)$  if and only if each identity in  $Ax$  is valid in  $T$ . In other words, we want to show that  $Ax$  axiomatizes the variety  $V(K)$ .

Our method to prove that a set  $Ax$  (along with the axioms for iteration theories) correctly axiomatizes a variety  $V$  will be the same in each case. We will prove two facts. For any ranked set  $\Sigma$ , let  $\approx$  denote the least iteration theory congruence on  $\Sigma\text{tr}$  such that  $fh \approx f'h$  for each equation  $f = f'$  in  $Ax$  and each homomorphism  $h: \Omega\text{term} \rightarrow \Sigma\text{tr}$ . First we prove that if  $\varphi: \Sigma\text{tr} \rightarrow A$  is any iteration theory morphism from  $\Sigma\text{tr}$  into a theory  $A$  in  $V$ , then  $\approx$  is contained in  $\ker(\varphi)$ , the iteration theory congruence induced by  $\varphi$  on  $\Sigma\text{tr}$ . Second, we prove that the quotient theory  $\Sigma\text{tr}/\approx$  is itself in  $V$ . The Basic Theorem below shows that these facts are sufficient to show that  $\Sigma\text{tr}/\approx$  is freely generated by  $\Sigma$  in  $V$ , and thus the axioms are complete. The difficult part is proving that  $\Sigma\text{tr}/\approx$  is itself in the variety. We will do this in two steps: by finding an explicit description for representatives of the  $\approx$ -congruence classes, and then using this description to show how to embed the theory in one of the generating theories of the variety. Thus, *in each variety considered here, the free structures are subtheories of the generating theories.*

**BASIC THEOREM.** *Suppose that for a given variety  $V$  and set of equational axioms  $Ax$ , the following conditions hold:*

(1) *If  $A \in V$ , then for any iteration theory morphism  $\varphi: \Sigma\text{tr} \rightarrow A$ ,  $t\varphi = t'\varphi$  whenever  $t \approx t'$ .*

(2) *The quotient theory  $\Sigma\text{tr}/\approx$  is itself in  $V$ , where  $\approx$  is defined as above. Then  $\Sigma\text{tr}/\approx$  is freely generated in  $V$  by the composite  $\eta \cdot \kappa_{\approx}$ , where  $\eta: \Sigma \rightarrow \Sigma\text{tr}$  is the inclusion and  $\kappa_{\approx}: \Sigma\text{tr} \rightarrow \Sigma\text{tr}/\approx$  is the canonical map taking a tree  $t$  to its equivalence class. Further, an iteration theory  $T$  is in  $V$  if and only if each axiom is valid in  $T$ .*

*Proof.* Let  $\equiv$  be the intersection of all congruences on  $\Sigma\text{tr}$  of the form  $\ker(\varphi)$ , for  $\varphi: \Sigma\text{tr} \rightarrow A$  an iteration theory morphism whose target is in  $V$ . Then it is easy to see that  $\Sigma\text{tr}/\equiv$  is the free theory in  $V$  freely generated by

$$\eta \cdot \kappa_{\equiv}: \Sigma \rightarrow \Sigma\text{tr} \rightarrow \Sigma\text{tr}/\equiv.$$

By (1), there is a unique iteration theory morphism  $\alpha: \Sigma\text{tr}/\approx \rightarrow \Sigma\text{tr}/\equiv$  such that

$$\begin{array}{ccc} \Sigma\text{tr} & \xrightarrow{\kappa_{\equiv}} & \Sigma\text{tr}/\equiv \\ \kappa_{\approx} \downarrow & \nearrow \alpha & \\ \Sigma\text{tr}/\approx & & \end{array}$$

commutes. By (2), since  $\Sigma\text{tr}/\equiv$  is freely generated by  $\eta \cdot \kappa_{\equiv}: \Sigma \rightarrow \Sigma\text{tr} \rightarrow \Sigma\text{tr}/\equiv$ , there is a unique iteration theory morphism, say  $\beta$ , such that

$$\begin{array}{ccc} \Sigma\text{tr} & \xrightarrow{\kappa_{\equiv}} & \Sigma\text{tr}/\equiv \\ \kappa_{\approx} \downarrow & \nearrow \beta & \\ \Sigma\text{tr}/\approx & & \end{array}$$

commutes. But then both  $\alpha$  and  $\beta$  are necessarily isomorphisms, thus proving the first part of the theorem. The second statement follows from the fact that if each axiom is

valid in  $T$ , then  $T$  is a quotient of a theory of the form  $\Sigma\text{tr}/\approx$ , for a sufficiently large ranked set  $\Sigma$ . Thus  $T$  is in  $V$ , so the theorem is proved.

*Example. Tree varieties.* Let  $V$  denote the variety generated by the iteration theories  $(\Sigma\text{TR}, \perp)$ , for each  $\Sigma$ . We show that  $V$  is axiomatized by the set of axioms for all iteration theories, i.e.,  $Ax$  is empty and  $\approx$  is the identity congruence. Indeed, by our method, there is nothing to do to satisfy (1) above. And, since for any  $\Sigma$ , the theory  $\Sigma\text{tr}$  is a subtheory of  $(\Sigma_{\perp}\text{TR}, \perp)$ ,  $\Sigma\text{tr}$  is in  $V$ . Since the tree theories  $\Sigma\text{tr}$  are free in the class of all iteration theories [BEW],  $V$  is the variety of all iteration theories.

We will use a slight generalization of the above theorem as well. For each ranked set  $\Sigma$ , let  $\Gamma$  be a ranked set disjoint from  $\Sigma$  and let  $\Sigma_{\Gamma}$  denote the ranked set  $\Sigma \cup \Gamma$ . The corresponding set of terms over  $\Omega_{\Gamma}$  has constants for each letter in  $\Gamma$  as well. Suppose that  $K(\Gamma)$  is the class of iteration theories  $A$  such that for each  $\gamma \in \Gamma_n$  there is a distinguished morphism  $\gamma_A: 1 \rightarrow n$  in  $A$ . Morphisms between theories in  $K(\Gamma)$  preserve these distinguished morphisms. Homomorphisms from terms to theories in  $K(\Gamma)$  preserve constants in  $\Gamma$ . It is easy to see that  $\Sigma_{\Gamma}\text{tr}$  is freely generated in  $K(\Gamma)$  by  $\Sigma$ .

**COROLLARY.** *Suppose that for a given variety  $V$  of theories in  $K(\Gamma)$ , and a set of equational axioms  $Ax$ , written using terms in  $\Omega_{\Gamma}$ , the following two statements hold:*

- (3.1) *If  $A \in V$ , then for any iteration theory morphism  $\varphi: \Sigma_{\Gamma}\text{tr} \rightarrow A$ ,  $t\varphi = t'\varphi$  whenever  $t \approx t'$ .*
- (3.2) *The quotient theory  $\Sigma_{\Gamma}\text{tr}/\approx$  is itself in  $V$ , where  $\approx$  is as above. Then  $\Sigma_{\Gamma}\text{tr}/\approx$  is freely generated in  $V$  by the composite  $\eta \cdot \kappa$ , where  $\eta: \Sigma \rightarrow \Sigma_{\Gamma}\text{tr}$  is the inclusion and  $\kappa: \Sigma_{\Gamma}\text{tr} \rightarrow \Sigma_{\Gamma}\text{tr}/\approx$  is the canonical map taking a tree  $t$  to its  $\approx$ -equivalence class.*

**III. Axiomatizing sequacious functions.** In this section, we axiomatize the variety SEQ of iteration theories generated by the theories  $(\text{Seq}(A), \perp)$ , for each set  $A$ . The axioms turn out to be particularly simple: in addition to the axioms for iteration theories, we need add no new axioms. Thus, for each ranked set  $\Sigma$ , the congruence  $\approx$  in the Basic Theorem is the least congruence on  $\Sigma\text{tr}$ , the identity congruence. We need show only that  $\Sigma\text{tr}$  is in SEQ. This fact will follow immediately from the theorem below.

In the proof of Theorem 1 we will make use of sets  $B$  and  $A_0 = B^*$ , where

$$B = \cup (\Sigma_n \times [n]: n \geq 1) \cup \Sigma \cup \{\perp\}.$$

Recall that  $\perp$  is a letter not in  $\Sigma$  used to define  $1_1^{\dagger}$ . The use of  $A_0$  causes a notational problem. The elements of  $A_0$  are finite sequences of elements of  $B$ , and sequacious functions on  $A_0$  involve sequences of elements of  $A_0$ . We will denote concatenation of words on  $B$  by juxtaposition, and will use  $(a_1, \dots, a_k)$  to denote a finite sequence in  $A_0$ .

**THEOREM 1.** *For  $A_0$  as above, there is an injective iteration theory morphism*

$$\varphi_0: \Sigma\text{tr} \rightarrow (\text{Seq}(A_0), \perp).$$

*Proof.* Since this argument is a model for many to come, we will give a detailed proof. Since  $\Sigma\text{tr}$  is freely generated by  $\Sigma$ , to define  $\varphi_0$  we need specify the value of  $\varphi_0$  only on each letter in  $\Sigma$ .

We define  $\sigma\varphi_0$  when  $\sigma \in \Sigma_n$ ,  $n \geq 0$ , as the sequacious function determined by the following conditions (recall I.5.2). Let  $\gamma$  be a word on  $B$ , say  $\gamma = \delta_1\delta_2 \cdots \delta_k$ ,  $k \geq 0$ , where  $\delta_i \in B$ ,  $i = 1, \dots, k$ . Then, if  $k > 0$  and  $\delta_1 = (\sigma, i_1) \in \Sigma_n \times [n]$ ,  $n > 0$ ,

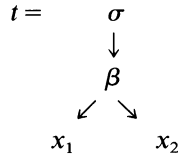
$$(1.1) \quad \langle \gamma, \sigma\varphi_0 \rangle = ((\gamma, \delta_2 \cdots \delta_k), i_1);$$

otherwise, if  $k$  or  $n$  is 0 or  $\delta_1 \neq (\sigma, j)$ ,

$$\langle \gamma, \sigma\varphi_0 \rangle = (\gamma, \sigma).$$

Thus, for all  $\gamma$  in  $A_0$ , all  $\sigma$  in  $\Sigma_n, n \geq 0$ , the value of  $\sigma\varphi_0$  on  $\gamma$  is an element  $A_0^2 \times [n] \cup A_0^2$ . The point of this construction is that the sequacious function  $F_t = t\varphi_0$  determined by the tree  $t$  determines the tree  $t$  itself in the following sense. If  $\gamma$  codes the label of a path in  $t$  to a vertex  $v$ ,  $F_t$  on  $\gamma$  will produce the sequence of words obtained from  $\gamma$  by deleting, from left to right, the letters in  $\gamma$  perhaps followed, at the end, by the element  $vt$ . On words that do not code a path in  $t$ ,  $F_t$  will abort.

*Example.* Let  $\sigma$  be a letter in  $\Sigma_1$  and suppose that  $\alpha$  and  $\beta$  are in  $\Sigma_2$ . Let  $t$  be the finite tree  $\sigma \cdot \beta : 1 \rightarrow 2$ . Then



and

$$\begin{aligned}
 (\sigma, 1)(\beta, 2)F_t &= (((\sigma, 1)(\beta, 2), (\beta, 2), \lambda), 2) \quad \text{in } A_0^3 \times [2], \\
 (\sigma, 1)(\alpha, 1)F_t &= ((\sigma, 1)(\alpha, 1), (\alpha, 1), \beta) \quad \text{in } A_0^3.
 \end{aligned}$$

In order to prove the resulting iteration theory morphism  $\varphi_0$  is injective, we restate some of the preceding definitions in the form of a proposition. In (2.3) below, when  $\gamma \in A_0$  and  $\delta = (\delta', j) \in A_0^+ \times [p]$ , we will understand  $(\gamma, \delta)$  to mean  $((\gamma, \delta'), j) \in A_0^+ \times [p]$ .

**PROPOSITION 2.** *Suppose that  $t : 1 \rightarrow p$  is a tree in  $\Sigma \text{tr}$ . Write  $F_t$  for  $t\varphi_0$ , and let  $\gamma$  be an element of  $A_0$ :*

- (2.1) *If  $t = x_j$ , then  $\gamma F_t = (\gamma, j)$ ;*
- (2.2) *If  $t = \sigma \cdot 0_n$ , for some  $\sigma$  in  $\Sigma_0 \cup \{\perp\}$ , then  $\gamma F_t = (\gamma, \sigma)$ ;*
- (2.3) *If  $t = \sigma \cdot \langle t_1, \dots, t_n \rangle$ , for some  $\sigma$  in  $\Sigma_n, n \geq 1$ , and if  $\gamma = (\sigma, i)^\wedge \gamma'$ , then  $\gamma F_t = (\gamma, \gamma' F_{t_i})$ ;*

*otherwise*

$$\gamma F_t = (\gamma, \sigma).$$

We now show that  $\varphi_0 : \Sigma \text{tr} \rightarrow \text{Seq}(A_0)$  is injective. Assume that  $t$  and  $t'$  are distinct trees  $1 \rightarrow p$  in  $\Sigma \text{tr}$ . We use induction on the length of the shortest word  $v$  such that  $vt \neq vt'$  to prove that  $\gamma F_t \neq \gamma F_{t'}$ , for some element  $\gamma$  in  $A_0$ .

*Case 1.*  $v = \lambda$ . In this case the labels of the roots of the two trees are distinct. Suppose, say, that  $t = \sigma \cdot \langle t_1, \dots, t_k \rangle$  and  $t' = \sigma' \cdot \langle t'_1, \dots, t'_j \rangle$ . One of the letters  $\sigma$  and  $\sigma'$  may be  $\perp$ . Then by (2.2) and (2.3),

$$\lambda F_t = (\lambda, \sigma) \neq (\lambda, \sigma') = \lambda F_{t'}.$$

If  $t' = x_j : 1 \rightarrow p$ , then

$$\lambda F_t = (\lambda, \sigma) \neq (\lambda, j) = \lambda F_{t'}.$$

If  $t$  is also a variable, say  $x_k, \lambda F_t = (\lambda, k)$ . The basis step is complete.

Case 2.  $v = iw$ ,  $t = \sigma \cdot \langle t_1, \dots, t_k \rangle$  and  $t' = \sigma \cdot \langle t'_1, \dots, t'_k \rangle$ , for some  $k \geq j$ . Suppose that  $t_i \neq t'_i$ , where  $1 \leq i \leq k$ . Then by the induction hypothesis there is some element  $\gamma'$  in  $A_0$  with

$$\gamma'F_{t_i} \neq \gamma'F_{t'_i}.$$

If we let  $\gamma = (\sigma, i)\gamma'$ , then

$$\gamma F_t = (\gamma, \gamma'F_{t_i}) \neq (\gamma, \gamma'F_{t'_i}) = \gamma F_{t'}.$$

The proof is complete.

We have completed the proof of the following theorem.

**THEOREM 3.** *For any ranked set  $\Sigma$ , the iteration theory  $\Sigma\text{tr}$  is in the variety SEQ, since it is isomorphic to a subtheory of a theory of the form  $(\text{Seq}(A_0), \perp)$ . Hence, the variety SEQ is the variety of all iteration theories.*

**IV. Partial functions.** In this section two main theorems are proved. First, an axiomatization is given (Theorem 12) for the variety PFN generated by the class of all iteration theories of the form  $\text{Pfn}(A)$ . In the course of the proof, the free iteration theories in this variety are explicitly exhibited. Second, it is proved (Theorem 13) that there is only one way to define the dagger operation  $\dagger$  in any theory  $\text{Pfn}(A)$  so that the resulting structure is an iteration theory. The latter result solves problem 3.9 in Part 2 of [BEW] and is analogous to Theorem 4.3 in [AM].

**PROPOSITION 1.** *There is a unique morphism  $1 \rightarrow 0$  in any iteration theory  $T$  in PFN.*

*Proof.* This condition is evidently an equational one, viz.,

$$(1.1) \quad f = g \quad \text{for all } f, g: 1 \rightarrow 0,$$

which is satisfied in any theory of the form  $\text{Pfn}(A)$  and hence in each theory in PFN. We now set out to show that this condition, together with the axioms for iteration theories, axiomatizes PFN. Let  $\approx$  denote the least iteration theory congruence on  $\Sigma\text{tr}$  such that

$$(2) \quad f \approx g,$$

for all  $f, g: 1 \rightarrow 0$ .

**PROPOSITION 3.** *If  $\varphi: \Sigma\text{tr} \rightarrow \text{Pfn}(A)$  is any iteration theory morphism, then  $\approx$  is contained in  $\ker(\varphi)$ .*

*Proof.* In  $\text{Pfn}(A)$ , a morphism  $f: 1 \rightarrow 0$  is a partial function

$$A \rightarrow \emptyset.$$

Thus  $f$  must be the empty function.

**FACT 4.** It is proved in [BT] that the least theory congruence such that (2) holds is also the least *iteration theory* congruence such that (2) holds.

**DEFINITION 5.** A tree  $t: 1 \rightarrow p$ ,  $p \geq 0$  in  $\Sigma\text{tr}$  is **co-accessible** if no leaf of  $t$  is labeled by a letter in  $\Sigma_0$  and, for each nonleaf vertex  $v$  in  $\text{dom}(t)$ , there is some  $u$  in  $N^*$  with  $(vu)t = x_i$ ; i.e., there is some path from  $v$  to a leaf of  $t$  labeled  $x_i$ , for some  $i \in [p]$ .

Thus a tree  $1 \rightarrow 0$  is coaccessible if and only if the root is a leaf labeled  $\perp$ .

We introduce additional terminology. If  $t: 1 \rightarrow p$  is a tree in  $\Sigma\text{tr}$ , and  $v \in N^*$  is a vertex in  $\text{dom}(t)$ , then we define the “label of the path to  $v$  in  $t$ ,”  $\text{lab}(v)$  for short, as follows.

**DEFINITION 6.** If  $v$  is a vertex in  $\text{dom}(t)$ , then  $\text{lab}(v)$  in  $t$  is the word on the alphabet  $\cup (\Sigma_n \times [n]: n \geq 1)$  defined by induction on the length of  $v$  as follows:

$$(6.1) \quad \text{If } v = \lambda, \text{ then } \text{lab}(v) = \lambda \text{ } (\lambda = \text{empty word}):$$

$$(6.2) \quad \text{If } v = iw, \text{ and } t = \sigma \cdot \langle t_1, \dots, t_n \rangle \text{ then } \text{lab}(v) = (\sigma, i)^\wedge \text{lab}(w), \text{ where } \text{lab}(w) \text{ is the label of } w \text{ in } t_i.$$

PROPOSITION 7. *If  $t$  and  $t'$  are distinct co-accessible trees  $1 \rightarrow p$  in  $\Sigma\text{tr}$ , then there is a leaf vertex  $v$  of  $t$ , say, labeled  $x_i$ , for some  $i \in [p]$ , such that  $\text{lab}(v)$  is not the label of a path to a leaf labeled  $x_i$  in  $t'$ .*

*Proof.* Since  $t$  and  $t'$  are distinct, there is some vertex  $w$  in the domain of both trees such that  $wt \neq wt'$ . We may assume that  $wt$  is not  $\perp$ . If  $w$  is not a leaf of  $t$  labeled  $x_i$  then some extension  $v = wu$  is a leaf vertex of  $t$  labeled  $x_i$ . But then  $\text{lab}(v) = \text{lab}(w) \wedge \text{lab}(u)$  and  $\text{lab}(v)$  is the required path label.

We will need the following terminology in the proof of the next proposition. In any theory, we say a morphism  $f: n \rightarrow p$  **factors through 0** if there is a morphism  $g: n \rightarrow 0$  such that  $f = g \cdot 0_p$ . In the theory  $\Sigma\text{tr}$ , if  $v$  is a vertex of the tree  $t: 1 \rightarrow p$ , we call  $v$  a **0-vertex** if  ${}_v t$  factors through 0 but for no proper prefix  $w$  of  $v$  does  ${}_w t$  factor through 0.

PROPOSITION 8. *For any tree  $t: 1 \rightarrow p, p \geq 1$ , in  $\Sigma\text{tr}$ , there is some co-accessible tree  $t'$  with  $t \approx t'$ .*

*Proof.* Let  $t_1, \dots, t_k$  be all subtrees of  $t$  which are distinct from  $\perp$  and which factor through 0. If  $k = 0$ , then  $t$  itself is co-accessible. Let  $X$  be the set of all 0-vertices  $v$  such that  ${}_v t = t_k$ . Let  $t^\wedge: 1 \rightarrow p+1$  denote the tree obtained from  $t$  by relabeling all vertices  $v$  in  $X$  as  $x_{p+1}$  and deleting the remaining vertices from the subtree  ${}_v t$ . Since  $t^\wedge$  has finitely many subtrees,  $t^\wedge$  is in  $\Sigma\text{tr}$ . Also, by construction,

$$(8.1) \quad t = t^\wedge \cdot \langle 1_p, t_k \rangle.$$

Let  $f = t^\wedge \cdot \langle 1_p, \perp \cdot 0_p \rangle$ . Then

$$(8.2) \quad t \approx f$$

and  $f$  has at least one fewer subtrees distinct from  $\perp$  which factor through 0. In this way, in finitely many steps we obtain a co-accessible tree  $t'$  with  $t' \approx t$ .

In the next argument, we give a construction similar to that used to prove Theorem III.1.

PROPOSITION 9. *There is a set  $A_0$  and a theory morphism  $\varphi_0: \Sigma\text{tr} \rightarrow \text{Pfn}(A_0)$  such that if  $t$  and  $t'$  are distinct co-accessible trees  $1 \rightarrow p$ , then  $t\varphi_0 \neq t'\varphi_0$ .*

*Proof.* Let  $B = \cup (\Sigma_n \times [n]: n \geq 1)$  and let  $A_0 = B^*$ , the set of all words on  $B$ . Again, to define  $\varphi_0$  we need only define the image  $\sigma\varphi_0$  of each letter in  $\Sigma$ . On  $\perp$  and the letters in  $\Sigma_0$  the value of  $\sigma\varphi_0$  is forced to be the empty function. Let  $w = (\delta_1, i_1) \cdots (\delta_k, i_k)$ ,  $k \geq 0$ . If  $\sigma$  is in  $\Sigma_n, n \geq 1$ , then if  $k \geq 1$  and  $\delta_1 = \sigma$ ,  $\sigma\varphi_0$  is defined by

$$(9.1) \quad \langle w, \sigma\varphi_0 \rangle = ((\delta_2, i_2) \cdots (\delta_k, i_k), i_1);$$

otherwise, if  $k = 0$  or  $\delta_1 \neq \sigma$ ,

$$\langle w, \sigma\varphi_0 \rangle \text{ is undefined.}$$

(Thus,  $\langle \lambda, \sigma\varphi_0 \rangle$  is undefined.)

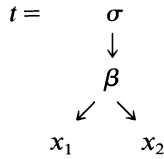
CLAIM.

$$(9.2) \quad \text{If } w \text{ is } \text{lab}(v), \text{ where } v \text{ is a leaf of } t \text{ labeled } x_i, \text{ then } \langle w, t\varphi_0 \rangle = (\lambda, i).$$

$$(9.3) \quad \text{If } w \text{ is not the label of a path in } t \text{ to a leaf labeled by some } x_i, \langle w, t\varphi_0 \rangle \text{ is undefined.}$$

The claim is easily proved by induction on the length of  $w$ . Now the proof is completed by using Propositions 7 and 8.

*Example.* Let  $\sigma$  be a letter in  $\Sigma_1$  and suppose that  $\alpha$  and  $\beta$  are in  $\Sigma_2$ . Let  $t$  be the finite tree  $\sigma \cdot \beta : 1 \rightarrow 2$ . Then, writing  $t\varphi_0$  as  $F_t$ ,



and

$$\begin{aligned}
 (\sigma, 1)(\beta, 2)F_t &= (\lambda, 2), \\
 (\sigma, 1)F_t &= \lambda F_\beta \text{ is undefined.}
 \end{aligned}$$

PROPOSITION 10. For any tree  $t : 1 \rightarrow p$  there is a unique co-accessible tree  $t' : 1 \rightarrow p$  with  $t \approx t'$ .

*Proof.* The proof follows from Propositions 9, 3, and 8.

PROPOSITION 11.  $\Sigma\text{tr}/\approx$  belongs to PFN since  $\Sigma\text{tr}/\approx$  is isomorphic to a sub-iteration theory of  $\text{Pfn}(A_0)$ , where  $A_0$  is the set defined in Proposition 9.

By the Basic Theorem II.1, we obtain the desired result.

THEOREM 12.

(12.1)  $\Sigma\text{tr}/\approx$  is the free theory, freely generated by  $\Sigma$ , in the variety PFN;

(12.2) the axiom (1.1) axiomatizes PFN.

**Uniqueness.** In [BEW], it was asked whether there is more than one way to define the iteration operation  $\dagger$  so that the theory of partial functions  $\text{Pfn}(A)$  is an iteration theory. We devote the remainder of this section to answering the question negatively.

UNIQUENESS THEOREM 13. In each theory  $\text{Pfn}(A)$  there is only one way to define  $\dagger$  on partial functions  $f : n \rightarrow p + n$  in such a way that the operation  $\dagger$  satisfies all valid iteration theory identities.

We may not define  $f^\dagger$  arbitrarily on  $f : n \rightarrow p + n$ , since for example the fixed-point identity (14) must hold.

In view of the pairing identity (see below), we need only consider the case that  $f : 1 \rightarrow p + 1$ .

Let  $f : A \rightarrow A \times [p + 1]$  be a partial function in  $\text{Pfn}(A)$ . Let  $a \in A$ . We consider the following finite or infinite “ $f$ -sequence for  $a$ ”:

$$a = a_0, a_1, a_2, \dots$$

where

$$a_0 f = (a_1, p + 1), a_1 f = (a_2, p + 1), \dots$$

Either for some  $i$ ,  $a_i f$  is undefined (we say “ $f$  is eventually undefined on  $a$ ”) or  $a_i f = (b, j)$ , for some  $j \in [p]$  (we say “ $f$  is eventually successful on  $a$  with value  $(b, j)$ ”), or the sequence is infinite. In the last case, we say “ $f$  cycles on  $a$ .”

Let

$$\begin{aligned}
 S &= \{a \in A : \text{the } f\text{-sequence for } a \text{ is finite}\}, \\
 C &= A - S = \{a \in A : f \text{ cycles on } a\}.
 \end{aligned}$$

It is easy to see that the value of  $f^\dagger$  is determined on the elements of  $S$  by the fixed-point identity

$$(14) \quad f^\dagger = f \cdot \langle 1_p, f^\dagger \rangle.$$

Indeed, if  $a \in S$ , then  $f^\dagger$  is undefined on  $a$  if  $f$  is eventually undefined on  $a$ , and  $af^\dagger = (b, j)$  if  $f$  is eventually successful on  $a$  with value  $(b, j)$ . Thus, the only question unresolved by this particular identity is how  $f^\dagger$  behaves on elements in  $C$ . We wish to show that if  $f$  cycles on  $a$ ,  $f^\dagger$  must be undefined on  $a$ .

Using the following identity:

$$(15) \quad [f \cdot (\beta + 1_1)]^\dagger = f^\dagger \cdot \beta,$$

where  $f: 1 \rightarrow p+1$  and  $\beta: p \rightarrow 1$ , we see that we may also assume that  $p = 1$ , since  $f$  cycles on  $a$  if and only if  $f \cdot (\beta + 1_1)$  cycles on  $a$ .

In the discussion we will use two versions of the **pairing identity**: for  $f: n \rightarrow p + n + m$ ,  $g: m \rightarrow p + n + m$ ,

$$(16) \quad \langle f, g \rangle^\dagger = \langle h^\dagger, g^\dagger \cdot \langle 1_p, h^\dagger \rangle \rangle,$$

where  $h = f \cdot \langle 1_{p+n}, g^\dagger \rangle$ ;

$$(17) \quad \langle f, g \rangle^\dagger = \langle (f \cdot \rho)^\dagger \cdot \langle 1_p, k^\dagger \rangle, k^\dagger \rangle,$$

where  $k = g \cdot \langle 1_p + 0_m, (f \cdot \rho)^\dagger, 0_p + 1_m \rangle$ , and  $\rho: p + n + m \rightarrow p + m + n$  is the base permutation

$$\rho = 1_p + \langle 0_m + 1_n, 1_m + 0_n \rangle.$$

(See [BGR] or [BEW] for a derivation of the pairing lemmas.) Note that (16) is I.(1.7). We shall also use the following fact:

$$(18) \quad (f^2)^\dagger = f^\dagger,$$

for all  $f: n \rightarrow p + n$ , where

$$f^2 = f \cdot \langle 1_p + 0_n, f \rangle.$$

Now let  $f: A \rightarrow A \times [2]$  be a partial function. Define the partial function  $f': A \rightarrow A \times [3]$  as follows:

$$\begin{aligned} xf' &= (y, 3) \quad \text{if } xf = (y, 2) \text{ and } f \text{ cycles or is eventually undefined on } x, \\ &= xf \quad \text{if } f \text{ is eventually successful on } x, \\ &= \text{undefined,} \quad \text{otherwise.} \end{aligned}$$

Note that

$$(19) \quad \text{If } \beta: 3 \rightarrow 2 \text{ is the base morphism } \langle 1_1 + 0_1, 0_1 + 1_1, 0_1 + 1_1 \rangle \text{ then, } f' \cdot \beta = f; \text{ also, if } f' \text{ is eventually successful on } x, \text{ then } f' \text{ is "immediately successful on } x," \text{ i.e., } xf' = (y, j), \text{ for some } j \in [2].$$

Now, by the pairing identity (17) above,

$$(0_1 + 1_1) \cdot \langle 0_2 + 1_1, f' \rangle^\dagger = k^\dagger,$$

where

$$\begin{aligned} k &= f' \cdot \langle 1_1 + 0_1, (0_1 + 1_1 + 0_1)^\dagger, 0_1 + 1_1 \rangle \\ &= f' \cdot \langle 1_1 + 0_1, 0_1 + 1_1, 0_1 + 1_1 \rangle \\ &= f' \cdot \beta = f. \end{aligned}$$

But by (16),

$$(0_1 + 1_1) \cdot \langle 0_2 + 1_1, f' \rangle^\dagger = f'^\dagger \cdot \langle 1_1, h^\dagger \rangle,$$

for some  $h$ . Hence, if  $f$  cycles on  $x$ , so does  $f'$ , and if  $xf'^\dagger$  is defined, so is  $xf'^\dagger$ .



The previous discussion and the identity (15) allow us to assume that  $f: A \rightarrow A \times [2]$  is a partial function such that

- (20) If  $f$  is eventually successful on  $x$ , then  $xf = (y, 1)$  for some  $y$ ; i.e.,  $f$  is immediately successful on  $x$ . Further, there is some  $a_0 \in A$  such that  $f$  cycles on  $a_0$  and  $a_0f^\dagger$  is defined.

Suppose that

(21) 
$$a_0f^\dagger = (b, 1),$$

and let  $c \in A$  be some element such that  $c \neq b$ . Let  $h_c: A \rightarrow A$  be the constant function with value  $(c, 1)$ , and let

$$g = f \cdot (h_c + 1_1),$$

so that

$$g^\dagger = f^\dagger \cdot h_c.$$

Thus,  $a_0g^\dagger = a_1g^\dagger = (c, 1)$ , by the fixed-point identity (14), where  $a_0f = (a_1, 2)$ .

We now define the two partial functions  $f_1$  and  $f_2: 1 \rightarrow 3$  as follows:

$$f_1 = f \cdot (1_1 + 0_1 + 1_1) \quad \text{and} \quad f_2 = g + 0_1.$$

Using version (17) of the pairing identity, we see that

$$(1_1 + 0_1) \cdot \langle f_1, f_2 \rangle^\dagger = f \cdot \langle 1_1, (g^2)^\dagger \rangle,$$

by property (20) of  $f$ . Further,

$$(1_1 + 0_1) \cdot \langle f_1, f_2 \rangle^\dagger = f \cdot \langle 1_1, (g^2)^\dagger \rangle = f \cdot \langle 1_1, g^\dagger \rangle,$$

by (18) above. But using version (16) and property (20),

$$(1_1 + 0_1) \cdot \langle f_1, f_2 \rangle^\dagger = (f^2)^\dagger = f^\dagger.$$

But then,

$$a_0f^\dagger = a_0f \cdot \langle 1_1, g^\dagger \rangle = a_1g^\dagger;$$

and the value of the right-hand side is  $a_1g^\dagger = (c, 1)$ , contradicting (21).

We have thus proved the following: if the  $\dagger$ -operation is defined on  $\text{Pfn}(A)$  so that the resulting theory is an iteration theory, then there is only one way to define it on any given partial function  $f: 1 \rightarrow p + 1$ ; namely, if  $f$  cycles on  $x$ ,  $f^\dagger$  is undefined at  $x$ . Otherwise, the value  $xf^\dagger$  is determined by the fixed-point identity.

When defining the element  $c$  above (after (21)), we assumed that the set  $A$  contains at least two elements. In the case that  $A$  is a singleton, there is a much easier argument. Indeed, if  $f: 1 \rightarrow 2$  cycles on  $a_0$  but  $a_0f^\dagger$  is defined, consider the function

$$g = f \cdot (\perp \cdot 0_1 + 1_1).$$

Then  $g = f$ , since  $a_0f = (a_0, 2)$ , and  $a_0$  is the only element in  $A$ . But

$$g^\dagger = f^\dagger \cdot \perp \cdot 0_1,$$

by (15), so that  $a_0f^\dagger$  is undefined, a contradiction. The Uniqueness Theorem 13 is proved.

**V. Sequacious functions with predicates.**

**A. The theory of one generalized predicate.**

DEFINITION 1. Let  $T$  be the iteration theory  $(\text{Seq}(A), \perp)$  (i.e.,  $1_1^\dagger = \perp$ ). Let  $s$  be an integer greater than 1. A sequacious function  $\Delta : 1 \rightarrow s$  in  $T$  is a (generalized) predicate if, for all  $a \in A$ , there is some  $i \in [s]$  with

$$a\Delta = (a, i).$$

When we wish to emphasize that  $\Delta$  will have a fixed interpretation as a generalized predicate in  $T$  we will write

$$\text{Seq}(A, \perp, \Delta)$$

for this theory. Let  $\text{SEQ}(\Delta)$  be the variety of all iteration theories generated by theories  $\text{Seq}(A, \perp, \Delta)$ , for each  $A$ . In this section we will show that the identities (a.1)–(a.3) below axiomatize  $\text{SEQ}(\Delta)$ .

PROPOSITION 2. Any generalized predicate  $\Delta : 1 \rightarrow s$  satisfies the following identities in  $T$ .

(a.1)  $\Delta \cdot \langle 1_1, \dots, 1_1 \rangle = 1_1,$

(a.2)  $\Delta \cdot (\Delta + \dots + \Delta) = \Delta \cdot \rho,$

where there are  $s$   $\Delta$ 's in the sum and  $\rho : s \rightarrow s^2$  is the base function taking  $i \in [s]$  to  $(i-1)s + i$ ;

(a.3)  $(\Delta \cdot \gamma)^\dagger = \Delta \cdot \gamma \cdot (1_p + \perp),$

where  $\gamma : s \rightarrow p+1$  is any base function.

If  $\Sigma$  is any ranked set, let  $\Sigma_\Delta$  be the ranked set obtained by adding a new symbol  $\Delta$  of rank  $s$  to  $\Sigma$  (as well as the letter  $\perp$  to  $\Sigma_0$ ). Let  $\approx$  be the least iteration theory congruence on  $\Sigma_\Delta\text{tr}$  such that

(a.1')  $\Delta \cdot \langle 1_1, \dots, 1_1 \rangle \approx 1_1,$

(a.2')  $\Delta \cdot (\Delta + \dots + \Delta) \approx \Delta \cdot \rho,$

(a.3')  $(\Delta \cdot \gamma)^\dagger \approx \Delta \cdot \gamma \cdot (1_p + \perp),$

where  $\rho$  and  $\gamma$  are the base functions described in (a.2) and (a.3) above.

FACT 3.1. For each tree  $t : 1 \rightarrow p$ ,

$$t \approx \Delta \cdot \langle t, \dots, t \rangle.$$

FACT 3.2. Suppose that  $t = \Delta \cdot \langle \Delta \cdot u^1, \dots, \Delta \cdot u^s \rangle$ , where each  $u^i : s \rightarrow p$ . Then, writing  $u_j^i$  for the composite of the base  $j_s : 1 \rightarrow s$  with  $u^i : s \rightarrow p$ ,

$$t \approx \Delta \cdot \langle u_1^1, u_2^2, \dots, u_s^s \rangle.$$

We call a scalar tree  $t$  in  $\Sigma_\Delta\text{tr}$  “ $\Delta$ -separated” if for any word  $w$  in  $\text{dom}(t)$  and for each  $i \in [s]$ ,  $(wi)t \neq \Delta$  if  $wt = \Delta$ ;  $t$  is “alternating” if for each  $w$  in  $\text{dom}(t)$  of even length,  $wt = \Delta$ . Note that if  $t$  is both alternating and  $\Delta$ -separated then for  $w$  in  $\text{dom}(t)$ ,  $wt = \Delta$  if and only if the length of  $w$  is even. In particular,  $t$  is  $\Delta$ -rooted. The  $\Delta$ -separated alternating trees will be shown to be representatives of  $\approx$ -equivalence classes.

We state four lemmas, proved later, which are enough to prove the axiomatization theorem.

LEMMA 4. If  $\varphi : \Sigma_\Delta\text{tr} \rightarrow \text{Seq}(A, \perp, \Delta)$  is any iteration theory morphism mapping the letter  $\Delta : 1 \rightarrow s$  to the generalized predicate  $\Delta : 1 \rightarrow s$  in  $\text{Seq}(A, \perp, \Delta)$ , then  $t\varphi = t'\varphi$  whenever  $t \approx t'$ .

LEMMA 5. For every tree  $t:1 \rightarrow p$  in  $\Sigma_{\Delta}\text{tr}$  there is a  $\Delta$ -separated alternating tree  $t':1 \rightarrow p$  with  $t \approx t'$ .

LEMMA 6. There is a particular iteration theory morphism

$$\varphi_0: \Sigma_{\Delta}\text{tr} \rightarrow \text{Seq}(A_0, \perp, \Delta)$$

such that  $t\varphi_0 \neq t'\varphi_0$  if  $t$  and  $t'$  are distinct  $\Delta$ -separated, alternating trees.

LEMMA 7.  $\Sigma_{\Delta}\text{tr}/\approx$  is isomorphic to a subtheory of the theory  $\text{Seq}(A_0, \perp, \Delta)$  of the previous lemma.

Let  $\eta: \Sigma \rightarrow \Sigma_{\Delta}\text{tr}$  be the obvious inclusion function, and let  $\kappa_{\Delta}: \Sigma_{\Delta}\text{tr} \rightarrow \Sigma_{\Delta}\text{tr}/\approx$  be the canonical map taking a tree  $t$  to its  $\approx$ -congruence class.

THEOREM A.  $\Sigma_{\Delta}\text{tr}/\approx$  is freely generated by  $\eta \cdot \kappa_{\Delta}: \Sigma \rightarrow \Sigma_{\Delta}\text{tr}/\approx$  in  $\text{SEQ}(\Delta)$  and the identities (a.1)–(1.3) axiomatize  $\text{SEQ}(\Delta)$ .

Remark. From Lemmas 5–7 it follows that for each scalar tree  $t$  there is a unique  $\Delta$ -separated, alternating tree  $t'$  with  $t \approx t'$ , so that the free theories  $\Sigma_{\Delta}\text{tr}/\approx$  are concretely described by these trees.

We need only prove the lemmas.

*Proof of Lemma 5.* In fact, we show how to obtain a  $\Delta$ -separated alternating tree  $t'$  by an inductive procedure. The difficult part of the procedure ensures that no two  $\Delta$ -labeled vertices are adjacent. It will follow that if the procedure is applied to alternating trees, the result is also alternating. We know that any scalar tree in  $\Sigma_{\Delta}\text{tr}$  may be obtained from the atomic and base trees (i.e., those trees of height at most 1) using the operations of scalar iteration and composition in the form

$$t = t_0 \cdot \langle t_1, \dots, t_k \rangle,$$

where  $t_0:1 \rightarrow k$ , and  $t_i:1 \rightarrow p$ , for  $i = 1, \dots, k$ .

The base and atomic trees are clearly  $\Delta$ -separated. For each base  $\beta:1 \rightarrow p$

$$\beta \approx \Delta \cdot \langle \beta, \dots, \beta \rangle,$$

and the right-hand tree is alternating. If  $\sigma \in \Sigma_n$ , then

$$\sigma \approx \Delta \cdot \langle \sigma, \dots, \sigma \rangle \cdot (\Delta + \dots + \Delta) \cdot \gamma,$$

for an appropriate base  $\gamma$ , and again the right-hand tree is alternating. (There are  $s$   $\sigma$ 's and  $n$   $\Delta$ 's on the right-hand side.) The tree  $\Delta$  itself is  $\Delta$ -separated and alternating.

Now assume that  $t:1 \rightarrow p+1$  is  $\Delta$ -separated and alternating. We will show that there is a  $\Delta$ -separated alternating tree  $t'$ , with  $t' \approx t^{\dagger}$ .

Recall that  $t$  is  $\Delta$ -rooted. For notational simplicity, assume  $\Delta:1 \rightarrow 2$ , so that

$$t = \Delta \cdot \langle t_1, t_2 \rangle.$$

Case 1. One (or both) of  $t_i$ ,  $i = 1, 2$ , is the base  $x_{p+1}$ .

Suppose that just  $t_2$  is  $x_{p+1}$ . In this case, we use the fact that if

$$t = f \cdot \langle g, 0_p + 1_1 \rangle,$$

then  $t^{\dagger} = (f^{\dagger} \cdot g)^{\dagger}$ , and in this case  $f^{\dagger} \cdot g \approx \Delta \cdot \langle t_1, \perp \cdot 0_{p+1} \rangle$ . The details of the other cases are similar and are omitted.

Case 2. Otherwise (see Fig. 1 for an example). Call a leaf  $v$  of

$$t = \Delta \cdot \langle t_1, t_2 \rangle.$$

“ $i$ -bad” ( $i = 1, 2$ ) if  $v$  is labeled  $x_{p+1}$  and  $v$  is the  $i$ th successor of a vertex labeled  $\Delta$ . Let the trees  $\bar{t}, \bar{t}_1, \bar{t}_2:1 \rightarrow p+3$  be obtained from  $t, t_1, t_2$  by relabeling the  $i$ -bad leaves  $x_{p+1+i}$ ,  $i = 1, 2$ . Now define  $h:3 \rightarrow p+3$  by

$$h = \langle \bar{t}, \bar{t}_1, \bar{t}_2 \rangle.$$

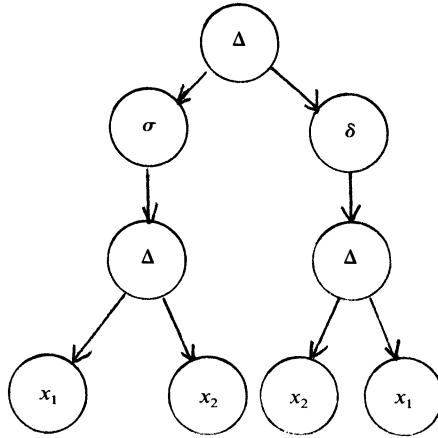


FIG. 1

(Figure 2 indicates the triple  $h$  for the tree in Fig. 1.) Note. If  $\beta : 3 \rightarrow 1$  is the unique base morphism, then  $\bar{t} \cdot \langle 1_p + 0_1, 0_p + \beta \rangle = t$ . A similar statement holds for  $\bar{t}_i, i = 1, 2$ .

CLAIM 1.  $h^\dagger : 3 \rightarrow p$  is  $\Delta$ -separated (i.e., each component of  $h^\dagger$  is  $\Delta$ -separated), since  $\bar{t}_1$  and  $\bar{t}_2$  are not  $\Delta$ -rooted and there is no  $\Delta$ -labeled vertex in  $h$  having a leaf labeled  $x_{p+1}$  as an immediate successor (see Fig. 3 for the same example).

Let  $g : 3 \rightarrow p+3$  be the composite of  $h$  with

$$f = \langle 1_{p+1} + 0_2, 0_{p+1} + \Delta, 0_{p+1} + \Delta \rangle.$$

CLAIM 2.

$$(*) \quad g^\dagger = \langle t^\dagger, t_1 \cdot \langle 1_p, t^\dagger \rangle, t_2 \cdot \langle 1_p, t^\dagger \rangle \rangle.$$

*Proof.* It is easy to see that there is a unique solution to the iteration equation for  $g$ . Let  $k : 3 \rightarrow p$  denote the morphism on the right-hand side of (\*). We will show that

$$g \cdot \langle 1_p, k \rangle = k.$$

Indeed,

$$\begin{aligned} f \cdot \langle 1_p, k \rangle &= \langle 1_p, t^\dagger, \Delta \cdot \langle t_1 \cdot \langle 1_p, t^\dagger \rangle, t_2 \cdot \langle 1_p, t^\dagger \rangle \rangle, \Delta \cdot \langle t_1 \cdot \langle 1_p, t^\dagger \rangle, t_2 \cdot \langle 1_p, t^\dagger \rangle \rangle \rangle \\ &= \langle 1_p, t^\dagger, t^\dagger, t^\dagger \rangle, \end{aligned}$$

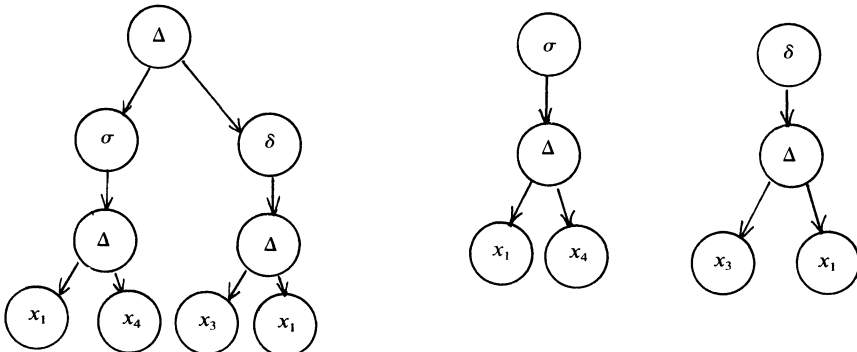


FIG. 2

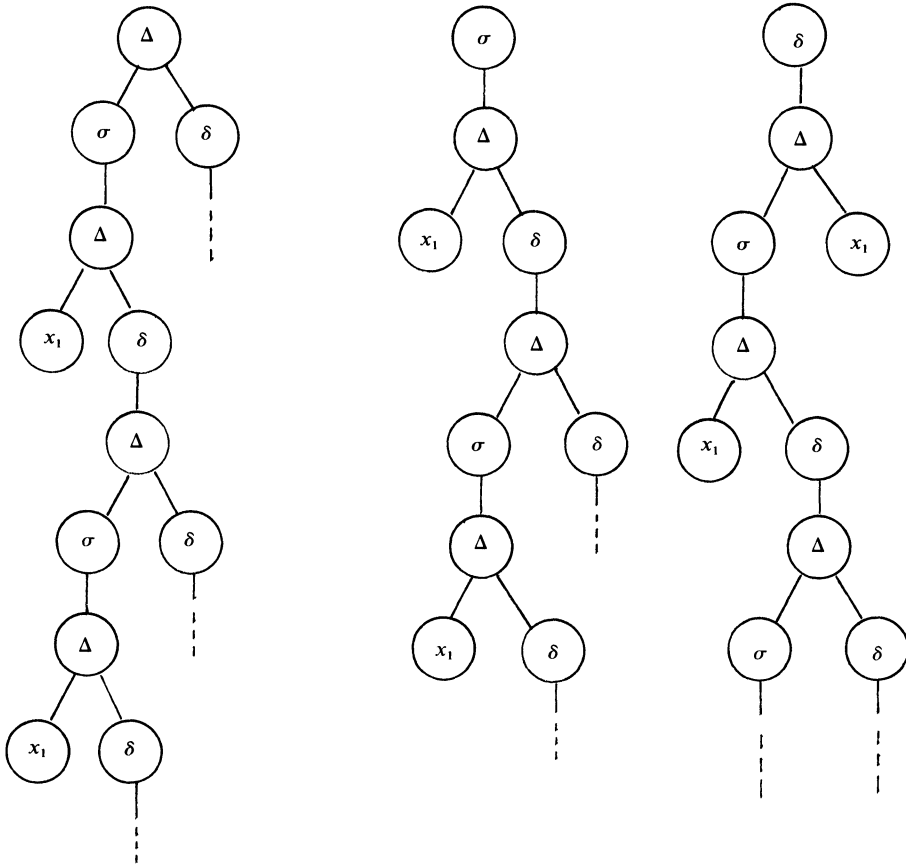


FIG. 3

since  $t = \Delta \cdot \langle t_1, t_2 \rangle$  and  $t \cdot \langle 1_p, t^\dagger \rangle = t^\dagger$ . But then,

$$\begin{aligned} h \cdot f \cdot \langle 1_p, k \rangle &= \langle \bar{t}, \bar{t}_1, \bar{t}_2 \rangle \cdot \langle 1_p, t^\dagger, t^\dagger, t^\dagger \rangle \\ &= \langle \bar{t}, \bar{t}_1, \bar{t}_2 \rangle \cdot \langle 1_p + 0_1, 0_p + \beta \rangle \cdot \langle 1_p, t^\dagger \rangle \\ &= \langle t, t_1, t_2 \rangle \cdot \langle 1_p, t^\dagger \rangle \\ &= \langle t^\dagger, t_1 \cdot \langle 1_p, t^\dagger \rangle, t_2 \cdot \langle 1_p, t^\dagger \rangle \rangle. \end{aligned}$$

Thus  $g^\dagger = k$ , as claimed.

CLAIM 3.  $g \approx h$ .

Indeed,  $g$  is obtained from  $h$  by attaching the tree  $\Delta \cdot \langle x_{p+2}, x_{p+3} \rangle$  to any leaf of the components of  $h$  labeled  $x_{p+1+i}$ ,  $i = 1, 2$ . But such leaves have immediate predecessors labeled  $\Delta$ , by construction. Using Facts 3.1 and 3.2 and the fact that  $h$  has only finitely many  $\Delta$ -rooted subtrees such that some successor of the root is labeled  $x_{p+1+i}$ ,  $i = 1, 2$ , it follows that  $g \approx h$ .

By Claim 3, and the fact that  $\approx$  is an iteration theory congruence, it follows that  $h^\dagger \approx g^\dagger$ . Thus the first component of  $g^\dagger$ , namely  $t^\dagger$ , is congruent to the first component of  $h^\dagger$ , which is a  $\Delta$ -separated tree.

Note. Since  $t$  was already alternating, the first component of  $h^\dagger$  is also alternating.

A similar but much simpler argument shows that if each tree  $t_i, i = 0, \dots, k$  is  $\Delta$ -separated and alternating, and

$$t = t_0 \cdot \langle t_1, \dots, t_k \rangle$$

then we can construct a  $\Delta$ -separated alternating tree  $t'$  congruent to  $t$ .

Lemma 4 is obvious. Lemma 7 follows immediately from Lemmas 6 and 5.

We only sketch the proof of Lemma 6, since it is similar to previous arguments. Let  $Tv$  denote  $[s]^*$ , the collection of all finite sequences of elements in  $[s]$  (which we think of as "truth values"), including the empty sequence  $\lambda$ . Let  $B$  denote the union of the sets  $\Sigma_n \times [n], n \geq 1$ , with the set  $\Sigma \cup \{\perp\}$ . Let  $C$  be  $B^*$ . Finally let  $A_0 = Tv \times C$ . We consider the theory  $\text{Seq}(A_0, \perp, \Delta)$ , in which the generalized predicate  $\Delta$  is defined below. In the following definitions, let

$$\xi = (b_1 b_2 \dots b_k, c_1 \dots c_j) \in A_0.$$

We let  $\tilde{\perp}$  denote the pair  $(\lambda, \perp)$  in  $A_0$ . The predicate  $\Delta$  in  $\text{Seq}(A_0, \tilde{\perp}, \Delta)$  is defined as follows:

$$\begin{aligned} \xi \Delta &= (\xi, b_1) \quad \text{if } k \geq 1, \\ &= (\xi, 1) \quad \text{otherwise.} \end{aligned}$$

For  $\sigma \in \Sigma_n, n \geq 0$ , let  $\sigma\varphi_0$  be the sequacious function determined by the condition

$$\begin{aligned} \xi \sigma\varphi_0 &= (\xi^\wedge(b_2 \dots b_k, c_2 \dots c_j), j_1) \quad \text{if } k > 0, j > 0 \quad \text{and} \quad c_1 = (\sigma, j_1); \\ &= \xi^\wedge(\lambda, \sigma) \quad \text{otherwise.} \end{aligned}$$

It is not difficult to show that if  $t$  and  $t'$  are distinct  $\Delta$ -separated alternating trees then  $t\varphi_0 \neq t'\varphi_0$ .

This concludes the proof of the lemmas and Theorem A is established.

**B. The theory of several binary predicates.** In this section, we show that the variety  $\text{SEQ}(\Pi)$  generated by theories of sequacious functions equipped with a finite set  $\Pi$  of binary predicates is isomorphic to the variety  $\text{SEQ}(\Delta)$ , for a generalized predicate of the appropriate rank.

Let  $\Pi = \{\pi_1, \pi_2, \dots, \pi_r\}$  be a set of new letters and let  $\Sigma_\Pi$  denote the ranked set obtained from  $\Sigma$  by adding the set  $\Pi$  to  $\Sigma_2$ .

For  $i = 1, \dots, r$  we define trees in  $\Sigma_\Pi \text{tr}$  as follows.

DEFINITION 8.

$$\begin{aligned} \Delta_1 &= \pi_1 : 1 \rightarrow 2, \\ \Delta_{j+1} &= \pi_{j+1} \cdot (\Delta_j + \Delta_j) : 1 \rightarrow 2^{j+1}. \end{aligned}$$

For  $k = 1, \dots, r$ , let  $\beta_k : 2^k \rightarrow 1$  denote the unique base morphism and by induction we define the base maps  $\nu_i^k : 2^k \rightarrow 2$ , for  $1 \leq i \leq k \leq r$ , as follows.

DEFINITION 9.

$$\nu_1^1 = 1_1 + 1_1 (= 1_2 : 2 \rightarrow 2),$$

for  $i \leq k$ ,

$$\nu_i^{k+1} = \langle \nu_i^k, \nu_i^k \rangle,$$

when  $i = k + 1$ ,

$$\nu_{k+1}^{k+1} = \beta_k + \beta_k,$$

as an abbreviation, we omit the superscript,

$$(9.a) \quad \nu_k = \nu_k^r \quad \text{for } 1 \leq k \leq r.$$

Let  $\equiv$  denote the least iteration theory congruence on  $\Sigma_{\Pi}$  tr such that

- (c.1)  $\pi_i \cdot \langle 1_1, 1_1 \rangle \equiv 1_1$  for each  $i \in [r]$ ,
- (c.2)  $\pi_i \cdot (\pi_i + \pi_i) \equiv \pi_i \cdot (1_1 + 0_2 + 1_1)$  for each  $i \in [r]$ ,
- (c.3)  $\pi_i \cdot (\pi_j + \pi_j) \equiv \pi_j \cdot (\pi_i + \pi_i) \cdot (1_1 + \rho + 1_1)$  for each  $i, j \in [r]$ ,
- (c.4)  $[\Delta_r \cdot \gamma]^\dagger \equiv \Delta_r \cdot \gamma \cdot (1_p + \perp)$ .

In (c.3),  $\rho: 2 \rightarrow 2$  is the nontrivial base permutation, and in (c.4),  $\gamma: 2^r \rightarrow p + 1$  is any base morphism;  $\Delta_r: 1 \rightarrow 2^r$  was defined in Definition 8. Below, the axiom (c.3) is used only when  $i \neq j$ .

Let  $\text{SEQ}(\Pi)$  denote the variety of iteration theories generated by the theories  $\text{Seq}(A, \perp, \Pi)$  in which each letter  $\pi_i$  in  $\Pi$  is interpreted as a binary predicate. We will prove that  $\Sigma_{\Pi}\text{tr}/\equiv$  is freely generated in  $\text{SEQ}(\Pi)$  by  $\Sigma$ . The proof will be easy once we show that, in fact, if  $s = 2^r$  and  $\Delta: 1 \rightarrow s$  is a new letter, then the theories  $\Sigma_{\Delta}\text{tr}/\approx$  and  $\Sigma_{\Pi}\text{tr}/\equiv$  are isomorphic. (Here  $\approx$  is the congruence on  $\Sigma_{\Delta}\text{tr}$  defined § A.) We need some preliminary facts.

PROPOSITION 10. For each pair  $i, k$  with  $1 \leq i \leq k \leq r$ ,

- (10.1)  $\Delta_k \cdot \beta_k \equiv 1_1$ ,
- (10.2)  $\Delta_k \cdot \nu_i^k \equiv \pi_i$ ,
- (10.3)  $\Delta_r \cdot \nu_i \equiv \pi_i$ ,
- (10.4)  $\nu_i \cdot \langle 1_1, 1_1 \rangle = \beta_r$ .

*Proof.* The proofs of (10.1) and (10.2) are by induction on  $k$ . When  $k = 1$ ,  $\Delta_1 = \pi_1$  and  $\nu_1^1$  is the identity base morphism. Also,  $\pi_1 \cdot \beta_1 \equiv 1_1$ , by (c.1). Assume that (10.1) is true for  $k$ . Then,

$$\begin{aligned} \Delta_{k+1} \cdot \beta_{k+1} &= \pi_{k+1} \cdot (\Delta_k + \Delta_k) \cdot \langle \beta_k, \beta_k \rangle \\ &\equiv \pi_{k+1} \cdot \langle 1_1, 1_1 \rangle \\ &\equiv 1_1. \end{aligned}$$

Now for (10.2), if  $i \leq k$ ,

$$\begin{aligned} \Delta_{k+1} \cdot \nu_i^{k+1} &= \pi_{k+1} \cdot (\Delta_k + \Delta_k) \cdot \langle \nu_i^k, \nu_i^k \rangle \\ &\equiv \pi_{k+1} \cdot \langle \pi_i, \pi_i \rangle \text{ by the induction hypothesis,} \\ &\equiv \pi_i \cdot (\pi_{k+1} + \pi_{k+1}) \cdot \langle 1_1 + 0_1, 1_1 + 0_1, 0_1 + 1_1, 0_1 + 1_1 \rangle \\ &\equiv \pi_i \cdot (1_1 + 1_1) \\ &= \pi_i. \end{aligned}$$

If  $i = k + 1$ , then

$$\begin{aligned} \Delta_{k+1} \cdot \nu_{k+1}^{k+1} &= \pi_{k+1} \cdot (\Delta_k + \Delta_k) \cdot (\beta_k + \beta_k) \\ &\equiv \pi_{k+1} \cdot (1_1 + 1_1) \\ &= \pi_{k+1}. \end{aligned}$$

The proof (10.2) is complete. Clearly, (10.3) is a special case of (10.2). Equation (10.4) is obvious, since there is a unique base morphism  $n \rightarrow 1$ , for  $n \geq 1$ .

We define an iteration theory morphism  $\varphi : \Sigma_{\Pi} \text{tr} \rightarrow \Sigma_{\Delta} \text{tr}$  (where  $\Delta : 1 \rightarrow s, s = 2^r$ ), as the unique morphism extending the following function:

$$\begin{aligned} \sigma &\mapsto \sigma \quad \text{for each } \sigma \text{ in } \Sigma, \\ \pi_i &\mapsto \Delta \cdot \nu_i, \end{aligned}$$

for  $1 \leq i \leq r$ .

PROPOSITION 11. For each pair of trees  $t, t' : n \rightarrow p$  in  $\Sigma_{\Pi} \text{tr}$ ,

$$\text{if } t \equiv t' \text{ then } t\varphi \approx t'\varphi.$$

Thus, if  $\kappa_{\Pi} : \Sigma_{\Pi} \text{tr} \rightarrow \Sigma_{\Pi} \text{tr} / \equiv$  and  $\kappa_{\Delta} : \Sigma_{\Delta} \text{tr} \rightarrow \Sigma_{\Delta} \text{tr} / \approx$  are the canonical morphisms, there is a unique iteration theory morphism  $\varphi^{\wedge} : \Sigma_{\Pi} \text{tr} / \equiv \rightarrow \Sigma_{\Delta} \text{tr} / \approx$  such that the following diagram commutes:

$$\begin{array}{ccc} \Sigma_{\Pi} \text{tr} & \xrightarrow{\varphi} & \Sigma_{\Delta} \text{tr} \\ \kappa_{\Pi} \downarrow & & \downarrow \kappa_{\Delta} \\ \Sigma_{\Pi} \text{tr} / \equiv & \xrightarrow{\varphi^{\wedge}} & \Sigma_{\Delta} \text{tr} / \approx \end{array}$$

*Proof of the Proposition.* It is enough to prove that  $t_i\varphi \approx t'_i\varphi$ , where  $t_i$  is the term on the left-hand side of (c.i) for  $i \in [4]$ , and  $t'_i$  is the right-hand term, since  $\ker(\varphi \cdot \kappa_{\Delta})$  is an iteration theory congruence on  $\Sigma_{\Pi} \text{tr}$ . Thus, we must prove the following facts:

$$(11.1) \quad \Delta \cdot \nu_i \cdot \langle 1_1, 1_1 \rangle \approx 1_1,$$

$$(11.2) \quad \Delta \cdot \nu_i \cdot (\Delta \cdot \nu_i + \Delta \cdot \nu_i) \approx \Delta \cdot \nu_i \cdot (1_1 + 0_2 + 1_1),$$

$$(11.3) \quad \Delta \cdot \nu_i \cdot (\Delta \cdot \nu_j + \Delta \cdot \nu_j) \approx \Delta \cdot \nu_j \cdot (\Delta \cdot \nu_i + \Delta \cdot \nu_i) \cdot (1_1 + \rho + 1_1),$$

$i \neq j, \quad \rho : 2 \rightarrow 2$  the nontrivial permutation.

$$(11.4) \quad \begin{aligned} &[\Delta \cdot \nu_r \cdot (\Delta \cdot \nu_{r-1} + \Delta \cdot \nu_{r-1}) \cdot \dots \cdot (\Delta \cdot \nu_1 + \dots + \Delta \cdot \nu_1) \cdot \gamma]^{\dagger} \\ &\approx [\Delta \cdot \nu_r \cdot (\Delta \cdot \nu_{r-1} + \Delta \cdot \nu_{r-1}) \cdot \dots \cdot (\Delta \cdot \nu_1 + \dots + \Delta \cdot \nu_1) \cdot \gamma] \cdot (1_p + \perp), \end{aligned}$$

where  $\gamma : 2^r \rightarrow p + 1$  is any base morphism.

*Proof of 11.1.*  $\Delta \cdot \nu_i \cdot \langle 1_1, 1_1 \rangle = \Delta \cdot \langle 1_1, \dots, 1_1 \rangle \approx 1_1$ , by (a.1).

*Proof of 11.2.* We use the base morphisms  $\Psi_i^{k,s}$  defined in the preliminaries. First,  $\nu_i \cdot (\Delta \cdot \nu_i + \Delta \cdot \nu_i) = \nu_i \cdot (\Delta + \Delta) \cdot (\nu_i + \nu_i)$ , and for any base morphism  $\nu : s \rightarrow 2$ ,

$$\nu \cdot (\Delta + \Delta) = (\Delta + \dots + \Delta) \cdot (\Psi_{1\nu}^{2,s}, \dots, \Psi_{s\nu}^{2,s}).$$

But applying (a.2),

$$\Delta \cdot \nu_i \cdot (\Delta \cdot \nu_i + \Delta \cdot \nu_i) \approx \Delta \cdot \rho \cdot (\Psi_{1\nu}^{2,s}, \dots, \Psi_{s\nu}^{2,s}) \cdot (\nu_i + \nu_i).$$

Calculating, for  $j \in [s]$ , the value of the base function  $\rho \cdot (\Psi_{1\nu}^{2,s}, \dots, \Psi_{s\nu}^{2,s}) \cdot (\nu_i + \nu_i)$  on  $j$  is four if  $j\nu_i = 2$  and 1 if  $j\nu_i = 1$ . Thus this function is exactly the function  $\nu_i \cdot (1_1 + 0_2 + 1_1)$ .

The proof of (11.3) is similar and is omitted.

*Proof of 11.4.* For  $k = 1, \dots, r$ , we define morphisms  $E_k : 1 \rightarrow 2^k$  as follows:

$$E_1 = \Delta \cdot \nu_1,$$

$$E_{k+1} = \Delta \cdot \nu_{k+1} \cdot (E_k + E_k).$$



Then

$$E_r = \Delta \cdot \nu_r \cdot (\Delta \cdot \nu_{r-1} + \Delta \cdot \nu_{r-1}) \cdot \dots \cdot (\Delta \cdot \nu_1 + \dots + \Delta \cdot \nu_1).$$

We will show that for each  $k \in [r]$ , there is a base morphism  $\sigma_k : 2^r \rightarrow 2^k$  such that

$$E_k \approx \Delta \cdot \sigma_k.$$

Indeed, when  $k = 1$ ,  $\sigma_1 = \nu_1$ , by definition

$$\begin{aligned} E_{k+1} &= \Delta \cdot \nu_{k+1} \cdot (E_k + E_k) \\ &\approx \Delta \cdot \nu_{k+1} \cdot (\Delta \cdot \sigma_k + \Delta \cdot \sigma_k) \\ &= \Delta \cdot \nu_{k+1} \cdot (\Delta + \Delta) \cdot (\sigma_k + \sigma_k) \\ &= \Delta \cdot (\Delta + \dots + \Delta) \cdot \xi \cdot (\sigma_k + \sigma_k), \end{aligned}$$

for a base  $\xi$  (see the preliminaries)

$$\approx \Delta \cdot \rho \cdot \xi \cdot (\sigma_k + \sigma_k),$$

where  $\rho : 2^r \rightarrow 2^{r+r}$  is the base morphism given in (a.2). Let

$$\sigma_{k+1} = \rho \cdot \xi \cdot (\sigma_k + \sigma_k).$$

Now to prove (11.4), we have that

$$\begin{aligned} &[\Delta \cdot \nu_r \cdot (\Delta \cdot \nu_{r-1} + \Delta \cdot \nu_{r-1}) \cdot \dots \cdot (\Delta \cdot \nu_1 + \dots + \Delta \cdot \nu_1) \cdot \gamma]^\dagger \\ &= [E_r \cdot \gamma]^\dagger \approx [\Delta \cdot \sigma_r \cdot \gamma]^\dagger \\ &\approx \Delta \cdot \sigma_r \cdot \gamma \cdot (1_p + \perp) \end{aligned}$$

by (a.3),

$$\begin{aligned} &\approx E_r \cdot \gamma \cdot (1_p + \perp) \\ &= [\Delta \cdot \nu_r \cdot (\Delta \cdot \nu_{r-1} + \Delta \cdot \nu_{r-1}) \cdot \dots \cdot (\Delta \cdot \nu_1 + \dots + \Delta \cdot \nu_1) \cdot \gamma] \cdot (1_p + \perp), \end{aligned}$$

completing the proof.

With somewhat more work, we can show that in fact,  $\sigma_r$  is the identity, so that

$$(11.5) \quad E_r \approx \Delta.$$

Define an iteration theory morphism  $\psi : \Sigma_\Delta \text{tr} \rightarrow \Sigma_{\Pi} \text{tr}$  as the unique morphism extending the function

$$\begin{aligned} &\sigma \mapsto \sigma \quad \text{for each } \sigma \text{ in } \Sigma, \\ &\Delta \mapsto \Delta_r (= \pi_r \cdot (\pi_{r-1} + \pi_{r-1}) \cdot \dots \cdot (\pi_1 + \dots + \pi_1) : 1 \rightarrow 2^r). \end{aligned}$$

PROPOSITION 12. For each pair of trees  $t, t' : n \rightarrow p$  in  $\Sigma_\Delta \text{tr}$ ,

$$\text{if } t \approx t' \text{ then } t\psi \equiv t'\psi.$$

Thus, as in Proposition 11, there is a unique iteration theory morphism

$$\psi^\wedge : \Sigma_\Delta \text{tr} / \approx \rightarrow \Sigma_{\Pi} \text{tr} / \equiv$$

such that

$$\begin{array}{ccc}
 \Sigma_{\Delta}tr & \xrightarrow{\psi} & \Sigma_{\Pi}tr \\
 \kappa_{\Delta} \downarrow & & \downarrow \kappa_{\Pi} \\
 \Sigma_{\Delta}tr/\approx & \xrightarrow{\psi^{\wedge}} & \Sigma_{\Pi}tr/\equiv
 \end{array}$$

commutes.

*Proof.* It is enough to prove that  $t_i\psi \equiv t'_i\psi$ , where  $t_i$  is the term on the left-hand side of (a.i),  $i = 1, 2, 3$ , and  $t'_i$  is the right-hand term, since  $\ker(\psi \cdot \kappa_{\Pi})$  is an iteration theory congruence on  $\Sigma_{\Delta}tr$ . Thus we must show that

(12.1)  $\Delta_r \cdot \langle 1_1, \dots, 1_1 \rangle \equiv 1_1,$

(12.2)  $\Delta_r \cdot (\Delta_r + \dots + \Delta_r) \equiv \Delta_r \cdot \rho,$

where there are  $s = 2^r$   $\Delta$ 's in the sum and  $\rho : s \rightarrow s^2$  is the base function taking  $i \in [s]$  to  $(i-1)s + i$  and

(12.3)  $(\Delta_r \cdot \gamma)^{\dagger} \equiv \Delta_r \cdot \gamma \cdot (1_p + \perp),$

where  $\gamma : s \rightarrow p+1$  is any base function.

*Proof of 12.1.* This is the case that  $k = r$  in (10.1) of Proposition 10. Formula (12.3) is the axiom (c.4). In order to prove (12.2), we may prove the following by induction on  $k$ :

(12.a)  $\Delta_k \cdot (\Delta_k + \dots + \Delta_k) \equiv \Delta_k \cdot \rho_k, \quad k \in [r],$

where  $\rho_k : 2^k \rightarrow 2^{2k}$  is the base morphism defined by

(12.b)  $i\rho_k = (i-1)2^k + i.$

Note that  $\rho_r = \rho$ , so that (12.a) will imply (12.2). In the induction step of the proof, we use (12.c), established by repeated use of (c.3):

(12.c)  $\Delta_k \cdot (\pi_{k+1} + \dots + \pi_{k+1}) \equiv \pi_{k+1} \cdot (\Delta_k + \Delta_k) \cdot \tau_k, \quad k \in [r].$

where  $\tau_k : 2^{k+1} \rightarrow 2^{k+1}$  is the base morphism defined as follows:

$$\begin{aligned}
 i\tau_k &= 2i-1 \quad \text{if } 1 \leq i \leq 2^k \\
 &= 2(i-2^k) \quad \text{if } 2^k < i \leq 2^{k+1}.
 \end{aligned}$$

We omit the remaining details.

PROPOSITION 13. For any tree  $t$  in  $\Sigma_{\Delta}tr$  and  $t'$  in  $\Sigma_{\Pi}tr$ ,

(14)  $t \approx t\psi\varphi,$

(15)  $t' \equiv t'\varphi\psi.$

Thus, both  $\varphi$  and  $\psi$  induce iteration theory isomorphisms between  $\Sigma_{\Delta}tr/\approx$  and  $\Sigma_{\Pi}tr/\equiv$ .

*Proof.* We must prove the following facts:

(15.a)  $\Delta \cdot \nu_r \cdot (\Delta \cdot \nu_{r-1} + \Delta \cdot \nu_{r-1}) \cdot \dots \cdot (\Delta \cdot \nu_1 + \dots + \Delta \cdot \nu_1) \approx \Delta,$

$$(15.b) \quad \pi_i \equiv \Delta_r \cdot \nu_i,$$

for each  $i \in [r]$ . But (11.5) is the former assertion and (10.3) is the latter.

Now it is an easy matter to prove Theorem B.

**THEOREM B.**  $\Sigma_{\Pi}tr/\equiv$  is freely generated in SEQ (II) by

$$\eta \cdot \kappa_{\Pi} : \Sigma \rightarrow \Sigma_{\Pi}tr/\equiv$$

where  $\eta : \Sigma \rightarrow \Sigma_{\Pi}tr$  is the inclusion. Thus, (c.1)–(c.4) axiomatize SEQ (II).

Indeed,  $\Sigma_{\Pi}tr/\equiv$  is in the variety since  $\Sigma_{\Delta}tr/\approx$  is isomorphic to a subtheory of  $Seq(A_0, \Pi, \perp)$ , when  $\Delta$  is interpreted as  $\Delta_r$ . Using Propositions 11 and 12 and Theorem A, we see that any rank-preserving function  $\Sigma \rightarrow Seq(A, \Pi, \perp)$  extends to a unique iteration theory morphism

$$\Sigma_{\Pi}tr/\equiv \rightarrow Seq(A, \Pi, \perp).$$

*Remark 16.* If  $\Pi$  is infinite, it follows from the fact that  $\Pi$  is the direct limit of its finite subsets that the union of the identities (c.1)–(c.4) correctly axiomatizes the variety SEQ (II) generated by theories  $Seq(A, \perp, \Pi)$ .

**VI. The variety PFN ( $\Delta$ ).** Let  $\Delta : A \rightarrow A \times [s]$  be a partial function,  $s \geq 2$ .  $\Delta$  is a predicate on  $A$  if, for each  $a \in A$ , there is some  $i \in [s]$  such that  $a\Delta = (a, i)$ . We let  $Pfn(A, \Delta)$  denote the iteration theory of all partial functions  $A \times [n] \rightarrow A \times [p]$ , in which  $\Delta$  denotes some fixed predicate.  $PFN(\Delta)$  is the variety generated by all theories of the form  $Pfn(A, \Delta)$ . In this section we will axiomatize  $PFN(\Delta)$  and find all free theories, using the Basic Theorem.

Let  $\Sigma$  be a fixed-ranked set, and let  $\Sigma_{\Delta}$  denote the ranked set obtained from  $\Sigma$  by adding a new letter  $\Delta$  to  $\Sigma_s$ . Let  $\approx$  be the least iteration theory congruence on  $\Sigma_{\Delta}tr$  such that 1) and 2) below hold.

1)  $\Delta$ -axioms:

$$(1.1) \quad \Delta \cdot \langle 1_1, \dots, 1_1 \rangle \approx 1_1,$$

$$(1.2) \quad \Delta \cdot (\Delta + \dots + \Delta) \approx \Delta \cdot \tau,$$

$$(1.3) \quad (\Delta \cdot \gamma)^{\dagger} \approx \Delta \cdot \gamma \cdot (1_p + \perp),$$

where  $\tau : s \rightarrow s^2$  is the base morphism taking  $i \in [s]$  to  $s(i-1) + i$ , and where  $\gamma : s \rightarrow p+1$  is any base morphism.

2)  $\perp$ -axiom:

$$(2.1) \quad f \approx \perp, \quad \text{all } f : 1 \rightarrow 0.$$

Note that 2) is equivalent to  $f \approx g$ , for all  $f, g : 1 \rightarrow 0$ . We show that the axioms corresponding to 1) and 2) suffice. The first lemma is obvious.

**LEMMA 3.** For any iteration theory morphism  $\varphi : \Sigma_{\Delta}tr \rightarrow Pfn(A, \Delta)$ , the congruence  $\approx$  is contained in  $\ker(\varphi)$ .

In order to show that  $\Sigma_{\Delta}tr/\approx$  is in the variety  $PFN(\Delta)$ , we introduce a slight variation of the notion of a “co-accessible tree.” Recall the definition of a  $\Delta$ -separated, alternating tree from § V.

**DEFINITION 4.** A  $\Delta$ -separated alternating tree  $t : 1 \rightarrow p$  is **almost co-accessible** if no leaf is labeled by an element of  $\Sigma_0$  and, for each nonleaf vertex  $v$  in  $\text{dom}(t)$  which is not labeled  $\Delta$ , there is an extension  $w = vu$  such that  $wt = x_i$ , for some  $i \in [p]$ .

Thus a  $\Delta$ -separated alternating tree  $1 \rightarrow 0$  is almost co-accessible if each successor of the root (which must be labeled  $\Delta$ ) is a leaf (labeled by  $\perp$ ). Further, if each subtree of the form  $\Delta \cdot \langle \perp \cdot 0_p, \dots, \perp \cdot 0_p \rangle$  is replaced by  $\perp$  in an almost co-accessible tree, the resulting tree is co-accessible. See Fig. 4 for an example of an almost co-accessible tree which is not co-accessible.

LEMMA 5. *For any tree  $t: 1 \rightarrow p$  in  $\Sigma_\Delta \text{tr}$  there is a  $\Delta$ -separated alternating almost co-accessible tree  $t'$  with  $t \approx t'$ .*

*Proof.* Using the  $\Delta$ -axioms, we can find a  $\Delta$ -separated alternating tree congruent to  $t$ , and using the  $\perp$ -axiom and only a slight modification of the argument in § IV, we can get an almost co-accessible tree congruent to  $t$ .

We again make use of the label  $\text{lab}(v)$  of a vertex of a tree in  $\Sigma_\Delta \text{tr}$  (see IV.(2)) to prove the following lemma. To avoid fatigue, in the next lemma we will say a tree  $t: 1 \rightarrow p$  is **admissible** if  $t$  is  $\Delta$ -separated, alternating, and almost co-accessible.

LEMMA 6. *There is a set  $A_0$  and a predicate  $\Delta_0$  on  $A_0$  and an iteration theory morphism  $\varphi: \Sigma_\Delta \text{tr} \rightarrow \text{Pfn}(A_0, \Delta_0)$  such that  $t\varphi \neq t'\varphi$  whenever  $t \neq t'$ , and both  $t$  and  $t'$  are admissible.*

*Proof.* Let  $B = \cup (\Sigma_n \times [n]: n \geq 1)$ , let  $C = \{\Delta\} \times [s]$  and define

$$A_0 = (C \cdot B)^* \cdot C,$$

the set of alternating words which begin and end with a letter of the form  $(\Delta, i)$  and every other letter has the form  $(\sigma, j)$ . Now we define the images  $\Delta\varphi = \Delta_0$  and  $\sigma\varphi$ , for each  $\sigma$  in  $\Sigma_n, n \geq 1$ .

Let  $w = (\Delta, i_1)v$ . The predicate  $\Delta_0$  is given by

$$w\Delta_0 = (w, i_1),$$

the partial function  $\sigma\varphi$  is defined by:

$$\begin{aligned} w\sigma\varphi &= (u, j) \quad \text{if } w = (\Delta, i)(\sigma, j)u \text{ for some } u \in A_0, \\ &= \text{undefined} \quad \text{otherwise.} \end{aligned}$$

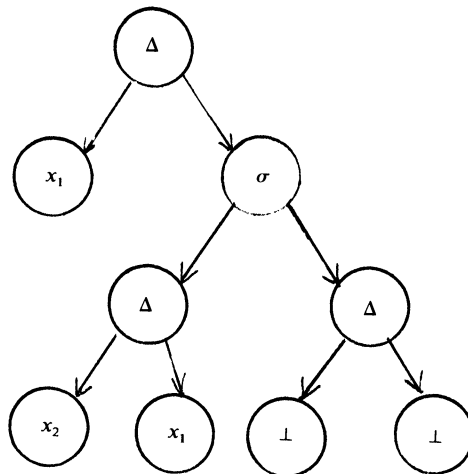


FIG. 4

The point of these definitions is the following:

If  $w = \text{lab}(v)$ , where  $v$  is a leaf labeled  $x_i$  of an admissible tree  $t: 1 \rightarrow p$ , and if  $v$  is the  $j$ th successor of a vertex (labeled  $\Delta$ ) then

$$(6.1) \quad \langle w, t\varphi \rangle = ((\Delta, j), i);$$

otherwise,

$$(6.2) \quad \langle w, t\varphi \rangle \text{ is undefined.}$$

Both of these statements may be proved by induction on the length of  $w$ . But if  $t$  and  $t'$  are distinct admissible trees  $1 \rightarrow p$ , there is a word  $w$  which is a label of a path to a leaf  $v$  of  $t$ , say, labeled  $x_i$  and which is either not the label of a path to a leaf of  $t'$  or is the label of a path to a leaf of  $t'$  labeled  $\perp$  or  $x_j, j \neq i$ . Thus the lemma is proved.

Together, the lemmas yield a proof of the main result of this section.

**THEOREM 7.**  $\Sigma_{\Delta} \text{tr} / \approx$  is freely generated by

$$\eta \cdot \kappa_{=} : \Sigma \rightarrow \Sigma_{\Delta} \text{tr} \rightarrow \Sigma_{\Delta} \text{tr} / \approx$$

in  $\text{PFN}(\Delta)$ . The identities (1) and (2) axiomatize  $\text{PFN}(\Delta)$ .

Just as in § V, we may alternatively consider the variety  $\text{PFN}(\Pi)$  of iteration theories generated by theories of the form  $\text{Pfn}(A, \Pi)$ , where  $\Pi = \{\pi_1, \dots, \pi_r\}$  is a set of binary predicates on  $A$ . Using the same methods as in § V, it is possible to show that this variety is the same as  $\text{PFN}(\Delta)$ , where  $\Delta: 1 \rightarrow 2^r$ .

**VII. Summary of the axioms.** The variety  $\text{SEQ}$  is the variety of all iteration theories and is axiomatized by the identities listed in I.(1.5)–I.(1.8).

$\text{PFN}$  is axiomatized by the one axiom schema:

$$(1) \quad f = g \quad \text{all } f, g: 1 \rightarrow 0.$$

$\text{SEQ}(\Delta)$  is axiomatized by the three schema:

$$(2) \quad \Delta \cdot \langle 1_1, \dots, 1_1 \rangle = 1_1,$$

$$(3) \quad \Delta \cdot (\Delta + \dots + \Delta) = \Delta \cdot \rho,$$

where there are  $s$   $\Delta$ 's in the sum and  $\rho: s \rightarrow s^2$  is the base function taking  $i \in [s]$  to  $(i-1)s + i$ , and

$$(4) \quad (\Delta \cdot \gamma)^\dagger = \Delta \cdot \gamma \cdot (1_p + \perp),$$

where  $\gamma: s \rightarrow p+1$  is any base function.

$\text{SEQ}(\Pi)$  is axiomatized by

$$(5) \quad \pi_i \cdot \langle 1_1, 1_1 \rangle = 1_1 \quad \text{for each } i \in [r],$$

$$(6) \quad \pi_i \cdot (\pi_i + \pi_i) = \pi_i \cdot (1_1 + 0_2 + 1_1) \quad \text{for each } i \in [r],$$

$$(7) \quad \pi_i \cdot (\pi_j + \pi_j) = \pi_j \cdot (\pi_i + \pi_i) \cdot (1_1 + \rho + 1_1) \quad \text{for each } i, j \in [r],$$

$$(8) \quad [\Delta_r \cdot \gamma]^\dagger = \Delta_r \cdot \gamma \cdot (1_p + \perp).$$

In (7),  $\rho: 2 \rightarrow 2$  is the nontrivial base permutation, and in (8),  $\gamma: 2^r \rightarrow p+1$  is any base morphism;  $\Delta_r: 1 \rightarrow 2^r$  was defined in § V, Definition 8.

$\text{PFN}(\Delta)$  and  $\text{PFN}(\Pi)$  are axiomatized by adding the axiom (1) to the axioms for  $\text{SEQ}(\Delta)$  and  $\text{SEQ}(\Pi)$ , respectively.

**Acknowledgments.** We would like to thank the referees for an extremely meticulous job of proofreading and stylistic criticism. The result of their efforts is a better paper.

## REFERENCES

- [ADJ 76] J. WRIGHT, J. THATCHER, E. WAGNER, AND J. GOGUEN, *Rational algebraic theories and fixed-point solutions*, in Proc. 17th Annual IEEE Symposium on Foundations of Computing, Houston, Texas, 1976, pp. 147-158.
- [ADJ 77] J. GOGUEN, J. THATCHER, E. WAGNER, AND J. WRIGHT, *Initial algebra semantics and continuous algebras*, J. Assoc. Comput. Mach., 24 (1977), pp. 68-95.
- [ADJ 78] E. WAGNER, J. THATCHER, AND J. WRIGHT, *Programming languages as mathematical objects*, in Mathematical Foundations of Computer Science 1977, Lectures Notes in Computer Science 64, Springer-Verlag, Berlin, New York, 1978, pp. 84-101.
- [ADJ 83] S. L. BLOOM, J. THATCHER, E. WAGNER, AND J. WRIGHT, *Recursion and iteration in continuous theories*, J. Comput. System Sci., 27 (1983), pp. 148-164.
- [AM] M. A. ARBIB AND E. MANES, *Partially additive categories and flow-diagram semantics*, J. Algebra, 62 (1980), pp. 203-227.
- [BG] D. BENSON AND I. GUESSARIAN, *Iterative and recursive matrix theories*, J. Algebra, 86 (1984), pp. 302-314.
- [BEW] S. L. BLOOM, C. C. ELGOT, AND J. B. WRIGHT, *Solutions of the iteration equation and extensions of the scalar iteration operation*, SIAM J. Comput., 9 (1980), pp. 26-45; *Vector iteration in pointed iterative theories*, SIAM J. Comput., 9 (1980), pp. 525-540.
- [BGR] S. L. BLOOM, S. GINALI, AND J. RUTLEDGE, *Scalar and vector iteration*, J. Comput. System Sci., 14 (1977), pp. 251-256.
- [BT] S. L. BLOOM AND R. TINDELL, *A note on zero congruences*, Acta Cybernet., 8 (1987), pp. 1-4.
- [BE] S. L. BLOOM AND Z. ESIK, *An axiomatization of flowchart schemes*, J. Comput. System Sci., 31 (1985), pp. 375-393.
- [BC] B. COURCELLE, *Fundamental properties of infinite trees*, Theoret. Comput. Sci., 25 (1983), pp. 95-169.
- [CCE] C. C. ELGOT, *Monadic computation and iterative algebraic theories*, in Logic Colloquium '73, Vol. 80, Studies in Logic, North-Holland, Amsterdam, 1975, pp. 175-230; reprinted in [EL].
- [EBT] C. C. ELGOT, S. L. BLOOM, AND R. TINDELL, *On the algebraic structure of rooted trees*, J. Comput. System Sci., 16 (1978), pp. 362-399.
- [CE-SP] C. C. ELGOT, *Structured programming, with and without go-to statements*, IEEE Trans. Software Engng., 2 (1976), pp. 41-53; reprinted in [EL].
- [EL] STEPHEN L. BLOOM, ED., *Calvin C. Elgot, Selected Papers*, Springer-Verlag, New York, Heidelberg, Berlin, 1982.
- [Es80] Z. ESIK, *Identities in iterative and rational algebraic theories*, in Computational Linguistics and Computer Languages XIV, 1980, pp. 183-207.
- [Es83] ———, *Algebras of iteration theories*, J. Comput. System Sci., 27 (1983), pp. 291-303.
- [Es85] ———, *On the weak equivalence of Elgot's flow-chart schemes*, Acta Cybernet., 7 (1985), pp. 147-154.
- [WL] F. W. LAWVERE, *Functorial semantics of algebraic theories*, in Proc. Nat. Acad. Sci. U.S.A., 50 (1963), pp. 869-872.
- [EN] E. NELSON, *Iterative algebras*, Theoret. Comput. Sci., 25 (1983), pp. 67-94.
- [St] GH. STEFANESCU, *On flowchart theories I*, J. Comput. System Sci., 35 (1987), pp. 163-191.
- [DT] D. TROEGER, *Metric iteration theories*, Fund. Inform., 5 (1982), pp. 187-216.

## ON THE COMPLEXITY OF COMPUTING THE VOLUME OF A POLYHEDRON\*

M. E. DYER† AND A. M. FRIEZE‡

**Abstract.** We show that computing the volume of a polyhedron given either as a list of facets or as a list of vertices is as hard as computing the permanent of a matrix.

**Key words.** volume, polyhedra, complexity, #P-complete

**AMS(MOS) subject classification.** 68A20

**1. Introduction.** Recently there has been some interest in establishing the computational complexity of determining the volume of convex bodies in  $\mathbb{R}^n$ . Elekes [2] and Bárány and Füredi [1] have shown that it is even difficult to closely approximate volumes of convex bodies defined by certain types of oracles. These hardness results complement the approximation algorithm of Lovász [4].

Lovász [4] also enquires about the complexity of computing the volume of a rational polytope given either by listing its facets or by listing its vertices. He conjectures that these problems are hard. In this paper we confirm Lovász's conjecture, and show that both problems are as hard as computing the matrix permanent, (see Valiant [6]). We cannot quite describe the problems as #P-complete, since they are not in the class #P as defined by Valiant [6]. However, since we have no wish to define yet another complexity class, we state our results relative to #P.

Let us now be more specific. Consider first Problem 1.

**Problem 1.** Let  $P = P(A, \mathbf{b}) = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$  be a polyhedron.  $A, \mathbf{b}$  have rational entries where  $A = (a_{ij}), i = 1, 2, \dots, m, j = 1, 2, \dots, n$ , and  $\mathbf{b} = (b_i), i = 1, 2, \dots, m$ . We shall use the notation of Schrijver [5] to describe problem size. Thus if  $x = p/q$  is rational where  $p, q > 0$  are relatively prime integers, then

$$\text{size}(x) = 1 + \lceil \log_2(|p| + 1) \rceil + \lceil \log_2(q + 1) \rceil$$

and

$$\text{size}(A, \mathbf{b}) = m(n + 1) + \sum_{i=1}^m \text{size}(b_i) + \sum_{i=1}^m \sum_{j=1}^n \text{size}(a_{ij}).$$

$L = \text{size}(A, \mathbf{b})$  will be used as our measure of problem size. Since we can (by linear programming) determine in polynomial time whether or not  $\text{vol}(P) = 0$  and whether or not  $\text{vol}(P) = \infty$ , we can assume without loss of generality that  $0 < \text{vol}(P) < \infty$ .

We shall prove two theorems. The first shows that computing  $\text{vol}(P)$  is #P-hard.

**THEOREM 1.** *Computing  $\text{vol}(P(A, \mathbf{b}))$  is #P-hard, even when  $A$  is totally unimodular.  $\square$*

There is a technical difficulty in stating the converse of this theorem. It may be encapsulated in the following problem.

**Problem.** Is  $\text{size}(\text{vol}(P))$  polynomially bounded in  $L$ ?  $\square$

It is not too difficult to show that  $\text{vol}(P)$  is a rational,  $p/q$ , say. We can also show that  $\text{vol}(P)$  cannot be too large. The difficulty is that  $q$  does not appear to have a

\* Received by the editors July 1, 1987; accepted for publication November 15, 1987.

† Department of Computer Studies, University of Leeds, Leeds LS2 9JT, United Kingdom.

‡ Department of Computer Science and Statistics, Queen Mary College, London E1 4NS, United Kingdom and the Department of Mathematics, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213.

bound polynomial in  $L$ . The problem stated above thus seems to be an open (and possibly difficult) question. We show later that the difficulty disappears if we restrict ourselves to any class of polyhedra whose vertices can be scaled to become integer lattice-points by a polynomial size transformation. We conjecture that the answer to our problem is in the affirmative, but we are at present only able to state the following result in terms of approximation.

**THEOREM 2.** *Let  $\varepsilon > 0$  be rational. Given a #P-complete oracle, in time polynomial in  $L$  and size  $(\varepsilon)$ , we can compute  $\hat{V}$  such that*

$$(1.1) \quad |\hat{V} - \text{vol}(P(\mathbf{A}, \mathbf{b}))| < \varepsilon. \quad \square$$

**COROLLARY 1.** *Suppose we restrict our attention in Theorem 2 to any class of polytopes for which  $\text{size}(\text{vol}(P))$  is bounded by a polynomial in  $L$ . Then, using a #P-complete oracle,  $\text{vol}(P)$  can be computed exactly in time polynomial in  $L$ .  $\square$*

Let us now consider Problem 2.

**Problem 2.** Let  $X = \{X_1, X_2, \dots, X_m\}$  be a set of rational points in  $\mathbb{R}^n$ . Let  $P(X)$  be the convex hull of  $X$ . We shall consider the problem of computing  $\text{vol}(P(X))$  when  $X$  is given as an  $n \times m$  matrix  $(x_{ij})$ ,  $i = 1, 2, \dots, n, j = 1, 2, \dots, m$ .  $\square$

**THEOREM 3.** *Computing  $\text{vol}(P(X))$  is #P-hard.  $\square$*

This time we have no difficulty in stating the converse result because, as we will show,  $\text{size}(\text{vol}(P(X)))$  is polynomially bounded in  $\text{size}(X)$ .

**THEOREM 4.** *Computing  $\text{vol}(P(X))$  is #P-easy.  $\square$*

**2. #P-hardness of Problem 1.** Let  $B = \{0, 1\}$  and  $C = B^n$ . We will consider a single linear inequality

$$(2.1) \quad \mathbf{a}^T \mathbf{x} \leq b$$

in  $\mathbb{R}^n$ , with  $\mathbf{a} > \mathbf{0}$  an integer vector. Define the polytope

$$(2.2) \quad P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^T \mathbf{x} \leq b, \mathbf{0} \leq \mathbf{x} \leq \mathbf{e}\}$$

where  $\mathbf{e}^T = (1, 1, \dots, 1)$ , and let

$$(2.3) \quad K = C \cap P.$$

We shall wish to regard (2.1) as being parameterised by  $b$ , in which case we will write  $P(b)$  or  $K(b)$  for emphasis. The following problem is known to be #P-hard (see [3]).

**#KNAPSACK**

**Input.** Positive integers  $a_1, a_2, \dots, a_n, b$ .

**Problem.** Determine  $N = |K|$ .

We Turing reduce #KNAPSACK to computing the volumes of certain polyhedra. For  $\mathbf{x} \in C$ , let  $|\mathbf{x}| = \mathbf{e}^T \mathbf{x}$  denote the number of 1's in  $\mathbf{x}$ .

Now consider the following problem, which may appear rather artificial, but will be required later.

**#PARITY**

**Input.** As for #KNAPSACK.

**Problem.** For  $i = 0, 1$  let  $N_i = |\{\mathbf{x} \in K : |\mathbf{x}| \equiv i \pmod{2}\}|$ . Determine

$$(2.4) \quad D = N_0 - N_1.$$

**LEMMA 1.** #PARITY is #P-hard.

*Proof.* Suppose we have a procedure for #PARITY. Let  $M = \mathbf{e}^T \mathbf{a} + 1$ , and consider the set of  $(n + 1)$  #PARITY problems, as above, corresponding to the inequalities

$$(2.5) \quad (\mathbf{a} + M\mathbf{e})^T \mathbf{x} \leq b + rM \quad (r = 0, 1, \dots, n).$$



Let  $N^{(r)} = |\{\mathbf{x} \in K : |\mathbf{x}| = r\}|$ . It follows, by an easy analysis, that the  $r$ th #PARITY problem defined by (2.5) determines the value

$$D^{(r)} = (-1)^r N^{(r)} + \sum_{j=0}^{r-1} (-1)^j \binom{n}{j}.$$

A straightforward calculation now yields

$$N = 2^{n-1} + \sum_{r=0}^n (-1)^r D^{(r)}.$$

Thus a polynomial number of calls to the #PARITY procedure, plus polynomial additional time, would solve #KNAPSACK.  $\square$

We now turn to the proof of Theorem 1.

*Proof of Theorem 1.* Assume we have some procedure for determining  $\text{vol}(P)$ . Let  $\Delta = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^T \mathbf{x} \leq b, \mathbf{x} \geq \mathbf{0}\}$ . Note that  $\Delta$  is bounded with  $\text{vol}(\Delta) = b^n / (n! \prod_{j=1}^n a_j)$ . Let  $U$  be the uniform measure on subsets of  $\Delta$  such that  $U(\Delta) = b^n$ . Note that  $U$  is related to volume for a subset  $E$  of  $\Delta$  by the equation  $\text{vol}(E) / \text{vol}(\Delta) = U(E) / U(\Delta)$ . Thus our procedure for determining  $\text{vol}(P)$  can easily be modified to determine  $U(P)$ , and we will assume this has been done. Let  $E_j = \{\mathbf{x} \in \Delta : x_j \leq 1\}$  for  $j = 1, 2, \dots, n$  and  $\bar{E}_j = (\Delta - E_j)$ . Then clearly  $P = \bigcap_{j=1}^n E_j$ . Now, for each  $\mathbf{v} \in C$ , define

$$\bar{E}_{\mathbf{v}} = \{\mathbf{x} \in \Delta : \mathbf{x} > \mathbf{v}\} = \bigcap_{v_j=1} \bar{E}_j.$$

Note that  $\bar{E}_{\mathbf{v}}$  is either empty or is an  $n$ -simplex (actually its closure is). The well-known inclusion-exclusion formula (see, e.g., [7]) now yields

$$U(P) = \sum_{\mathbf{v} \in C} (-1)^{|\mathbf{v}|} U(\bar{E}_{\mathbf{v}}).$$

It is easy to show that

$$\begin{aligned} U(\bar{E}_{\mathbf{v}}) &= 0 && \text{if } \mathbf{a}^T \mathbf{v} > b \\ &= (b - \mathbf{a}^T \mathbf{v})^n && \text{if } \mathbf{a}^T \mathbf{v} \leq b. \end{aligned}$$

Thus

$$U(P) = \sum_{\mathbf{v} \in K} (-1)^{|\mathbf{v}|} (b - \mathbf{a}^T \mathbf{v})^n.$$

Suppose that  $b$  is an integer, and  $\beta$  real such that  $b \leq \beta < b + 1$ . The integrality of  $\mathbf{a}$  then implies that  $K(\beta) = K(b)$ . Thus, writing  $F(\beta) = U(P(\beta))$ , it follows that

$$(2.6) \quad F(\beta) = \sum_{\mathbf{v} \in K(b)} (-1)^{|\mathbf{v}|} (\beta - \mathbf{a}^T \mathbf{v})^n.$$

Let  $\beta$  be a rational  $p/q$  with  $p, q > 0$  relatively prime. It follows from (2.6) that  $q^n F(\beta)$  is an integer such that  $q^n F(\beta) \leq (2p)^n$ . Thus  $\text{size}(F(\beta)) \leq n(\text{size}(\beta) + 1)$ . Thus the procedure for calculating  $U(P)$  can be used to determine  $F(\beta)$ , provided  $\beta$  has size polynomial in  $L$ . Now, expanding the terms in (2.6), we see that

$$(2.7) \quad F(\beta) = \sum_{i=0}^n \alpha_i \beta^{n-i}$$

where

$$(2.8) \quad \alpha_i = (-1)^i \binom{n}{i} \sum_{\mathbf{v} \in K(b)} (-1)^{|\mathbf{v}|} (\mathbf{a}^T \mathbf{v})^i \quad (i = 0, 1, \dots, n).$$

Observe, in particular, that  $\alpha_0 = \sum_{\mathbf{v} \in K(b)} (-1)^{|\mathbf{v}|} = D$ , as defined in (2.4). This is #P-hard to compute (Lemma 1). It follows, a fortiori, that it is #P-hard to determine the coefficients of the polynomial  $F$ . But we may do this as follows:

(i) Determine  $F(\beta) = U(P(\beta)) = n! \prod_{j=1}^n a_j \text{vol}(P(\beta))$  for  $\beta = b + k/(n + 1)$ ,  $k = 0, 1, \dots, n$ .

(ii) Solve (in polynomial time) the resulting system of linear equations determined by (2.7).

We observe that none of the numbers involved in the computations is very large. For example, it follows from (2.8) that the  $\alpha_i$  are integers such that  $|\alpha_i| \leq (4e^T \mathbf{a})^n$ .

There is one final point. The statement of Theorem 1 claims that it remains true when  $A$  is totally unimodular. This follows by making the substitution

$$y_j = a_j x_j \quad (j = 1, 2, \dots, n)$$

in the system (2.2) defining  $P$ . The constraints are then easily transformed into a totally unimodular system. The theorem now follows.  $\square$

*Remark.* The problem we have proved hard may be viewed in either of the following ways:

(i) If  $\mathbf{X}$  is a point chosen randomly from the uniform probability distribution on the unit hypercube, then it is hard to compute the probability that  $\mathbf{X}$  satisfies a single linear inequality.

(ii) Integrating the step-function

$$\begin{aligned} \psi(\mathbf{x}) &= 1 && (b - \mathbf{a}^T \mathbf{x} \leq 0) \\ &= 0 && (b - \mathbf{a}^T \mathbf{x} > 0) \end{aligned}$$

over the unit hypercube is  $\#P$ -hard.

Note that the function  $\psi$  in (ii) is not even continuous. However, we may integrate  $\psi$  explicitly over  $k$  variables, using (2.6). This gives the piecewise polynomial function

$$\psi_k(\mathbf{x}) = \sum_{\hat{\mathbf{v}} \in \mathcal{K}(b - \mathbf{a}^T \mathbf{x})} (-1)^{|\hat{\mathbf{v}}|} (b - \mathbf{a}^T \mathbf{x} - \hat{\mathbf{a}}^T \hat{\mathbf{v}})^k$$

where  $\hat{\mathbf{a}}, \hat{\mathbf{v}} \in \mathbb{R}^k$  and  $\mathbf{a}, \mathbf{x} \in \mathbb{R}^{n-k}$ . This function is readily shown to be of class  $C^{k-1}$ . Hence we have the conclusion that it is hard to integrate such a function over the unit hypercube. In fact, since the description of  $\psi_k$  continues to be of polynomial size for  $k = O(\log n)$ , the conclusion can be strengthened further.

**3.  $\#P$ -easiness of Problem 1.** We will go straight into the proof of Theorem 2.

*Proof of Theorem 2.* Since  $P$  is bounded, we have [5, Thm. 10.2]

$$(3.1) \quad P \subseteq \Gamma = \{\mathbf{x} \in \mathbb{R}^n : |x_j| \leq D = 2^{4n^2 L} (j = 1, 2, \dots, n)\}.$$

Now let  $s = \lceil mn^2 2^{4n^3 L + n} / \varepsilon \rceil$ , and divide  $\Gamma$  into  $s^n$  subcubes of side  $\delta = 2D/s$ . These subcubes fall into three classes:

- (i) interior (wholly) to  $P$ ;
- (ii) exterior (wholly) to  $P$ ;
- (iii) boundary.

We will estimate  $V = \text{vol}(P)$  by the total volume of the interior cubes,  $\hat{V}$ . Thus  $\hat{V} = I\delta^n$ , where  $I$  is the number of interior cubes. Clearly  $V \geq \hat{V}$ , and the error  $E = V - \hat{V}$  is bounded above by  $J\delta^n$ , where  $J$  is the number of boundary cubes. We show later that

$$(3.2) \quad J \leq mn^2 s^{n-1},$$

and hence  $J\delta^n \leq \varepsilon$  as required.

The counting machine  $\phi$  [6] which computes  $I$  works as follows. Each copy chooses one of the integer vectors  $\mathbf{t}$ , where  $1 \leq t_j \leq s$  ( $j = 1, 2, \dots, n$ ) in time proportional to  $n$  size ( $s$ ). It then tests, in polynomial time, whether the subcube

$$\Gamma' = \{\mathbf{x} \in \mathbb{R}^n : -D + (t_j - 1)\delta \leq x_j \leq -D + t_j\delta, j = 1, 2, \dots, n\}$$

is interior. Note that this is equivalent to

$$\frac{1}{2} \delta \sum_{j=1}^n |a_{ij}| \leq b_i + \sum_{j=1}^n a_{ij} \{D - (t_j - \frac{1}{2})\} \delta \quad (i = 1, 2, \dots, m).$$

It is clear that the sizes of all numbers involved, in particular  $s$ , are polynomial in  $L$ . The output of  $\phi$  is  $I$ , from which  $\hat{V}$  is calculated.

It remains only to prove (3.2). If

$$P = \{x \in \mathbb{R}^n : a_i^T x \leq b_i (i = 1, 2, \dots, m)\}$$

then let  $H_i = \{x \in \mathbb{R}^n : a_i^T x = b_i\}$ . Call a subcube  $\Gamma'$  of  $\Gamma$  *intersected* if  $H_i \cap \Gamma' \neq \emptyset$  for some  $1 \leq i \leq m$ . Let  $K$  be the number of intersected subcubes. Clearly  $K \geq J$ . Each subcube  $\Gamma'$  projects onto  $n$  squares  $\Gamma'_j$  in the coordinate planes, i.e., if  $\Gamma' = \{x \in \mathbb{R}^n : u_k \leq x_k \leq u_k + \delta (k = 1, 2, \dots, n)\}$  then  $\Gamma'_j = \{x : x_j = 0 \text{ and } u_k \leq x_k \leq u_k + \delta (k \neq j)\}$ . The total number of distinct squares is clearly  $ns^{n-1}$ . Construct a mapping from the intersected cubes to the squares as follows.

For each intersected cube  $\Gamma'$  choose some  $i$  such that  $H_i \cap \Gamma' \neq \emptyset$  (e.g., the least such  $i$ ). Choose  $j$  so that

$$|a_{ij}| = \max_{1 \leq k \leq n} |a_{ik}|.$$

Suppose two intersected cubes map to the same square with the same value of  $i$ . Then there exist points  $x, x'$  within the two cubes such that

$$\begin{aligned} a_i^T x &= a_i^T x' = b_i, \\ x_j &\geq x'_j \quad \text{and} \quad |x_k - x'_k| \leq \delta \quad (k \neq j). \end{aligned}$$

Thus  $x_j - x'_j \leq \sum_{k \neq j} |a_{ik}| \delta / |a_{ij}| \leq (n-1)\delta$ . Therefore at most  $n$  cubes can map onto the same square for a given  $i$ . Since there are only  $m$  values of  $i$ , at most  $mn$  cubes map onto the same square. Hence  $K \leq mn \times ns^{n-1} = mn^2 s^{n-1}$ , and (3.2) follows.

*Proof of Corollary 1.* Suppose we know that  $\text{size}(\text{vol}(P)) \leq p(L)$  for some polynomial  $p(L)$ . Take  $\varepsilon = 1/3p(L)^2$  in Theorem 2. We know that  $V = a/b$  for integers  $a, b \leq 2^{p(L)}$ , and  $V - \hat{V} \leq \varepsilon$ . Having computed  $\hat{V}$ , we can use continued fractions to compute  $V$  exactly in polynomial time (see, e.g., [5, Cor. 6.3a]).  $\square$

Finally, we give a simple example of a class of polyhedra for which  $\text{vol}(P)$  has polynomial size.

LEMMA 2. *If  $P$  is integral, then  $\text{size}(\text{vol}(P)) = O(L^3)$ .*

*Proof.* Let  $P = \{x \in \mathbb{R}^n : a_i^T x \leq b_i (i = 1, 2, \dots, m)\}$  and be such that every vertex of  $P$  has integer coordinates. Now  $P$  has a triangulation using only its own vertices. If  $\sigma$  is a simplex of this triangulation, then

$$\text{vol}(\sigma) = \frac{1}{n!} \det \begin{pmatrix} 1 & 1 & & 1 \\ v_0 & v_1 & \dots & v_n \end{pmatrix} = \frac{\nu_\sigma}{n!}, \quad \text{say,}$$

where  $v_i (i = 0, 1, \dots, n)$  are vertices of  $P$ . Now  $\nu_\sigma$  is an integer. Hence  $\text{vol}(P) = \nu/n!$ , with  $\nu = \sum_\sigma \nu_\sigma$ , i.e.;  $n! \text{vol}(P)$  is an integer. Now (3.1) implies

$$\text{vol}(P) \leq 2^{8n^2L+n}.$$

Hence

$$\text{size}(\text{vol}(P)) \leq 8n^2L + n + \lceil \log_2(n! + 1) \rceil + 1,$$

and the result follows.  $\square$

**4. #P-hardness of Problem 2.** We shall again reduce the counting problem #KNAPSACK to volume computations of the relevant form. We will make the following two additional assumptions for #KNAPSACK:

$$(4.1) \quad \text{There are no 0-1 solutions to } \mathbf{a}^T \mathbf{x} = b.$$

We can ensure this simply by replacing  $a_j$  by  $2a_j$  ( $j = 1, 2, \dots, n$ ) and  $b$  by  $2b + 1$  in (2.1). This does not affect the value of  $N$ .

$$(4.2) \quad b > \frac{1}{2} \mathbf{e}^T \mathbf{a}.$$

If this is not true on input, then we can add a variable  $x_{n+1}$  with  $a_{n+1} = \mathbf{e}^T \mathbf{a} - b + 1$ , and replace  $b$  by  $b + a_{n+1}$ . This adds  $2^n$  to the value of  $N$ .

Now let  $P_1 = P = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^T \mathbf{x} \leq b, \mathbf{0} \leq \mathbf{x} \leq \mathbf{e}\}$ . Then (4.2) implies that  $\frac{1}{2} \mathbf{e} = (\frac{1}{2}, \frac{1}{2}, \dots, \frac{1}{2})$  is an interior point of  $P_1$ . Now, substituting  $\mathbf{y} = 2\mathbf{x} - \mathbf{e}$ , transform  $P_1$  to

$$P_2 = \{\mathbf{y} \in \mathbb{R}^n : \mathbf{a}^T \mathbf{y} \leq b', -\mathbf{e} \leq \mathbf{y} \leq \mathbf{e}\},$$

where  $b' = 2b - \mathbf{e}^T \mathbf{a} > 0$ . Note that  $\mathbf{x} \in \{0, 1\}^n \cap P_1 \leftrightarrow \mathbf{y} \in \{-1, +1\}^n \cap P_2$ . So we have reduced #KNAPSACK to computing  $N = |Y|$ , where

$$\begin{aligned} Y &= \{\mathbf{y} \in P_2 : y_j = \pm 1 \ (j = 1, 2, \dots, n)\} \\ &= \{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(N)}\}, \text{ say.} \end{aligned}$$

The constraints defining  $P_2$  can be written as

$$(4.3) \quad \begin{aligned} &\mathbf{A}^T \mathbf{y} \leq 1, \\ &-y_j \leq 1, \quad y_j \leq 1 \quad (j = 1, 2, \dots, n) \end{aligned}$$

where  $\mathbf{A} = \mathbf{a}/b'$ . Now  $\mathbf{0}$  is an interior point of  $P_2$ . Consider the polar  $P_2^*$  [5, Chap. 9], where

$$(4.4) \quad \begin{aligned} P_2^* &= \{\mathbf{z} \in \mathbb{R}^n : \mathbf{z}^T \mathbf{u} \leq 1 \text{ for all } \mathbf{u} \in P_2\} \\ &= \text{conv}\{\mathbf{A}, \mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_n, -\mathbf{e}_1, -\mathbf{e}_2, \dots, -\mathbf{e}_n\}. \end{aligned}$$

The second equality in (4.4) merely states the well-known relationship between the facets of  $P_2$  and the vertices of  $P_2^*$ .

We show that  $N$  can be computed from  $\text{vol}(P_2^*)$  and  $\text{vol}(\hat{P}_2^*)$ , where  $\hat{P}_2^*$  is the polyhedron obtained by using  $(b + \frac{1}{2})$  in place of  $b$  in the definition of  $P_1$  before transforming to  $P_2^*$ . Since  $P_2^*, \hat{P}_2^*$  are defined as convex hulls, Theorem 3 will follow.

Now polarity yields a one-to-one correspondence between the (*nondegenerate*) vertices of  $P_2$  not lying in facet  $\{\mathbf{y} \in P_2 : \mathbf{A}^T \mathbf{y} = 1\}$  (i.e., members of  $Y$ ) and the (*simplicial*) facets of  $P_2^*$  not containing the vertex  $\mathbf{A}$ . Observe that the facet corresponding to  $\mathbf{y}^{(t)}$  has vertex set  $\Lambda^{(t)} = \{y_i^{(t)} \mathbf{e}_i : i = 1, 2, \dots, n\}$  for  $t = 1, 2, \dots, N$ . Thus there is a decomposition of  $P_2^*$  into simplices  $\sigma_1, \sigma_2, \dots, \sigma_N$ , where  $\sigma_t = \text{conv}(\{\mathbf{A}\} \cup \Lambda^{(t)})$ . Hence

$$\begin{aligned} \text{vol}(P_2^*) &= \sum_{t=1}^N \text{vol}(\sigma_t) \\ &= \frac{1}{n!} \sum_{t=1}^N \left| \det \begin{pmatrix} 1 & 1 & \dots & 1 \\ \mathbf{A} & y_1^{(t)} \mathbf{e}_1 & \dots & y_n^{(t)} \mathbf{e}_n \end{pmatrix} \right| \\ &= \frac{1}{n!} \sum_{t=1}^N |1 - \mathbf{A}^T \mathbf{y}^{(t)}| \quad \text{since } y_i^{(t)} = \pm 1 \\ &= \frac{2}{n!} \sum_{t=1}^N \left| \frac{b - \mathbf{a}^T \mathbf{x}^{(t)}}{2b - \mathbf{e}^T \mathbf{a}} \right| \end{aligned}$$

where  $\mathbf{x}^{(t)} = \frac{1}{2}(\mathbf{e} + \mathbf{y}^{(t)})$  is the zero-one solution to (2.1) which corresponds to  $\mathbf{y}^{(t)}$ . Thus, using (4.2),

$$\begin{aligned} \frac{1}{2} n! \text{vol}(P_2^*) &= \sum_{t=1}^N \frac{b - \mathbf{a}^T \mathbf{x}^{(t)}}{2b - \mathbf{e}^T \mathbf{a}} \\ &= \frac{bN - S}{2b - \mathbf{e}^T \mathbf{a}} \end{aligned}$$

where  $S = \sum_{t=1}^N \mathbf{a}^T \mathbf{x}^{(t)}$ . Furthermore

$$\frac{1}{2} n! \text{vol}(\hat{P}_2^*) = \frac{(b + \frac{1}{2})N - S}{2(b + \frac{1}{2}) - \mathbf{e}^T \mathbf{a}}$$

where we have used, in the notation of § 2,  $K(b) = K(b + \frac{1}{2})$ , so that  $N$  and  $S$  are unchanged.

Now  $N$  can be easily computed from  $\text{vol}(P_2^*)$  and  $\text{vol}(\hat{P}_2^*)$ . Finally note that  $S$  is an integer with  $0 < S \leq 2^n b$ , so that the numbers involved are of polynomial size. Thus Theorem 3 has been proved.  $\square$

**5. #P-easiness of Problem 2.** Let us first show that  $\text{size}(\text{vol}(P(X)))$  is polynomially bounded. Indeed we can prove quite straightforwardly that, for  $m > n \geq 1$ ,

$$(5.1) \quad \tau = \text{size}(\text{vol}(P(X))) \leq 3mn^2L$$

where  $L = \text{size}(X)$ . To see this, let  $\lambda$  be the least common multiple of the denominators of entries of  $X$ . Then  $\text{size}(\lambda) \leq mnL$ . Now  $\lambda X$  contains integer vectors and  $\text{vol}(P(\lambda X)) = \lambda^n \text{vol}(P(X))$ . Since  $P(\lambda X)$  can be decomposed into simplices (see proof of Lemma 2), it follows that  $n! \text{vol}(P(\lambda X))$  is an integer. Hence

$$\text{vol}(P(X)) = \frac{n! \lambda^n \text{vol}(P(X))}{n! \lambda^n}$$

expresses  $\text{vol}(P)$  as a ratio of two integers. Noting that  $\text{vol}(P(X)) \leq 2^{nL}$ , (5.1) follows.

The proof of Theorem 4 is similar to that of Theorem 2, and so we will only give an outline. Now

$$P(X) \subseteq \Gamma_1 = \{\mathbf{x} \in \mathbb{R}^n : |x_j| \leq 2^L (j = 1, 2, \dots, n)\}.$$

Let  $s = \lceil m^n n^2 2^{6mn^2L + n(L+1)+2} \rceil$  and divide  $\Gamma_1$  into  $s^n$  subcubes of size  $\delta_1 = 2^{L+1}/s$ . Our counting machine computes the number  $I_1$  of subcubes which intersect  $P(X)$ . The estimate of volume is then  $I_1 \delta_1^m / m!$ . Note that this time we use an overestimate, rather than an underestimate, of the volume.

We can now reduce the testing for the intersection of an individual subcube with  $P(X)$  to the solution of a single linear program.

To establish this claim, for simplicity let us assume we translate and scale the problem to that of testing the intersection of the cube  $\Sigma = \{\mathbf{x} \in \mathbb{R}^n : -1 \leq x_j \leq 1 (j = 1, 2, \dots, n)\}$ , with the polytope  $P = \text{conv}\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$ . Then  $\Sigma \cap P = \emptyset$  if and only if there exists a  $\boldsymbol{\gamma} \in \mathbb{R}^n$  such that

$$\sum_{i=1}^n |\gamma_i| < 1 \quad \text{and} \quad \boldsymbol{\gamma}^T \mathbf{x}_i \geq 1 \quad (i = 1, 2, \dots, m).$$

This can be tested by solving the linear program

$$\begin{aligned} &\text{minimise} \quad z = \mathbf{e}^T (\boldsymbol{\gamma}_1 + \boldsymbol{\gamma}_2) \\ &\text{subject to} \quad \mathbf{x}_i^T (\boldsymbol{\gamma}_1 - \boldsymbol{\gamma}_2) \geq 1 \quad (i = 1, 2, \dots, m) \end{aligned}$$

and by checking whether  $\min z < 1$ .

The error in the volume estimate is again bounded by the total volume of the subcubes which intersect the boundary of  $P(X)$ . Since  $P(X)$  has less than  $m^n$  facets, this is at most

$$m^n n^2 s^{n-1} \delta_1^n$$

(see the proof of Theorem 2 with  $m$  replaced by  $m^n$ ). By (5.1), this is at most  $2^{-2(\tau+1)}$ , and so  $\text{vol}(P(X))$  can be computed exactly using continued fractions.

**6. Remarks.** Our results leave open two interesting questions. The first is the problem raised in § 1 concerning the size of description of polyhedral volumes. The second is as to whether it remains hard to approximate the volume in either Problems 1 or 2; i.e., for some given  $\varepsilon > 0$ , is it hard to obtain an estimate  $\hat{V}$  of the volume  $V$  such that  $(1 - \varepsilon) < \hat{V}/V < (1 + \varepsilon)$ ? Our methods appear to shed little light on this issue, but we conjecture that this approximation problem is also hard. Finally, we observe that determining the volume of a polyhedron in *fixed* dimension is easy. We simply determine the complete face-lattice of the polyhedron, triangulate it, and then use the formula for the volume of a simplex.

#### REFERENCES

- [1] I. BÁRÁNY AND Z. FÜREDI, *Computing the volume is difficult*, in Proc. 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 442–447.
- [2] G. ELEKES, *A geometric inequality and the complexity of computing volume*, Discrete and Computational Geometry, to appear.
- [3] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
- [4] L. LOVÁSZ, *An Algorithmic Theory of Numbers, Graphs and Convexity*, CBMS-NSF Regional Conference Series in Applied Mathematics 50, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986.
- [5] A. SCHRIJVER, *Theory of Linear and Integer Programming*, John Wiley, Chichester, 1986.
- [6] L. G. VALIANT, *The complexity of enumeration and reliability problems*, SIAM J. Comput., 8 (1979), pp. 410–421.
- [7] S. G. WILLIAMSON, *Combinatorics for Computer Science*, Computer Science Press, Rockville, MD, 1985.

## FAULT TOLERANCE IN NETWORKS OF BOUNDED DEGREE\*

CYNTHIA DWORK†, DAVID PELEG†, NICHOLAS PIPPENGER†, AND ELI UPFAL†

**Abstract.** Achieving processor cooperation in the presence of faults is a major problem in distributed systems. Popular paradigms such as Byzantine agreement have been studied principally in the context of a complete network. Indeed, Dolev [*J. Algorithms*, 3 (1982), pp. 14–30] and Hadzilacos [*Issues of Fault Tolerance in Concurrent Computations*, Ph.D. thesis, Harvard University, Cambridge, MA, 1984] have shown that  $\Omega(t)$  connectivity is necessary if the requirement is that *all* nonfaulty processors decide unanimously, where  $t$  is the number of faults to be tolerated. We believe that in foreseeable technologies the number of faults will grow with the size of the network while the degree will remain practically fixed. We therefore raise the question whether it is possible to avoid the connectivity requirements by slightly lowering our expectations. In many practical situations we may be willing to “lose” some correct processors and settle for cooperation between the vast majority of the processors. Thus motivated, we present a general simulation technique by which vertices (processors) in almost any network of bounded degree can simulate an algorithm designed for the complete network. The simulation has the property that although some correct processors may be cut off from the majority of the network by faulty processors, the vast majority of the correct processors will be able to communicate among themselves undisturbed by the (arbitrary) behavior of the faulty nodes.

We define a new paradigm for distributed computing, almost-everywhere agreement, in which we require only that almost all correct processors reach consensus. Unlike the traditional Byzantine agreement problem, almost-everywhere agreement can be solved on networks of bounded degree. Specifically, we can simulate any sufficiently resilient Byzantine agreement algorithm on a network of bounded degree using our communication scheme described above. Although we “lose” some correct processors, effectively treating them as faulty, the vast majority of correct processors decide on a common value.

**Key words.** fault tolerance, communication, bounded-degree network, expander graph

**AMS(MOS) subject classifications.** 68M10, 68M15, 68R10

**1. Preliminaries.** In 1982 Dolev [D] published the following damning result for distributed computing: “Byzantine agreement is achievable only if the number of faulty processors in the system is less than one-half of the connectivity of the system’s network.” Even in the absence of malicious failures connectivity  $t+1$  is required to achieve agreement in the presence of  $t$  faulty processors [H].

The results are viewed as damning because of the fundamental nature of the Byzantine agreement problem. In this problem each processor begins with an initial value drawn from some domain  $V$  of possible values. At some point during the computation, during which processors repeatedly exchange messages and perform local computations, each processor must irreversibly decide on a value, subject to two conditions. No two correct processors may decide on different values, and if all correct processors begin with the same value  $v$ , then  $v$  must be the common decision value. (See [F] for a survey of related problems.) The ability to achieve this type of coordination is important in a wide range of applications, such as database management, fault-tolerant analysis of sensor readings, and coordinated control of multiple agents.

A simple corollary of the results of Dolev and Hadzilacos is that in order for a system to be able to reach Byzantine agreement in the presence of up to  $t$  faulty processors, every processor must be directly connected to at least  $\Omega(t)$  others. Such high connectivity, while feasible in a small system, cannot be implemented at reasonable cost in a large system.

As technology improves, increasingly large distributed systems and parallel computers will be constructed. However, in any forthcoming technology, the number of

\* Received by the editors June 17, 1986; accepted for publication (in revised form) November 3, 1987.

† IBM Almaden Research Center, San Jose, California 95120-6099.

faulty processors in a given system will grow with the size of the system, whereas the degree of the interconnection network by which the processors communicate will, for all practical purposes, remain fixed.

Despite these negative results, distributed systems are widely used and parallel computers are being built. This suggests that the correctness conditions for Byzantine agreement are too stringent to reflect practical situations. In particular, Byzantine agreement guarantees coordination among all correct processors, by necessarily omitting up to  $t$  faulty processors. In many situations it may suffice to guarantee agreement among all but  $O(t)$  processors. In other situations a simple majority consensus may suffice. Similarly, in clock synchronization, or in firing squad synchronization, it may suffice for a vast majority of the correct processors to be synchronized.

In the traditional paradigm for distributed computing described above, the correctness conditions describe the states of *all* nonfaulty processors. In this paper we propose a new paradigm for fault-tolerant computing in which correctness conditions are relaxed by “giving up for lost” those correct processors whose communication paths to the remainder of the network are excessively corrupted by faulty processors. Such a processor is called *poor*. While any network of bounded degree must contain some poor processors, in this paper we show that their number can often be kept quite small, even in networks of constant degree. Further, we argue that this type of cooperation may fit well most applications of, say, Byzantine agreement. All known algorithms guarantee only that if at most  $f \leq t < n/3$  processors fail then at least  $k \geq n - f$  processors will mutually agree on a value. Our results show that we can eliminate the costly connectivity condition requiring  $\Omega(nt)$  edges by employing an appropriately chosen bounded-degree network of  $n + O(t)$  processors and still guarantee agreement among  $n$  correct processors. Our paradigm admits deterministic solutions in networks of small constant degree to such fundamental problems as atomic broadcast, Byzantine agreement, and clock synchronization.

We present a general simulation technique by which for almost any regular graph  $G$ , the vertices (processors) of  $G$  can simulate an algorithm designed for a complete network in such a way that the number of poor processors in  $G$  is small. The crux of the simulation is a transmission scheme for simulating the point-to-point transmissions of the complete network by sending messages along several paths of  $G$  in such a way that there will always be a large set of correct processors capable of communicating among themselves as if they comprise a fully connected subnetwork, independent of the behavior of the faulty processors.

For consensus problems we can often do better than in the general simulation by employing a *compression procedure* based on the existence of *compressor graphs* [P]. This procedure is iterative and local in nature, and cannot by itself guarantee agreement. However, it can be used to “sharpen” dichotomies in that if a sufficiently large majority (e.g., all but  $O(t \log t)$ ) of the correct processors have the same value, then the procedure converges and strengthens this majority (e.g., to all but  $t + 1$ ).

Our model of computation is identical to that commonly used in the Byzantine literature. Specifically, each processor can be thought of as a (possibly infinite) state machine with special registers for communication with the outside world. The processors communicate by means of point-to-point links, which are assumed to be completely reliable. The entire system is synchronous, and can be thought of as controlled by a common clock. At each pulse of the common clock a processor may send a message on each of its incident communication links (possibly different messages on different links). Messages sent at one clock pulse are delivered before the next pulse.

For each of our transmission schemes there is a specific lower bound  $b$  on the number of clock pulses needed to simulate one complete round of message exchange



in the simulated network. For simplicity we assume that the common clock sends a “super-pulse” every  $b$  rounds. A processor simulates round  $r$  of the original algorithm at the  $r$ th super-pulse.

Since we cannot hope to solve the Byzantine agreement problem exactly on networks of bounded degree, we introduce the notion of *almost-everywhere agreement* (denoted a.e. agreement), in which all but a small number of the correct processors must choose a common decision value.

More precisely, a protocol  $P$  is said to achieve  $t$ -resilient  $X$  agreement, where  $X$  is any term, if in every execution of  $P$  in which at most  $t$  processors fail all but  $X$  of the correct processors eventually decide on a common value. Moreover, if all the correct processors share the same initial value then that must be the value chosen. Note that the traditional Byzantine agreement problem is just 0 agreement.

A protocol solves a.e. agreement if it solves  $X$  agreement for some  $X$  such that  $X/(n-t) \rightarrow 0$  as  $n \rightarrow \infty$ .

Our first result applies only to fail stop, omission, or authenticated Byzantine faults.

**THEOREM 1.** *For all  $r \geq 5$  there exists a constant  $\varepsilon = \varepsilon(r)$  such that for all  $t < \varepsilon n$  almost all  $r$ -regular graphs (i.e., all but a vanishingly small fraction of such graphs) admit a  $t$ -resilient algorithm for  $O(t)$  agreement.*

The remaining results apply to unauthenticated Byzantine failures.

**THEOREM 2.** *For all  $r \geq 5$ , almost all  $r$ -regular graphs admit a  $t$ -resilient algorithm for  $O(t)$  agreement, where  $t \leq n^{1-\varepsilon}$ , for some constant  $\varepsilon = \varepsilon(r)$ , where  $\varepsilon(r) \rightarrow 0$  as  $r \rightarrow n$ .*

The next theorems describe explicit graphs for which the set of poor processors is small.

**THEOREM 3.** *The  $n$  node butterfly network (degree 4; see § 2.3 for definition) admits a  $t$ -resilient  $O(t \log t)$ -agreement algorithm for  $t \leq cn/\log n$  for some constant  $c$ .*

The result of Theorem 3 can be improved for a family of networks obtained by superimposing a compressor of degree 5 on a butterfly network.

**THEOREM 4.** *There exists a constant  $c$  and a network of degree 9 that admits a  $t$ -resilient  $O(t)$ -agreement algorithm for  $t \leq cn/\log n$ .*

In the case of unauthenticated Byzantine failures, we achieve  $O(t)$  agreement only for  $t \leq cn/\log n$ . If  $t > O(n/\log n)$  then it is easy to show that the number of poor processors is linear in  $n$ . The existence problem for an  $O(t)$ -agreement algorithm in this case remains open. However, we solve this problem on graphs of unbounded but still relatively small degree.

**THEOREM 5.** *For every  $0 < \varepsilon < 1$  there exist a constant  $c = c(\varepsilon)$ , graphs  $G$  of degree  $O(n^\varepsilon)$ , and  $t$ -resilient  $O(t)$ -agreement algorithms for  $t \leq cn$ .*

Finally we present a purely combinatorial characterization of networks which admit  $p(t)$  agreement for any function  $p$ . When  $p(t) = 0$  our characterization coincides exactly with the  $(2t+1)$ -connectivity requirement for the traditional Byzantine agreement cited above [D].

**2. Simulation results.** In § 2.1 we describe a general strategy for simulating on one network any algorithm designed for another network, describing what we mean by “simulation.” In § 2.2 we discuss a general scheme for implementing our strategy, and in § 2.3 we make all of this more concrete by presenting the simulation of a complete network by a butterfly network. In § 2.4 we show that our general scheme can be implemented on almost all regular graphs of bounded degree. Finally, § 2.5 briefly discusses our results under more restrictive fault models.

**2.1. The general simulation.** For simplicity, we take the simulated network to be completely connected. Let  $A$  be an algorithm designed for a fully connected network  $H$ . Consider an arbitrary network  $G$  over the same set of vertices (processors) as in

$H$ , and suppose we wish to simulate  $A$  on  $G$ . We need only specify the simulation of communication between processors; a direct message from a processor  $u$  to its neighbor  $v$  in  $H$  can be simulated in  $G$  by sending the message from  $u$  to  $v$  through various paths, and supplying  $v$  with a method for determining the correct value of the message, e.g., by taking the value appearing in the majority of the paths. Taken together, the particular choice of paths and the supplied decision method constitute a *transmission scheme*. Of course, even if  $u$  and  $v$  are correct processors the faulty processors may be so placed that all or most of the paths from  $u$  to  $v$  are corrupted. Thus, even if a processor is correct, it may be unable to properly communicate with the other processors.

Let a transmission scheme for  $G$  be fixed. Let  $T$  be a subset of the vertices of  $G$  (think of  $T$  as the set of faulty processors). A pair of nodes  $(u, v) \in G$  is *successful* with respect to  $T$  if, whenever all the processors not in  $T$  follow the transmission scheme correctly, the simulation of a message transmission from  $u$  to  $v$  always succeeds (i.e.,  $v$  decides correctly on the value sent by  $u$ ). Let  $\text{POOR}(G, T)$  be a minimal set of correct nodes such that every pair of nodes,  $u, v \in T \cup \text{POOR}(G, T)$  is successful with respect to  $T$ . (Note that this set need not be unique.) Let  $p(G, t) = \max\{|\text{POOR}(G, T)| \text{ such that } T \subseteq V, |T| = t\}$ . As we will show in Theorem 2.1 there is a  $p(G, t)$ -agreement algorithm resilient to  $t$  failures for every graph  $G$  and suitable choice of  $t$ . We are therefore interested in finding graphs  $G$  for which  $p(G, t)$  is small. Such graphs are the subject of §§ 2.3 and 2.4.

**THEOREM 2.1.** *Let  $A$  be an algorithm for the traditional Byzantine agreement problem designed for network  $H$ , let  $G$  be a graph with the same number of vertices as  $H$ , and let  $TS$  be a transmission scheme for simulating on  $G$  message transmissions in  $H$ . Let  $A(TS)$  be the simulation of  $A$  on  $G$  using the transmission scheme  $TS$  to simulate messages sent on  $H$ . For every  $t$ , if  $A$  is guaranteed to work correctly on  $H$  in the presence of at most  $t + p(G, t)$  faults, then  $A(TS)$  achieves  $p(G, t)$  agreement on  $G$  in the presence of up to  $t$  faults.*

*Proof.* Let  $T$  be a set of faulty processors in  $G$ .  $A(TS)$  simulates the execution of  $A$  on  $H$  by simulating the processors in a one-one fashion. By definition, the processors not in  $\text{POOR}(G, T)$  can communicate among themselves as if they comprised a fully connected subnetwork, so the simulated communication among this set of processors is successful. The behavior of the correct processors in  $\text{POOR}(G, T)$  may appear to be faulty. These are the processors we give up for lost. Since  $A$  is guaranteed to work correctly even in the presence of  $t + p(G, t) \geq |\text{POOR}(G, T)|$  failures we are done.  $\square$

In order to use the transmission schemes described here the processors must have some knowledge of the topology of the system. The amount of knowledge needed, and how this quantity depends on the types of faults considered, are subjects for further research.

**2.2. A class of transmission schemes.** We now describe in more detail a specific class of transmission schemes, called *three-phase* transmission schemes. Let  $G$  be any network in which we may specify the following sets. For every node  $v$  we specify sets of processors  $\Gamma_{\text{in}}(v), \Gamma_{\text{out}}(v) \subseteq V$ , each of fixed (but not necessarily constant) size  $s$ . For each node  $w$  in  $\Gamma_{\text{in}}(v)$  ( $\Gamma_{\text{out}}(v)$ ) a path from  $w$  to  $v$  ( $v$  to  $w$ ) is specified. In addition, for each ordered pair of nodes  $(u, v)$  we specify  $s$  vertex-disjoint paths from  $\Gamma_{\text{out}}(u)$  to  $\Gamma_{\text{in}}(v)$ .

The transmission of a message  $x$  from  $u$  to  $v$  consists of three phases. In the first phase the message is broadcast from  $u$  to every node in  $\Gamma_{\text{out}}(u)$  through the specified paths. Thus, at the end of the first phase a copy of  $x$  is received by all nodes of  $\Gamma_{\text{out}}(u)$ .

(Those processors in  $\Gamma_{out}(u)$  whose path from  $u$  contains faults may have received an incorrect version of the message, or nothing at all.) In the second phase the  $s$  (possibly corrupted) copies of  $u$ 's message are sent along the vertex-disjoint paths from  $\Gamma_{out}(u)$  to  $\Gamma_{in}(v)$ . In the third phase these (possibly corrupted) copies of  $u$ 's message are routed to  $v$  along the specified paths from  $\Gamma_{in}(v)$  to  $v$ . Thus,  $v$  receives  $s$  (possibly corrupted) copies of  $u$ 's message. Finally,  $v$  takes the value appearing in the majority of the copies that arrived to be the actual message. (If no majority exists then a default value is taken.) Clearly,  $v$  could be making a mistake, even if it is a correct processor.

Let  $T$  be some set of nodes on the network (again, think of  $T$  as the set of faulty processors). A node  $u$  is said to be *out-bad* with respect to  $T$  if at least  $\frac{1}{8}$  of the specified paths to  $\Gamma_{out}(u)$  contain a vertex in  $T$ . Intuitively, if  $T$  is the set of faulty processors and  $u$  is out-bad with respect to  $T$ , then at least  $\frac{1}{8}$  of the processors in  $\Gamma_{out}(u)$  may receive a corrupted version of the message  $x$ . Similarly we say  $v$  is *in-bad* with respect to  $T$  if at least  $\frac{1}{8}$  of the paths from  $\Gamma_{in}(v)$  to  $v$  pass through some node of  $T$ . Let  $BAD(G, T)$  be the set of nodes in the network  $G$  which are (either out- or in-) bad with respect to  $T$ . Let  $b(G, t) = \max \{ |BAD(G, T)| \text{ such that } T \subseteq V, |T| = t \}$ . Recall that, informally,  $p(G, t)$  is an upper bound on the number of poor processors in  $G$  when no more than  $t$  processors fail. The next claim bounds the number of poor processors in terms of the number of bad processors.

CLAIM 2.2. *Let  $s$  be the size of the sets  $\Gamma$ . For all  $t < s/4$ ,  $p(G, t) \leq b(G, t)$ .*

*Proof.* We will prove that for any  $T \subseteq V$ ,  $POOR(G, T) \subseteq BAD(G, T)$ . The claim then follows immediately from the definitions. Let  $T$  be some subset of  $V$  of size at most  $t$ . We will show that for any two processors  $u$  and  $v$  not in  $BAD(G, T)$ , any message  $m$  sent from  $u$  to  $v$  according to the three-phase transmission scheme is correctly received by  $v$ . This follows from the following argument: since  $u$  is not out-bad,  $m$  will reach at least  $\frac{7}{8}$  of its first-phase destinations correctly. Thus at the end of the first phase, at most  $s/8$  copies of  $m$  have been corrupted. In the second phase, since every copy is transmitted along disjoint paths, at most  $t$  more copies might get hurt. Finally in the third phase, since  $v$  is not in-bad, at most an additional  $s/8$  copies can be corrupted. Over all, at most  $s/4 + t < s/2$  copies of the message might be lost. Therefore the majority of the copies will arrive intact, so  $v$  will recognize the message correctly.  $\square$

*Remark.* It will sometimes be necessary to consider a more relaxed type of three-phase transmission scheme in which each node  $v \in (\Gamma_{out}(u) \cup \Gamma_{in}(w))$  may appear on at most two of the paths between these two sets (once as an endpoint and at most once more as an intermediate node). The paths have to remain otherwise disjoint. For such a scheme we can prove that if  $s$  is the size of the sets  $\Gamma$ , then for all  $t < s/8$ ,  $p(G, t) \leq b(G, t)$ .

**2.3. Simulation of a complete network on the butterfly.** In this section we show how to simulate a complete network on a simple degree-4 network known as the butterfly [U]. All we have to do is specify a transmission scheme. This is done losing at most  $O(t \log t)$  processors in the presence of  $t$  faults (i.e., at most  $O(t \log t)$  processors will be bad as defined in § 2.2).

An  $m$ -butterfly is a communication graph  $G_m = (V_m, E_m)$  with  $V_m = \{(a, i) | 0 \leq a \leq m-1, 0 \leq i \leq 2^m-1\}$ . The set of edges  $E_m$  connects  $(a, i)$  to  $(b, j)$  if and only if  $b = (a+1) \pmod m$  and  $j$  is either identical to  $i$  or differs from it in the  $a$ th least significant bit.

On the butterfly we use a version of the three-phase scheme for transmitting a message from  $u = (a, i)$  to  $v = (b, j)$ . To do this we need only specify the out- and

in-sets and the three sets of paths. All messages are sent only through forward links (i.e., from a node  $(c, k)$  to a node  $((c+1) \pmod m, l)$  but not in the other direction). We take  $s = 2^m$  and define the sets  $\Gamma$  by  $\Gamma_{in}((c, k)) = \Gamma_{out}((c, k)) = \{(c, l) \mid 0 \leq l \leq 2^m - 1\}$ .

The paths are defined as follows. The forward edges of any node  $u$  span a full binary tree of height  $m$  whose leaves are all the nodes in  $\Gamma_{out}(u)$ , and the paths are chosen according to this tree. The second-phase paths connect every node  $(a, k)$  in  $\Gamma_{out}(u)$  to  $(b, k)$  in  $\Gamma_{in}(v)$  by  $(b - a) \pmod m$  edges connecting every intermediate node  $(c, k)$  to  $((c+1) \pmod m, k)$ . The third-phase paths from  $\Gamma_{in}(u)$  to  $u$  are defined in a way similar to the first-phase paths, using the dual tree based on the backward edges. More specifically, the path from a node  $(c, k)$  is directed to its neighbor  $((c+1) \pmod m, l)$  with the  $c$ th bit of  $l$  matching that of  $j$ .

This completes the description of a three-phase transmission scheme for the butterfly network. We now analyze the resiliency of this scheme.

CLAIM 2.3.  $b(G_m, t) = O(t \log t)$ .

*Proof.* Let us measure the amount of “damage” that a faulty processor  $p$  can cause to correct senders in the first phase. Keeping  $p$  fixed and looking at all possible senders  $u$  whose paths to  $\Gamma_{out}(u)$  contain  $p$  we see that  $p$  can block at most  $1/2^i$  of the outbound paths for its  $2^i$  “distance  $i$ ” neighbors. Summing up to distance  $\log 16t$ , the total damage caused by  $t$  faulty processors (measured in “number of dominated paths,” or “number of destroyed copies”) is

$$t \left( 2^{\frac{1}{2}} + 4^{\frac{1}{4}} + \dots + 2^{\log 16t} \frac{1}{2^{\log 16t}} \right) 2^m = t (\log 16t) 2^m.$$

The number of nodes that might lose  $\frac{1}{16}$  or more of their paths due to interruptions of distance  $\log 16t$  or less is therefore bounded by  $t \log 16t (2^m / (2^m / 16)) = 16t \log 16t$ . Now, let  $u$  be any processor for which less than  $\frac{1}{16}$  of the paths from  $u$  to  $\Gamma_{out}(u)$  contain a faulty processor at distance  $\log 16t$  or less from  $u$ . We claim  $u$  cannot be out-bad. This would imply that the number of out-bad nodes is also bounded by  $16t \log 16t$ . To prove the claim, note that if less than  $\frac{1}{16}$  of  $u$ 's out-bound paths are corrupted at distance  $\log 16t$  or less, then even if we assume the existence of up to  $t$  distinct faulty processes at distance  $\log 16t$  or more from  $u$ , these faults damage at most  $t 2^m / 2^{\log 16t} = 2^m / 16$  of the out-bound paths from  $u$ , so less than  $\frac{1}{8}$  of the paths from  $u$  to  $\Gamma_{out}(u)$  are corrupted, and by definition  $u$  is not out-bad. The analysis for the in-bad nodes is identical. Thus,  $b(G_m, t) \leq 32t \log 16t$ .  $\square$

COROLLARY 2.4.  $p(G_m, t) = O(t \log t)$ .

*Proof.* The proof is immediate from Claims 2.2 and 2.3.  $\square$

COROLLARY 2.5. For all  $m$ , there exists a  $t$ -resilient  $O(t \log t)$ -agreement algorithm running on  $G_m$  for  $t \leq cn / \log n$  for some constant  $c$ .

*Proof.* The proof is immediate from Claims 2.1 and 3.1.  $\square$

The next claim shows that this bound on almost-everywhere agreement is optimal for the butterfly network.

CLAIM 2.6.  $p(G_m, t) = \Omega(t \log t)$ .

*Proof.* Given  $t$ , let  $k = \lceil \log t \rceil$ , and choose the faulty nodes to be  $\{(i, j) \mid i = 0, k - 1, j = 0, \dots, 2^{k-1} - 1\}$ . This choice completely disconnects the set of nodes  $\{(a, b) \mid a = 0, \dots, k - 1, b = 0, \dots, 2^{k-1} - 1\}$ , which is of size  $\geq t \log t$ , from the remainder of the network.  $\square$

**2.4. Almost all regular graphs have good transmission schemes.** Let  $H(r, n)$  be the set of regular graphs of degree  $r$  and size  $n$ . Let  $h(r, n) = |H(r, n)| \geq (n/r)^{nr/2} [\text{Bo}]$ . We will show that for almost every  $r$ -regular graph  $G$ ,  $p(G, t) \in O(t^{1+\delta} \log t)$  for large

values of  $t$ . This is done by first showing that for all  $r \geq 5$  almost  $r$ -regular graphs  $G$  have three-phase transmission schemes for which  $b(G, t) \in O(t^{1+\delta} \log t)$  for  $t \in O(n^{1-\epsilon})$ , for appropriate  $\delta, \epsilon$ .

LEMMA 2.7. *For every  $r \geq 5$  there exists a constant  $\alpha = \alpha(r)$ ,  $0 < \alpha < 1$ , such that  $h(r, n)(1 - O(n^{-1/4}))$  of the  $h(r, n)$   $r$ -regular graphs have the following expansion property: Every subset of vertices  $U$  such that  $|U| \leq \alpha n$  has at least  $|U|(r - 3)$  neighbors outside  $U$ .*

*Proof.* We turn the set  $H(r, n)$  into a probability space by giving every graph  $G \in H(r, n)$  the same probability. A proof that a random graph  $G \in H(r, n)$  possesses the properties (1) and (2) with probability at least  $1 - O(n^{-1})$  implies Lemma 2.7.

The study of the probability space  $H(r, n)$  presents a special difficulty since no explicit representation of the set  $H(r, n)$  or a direct procedure for constructing a random element in  $H(r, n)$  are available. Recently, Bollobás [Bo] derived a method for studying  $H(r, n)$  through a related, more simple probability space  $\Phi(r, n)$ . Our proof is based on this new approach.

We start with a set,

$$W(r, n) = \{(v, i) | v = 1, \dots, n, i = 1, \dots, r\}.$$

A configuration  $F$  is a partition of  $W$  into  $rn/2$  pairs  $\{(v, i), (v', i')\}$ . Let  $\Phi(r, n)$  be the set of all configurations of  $W(r, n)$ . A configuration  $F$  defines an  $r$ -degree multigraph (with possible self loops), with vertex set  $V = \{1, \dots, n\}$  in which  $v$  is connected to  $v'$  if  $F$  includes a pair  $\{(v, i), (v', i')\}$  for some  $1 \leq i, i' \leq r$ .

Let  $\Psi(r, n)$  be the set of all configurations that define a proper  $r$ -regular graph. Bollobás was able to show that  $|\Psi(r, n)| \approx e^{(-r^2-1)/4} |\Phi(r, n)|$ . Thus, if we turn  $\Psi(r, n)$  and  $\Phi(r, n)$  into probability spaces by giving all their elements equal probability, an event that holds with probability  $1 - O(n^{-1})$  in  $\Psi(r, n)$  also holds with probability  $1 - O(n^{-1})$  in  $\Phi(r, n)$ . Furthermore, each graph  $G \in H(r, n)$  is obtained from precisely  $(r!)^n$  configurations. Thus, to prove that an event holds with probability  $1 - O(n^{-1})$  in  $H(r, n)$  it is enough to prove this result in the space  $\Phi(r, n)$ , where the analysis is substantially easier.

We first obtain a lower bound for the number of edges with at least one endpoint in a given set of vertices  $U$ . Let  $\beta = \frac{3}{2}$  and  $\alpha \leq e^{-2r} r^{-4}$ . Let  $E_1$  be the event: *There is a subset of vertices  $U$ ,  $|U| = k \leq \alpha n$ , that are endpoints of less than  $(r - \beta)k$  distinct edges.* The event  $E_1$  implies that at least  $\beta k$  edges in the graph connect two vertices of  $U$ . Thus,

$$\text{Prob}(E_1) \leq \sum_{k \leq \alpha n} \binom{n}{k} \binom{rk}{\beta k} \left(\frac{k}{n}\right)^{\beta k}.$$

Since

$$\binom{n}{k} \leq \left(\frac{ne}{k}\right)^k,$$

$$\text{Prob}(E_1) \leq \sum_{k \leq \alpha n} \left(\frac{en}{k}\right)^k (er)^{\beta k} \left(\frac{k}{n}\right)^{\beta k}.$$

For  $k \leq \log^3 n$ , each of the terms in the sum is bounded by  $O(n^{-1/2})$ ; thus, the sum of the first  $\log^3 n$  terms is bounded by  $O(n^{-1/3})$ . For  $k \geq \log^3 n$  each term in the sum is bounded by

$$(e^{(1+\beta)} r^\beta \alpha^{(\beta-1)})^k \leq n^{-2}$$

by the choice of  $\alpha$  and  $\beta$ . There are less than  $n$  terms in the sum; thus,  $\text{Prob}(E_1) = O(n^{-1/3})$ .

Let  $E_2$  be the event: *There exists a subset of vertices  $U, |U| \leq \alpha n$  that has less than  $|U|(r-3)$  neighbors outside  $U$ .* Conditioning on  $\bar{E}_1$ , the complement of the event  $E_1$ , we know that the vertices of  $U$  are endpoints of at least  $(r-\beta)|U|$  edges. We bound  $\text{Prob}(E_2|\bar{E}_1)$  by the probability that the other endpoints of all these edges are in a set of size  $(r-3)|U|$ . Thus,

$$\begin{aligned} \text{Prob}(E_2|\bar{E}_1) &\leq \sum_{k \leq \alpha n} \binom{n}{k} \binom{n}{(r-3)k} \left(\frac{k(r-3)}{n}\right)^{(r-\beta)k} \\ &\leq \left(\frac{ne}{k}\right)^k \left(\frac{ne}{(r-3)k}\right)^{(r-3)k} \left(\frac{(r-3)k}{n}\right)^{(r-\beta)k} \\ &\leq \left(e^{(r-2)}(r-3)^{(3-\beta)} \left(\frac{(r-3)k}{n}\right)^{(2-\beta)}\right)^k. \end{aligned}$$

Again we distinguish between two cases. If  $k \leq \log^3 n$  then since  $2-\beta = \frac{1}{2}$ , each term in the sum is bounded by  $O(n^{-1/2})$ . If  $k \geq \log^3 n$  then each term is bounded by  $(e^r r^{3/2} (r\alpha)^{1/2})^k$ , and thus, by the choice of  $\alpha$ , is bounded by  $O(n^{-2})$ .

Thus, the expansion property holds with probability  $\text{Prob}(\bar{E}_2) \geq \text{Prob}(\bar{E}_2|\bar{E}_1)$   $\text{Prob}(\bar{E}_1) \geq 1 - O(n^{-1/3})$ .  $\square$

**CLAIM 2.8.** *Let  $G = (V, E)$  be a graph satisfying the following expansion condition: Every subset of vertices  $U$  such that  $|U| \leq \alpha n$  has at least  $2|U|$  neighbors outside  $U$ . Then  $G$  has the following superconcentration property: Every two subsets of vertices  $U, W$  of size  $|U| = |W| \leq \alpha n$  are connected by  $|U|$  paths which are vertex-disjoint except that a node in  $U \cup W$ , in addition to being an endpoint, might appear as an intermediate node on one other path. item.*

*Proof.* We use the following extension of Menger’s theorem.

**THEOREM** [Br, p. 167]. *If  $a$  and  $b$  are two nonadjacent vertices of a simple graph  $G$ , then the maximum number of vertex-disjoint paths between  $a$  and  $b$  equals  $c_G(a, b) = \min_A |A|$ , where the minimization is taken over all vertex sets  $A$  that do not contain either  $a$  or  $b$  and whose removal disconnects  $a$  from  $b$ .*

Assume that our graph  $G = (V, E)$  contains a pair of sets  $U, W$  that do not satisfy the property. Let  $k = |U| = |W|$ . Consider the graph  $H = (V', E')$  obtained from  $G$  as follows: The set of vertices  $V' = V + U' + W' + \{s, t\}$ ,  $|U'| = |W'| = k$ . The vertex  $s$  is connected to all  $u', u' \in U'$  and the vertex  $t$  is connected to all  $w', w' \in W'$ . The connections between the vertices  $V$  are as in the graph  $G$ , and the vertices in  $U' \cup W'$  have the same connections as the corresponding vertices in  $U \cup W$ . Existence of  $k$  disjoint paths connecting  $s$  to  $t$  implies the superconcentration property. The above theorem implies that if there are no  $k$  vertex-disjoint paths connecting  $s$  to  $t$  then there is a set  $A, |A| < k$  that disconnects  $s$  from  $t$ .

We will prove that the existence of such a set in  $H$  violates the expansion property of  $G$ . Assume that a set  $A, |A| < k$  disconnects  $s$  from  $t$ . Let  $a = |U \cap A|$ ,  $b = |V \cap A|$  and  $c = |W \cap A|$ ,  $a + b + c < k$ . By the expansion property  $A \cap V$  can disconnect only a set of size  $\frac{1}{2}b$  from the rest of the set  $V$ . Thus, the set  $V - A$  has a connected component  $V'$  of size at least  $|V| - \frac{3}{2}b$ . By the expansion property the set  $U - A$  has at least  $2(k - a)$  neighbors in  $V$  and the set  $W - A$  at least  $2(k - c)$  neighbors in  $V$ . Since  $a + b + c < k$ ,  $2(k - a) > \frac{3}{2}b$  and  $2(k - c) > \frac{3}{2}b$ ; thus both  $U - A$  and  $W - A$  are connected to the connected component  $V'$  and there is a path connecting  $s$  to  $t$  that does not use the set  $A$ .  $\square$

**CLAIM 2.9.** *For all  $r \geq 5$ , for every graph  $G$  in  $H(r, n)$ , if  $G$  has the expansion and superconcentration properties, then there exists a three-phase transmission scheme for  $G$*

subject to  $p(G, t) \in O(t^{1+\delta} \log t)$  for  $t \in O(n^{1-\epsilon})$ , for some  $0 < \delta < \epsilon < 1$ ,  $\epsilon = \epsilon(r)$ , where  $\epsilon \rightarrow 0$  as  $r \rightarrow \infty$ .

*Proof.* In this proof all logarithms are to the base  $r-3$ . Let  $d = \lceil \log t \rceil$  and let  $\delta = \log((r-1)/(r-3))$ . For each vertex  $u$  in  $G$ , let  $D(u)$  denote the set of vertices in  $G$  at distance  $d+4$  from  $u$ . By the expansion property, and the fact that  $r-3 \geq 2$ ,  $|D(u)| \geq 16(r-3)^d$ . We choose  $\Gamma(u) = \Gamma_{\text{in}}(u) = \Gamma_{\text{out}}(u)$  to be an  $s = 9(r-3)^d \leq \alpha n$  element subset of  $D(u)$ . This leaves us free to use the same inbound paths as outbound paths. Specifically, we choose an arbitrary breadth-first search tree of  $G$  rooted at  $u$ , and use its branches to define paths between  $u$  and the elements of  $\Gamma(u)$ . Finally, by the superconcentration property there are  $s$  ‘‘almost vertex disjoint’’ paths between  $\Gamma(v)$  and  $\Gamma(w)$  for every two nodes  $v, w$  of  $G$ . For each pair of nodes we specify a particular set of such paths. This completes the specification of the transmission scheme.

It remains to analyze the damage that can be caused by the faulty processors and to show that it cannot be too large. Let  $u$  be an arbitrary correct processor and let  $p$  be a faulty processor at distance  $i$  from  $u$ . Then  $p$  can affect the paths from  $u$  to at most  $(r-1)^{d+4-i}$  elements of  $\Gamma(u)$ . At most  $r(r-1)^{i-1}$  vertices  $u$  are at distance  $i$  from  $p$ . Thus  $p$  can corrupt at most  $r(r-1)^{d+3}$  elements in sets  $\Gamma(u)$  for vertices  $u$  at distance  $i$  from  $p$ . Summing up for all distances  $i \leq d$  and all faulty processors we see that the faulty processors can corrupt at most  $tdr(r-1)^{d+3}$  paths in total. Therefore, assuming that this damage is distributed in a worst-case fashion for the transmission scheme (optimal for an adversary),

$$\begin{aligned} b(G, t) &\leq \frac{t \log tr(r-1)^{d+3}}{9(r-3)^d/8} \\ &= \frac{8r(r-1)^3}{9} t^{1+\delta} \log t. \end{aligned}$$

Finally note that by the remark following Claim 2.2 and the fact that  $s/8 > (r-3)^d \geq t$ , the same bound applies for  $p(G, t)$ . Note also that this bound limits the results to  $t \in O(n^{1-\epsilon})$  for an appropriate  $\delta < \epsilon < 1$ .  $\square$

**COROLLARY 2.10.** *For every  $r \geq 5$  almost every  $r$ -regular graph has a  $t$ -resilient  $O(t^{1+\delta} \log t)$ -agreement algorithm, for  $t \in O(n^{1-\epsilon})$ , for some  $0 < \delta < \epsilon < 1$ .  $\square$*

**2.5. Results for faults of restricted severity.** In this section we briefly mention our results for the failstop, omission, and authenticated Byzantine models. As with the unrestricted Byzantine failures (without authentication) our approach is to devise an appropriate transmission scheme. However, in the failure models considered here two correct processors can communicate provided they are connected by even one uncorrupted path. Thus the poor processors will be only those completely disconnected from the major portion of the graph. Now, if a graph has good expansion properties, then it is impossible for  $t$  processors to disconnect more than  $O(t)$  processors. Combining this with the fact that almost all regular graphs of degree at least 5 enjoy such properties (Lemma 2.7), we obtain the following result.

**THEOREM 2.11.** *Assuming an authentication mechanism or nonmalicious faults (failstop, omission), for all  $r \geq 5$  there exists a constant  $c = c(r)$  such that for all  $t \leq cn$ ,*

- (1) *almost all  $r$ -regular graphs have transmission schemes in which there are at most  $O(t)$  poor processors, and*
- (2) *almost all  $r$ -regular graphs admit a  $t$ -resilient algorithm for  $O(t)$  agreement.*

**3. Byzantine agreement on compressor graphs.** Until this point we have considered only techniques based on simulations of complete-network algorithms on our bounded

degree networks. A different approach to reaching agreement in a bounded degree network can be based on distributed protocols of a local nature. In this section we consider such an approach and analyze its behavior on a compressor graph.

A graph  $G$  is a  $\theta$ -compressor for constant  $\theta < 1$ , if for every subset of vertices  $U$  of size  $|U| \leq \theta n$ , the set  $\{v|v \text{ has at least half of its neighbors in } U\}$  has cardinality at most  $|U|/2$ . An explicit construction of compressors of a (very high) fixed degree (about  $8^{17}$ ) has been shown by Pippenger [P]. Using a recent result of Lubotzky, Phillips, and Sarnak [LPS] the degree of explicitly constructed compressors can be reduced to about 64.

Our interest in compressors follows from our ability to use the compression property to “sharpen” almost-everywhere agreement to achieve  $O(t)$  agreement (which is asymptotically optimal, as  $t$  faults can always completely disconnect  $\Omega(t)$  correct processors). Ultimately, our approach will be to use one of the simulation techniques of the previous section to obtain  $p(G, t)$  agreement and then to sharpen this to  $O(t)$  agreement by using the compressor properties of the graph.

We consider the following scheme for agreement, called the *compression procedure*. This procedure is based on simple rounds of the following form. In every round, every correct processor

- (1) sends its value to all its neighbors,
- (2) receives the values of all its neighbors, and
- (3) chooses as its new value the value held by a majority of its neighbors.

The procedure terminates after some fixed number of rounds.

Let  $G = (V, E)$  be a  $\theta$ -compressor of size  $n$ , and let  $T$  be the set of faulty processors in the network,  $|T| \leq t \leq \theta n/2$ .

LEMMA 3.1. *If there are  $(1 - \theta)n$  correct processors which share the same initial value  $x$ , then after applying the compression procedure for  $\log n$  rounds, at most  $t + 1$  correct processors will have a value different from  $x$ .*

*Proof.* Let  $V_k$  denote the set of correct processors whose value differs from  $v$  after  $k$  rounds of the majority procedure. Let  $A_k = ((2^k - 1)t + |V_0|)/2^k$ , for  $k \geq 0$ . We begin by bounding  $A_k$  in terms of  $n$  and  $t$ .

CLAIM 3.1.1. *For every  $k \geq 0$ ,  $A_k + t \leq \theta n$ .*

*Proof.* There are two cases. If  $|V_0| \leq t$  then for every  $k \geq 0$ ,  $A_k \leq ((2^k - 1)t + t)/2^k = t$  and we are done by the assumption on  $t$ . If  $|V_0| > t$  we prove the claim by induction on  $k$ . For  $k = 0$ ,  $A_k = |V_0|$ , and  $A_k + t = |V_0| + t \leq \theta n$  by the assumption on the initial state of the network. Assume the claim inductively for  $k$ . Since  $A_{k+1} = A_k + (t - |V_0|)/2^{k+1} < A_k$  we are done.  $\square$

We next bound  $V_k$  in terms of  $A_k$ .

CLAIM 3.1.2. *For every  $k \geq 0$ ,  $|V_k| \leq A_k$ .*

*Proof.* We prove the claim by induction on  $k$ . For  $k = 0$  the claim is trivial. Assume the claim for  $k$  inductively, and consider the case of  $k + 1$ . The set  $V_{k+1}$  will contain all correct processors whose majority of neighbors resides in  $T \cup V_k$ . By the inductive hypothesis this set has cardinality at most  $t + A_k$ , which, by Claim 3.1.1, is at most  $\theta n$ . We may therefore apply the compression property to obtain  $|V_{k+1}| \leq (t + A_k)/2 = A_{k+1}$ .

Taking  $k = \log n$  we obtain  $|V_k| \leq A_k \leq t + 1$ . This completes the proof of Claim 3.1.1.  $\square$

LEMMA 3.2. *For every  $r \geq 5$  there exists a constant  $0 < \theta < 1$ , such that  $h(r, n)(1 - O(n^{-1}))$  of the  $h(r, n)$   $r$ -regular graphs have the compression property for all subsets  $U$  of size  $|U| \leq \theta n$ .*

*Proof.* We use the same setting and notation as in the proof of Lemma 2.7. Define the event  $E_3$ : *There exists a set  $U$ ,  $|U| \leq \theta n$  and a set  $W$ ,  $|W| \leq |U|/2$  such that half of*



the edges incident on  $W$  have one endpoint in  $W$  and the other in the set  $U$ . Clearly if a graph has a set  $U$  that violates the compression property it is included in the event  $E_3$ . To estimate the probability of the event  $E_3$  we condition first on the complement of the event  $E_1$  (defined in the proof of Lemma 2.7). Under the condition  $\bar{E}_1$ , the vertices of the set  $W$  are incident to at least  $(r-\beta)|W|$  edges, and to satisfy the condition  $E_3$  at least half of these edges have their other endpoint in the set  $U$ :

$$\begin{aligned} \text{Prob}(E_3|\bar{E}_1) &\leq \sum_{k \leq \theta_n} \binom{n}{k} \binom{n}{k/2} \binom{(r-\beta)k}{((r-\beta)/2)k} \left(\frac{rk}{rn}\right)^{((r-\beta)/2)k} \\ &\leq \left(\frac{ne}{k}\right)^k \left(\frac{2ne}{k}\right)^{k/2} \left(\frac{(r-\beta)k}{((r-\beta)/2)k}\right)^{(r-\beta)k} \left(\frac{k}{n}\right)^{((r-\beta)/2)k} \\ &\leq \left(e^{3k/2} 2^{(r-\beta+1/2)} \left(\frac{k}{n}\right)^{(r-\beta-3/2)}\right)^k \end{aligned}$$

For  $k \leq \log^3 n$  each term in the sum is bounded by  $O(n^{-1})$ . For  $k \geq \log^3 n$  each term is bounded by  $(e^{r+1/2} \theta^{r-3/2})^k \leq n^{-2}$ , since  $\beta = \frac{3}{2}$  and  $e^{-2}$ . Thus,  $\text{Prob}(\bar{E}_3) \geq \text{Prob}(\bar{E}_3|\bar{E}_1) \text{Prob}(\bar{E}_1) \geq 1 - O(n^{-1})$ .  $\square$

Combining this with the transmission schemes described in § 2.4, we prove the following.

**COROLLARY 3.3.** *For every  $r \geq 5$ ,  $h(r, n)(1 - O(n^{-1}))$  of the  $h(r, n)$   $r$ -regular graphs admit  $O(t)$  agreement, where  $t \in O(n^{1-\epsilon})$  for some  $0 < \epsilon < 1$ .*

*Proof.* By Lemmas 2.7 and 3.2 almost every  $r$ -regular graph enjoys the expansion, compression, and superconcentration properties. Let  $G$  be any such graph. Using the three-phase transmission scheme of § 2.4 to simulate any standard Byzantine agreement designed for the complete network, we achieve  $O(t^{1+\delta} \log t)$  agreement, so at the end of the simulation at least  $1 - \theta$  of the correct processors agree on a value. Since this satisfies the conditions of Lemma 3.1 we may apply that lemma to obtain the desired  $O(t)$ -agreement.  $\square$

**COROLLARY 3.4.** *There exists a family of degree 9 networks that admit  $t$ -resilient  $O(t)$  agreement for  $t \leq \alpha n / \log n$ , for some constant  $\alpha$ .*

*Proof.* Since a butterfly has degree 4 and a compressor has degree 5, there exists a graph of degree 9 which is both a butterfly and a compressor. We first use the result of Corollary 2.5 to get an  $O(t \log t)$  agreement on the butterfly graph, then we use the compressor to sharpen this agreement.  $\square$

**4. Almost-everywhere agreement on networks of unbounded degree.** On a complete network,  $t$ -resilient agreement protocols exist for all  $t < n/3$ . In contrast, our previous algorithms can tolerate only  $O(n/\log n)$  failures. The situation for higher values of  $t$  on bounded degree networks remains open. In this section we present an approach for reaching agreement on networks of unbounded but small degree (which is nevertheless much lower than  $n$  or  $t$ ).

**THEOREM 4.1.** *For every  $\epsilon > 0$  there exist a constant  $c = c(\epsilon)$ , graphs  $G$  of degree  $O(n^\epsilon)$  and  $t$ -resilient  $O(t)$ -agreement algorithms for  $t \leq cn$ .*

*Proof.* For simplicity, we discuss only the case  $\epsilon = \frac{1}{2}$ . Let  $G = (V, E)$  be defined as follows.  $V$  contains  $n = m^2$  nodes, partitioned into  $m$  pairwise disjoint committees  $A_i, 1 \leq i \leq m$ , each of size  $m$ . Each committee forms a clique. In addition, every two committees  $A_i$  and  $A_j$  are connected by  $m$  edges which form a matching between them. On top of that, we optionally add edges so as to make the graph into a  $\theta$ -compressor. As discussed earlier, this property can be achieved by a graph of bounded degree. Thus the degree of any node in the resulting graph will be  $O(n^{1/2})$ .

Let  $P$  be any  $t$ -resilient protocol for the traditional Byzantine agreement problem on a complete network [DFFLS], [PSL], [LSP], guaranteeing agreement whenever the number of faulty processors is smaller than  $n/3$ .

The algorithm we give consists of two main stages plus an additional optional stage which uses the compressor edges. In the first stage, each committee privately runs  $P$  (appropriately scaled) among its own members. We say a committee is *good* if fewer than  $\frac{1}{4}$  of its members are faulty. Clearly the good committees will reach agreement among themselves. In the next stage, each committee acts as a single processor and the whole graph is viewed as a clique of  $m$  (composite) processors which simulate an execution of  $P$ . The final optional stage is a compression step, as described in § 3.

The communication between committees is performed by a special protocol COM which guarantees that processors within a good committee behave uniformly. According to this protocol a committee  $A$  sends a message  $x$  to  $B$  as follows. Every processor in  $A$  sends  $x$  to its neighbor in  $B$ . Every node in  $B$  now executes the following two steps:

- (1) Exchanges with all other processors in  $B$  the value received from  $A$ ;
- (2) Adopts as its updated value of  $x$  that message received from the majority of the nodes in  $B$  (ties are broken arbitrarily).

Finally, the nodes of  $B$  run  $P$  on the adopted values for  $x$ . The value agreed upon by the end of the run is taken to be the message sent by  $A$ .

We now analyze the behavior of this algorithm. Call a committee  $A_i$  *good* if  $t_i < m/4$ , where  $t_i$  is the number of actual faults in  $A_i$ , and *bad* otherwise.

CLAIM 4.2. *In every communication step (i.e., every execution of COM), if the committee  $A_i$  receiving a message is good then all the good processors in it will agree on the same received value.*

*Proof.* This follows from the properties of  $P$  and the fact that  $t_i < m/3$ .  $\square$

COROLLARY 4.3. *In every round of the second stage of the main algorithm, the elements of a good committee will have the same view and will send out the same messages.*  $\square$

CLAIM 4.4. *In every communication step COM, if the receiving committee  $A_i$  and the sending committee  $A_j$  are good, then all the good processors in  $A_j$  will agree on the value that was sent by (the good processors of)  $A_i$ .*

*Proof.* Since all the good processors of  $A_i$  send the same message  $X$ , and  $t_i, t_j < m/4$ , the number of good processes in  $A_j$  receiving  $X$  is at more than  $m/2$ . Therefore after the majority step of COM, all the correct processes in  $A_j$  will have the same value  $X$ . By the unanimity property of the algorithm  $P$ , all the correct processors in  $A_j$  will eventually agree on  $X$ .  $\square$

CLAIM 4.5. *If  $t < n/12$  then in the end of the basic main algorithm (without the final compression step), agreement will be reached between most of the good processors; at most  $3t$  good processors will reach a wrong decision.*

*Proof.* Denote the number of bad committees by  $b$ . Clearly  $b \leq 4t/m$  (otherwise, since every bad committee has at least  $m/4$  faults, there are more than  $t$  faults overall). Hence  $b < (4/m)(n/12) = m/3$ . Therefore the main algorithm will guarantee that all the good committees will reach agreement, where in every good committee all good processors will have the right value. The “confused” good processors are at most those in the bad committees, whose number is bounded by  $b(m - m/4) \leq (3m/4)(4t/m) = 3t$ .  $\square$

CLAIM 4.6. *If  $t < \theta n/4$  then in the end of the algorithm (with the final compression step), agreement will be reached between most of the good processors; at most  $t + 1$  good processors will reach a wrong decision.*

*Proof.* Upon termination of the basic algorithm, the number  $l$  of good processors holding a wrong value is bounded by  $3t$ , as shown in the previous claim. Thus  $t < \theta n/2$ , and  $l + t \leq 4t < \theta n$ , and by Claim 2.3, after the compression procedure at most  $t + 1$  good processors will have a wrong value.  $\square$

This algorithm can be naturally extended for graphs of degree  $O(n^{1/k})$  for  $k = 3, 4, \dots$ , by dividing the graph into committees of committees of committees, etc., with an appropriate definition of a good committee on every level. We omit the details.

**5. Combinatorial characterization of fault-tolerant networks.** There is a necessary and sufficient condition for a system to be  $t$ -resilient in terms of the combinatorial properties of its communication graph. In this section we derive a combinatorial characterization of graphs that admit  $p(t)$  agreement.

For any agreement protocol  $P$  let  $P(T)$  be any maximal set of correct processors that always reach agreement under the protocol  $P$ , independent of the behavior of the processors in  $T$  (thought of as faulty).

**THEOREM 5.1.** *Let  $G$  be a communication graph, let  $\{T_i\}_{i=1}^k$  be the family of all possible sets of faulty processors in  $G$ , and let  $\{A(T_i)\}_{i=1}^k$  be a family of sets of processors in  $G$ . There exists a protocol  $P$  subject to  $P(T_i) = A(T_i)$  for  $i = 1, \dots, k$ , if and only if for every pair of processors  $u, v \in A(T_i) \cap A(T_j)$ , the set  $T_i \cup T_j$  does not disconnect  $u$  from  $v$  in  $G$ .*

*Comment.* Note that if for all sets  $T$  of at most  $t$  processors  $A(T)$  is the complement of  $T$ , then our characterization yields the  $2t + 1$  connectivity condition for the traditional Byzantine agreement problem.

*Sketch of proof.* To prove necessity, we show that if there exist sets  $T_i$  and  $T_j$  which jointly (but not individually) can disconnect correct processors  $u$  and  $v$ , then there exist two scenarios, indistinguishable to  $v$ , such that in one scenario  $T_i$  is faulty and  $u$  decides on a value  $a$ , while in the second scenario  $T_j$  is faulty and  $u$  decides on a value  $b$ .

We prove sufficiency by constructing an algorithm with the desired properties. We briefly describe a few points of our construction.

A processor  $u$  transmits a message to  $v$  by sending it along all simple paths from  $u$  to  $v$ . As the message passes from site to site, each processor appends the name of the processor from which the message was received. Thus, a message that passes through faulty processors contains the name of at least one such processor (the last one). The processor  $v$  searches for a set  $T_i$  such that all the messages not passing through this set are consistent and both  $u$  and  $v$  are in  $A(T_i)$ . Let  $T$  be the set of faulty processors in a particular execution of our algorithm. If  $u$  and  $v$  are in  $A(T)$  then  $v$  will try this set and extract the correct value. Crucial to our algorithm is that  $v$  will never extract an incorrect value. This is because by assumption, for all other relevant sets  $T_j$ ,  $T \cup T_j$  will not disconnect  $u$  from  $v$ . Thus,  $v$  receives the message via at least one fault-free path. Therefore, the faulty processors can at most create an inconsistent set of values, from which  $v$  extracts nothing.  $\square$

**6. Conclusions and open problems.** We propose a new paradigm for fault-tolerant computing, in which correctness conditions are relaxed by accepting the loss of a small number of correct processors.

This paper concentrates on demonstrating the feasibility of this new approach and its advantages. In particular, we show that a simple distributed algorithm can achieve agreement among almost all the correct processors in circumstances in which Byzantine agreement is not achievable.

Many problems remain open for further investigation. We mention the following:

(1) What is the minimum number of steps and messages required to achieve  $X$  agreement? Can the efficiency of the algorithms presented in this paper be significantly improved?

(2) The algorithms presented in this work assume that all the processors know the topology of the communication network and the communication schemes used by all other processors. Is this requirement essential for achieving  $X$  agreement?

(3) Can  $X$  agreement be achieved on a network of bounded degree in the presence of more than  $n/\log n$  (malicious) faults? Note that in this case most of the communication paths between most pairs of processors include at least one faulty processor.

**Acknowledgment.** We wish to thank M. Ajtai for helpful discussions.

#### REFERENCES

- [Br] C. BERGE, *Graphs and Hypergraphs*, second edition, North-Holland, Amsterdam, 1979.
- [Bo] B. BOLLOBÁS, *Random graphs*, in *Combinatorics*, London Mathematical Society Lecture Notes 52, Cambridge University Press, Cambridge, 1981, pp. 80–102.
- [D] D. DOLEV, *The Byzantine generals strike again*, *J. Algorithms*, 3 (1982), pp. 14–30.
- [DFFLS] D. DOLEV, M. J. FISCHER, R. FOWLER, N. A. LYNCH, AND R. STRONG, *An efficient algorithm for Byzantine agreement without authentication*, *Inform. and Control*, 52 (1982), pp. 256–274.
- [F] M. J. FISCHER, *The consensus problem in unreliable distributed systems*, Tech. Report 273, Dept. of Computer Science, Yale University, New Haven, CT, 1983.
- [GG] O. GABBER AND Z. GALIL, *Explicit construction of linear-sized superconcentrators*, *J. Comput. System Sci.*, 22 (1981), pp. 407–420.
- [H] V. HADZILACOS, *Issues of fault tolerance in concurrent computations*, Ph.D. thesis, Harvard University, Cambridge, MA, 1984.
- [LPS] A. LUBOTZKY, R. PHILLIPS, AND P. SARNAK, *Ramanujan conjecture and explicit constructions of expanders*, in *Proc. 18th Annual ACM Symposium on Theory of Computing*, 1986, pp. 240–246.
- [LSP] L. LAMPORT, R. SHOSTAK, AND M. PEASE, *The Byzantine generals problem*, *ACM Trans. Prog. Lang. Syst.*, 4 (1982), pp. 382–401.
- [PSL] M. PEASE, R. SHOSTAK, AND L. LAMPORT, *Reaching agreement in the presence of faults*, *J. ACM*, 27 (1980), pp. 228–234.
- [P] N. PIPPENGER, *On networks of noisy gates*, in *Proc. 26th Annual ACM Symposium on Foundations of Computer Science*, 1985, pp. 31–38.
- [U] J. D. ULLMAN, *Computational Aspects of VLSI*, Computer Science Press, Potomac, MD, 1984.

## NATURAL SELF-REDUCIBLE SETS\*

ALAN L. SELMAN†

**Abstract.** To every set  $A$  in NP we associate a (collection of) natural self-reducible set(s). We prove that every disjunctive-self-reducible set in NP is  $\cong_a^P$ -equivalent to one of these natural self-reducible sets and we show that if certain questions about these sets could be answered, then several significant open questions about the fine structure of NP would be solved.

**Key words.** NP, self-reducible, P-selective, P-separable, reducibilities, P-isomorphic

**AMS(MOS) subject classification.** 68Q15

**1. Introduction.** All “natural” NP-complete sets are self-reducible but it is not known whether *all* NP-complete sets are self-reducible. Self-reducibility is that property of sets which, for example, makes the problem of finding a satisfying assignment for a formula of propositional logic polynomially Turing equivalent to the problem of deciding whether a formula has a satisfying assignment. The theorems of Berman [Ber78] and Mahaney [Mah82] use the self-reducibility property extensively.

Our motivation is to reveal the mathematical structure of NP. Although hundreds of NP-complete problems are known, even elementary matters such as whether the union of every pair of disjoint NP-complete sets is NP-complete remains an open question. We suggest examination of self-reducible sets in order to gain insight into questions of this kind. To every set  $A$  in NP we associate a (collection of) natural self-reducible set(s), prefix  $(R_A)$ . Prefix  $(R_A)$  may be thought of as the set of all prefixes of accepting computations of some nondeterministic polynomial time-bounded Turing machine that accepts  $A$ . If we could answer certain questions about these sets, then several significant open problems about the fine structure of NP would be solved. Along the way some fundamental facts about self-reducible sets are obtained. We prove that every disjunctive-self-reducible set  $A$  is  $\cong_a^P$ -equivalent to an appropriate prefix  $(R_A)$ . (An immediate consequence is the known result that every disjunctive-self-reducible set belongs to NP [Ko83].)

**2. Preliminaries.** All languages considered are over the finite alphabet  $\Sigma = \{0, 1\}$ , and  $\#$  is a symbol that is not in the alphabet  $\{0, 1\}$ . The reader is assumed to be familiar with standard concepts and notation of polynomial time-bounded complexity theory [HU79], [GJ79], but a number of specialized notions are described below.

Let  $E = \cup_c \text{DTIME}(2^{cn})$ , and let  $NE = \cup_c \text{NTIME}(2^{cn})$ . It is not known whether  $E = NE$ , but it is a well-known fact that  $E \neq NE$  implies  $P \neq NP$  [Boo74a], [JS74]. A *tally language* is a subset of  $\{1\}^*$ . A set  $S$  is *sparse* if there is a polynomial  $p$  such that for every positive integer  $n$ ,

$$\|S \cap \{0, 1\}^n\| \leq p(n).$$

Obviously, tally languages are sparse sets. It is known that  $E = NE$  if and only if every tally language in NP belongs to P [Boo74b], [HHH74], and Hartmanis [Har83] has recently shown that  $E = NE$  if and only if every sparse set in NP belongs to P. Also, it is known that no NP-complete sets are sparse unless  $P = NP$  [Mah82].

---

\* Received by the editors August 4, 1986; accepted for publication (in revised form) November 2, 1987. This research was supported in part by the National Science Foundation under grant DCR86-96082 and by the National Security Agency under grant MDA904-87-H-2020.

† College of Computer Science, Northeastern University, Boston, Massachusetts 02115.

**2.1. Reducibility concepts.** We use the notation  $\leq_T^P$  and  $\leq_m^P$  to denote polynomial time-bounded Turing and many-one reducibility, respectively. For an arbitrary polynomial time-bounded reducibility  $\leq_r^P$ , a set  $L$  is  $\leq_r^P$ -hard for complexity class  $C$  if every set in  $C$  is  $\leq_r^P$ -reducible to  $L$ , and a set  $L$  is  $\leq_r^P$ -complete for  $C$  if  $L$  is  $\leq_r^P$ -hard for  $C$  and  $L \in C$ . By consensus, “NP-complete” means  $\leq_m^P$ -complete for NP, and “NP-hard” means  $\leq_T^P$ -hard for NP.

It is known that  $\leq_T^P$  and  $\leq_m^P$  are not identical on the class of recursive sets [LLS75]. If  $P \neq NP$ , then there exist NP-hard sets that are not  $\leq_m^P$ -hard [Lon82]. However, it is not known whether there exist  $\leq_T^P$ -complete sets in NP that are not already NP-complete.

A number of specialized polynomial time-bounded reducibilities have been studied [LLS75]; of these we need to know disjunctive reducibility,  $\leq_d^P$ . A simple characterization follows:  $A \leq_d^P B$  if and only if there is a polynomial-time computable function

$$f: \{0, 1\}^* \mapsto (\neq \{0, 1\}^*)^*$$

such that for each input word  $x$ ,  $x \in A$  if and only if  $\{y_1, \dots, y_k\} \cap B \neq \emptyset$ , where  $f(x) = \#y_1 \dots \#y_k$ .

Reference will be made to truth-table and positive truth-table reducibilities also. The uninitiated reader need only be aware of the following inclusions:

$$\leq_m^P \subseteq \leq_d^P \subseteq \leq_{pt}^P \subseteq \leq_u^P \subseteq \leq_T^P.$$

**2.2. Self-reductions.** Self-reducibility notions have appeared in the literature in a variety of guises, but the most useful definition is due to Meyer and Paterson [MP79]. Their definition follows.

DEFINITION 1. A partial order  $<$  on  $\Sigma^*$  is OK if and only if

(i) every strictly decreasing chain is finite, and there is a polynomial  $p$  such that every finite  $<$ -decreasing chain is shorter than  $p$  of the length of its maximum element, and

(ii)  $x < y$  implies  $|x| \leq q(|y|)$ , for some polynomial  $q$ , and all  $x$  and  $y$  in  $\Sigma^*$ .

DEFINITION 2. A set  $A$  is self-reducible if and only if there is an OK partial order and a query machine  $M$  such that  $M$  accepts  $A$  in polynomial time with oracle  $A$ , and  $M$  is restricted so that on any input  $x$  in  $\Sigma^*$ ,  $M$  asks its oracle only about words strictly less than  $x$  in the partial order.

Our concern is with sets that are  $d$ -self-reducible, i.e., the query machine  $M$  also provides a  $\leq_d^P$ -reduction. This means that on every input word  $x$ , the query machine  $M$  either

(i) decides membership of  $x$  in  $A$  in polynomial time without queries to the oracle, or

(ii) computes a set of queries  $\{z_1, \dots, z_k\}$  in polynomial time so that

$$x \in A \Leftrightarrow \{z_1, \dots, z_k\} \cap A \neq \emptyset.$$

All of the known NP-complete combinatorial problems are  $d$ -self-reducible, and Meyer and Paterson [MP79] and Schnorr [Sch76] demonstrated  $d$ -self-reducibility of several problems in NP that are not known to be either in P or NP-complete.

**3. Natural self-reducible sets.** Given a set  $A \in NP$  there is an associated polynomial-time recognizable relation  $R_A$  and an associated polynomial  $p_A$  such that

$$(1) \quad A = \{x \mid \exists y |y| = p_A(|x|) \text{ and } R_A(x, y)\}.$$

A string  $y$  of length  $p_A(|x|)$  such that  $R_A(x, y)$  holds is called a *proof* that  $x \in A$ . We will be performing various manipulations on these proofs, and so for purely technical reasons it is convenient to assume that associated polynomial-time relations also satisfy the following condition:

$$(2) \quad \forall z_{|z| \geq p_A(|x|)} [R_A(x, z) \leftrightarrow R_A(x, y), \text{ where } y \text{ is the prefix of } z \text{ of length } p_A(|x|)].$$

We make this assumption without loss of generality because every polynomial-time recognizable relation  $R_A$  that satisfies property (1) can be extended so that it also satisfies condition (2).

DEFINITION 3.  $\text{Prefix}(R_A) = \{x \# u \mid \exists v \mid uv = p(|x|) \text{ and } R_A(x, uv)\}$ .

PROPOSITION 1. (i)  $\text{Prefix}(R_A)$  is *d-self-reducible*. (Namely,  $x \# u \in \text{Prefix}(R_A) \leftrightarrow (x \# u0 \in \text{Prefix}(R_A) \text{ or } x \# u1 \in \text{Prefix}(R_A)) \text{ or } R_A(x, u)$ .)

(ii)  $A \leq_m^P \text{Prefix}(R_A)$ . (Namely,  $x \in A \leftrightarrow x \# \in \text{Prefix}(R_A)$ .)

(iii) If  $A$  is NP-complete, then  $\text{Prefix}(R_A)$  is NP-complete.

(iv) If  $A$  is  $\leq_T^P$ -complete for NP, then  $\text{Prefix}(R_A)$  is  $\leq_T^P$ -complete for NP.

A number of open questions about  $\text{Prefix}(R_A)$  will be raised. From the possible answers to these questions, inferences will be drawn about structural properties of NP.

**3.1. Disjoint pairs of NP-sets.** The following are two well-known results about disjoint pairs of recursively enumerable sets (cf. [Sho71]).

(1) For every nonrecursive recursively enumerable set  $A$  there exist sets  $B_1$  and  $B_2$  such that  $A = B_1 \cup B_2$ ,  $B_1 \cap B_2 = \emptyset$  and  $B_1$  and  $B_2$  are both recursively enumerable and nonrecursive.

(2) If  $B_1$  and  $B_2$  are disjoint recursively enumerable sets, then  $\mathbf{d}(B_1 \cup B_2) = \mathbf{d}(B_1) \cup \mathbf{d}(B_2)$ . In particular,  $B_1 \leq_T B_1 \cup B_2$  and  $B_2 \leq_T B_1 \cup B_2$ .

Now the analogue of result (1) to polynomial-time complexity holds, for Ladner [Lad75] has shown that for every set  $A$  in NP-P, there exist disjoint sets  $B_1$  and  $B_2$  in NP-P such that  $A = B_1 \cup B_2$ .

Consider the proof of result (2): To decide  $B_1$  with oracle  $B_1 \cup B_2$ , on input  $x$ , if  $x \notin B_1 \cup B_2$ , then reject, else simultaneously enumerate  $B_1$  and  $B_2$  until  $x$  is output. Accept if the enumeration of  $B_1$  outputs  $x$  and reject if the enumeration of  $B_2$  outputs  $x$ .

The proof just given certainly suggests that the analogue of result (2) is false. More to the point perhaps, if  $B_1 \leq_T^P B_1 \cup B_2$  for every disjoint pair of sets  $B_1$  and  $B_2$  in NP, then  $\text{NP} \cap \text{co-NP} = \text{P}$ . (To see this assertion, let  $B_1 \in \text{NP} \cap \text{co-NP}$ , let  $B_2 = \overline{B_1}$ , and observe that  $B_1 \cup B_2 = \Sigma^*$  is in P.) Since it is unlikely that  $\text{NP} \cap \text{co-NP} = \text{P}$ , it seems reasonable to conjecture (assuming  $\text{P} \neq \text{NP}$ ) that there exist disjoint sets  $B_1$  and  $B_2$  in NP-P such that  $B_1 \cup B_2 \in \text{NP-P}$  and  $B_1 \not\leq_T^P B_1 \cup B_2$ . Neither Ladner's proof [Lad75] of the analogue of result (1) nor Schöning's elegant treatment [Sch82] yields a proof of this conjecture, for their techniques give  $B_1 \leq_m^P B_1 \cup B_2$  and  $B_2 \leq_m^P B_1 \cup B_2$ .

Let  $A$  and  $B$  be disjoint sets in NP-P. Let  $p_A$  and  $p_B$  be associated polynomials, and let  $R_A$  and  $R_B$  be any polynomial-time recognizable associated relations. Let  $p_{A \cup B} = \max\{p_A, p_B\}$ . Since  $R_A$  and  $R_B$  satisfy both of the defining conditions (1) and (2), it follows readily that  $A \cup B = \{x \mid \exists y \mid y = p_{A \cup B}(|x|) \text{ and } [R_A(x, y) \vee R_B(x, y)]\}$ . Furthermore,  $[R_A(x, y) \vee R_B(x, y)]$  satisfies condition (2), where  $p_{A \cup B}$  is the appropriate bound. Therefore,  $[R_A(x, y) \vee R_B(x, y)]$  is an associated relation for  $A \cup B$ . Hence, it is consistent with our notation to define  $R_{A \cup B}$  to be the relation  $R_A \cup R_B$ .

LEMMA 2.  $\text{Prefix}(R_A) \leq_T^P \text{Prefix}(R_{A \cup B})$ , if  $A \cap B = \emptyset$ .

*Proof.* The following procedure gives the reduction.

```

input  $x \# u$ ;
if  $x \# u \notin \text{prefix}(R_{A \cup B})$ 
  then reject
else begin
   $y := u$ ;
  while  $\neg R_{A \cup B}(x, y)$  do
    if  $x \# y0 \in \text{prefix}(R_{A \cup B})$ 
      then  $y := y0$ 
    else if  $x \# y1 \in \text{prefix}(R_{A \cup B})$ 
      then  $y := y1$ 
    end;
  if  $R_A(x, y)$ 
    then accept
  else reject

```

□

The following theorem follows from the lemma and Proposition 1(iv).

**THEOREM 3.** *If  $A$  and  $B$  are  $\leq_T^P$ -complete for NP,  $A \cap B = \emptyset$ , then  $\text{prefix}(R_{A \cup B})$  is  $\leq_T^P$ -complete for NP.*

**THEOREM 4.** *If  $\text{prefix}(R_A) \cap \text{prefix}(R_B) = \emptyset$ , then  $\text{prefix}(R_A) \leq_T^P \text{prefix}(R_A) \cup \text{prefix}(R_B)$ .*

Thus, the analogue of result (2) about disjoint pairs of recursively enumerable sets is true when restricted to prefix sets. For the proof, simply observe that  $\text{prefix}(R_A) \cup \text{prefix}(R_B) = \text{prefix}(R_{A \cup B})$  when  $\text{prefix}(R_A)$  and  $\text{prefix}(R_B)$  are disjoint. In particular, the analogue of result (2) for  $\leq_T^P$ -complete sets is true when attention is restricted to prefix sets, i.e., the union of two disjoint  $\leq_T^P$ -complete prefix sets is  $\leq_T^P$ -complete. We would like to know whether this is true in general. Because of the following easy-to-prove proposition, we caution that it probably would be difficult to prove, if it is true.

**PROPOSITION 5.** *If the union of every pair of disjoint sets  $\leq_T^P$ -complete for NP is a  $\leq_T^P$ -complete set for NP, then  $P \neq \text{NP}$  implies  $\text{NP} \neq \text{co-NP}$ .*

**3.2. P-inseparable sets in NP.** A disjoint pair of sets in NP,  $A$  and  $B$ , are *P-separable* if there exists a set  $L$  in P such that  $A \subseteq L$  and  $B \subseteq \bar{L}$ ; they are *P-inseparable* otherwise.

It is known that there exist public-key cryptosystems that cannot be cracked in polynomial time only if there exist NP-complete P-inseparable sets, and that  $P \neq \text{UP}$  implies the existence of NP-complete P-inseparable sets [GS84]. It is not known whether this condition follows from the (presumably) weaker hypothesis  $P \neq \text{NP}$ . The following proposition states that if there exist disjoint NP-complete ( $\leq_T^P$ -complete for NP) sets  $A$  and  $B$  such that  $A \cup B$  is not NP-complete (not  $\leq_T^P$ -complete, respectively), then  $A$  and  $B$  are P-inseparable. Thus, if the  $\leq_T^P$ -complete sets for NP are not closed under disjoint union, then that fact has an equally interesting structural implication (namely, the existence of  $\leq_T^P$ -complete for NP P-inseparable sets).

**PROPOSITION 6.** *Let  $A \cup B \neq \Sigma^*$ . If  $A$  and  $B$  are P-separable sets in NP, then  $A \leq_m^P A \cup B$  and  $\text{prefix}(R_A) \leq_m^P \text{prefix}(R_{A \cup B})$ . As a consequence, if  $A$  is NP-complete ( $\leq_T^P$ -complete for NP), then  $A \cup B$  is NP-complete ( $\leq_T^P$ -complete for NP, respectively).*

*Proof.* Let  $L \in P$  be a set that separates  $A$  and  $B$ ,  $A \subseteq L$  and  $B \subseteq \bar{L}$ . Let  $a$  be a string in the complement of  $A \cup B$ . Define  $f(x) = \text{if } x \in L \text{ then } x \text{ else } a$ . Then,  $x \in A \leftrightarrow f(x) \in A \cup B$  and so  $A \leq_m^P A \cup B$ . Noting that  $x \# u \in \text{prefix}(R_A) \leftrightarrow x \# u \in \text{prefix}(R_{A \cup B})$  and  $x \in A$ , it follows that  $x \# u \in \text{prefix}(R_A) \leftrightarrow f(x) \# u \in \text{prefix}(R_{A \cup B})$ . Thus,  $\text{prefix}(R_A) \leq_m^P \text{prefix}(R_{A \cup B})$ . □

The case that there exist disjoint NP-complete sets  $A$  and  $B$  whose union is  $\Sigma^*$  is easy to analyze. In this case  $A = \bar{B}$  and so it must be the case that  $\text{NP} = \text{co-NP}$ .



$A \cup B$  is not NP-complete regardless of whether  $A$  and  $B$  are P-inseparable because only  $\Sigma^*$  can be  $\leq_m^P$ -reducible to  $\Sigma^*$ . However, it remains true that if  $A$  and  $B$  are P-separable, then  $A \cup B$  is  $\leq_T^P$ -complete for NP. Namely,  $A$  and  $B$  are P-separable if and only if  $A \in P$ .  $A \in P$  implies  $P = NP$ , from which it follows that  $\Sigma^*$  is  $\leq_T^P$ -complete for NP.

**3.3. Disjunctive-self-reducible sets in NP.** How specialized are the  $d$ -self-reducible sets prefix  $(R_A)$ ? The answer is they are not very specialized—every  $d$ -self-reducible set can be put into this form.

**THEOREM 7.** *If  $A$  is  $d$ -self-reducible, then there is an associated polynomial-time relation  $R_A$  such that  $A \equiv_d^P \text{prefix}(R_A)$ .*

*Proof.* Let  $A$  be  $d$ -self-reducible, and let the reduction be given by the polynomial time computable function  $f$ . A self-reducing tree for each  $x \in \Sigma^*$  is constructed as follows (cf. [Ko83]). The root of the tree is  $x$ . For each node  $y$ , its children are the smaller strings  $z_1, \dots, z_k$ . Note that the self-reducing tree for  $x$  is polynomial depth-bounded and that  $x \in A$  if and only if this tree has a path all of whose nodes are in  $A$ . A path has all of its nodes in  $A$  if and only if the leaf is in  $A$ , and this occurs if and only if the self-reducing machine for  $A$  accepts the leaf without generating further queries.

Now define  $R_A(x, y) \equiv [y \text{ is a path in the self-reducing tree of } x \text{ such that the leaf is accepted by the self-reducing machine for } A]$ .  $R_A$  is clearly a polynomial-time recognizable relation, and

$$x \in A \leftrightarrow \exists y |y| = p_A(|x|) \text{ and } R_A(x, y),$$

for some polynomial  $p_A$ . (Existence of  $p_A$  is guaranteed since  $f$  computes relative to some OK partial order.) Recall that by definition  $R_A$  and  $p_A$  must satisfy two conditions in order for  $R_A$  to be an associated polynomial-time recognizable relation for  $A$ . We just showed that  $R_A$  and  $p_A$  satisfy condition (1). For this reason, without possibility of confusion, let  $R_A$  denote the extension of  $R_A$  that satisfies condition (2), so that the resulting relation  $R_A$  is then an associated polynomial-time recognizable relation for  $A$ .

We know already that  $A \leq_m^P \text{prefix}(R_A)$ .

A string  $x \neq u$  belongs to  $\text{prefix}(R_A)$  if and only if  $u$  is a prefix of an accepting path in the self-reducing tree of  $x$ . Thus, a string  $x \neq u$  belongs to  $\text{prefix}(R_A)$  if and only if the string  $u$  encodes a sequence  $y_1, \dots, y_k, y_{\text{short}}$  such that

- (i)  $y_1 = x$ ;
- (ii) for all  $i, i = 1, \dots, k-1, y_{i+1}$  is a string produced by  $f$  on input  $y_i$ ;
- (iii) if  $y_{\text{short}} = \lambda$ , then  $y_k \in A$ , and if  $y_{\text{short}} \neq \lambda$ , then there is a string  $z$  such that  $y_{\text{short}}$  is a prefix of  $z, z$  is a string produced by  $f$  on input  $y_k$ , and  $z \in A$ .

If  $y_{\text{short}} = \lambda$ , then  $y_k$  is the only oracle call to  $A$ . If  $y_{\text{short}} \neq \lambda$ , then the queries to  $A$  are all of the children of  $y_k$  in the self-reducing tree of  $x$  that start with  $y_{\text{short}}$ . Since  $x \in A$  if and only if one of these children is in  $A$ , it follows that  $\text{prefix}(R_A) \leq_d^P A$ .  $\square$

**COROLLARY 8 [Ko83]** *If  $A$  is  $d$ -self-reducible, then  $A$  belongs to NP.*

Now we will gain further insight by comparing  $d$ -self-reducibility with other properties of sets in NP. A set  $A$  is defined in [Sel79] to be  $p$ -selective if there is a function  $f: \Sigma^* \times \Sigma^* \mapsto \Sigma^*$  that satisfies each of the following:

- (1)  $f$  is computable in polynomial time;
- (2)  $f(x, y) = x$  or  $f(x, y) = y$ ;
- (3)  $x \in A$  or  $y \in A \rightarrow f(x, y) \in A$ .

The following proposition summarizes known results about  $p$ -selective sets that we will use.

PROPOSITION 9. (i) [Sel79] *For every tally language  $T$ , there exists a  $\leq_T^P$ -equivalent  $p$ -selective set  $A$ , and if  $T \in \text{NP}$ , then  $A \in \text{NP}$ . Therefore, if  $E \neq \text{NE}$ , then there exist  $p$ -selective sets in  $\text{NP} - \text{P}$ .*

(ii) [Sel82] *If a set is both  $p$ -selective and  $d$ -self-reducible, then it belongs to  $\text{P}$ .*

(iii) [Sel82] *If  $A \leq_{\text{PIT}}^P B$  and  $B$  is  $p$ -selective, then  $A$  is  $p$ -selective.*

(iv) [Sel82]  *$A \in \text{P}$  if and only if  $A \leq_{\text{PIT}}^P \bar{A}$  and  $A$  is  $p$ -selective.*

By assertions (i) and (ii) of Proposition 9, it is fairly clear that not all sets in  $\text{NP} - \text{P}$  are  $d$ -self-reducible. In [BB86] the notion of self-producible circuit is defined and it is proved that a set  $A$  has self-producible circuits if and only if  $A$  is  $\leq_T^P$ -equivalent to a tally language. Therefore,  $E \neq \text{NE}$  implies there is a  $p$ -selective set in  $\text{NP} - \text{P}$  that has self-producible circuits. Thus, it is fairly clear that not even all sets with self-producible circuits in  $\text{NP} - \text{P}$  are  $d$ -self-reducible. However, these remarks do not explain anything about the  $\text{NP}$ -complete sets because Ko [Ko83] has shown that  $\text{NP}$ -complete sets cannot be  $p$ -selective unless the polynomial hierarchy collapses to  $\Sigma_2^P$ . (Specifically, Ko showed that  $p$ -selective sets have sparse oracles. The conclusion follows from [KL80].)

The question of whether all  $\text{NP}$ -complete sets are  $d$ -self-reducible is somewhat related to the Isomorphism Conjecture of Berman and Hartmanis [BH77]. The following proposition is presented without proof, for its proof is straightforward and it is probably known to others.

PROPOSITION 10. *If  $A$  is  $d$ -self-reducible and  $A \equiv_{\text{iso}}^P B$ , then  $B$  is  $d$ -self-reducible.*

All of the known  $\text{NP}$ -complete combinatorial problems are paddable and every two paddable  $\text{NP}$ -complete sets are  $\text{P}$ -isomorphic [BH77], [MY85]. The Isomorphism Conjecture asserts that all  $\text{NP}$ -complete sets are  $\text{P}$ -isomorphic. The Satisfiability Problem, for example, is paddable and is  $d$ -self-reducible also. Hence, the following corollary follows.

COROLLARY 11. *Every paddable  $\text{NP}$ -complete set is  $d$ -self-reducible.*

Therefore, all of the known  $\text{NP}$ -complete combinatorial problems are  $d$ -self-reducible. If there exist  $\text{NP}$ -complete sets that are not  $d$ -self-reducible, then there exist  $\text{NP}$ -complete sets that are not paddable and so the Berman-Hartmanis conjecture is false.

Despite Theorem 7 and Corollary 11, we have not been able to extend Theorem 4 to arbitrary  $d$ -self-reducible sets. In particular, we do not know whether the union of every pair of disjoint  $d$ -self-reducible sets  $\leq_T^P$ -complete for  $\text{NP}$  is a  $\leq_T^P$ -complete set for  $\text{NP}$ . By Proposition 5 and Corollary 11, if the two problematic assertions hold, the Berman-Hartmanis conjecture is true and the union of every pair of disjoint  $d$ -self-reducible sets  $\leq_T^P$ -complete for  $\text{NP}$  is a  $\leq_T^P$ -complete set for  $\text{NP}$ , then  $\text{P} \neq \text{NP}$  implies  $\text{NP} \neq \text{co-NP}$ .

Joseph and Young [JY85] have invented a class of  $\text{NP}$ -complete sets which are not obviously  $d$ -self-reducible, and hence are not obviously paddable. Very few structural properties are known for these sets. But, it is proved in [JY85] that if one of these  $\text{NP}$ -complete sets is not  $d$ -self-reducible, then there exist one-way functions.

By the way,  $d$ -self-reducibility and paddability are not equivalent notions on  $\text{NP}$ . To see this, let  $A$  be any  $p$ -selective set in  $\text{NP} - \text{P}$ . By Proposition (iii), no  $d$ -self-reducible set can be  $\leq_m^P$ -equivalent to  $A$ , but  $A \equiv_m^P A \times \Sigma^*$ , which is paddable by definition.

Since not all sets in  $\text{NP}$  are  $d$ -self-reducible, let us raise the question of whether all sets in  $\text{NP}$  are  $\leq_T^P$ -equivalent to a  $d$ -self-reducible set. If this is so, then, assuming  $E \neq \text{NE}$ , reducibilities could be distinguished in  $\text{NP}$  in a very strong way.

PROPOSITION 12. *Statement (a) implies statement (b).*

- (a) Every set  $A$  that belongs to  $NP-P$  is  $\cong_T^P$ -equivalent to some  $d$ -self-reducible set.
- (b)  $E \neq NE$  implies

$$\exists C, D [C \in NP, D \in NP, C \cong_T^P D, \text{ and } C \not\cong_{pitt}^P D].$$

*Proof.* Apply Proposition 9(i) to obtain a  $p$ -selective set  $D$  that belongs to  $NP-P$ . Let  $C$  be a  $\cong_T^P$ -equivalent  $d$ -self-reducible set. Then, by Proposition 9(ii) and (iii),  $C \not\cong_{pitt}^P D$ .  $\square$

(Let us compare Proposition 12 with Corollary 16 of [Sel82]. The latter result uses a different hypothesis,  $NP \cap \text{co-}NP \neq P$ , to obtain essentially the same conclusion as in Proposition 12. That assumption is used in order to obtain a set  $C$  in  $NP \cap NP-P$  that is not  $p$ -selective. Then, letting  $D = \bar{C}$ ,  $C \cong_T^P D$  is trivial and  $C \not\cong_{pitt}^P D$  follows from assertion (iv) of Proposition 9. In contrast, the sets  $C$  and  $D$  that are obtained in Proposition 12 are by assumption two different  $\cong_T^P$ -equivalent sets that belong to  $NP$  and the conclusion follows from Proposition 9(iii).)

If  $NP$  contains disjoint sets  $A$  and  $B$  such that  $\text{prefix}(R_A) \not\cong_m^P \text{prefix}(R_{A \cup B})$ , then reducibilities could be distinguished on  $NP$ .

PROPOSITION 13 (i) *If there exist disjoint sets  $A$  and  $B$  in  $NP-P$  such that  $\text{prefix}(R_A) \not\cong_m^P \text{prefix}(R_{A \cup B})$ , then " $\cong_T^P$ " and " $\cong_m^P$ " differ on  $NP$ .*

(ii) *If there exist disjoint  $\cong_T^P$ -complete sets  $A$  and  $B$  such that  $\text{prefix}(R_A) \not\cong_m^P \text{prefix}(R_{A \cup B})$ , then " $\cong_T^P$ -complete" and " $\cong_m^P$ -complete" are distinct.*

*Proof.* The first claim follows from Lemma 2. For the second claim, by Theorem 3,  $\text{prefix}(R_{A \cup B})$  is  $\cong_T^P$ -complete for  $NP$ , but by the hypothesis, it is not  $\cong_m^P$ -complete.  $\square$

Is every set in  $NP-P$   $\cong_m^P$ -equivalent to a  $d$ -self-reducible set? We have already noted that the answer is *no*, for if  $E \neq NE$ , then  $NP-P$  contains a  $p$ -selective set  $A$  and no  $d$ -self-reducible set can be  $\cong_m^P$ -equivalent to  $A$ . It follows that for every associated relation  $R_A$ ,  $\text{prefix}(R_A) \not\cong_m^P A$ .

Perhaps every set  $A$  is  $\cong_T^P$ -equivalent to  $\text{prefix}(R_A)$  for *some* appropriate associated relation  $R_A$ . Is  $\text{prefix}(R_A) \cong_T^P A$  for every  $A$  and *every* associated relation  $R_A$ ? Assuming  $P \neq NP$ , the answer is *no*. The universal set  $\Sigma^*$  has an associated relation  $R_{\Sigma^*}$  that defines  $\Sigma^*$  as a member of  $NP$  such that  $\text{prefix}(R_{\Sigma^*})$  is  $NP$ -complete, and therefore  $\text{prefix}(R_{\Sigma^*}) \not\cong_T^P \Sigma^*$ . The proof is easy. Let  $SAT$  denote the  $NP$ -complete Satisfiability Problem. Let  $R_{\Sigma^*} = \{(x, y) | y = 0^{|x|^3} \text{ or } y = 1^{|x|^3 - |z|} z \text{ and } z \text{ is a satisfying assignment of } x\}$ .  $R_{\Sigma^*}$  is clearly an associated polynomial-time recognizable relation for  $\Sigma^*$ .  $\text{Prefix}(R_{\Sigma^*})$  is  $NP$ -complete because  $x \in SAT \leftrightarrow x \# 1 \in \text{prefix}(R_{\Sigma^*})$ .

The following theorem uses this construction in order to obtain a set  $A$  in  $NP-P$  and an associated polynomial-time recognizable relation  $R_A$  such that  $\text{prefix}(R_A)$  is  $\cong_T^P$ -complete for  $NP$  but  $A$  is not  $\cong_T^P$ -complete for  $NP$ .

THEOREM 14. *Assume  $NP \neq P$ . There exists a set  $A \in NP-P$  and an associated relation  $R_A$  such that  $\text{prefix}(R_A)$  is  $\cong_T^P$ -complete for  $NP$  and  $A$  is not  $\cong_T^P$ -complete for  $NP$ . Thus,  $\text{prefix}(R_A) \not\cong_T^P A$ .*

*Proof.* Choose  $L \in NP-P$ . Choose  $B \in NP-P$  such that  $B \cong_T^P L$  but  $L \not\cong_T^P B$  [Lad75]. Let  $B_0 = \{0x | x \in B\}$  and let  $B_1 = \{1x | x \in \Sigma^*\}$ . Let  $R_{\Sigma^*}$  be an associated polynomial-time recognizable relation for  $\Sigma^*$  such that  $\text{prefix}(R_{\Sigma^*})$  is  $NP$ -complete. Let  $R_{B_0}$  be any associated relation for  $B_0$  and define  $R_{B_1}$  so that

$$R_{B_1}(ax, y) \equiv [a = 1 \wedge R_{\Sigma^*}(x, y)].$$

Noting that  $B_0 \cap B_1 = \emptyset$ , define  $A = B_0 \cup B_1$  and let  $R_A = R_{B_0} \cup R_{B_1}$ .

Clearly  $A \cong_T^P B$ , because  $ax \in A \leftrightarrow (a = 0 \wedge x \in B) \vee a = 1$ , and  $x \in B \leftrightarrow 0x \in A$ . Therefore,  $A$  is not  $\cong_T^P$ -complete for  $NP$ .

Now we will show that  $\text{prefix}(R_{\Sigma^*}) \leq_T^P \text{prefix}(R_A)$ . Since  $\text{prefix}(R_{\Sigma^*})$  is NP-complete, this will complete the proof:

$$\begin{aligned} \text{prefix}(R_{\Sigma^*}) &\leq_T^P \text{prefix}(R_{B_1}) && \text{(by definition of } R_{B_1}) \\ &\leq_T^P \text{prefix}(R_{B_0 \cup B_1}) && \text{(by Lemma 2)} \\ &= \text{prefix}(R_A). \end{aligned}$$

## REFERENCES

- [BB86] J. BALCÁZAR AND R. BOOK, *On generalized Kolmogorov complexity*, in Proc. 3rd Annual Symposium on Theoretical Aspects of Computer Science, Springer-Verlag, Berlin, New York, 1986, pp. 334–340.
- [Ber78] P. BERMAN, *Relationships between density and deterministic complexity of NP-complete languages*, in Proc. 5th Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1978, pp. 63–71.
- [BH77] L. BERMAN AND H. HARTMANIS, *On isomorphisms and density of NP and other complete sets*, SIAM J. Comput., 6 (1977), pp. 305–322.
- [Boo74a] R. BOOK, *Comparing complexity classes*, J. Comput. System Sci., 9 (1974), pp. 213–219.
- [Boo74b] —, *Tally languages and complexity classes*, Inform. and Control, 26 (1974), pp. 186–193.
- [GJ79] M. GAREY AND D. JOHNSON, *Computers and Intractability: A Guide to The Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
- [GS84] J. GROLLMANN AND A. SELMAN, *Complexity measures for public-key cryptosystems*, in Proc. 25th IEEE Symposium on Foundations of Computer Science, 1984, pp. 495–503.
- [Har83] J. HARTMANIS, *On sparse sets in NP–P*, Inform. Process. Lett., 16 (1983), pp. 53–60.
- [HHH74] J. HARTMANIS AND H. HUNT III, *The LBA problem and its importance in the theory of computing*, SIAM-AMS Proceedings, 7 (1974), pp. 1–26.
- [HU79] J. HOPCROFT AND J. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.
- [JS74] N. JONES AND A. SELMAN, *Turing machines and the spectra of first-order formulas*, J. Symbolic Logic, 29 (1974), pp. 139–150.
- [JY85] D. JOSEPH AND P. YOUNG, *Some remarks on witness functions for non-polynomial and non-complete sets in NP*, Theoret. Comput. Sci., 39 (1985), pp. 225–237.
- [KL80] R. KARP AND R. LIPTON, *Some connections between nonuniform and uniform complexity classes*, in Proc. 12th ACM Symposium on Theory of Computing, 1980, 302–309.
- [Ko83] K. KO, *On self-reducibility and weak P-selectivity*, J. Comput. System Sci., 26 (1983), 209–211.
- [Lad75] R. LADNER, *On the structure of polynomial time reducibility*, J. Assoc. Comput. Mach., 22 (1975), 155–171.
- [LLS75] R. LADNER, N. LYNCH, AND A. SELMAN, *A comparison of polynomial time reducibilities*, Theor. Comput. Sci., 1 (1975), pp. 103–123.
- [Lon82] T. LONG, *Strong nondeterministic polynomial-time reducibilities*, Theoret. Comput. Sci., 21 (1982), pp. 1–25.
- [Mah82] S. MAHANEY, *Sparse complete sets for NP: solution of a conjecture of Berman and Hartmanis*, J. Comput. System Sci., 25 (1982), pp. 130–143.
- [MP79] A. MEYER AND M. PATERSON, *With what frequency are apparently intractable problems difficult?*, Tech. Report MIT/LCS/TM-126, Massachusetts Institute of Technology, Cambridge, MA, 1979.
- [MY85] S. MAHANEY AND P. YOUNG, *Orderings of polynomial isomorphism types*, Theoret. Comput. Sci., 39 (1985), pp. 207–224.
- [Sch76] C. SCHNORR, *Optimal algorithms for self-reducible problems*, in Proc. 3rd Colloquium on Automata, Languages, and Programming, Edinburgh, University Press, Edinburgh, July 1976, pp. 322–337.
- [Sch82] U. SCHÖNING, *A uniform approach to obtain diagonal sets in complexity classes*, Theor. Comput. Sci., 18 (1982), pp. 95–103.
- [Sel79] A. SELMAN, *P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP*, Math. Systems Theory, 13 (1979), pp. 55–65.
- [Sel82] —, *Reductions on NP and P-selective sets*, Theoret. Comput. Sci., 19 (1982), pp. 287–304.
- [Sho71] J. SHOENFIELD, *Degrees of Unsolvability. Mathematics Studies*, North Holland, Amsterdam, 1971.

## AXIOMATISING FINITE CONCURRENT PROCESSES\*

MATTHEW HENNESSY†

**Abstract.** We re-examine the well-known observational equivalence between processes with a view to modifying it so as to distinguish between concurrent and purely nondeterministic processes.

Observational equivalence is based on atomic actions or observations. In the first part of this paper we generalise these atomic observations so that arbitrary processes may act as observers. We show, for a particular language based on finite (CCS) terms, that the generalised equivalence coincides with observational equivalence; the more powerful observers do not lead to a finer equivalence.

In the second part of the paper we consider observers which can distinguish the beginning and ending of atomic actions. The resulting equivalence distinguishes a concurrent process from the purely nondeterministic process obtained by interleaving its possible actions. We give a complete axiomatisation for the congruence generated by the new equivalence.

**Key words.** concurrent processes, observational equivalence, noninterleaving

**CR classification.** F.3.2

**Introduction.** In [Mil1], [Mil2] Milner has developed a semantic theory of processes called observational or bisimulation equivalence. Intuitively two processes are deemed to be equivalent if there is no observational method for distinguishing them. We observe a process by communicating with it and this act of observation changes the subsequent behaviour of the process. A suitable formulation of these ideas leads to observational equivalence. It is the basis of a very elegant semantic theory of processes and has gained considerable popularity. When applied to algebraic languages for describing processes it can be characterised equationally [He]. This in turn leads to transformational proof techniques for showing that a process implementation meets its specification [Mil1].

However, this semantic theory does not treat concurrency as a primitive independent notion: every concurrent process is equivalent to a nondeterministic but sequential process. Consider, for example, the language CCS [Mil1]. Here  $p|q$  represents the process which consists of two subprocesses in parallel,  $p$  and  $q$ . As simple examples of such subprocesses consider  $aNIL$ ,  $bNIL$ , respectively. These are subprocesses which can only perform one action (called  $a$ ,  $b$ , respectively) and then die. The parallel process  $aNIL|bNIL$  is observationally equivalent to a sequential nondeterministic process, obtained by interleaving the possible atomic actions of the two subprocesses  $aNIL$ ,  $bNIL$ . In CCS this equivalent process is written as  $abNIL + baNIL$ . In general, no matter how complicated  $p$  and  $q$  might be, such a reduction is always possible although the resulting nondeterministic process may not be syntactically expressible in the description language under consideration.

The popularity of observational equivalence (and related equivalences) bears witness to the usefulness of a semantic theory based on interleaving, where concurrency is reduced to nondeterminism. Nevertheless, at an intuitive level the reduction of concurrency to nondeterminism seems unsatisfactory; surely one important property

---

\* Received by the editors October 23, 1981; accepted for publication (in revised form) October 16, 1987. This paper is based on a preprint entitled, "On the Relationship between Time and Interleaving," which was based on work carried out while the author was visiting l'Ecole des Mines, Sophia-Antipolis, France, in 1980. This work was supported by Sciences and Engineering Research Council grants GR/D/97368 and GR/E05360.

† Computer Science, School of Mathematical and Physical Sciences, University of Sussex, Falmer, Brighton BN1 9QH, United Kingdom.

of a concurrent process is the fact that it is composed of independent and possibly distributed processes.

The aim of this paper is to investigate semantic equivalences which do not reduce concurrency to nondeterminism. More specifically, we would like to use the basic paradigm of observational equivalence to develop behavioural equivalences which reflect in some way the concurrent nature of processes.

To continue this discussion it is necessary to describe in more detail observational equivalence. Let  $P$  denote a set of processes. The operational behaviour of these processes may be defined by describing the set of moves or actions which they can perform. By and large we will not be concerned with the nature of these actions nor with how we assign specific actions to specific processes, although we will see an example later in the paper. At this stage it suffices to say that for each action  $a$  there is a binary relation  $\overset{a}{\Rightarrow}$  over processes;  $p \overset{a}{\Rightarrow} q$  means that the process  $p$  may perform the action  $a$  and performing this action may transform it into the process  $q$ . In general  $\overset{a}{\Rightarrow}$  will be a relation because not every process can perform every action and individual processes may be capable of performing an action in various different ways.

Observational equivalence,  $\approx$ , is defined with respect to such a description of the operational behaviour of processes. It satisfies the following property:

(\*) if  $p \approx q$  and  $p \overset{a}{\Rightarrow} p'$ , then  $q \overset{a}{\Rightarrow} q'$  for some  $q'$  such that  $p' \approx q'$ .

Here  $p \overset{a}{\Rightarrow} p'$  may be termed an observation and so (\*) may be interpreted as saying that if  $p$  and  $q$  are observationally equivalent then they have similar observations. In fact  $\approx$  may be defined to be the largest symmetric relation which satisfies (\*). So, contrapositively, if  $p$  is not equivalent to  $q$  there is some observation of one which has no equivalent observation of the other.

When we apply this definition to languages such as CCS we are unable to distinguish concurrency from interleaving. Let us now look at how, while retaining the basic paradigm (\*), we might modify the definition so as to obtain a more discriminating equivalence. It uses the basic observations  $p \overset{a}{\Rightarrow} p'$  and we first consider the possibility of employing similar but more complex observations. The basic observation  $p \overset{a}{\Rightarrow} p'$  may be viewed as a successful communication between an unidentified external process and the process under observation,  $p$ . Indeed, in the usual formulation of the operational semantics of languages such as CCS, communication is defined as mutual observation. In this formulation  $p \overset{a}{\Rightarrow} p'$  is true exactly when a process such as  $aq$  can communicate with  $p$ . The effect of this communication on  $p$  is to transform it into  $p'$ . It also transforms the observer  $aq$  into  $q$ . So the total effect of the observation or communication may be described by

$$\langle p, aq \rangle \rightarrow \langle p', q \rangle.$$

However, in the definition of observational equivalence given above in (\*) no use is made of the residual of the observer. From the point of view of the definition we may take as observers the simple atomic processes  $aNIL$ ; the only observations used are of the form

$$\langle p, aNIL \rangle \rightarrow \langle p', NIL \rangle.$$

This reformulation of observational equivalence leads to the obvious generalisation where we allow as observers arbitrarily complicated processes. This includes observers which consist of many subprocesses each independently probing the process under observation.

Such a generalised observational equivalence is defined and investigated in § 1. The main theorem of this section states that this increase in the power of observers does not result in a more discriminating equivalence; the equivalence defined using arbitrary observers coincides with the equivalence defined using atomic observers only. This theorem is proved for a particular language based on CCS. However, the proof is so general that it should be widely applicable.

Faced with this setback we must examine more carefully the implicit assumptions underlying observational equivalence. One assumption is that observers, no matter how complicated, have only a monolithic view of the process under observation. They never get a separate view of subparts of the process nor are they allowed to experiment independently on these distinct subparts. In a separate paper we investigate generalisations of observational equivalence where observers are endowed with such powers. Here we weaken a different underlying assumption.

In the usual operational semantics of languages such as CCS, and the related observational equivalence, communication (or observation) is taken to be an instantaneous indivisible event. However it can be argued that there are at least two events: the communication as seen by the two processes in question. We postulate that these two events are different or that they take a certain amount of time. Rather than associate specific durations with actions, we introduce observers which can distinguish the initialisation of a communication from its termination. The observational equivalence based on these new observers is more discriminating. In particular, it can distinguish concurrency from nondeterminism.

The new equivalence is the topic of § 2. It is defined for a simple language for describing finite communicating processes, based on CCS. The main result of this section is that the related congruence can be completely characterised by a set of equations. An equational characterisation is one of the main attractions of the standard observational equivalence as such equations form the basis for transformational proof systems for processes. Thus our result provides a basis for a similar type of proof system which takes into consideration the concurrent nature of processes.

In a final section we briefly examine other formulations of our new equivalence and related work.

**1. Generalised observational equivalence.** In this section we reformulate the standard definition of observational equivalence so as to allow arbitrarily complex observers. The definition and some general consequences are given in § 1.1. In the next section it is applied to a CCS-like language and we prove that in this setting it is sufficient to consider simple atomic observers; the addition of the more complex observers does not add to the distinguishing power of the equivalence. In the final section we prove that our reformulation of observational equivalence does indeed coincide with the standard version of observational equivalence, as defined, say, in [Mil2]. We also provide an equational characterisation of the associated congruence.

**1.1.** A general notion of observation must take into account not only the effect the observation has on the observed but also on the observers themselves. Let  $O, P$  be sets of *observers*, *processes*, respectively. For the moment we are not interested in the composition of these objects. We use  $C$  to denote  $O \times P$  which we call the set of *configurations*. A set of observations is given by any relation  $\rightarrow \subseteq C \times C$ . We write  $\langle o, p \rangle \rightarrow \langle o', p' \rangle$  rather than  $\langle \langle o, p \rangle, \langle o', p' \rangle \rangle \varepsilon \rightarrow$  as it is more suggestive. It means that the act of  $o$  observing  $p$  changes the observer  $o$  into  $o'$  and the observed process  $p$  into  $p'$ . We define an equivalence of “indistinguishability” between processes,  $\sim_P$ , and a similar equivalence between observers,  $\sim_O$ . Intuitively  $p \sim_P p'$  means that there is no

way of observing any different behaviour between  $p$  and  $p'$  while  $o \sim_O o'$  means that both  $o$  and  $o'$  have exactly the same observational power. Following these intuitive ideas we would expect these relations to satisfy the following:

(a) If  $p \sim_P p'$ , then whenever  $\langle o, p \rangle \rightarrow \langle o', q \rangle$  there exists a configuration  $\langle o'', q' \rangle$  such that  $\langle o, p' \rangle \rightarrow \langle o'', q' \rangle$  where  $o' \sim_O o''$  and  $q \sim_P q'$ .

(b) If  $o \sim_O o'$ , then whenever  $\langle o, p \rangle \rightarrow \langle o'', q \rangle$  there exists a configuration  $\langle o''', q' \rangle$  such that  $\langle o'p \rangle \rightarrow \langle o''', q' \rangle$ , where  $o'' \sim_O o'''$  and  $q \sim_P q'$ .

In fact, we take (a), (b) (and their symmetric counterparts) to be the definitions of  $\sim_P, \sim_O$ . The inherent circularity of the definition is handled by using maximal fixpoints.

In the formal definition we use the following notation. For  $R_1 \subseteq O \times O, R_2 \subseteq P \times P$  let  $R_1 \times R_2 \subseteq C \times C$  be defined by  $\langle \langle o, p \rangle, \langle o', p' \rangle \rangle \in R_1 \times R_2$  if  $\langle o, o' \rangle \in R_1$  and  $\langle p, p' \rangle \in R_2$ .

DEFINITION 1.1.1. For  $R_1 \subseteq O \times O, R_2 \subseteq P \times P$  define  $D_1(R_1, R_2) \subseteq O \times O, D_2(R_1, R_2) \subseteq P \times P$  by the following:

(i)  $\langle o, o' \rangle \in D_1(R_1, R_2)$  if for every  $p$  in  $P$   $\langle o, p \rangle \rightarrow c$  implies  $\langle o', p \rangle \rightarrow c'$  for some configuration  $c'$  such that  $\langle c, c' \rangle \in R_1 \times R_2$ .

(ii)  $\langle p, p' \rangle \in D_2(R_1, R_2)$  if for every  $o$  in  $O$   $\langle o, p \rangle \rightarrow c$  implies  $\langle o, p' \rangle \rightarrow c'$  for some configuration  $c$  such that  $\langle c, c' \rangle \in R_1 \times R_2$ .  $\square$

Viewed as a tuple of functions over relations  $D_1, D_2$  are monotonic with respect to set inclusion. Therefore we may take  $\sim_O, \sim_P$  to be the maximal pair of symmetric relations which satisfies the equations

$$R_1 = D_1(R_1, R_2), \quad R_2 = D_2(R_1, R_2).$$

It is straightforward to show that both  $\sim_O$  and  $\sim_P$  are equivalence relations and by definition they satisfy (a) and (b) above. Readers familiar with the standard definition of observational equivalence should recognise that  $\sim_P, \sim_O$  are attempts at a straightforward generalisation to include more complicated observers. In § 1.3 we will actually see that the standard observational equivalence may be obtained from ours by restricting attention to what we will call atomic observers.

We first put a natural restriction on our systems of observations. We wish to examine how processes react in environments of other processes. Therefore we assume  $O = P$ . Further, we assume that “observing” and “being observed” are the same. So we assume that every observation relation  $\rightarrow$  satisfies

$$\langle o, p \rangle \rightarrow \langle o', p' \rangle \quad \text{if and only if} \quad \langle p, o \rangle \rightarrow \langle p', o' \rangle.$$

LEMMA 1.2. *If  $O = P$  and  $\rightarrow$  satisfies the above condition, then  $\sim_O$  and  $\sim_P$  coincide and are the maximal fixpoints of both of the equations*

$$R = D_1(R, R) \quad \text{and} \quad R = D_2(R, R).$$

*Proof.* Because of the symmetry of  $\rightarrow$  it is easy to show

$$(*) \quad \sim_O \subseteq D_2(\sim_P, \sim_O),$$

$$(**) \quad \sim_P \subseteq D_2(\sim_P, \sim_O),$$

from which it follows that  $\sim_O = \sim_P$ .

Let  $X$  denote the maximal fixpoint of  $R = D_1(R, R)$ . Because of (\*) it now follows that  $\sim_P \subseteq X$ . To show the converse, it is sufficient to prove  $X \subseteq D_1(X, X)$  which once again follows from the symmetry of  $\rightarrow$ .  $\square$

In the case when  $O$  and  $P$  coincide the resulting equivalence,  $\sim_O = \sim_P$ , is called *generalised bisimulation equivalence*, and will usually be denoted by  $\sim_O$ . We wish to



show that in fact there is no need to take observations with respect to the entire set of observers; a much smaller set will generate exactly the same equivalence. Note that the definition of  $\sim_O, \sim_P$  can be parameterised with respect to an arbitrary subset of observers. We are only interested in one particular subset, which we now explain.

An observer  $o$  is *blind* if for every  $p$  in  $P$   $\langle o, p \rangle \rightarrow \langle o', p' \rangle$  implies  $o'$  coincides with  $o$ ; a blind observer never changes state and therefore cannot perceive any difference in the process being observed. The observer  $o$  is *atomic* if for every  $p$  in  $P$   $\langle o, p \rangle \rightarrow \langle o', p' \rangle$  implies  $o'$  is blind. So atomic observers may make at most one observation. Let  $A \subseteq O$  denote the set of atomic observers. Definition 1.1.1 can now be modified so as to apply to  $A$  rather than  $O$ :

For  $R_1 \subseteq A \times A, R_2 \subseteq P \times P$  let  $E_1(R_1, R_2) \subseteq A \times A, E_2(R_1, R_2) \subseteq P \times P$  be defined by the following.

(i)  $\langle o, o' \rangle \in E_1(R_1, R_2)$  if for every  $p$  in  $P$   $\langle o, p \rangle \rightarrow c$  implies  $\langle o', p \rangle \rightarrow c'$  for some configuration  $c'$  such that  $\langle c, c' \rangle \in R_1 \times R_2$ .

(ii)  $\langle p, p' \rangle \in E_2(R_1, R_2)$  if for every  $o \in A$   $\langle o, p \rangle \rightarrow c$  implies  $\langle o, p' \rangle \rightarrow c'$  for some configuration  $c'$  such that  $\langle c, c' \rangle \in R_1 \times R_2$ .

Let  $\sim_{OA}, \sim_A$  be the maximal symmetric solutions to the pair of equations

$$R_1 = E_1(R_1, R_2), \quad R_2 = E_2(R_1, R_2).$$

The equivalence on observers  $\sim_{OA}$  is uninteresting but  $\sim_A$  is a formalisation of the idea that processes are equivalent unless they can be distinguished by atomic observers. In the next section we show  $\sim_P$  and  $\sim_A$  coincide, at least when applied to a particular language.

**1.2. A simple language.** The language used is a simple extension of “finite CCS.” The only innovation is an asymmetric version of the usual parallel combinator  $|$ , which we denote by  $\vdash$ . Intuitively  $p \vdash q$  acts in exactly the same way as  $p|q$  except that the very first action should involve the subprocess  $p$ .

Let  $\text{Act}$  be a set of actions which as usual we assume has the form  $\{\tau\} \cup \Lambda \cup \bar{\Lambda}$ , where

- $\Lambda$  is a given set of names,
- $\bar{\Lambda} = \{\bar{a}, a \in \Lambda\}$ , the complements of names in  $\Lambda$ ,
- $\tau$  is a distinguished action.

The distinguished action  $\tau$  is used to denote internal invisible actions and in our language these occur when subprocesses communicate with each other. Communication is modelled by the simultaneous occurrence of an action such as  $a$  with its complement  $\bar{a}$ . Notationally it is convenient to assume that complementation is symmetric, i.e.,  $\bar{\bar{a}} = a$ . We use  $\mu, \gamma$ , etc., to range over  $\text{Act}$  unless otherwise stated, and  $a, b$ , etc., to range over the restricted set of visible actions  $\Lambda \cup \bar{\Lambda}$ . We will sometimes use  $\text{VAct}$  to denote this set,  $\Lambda \cup \bar{\Lambda}$ . The language we use is given by the BNF definition

$$p := \text{NIL} \mid \mu p \mid p + q \mid p \mid q \mid p \vdash q.$$

For the remainder of the paper we let  $P$  denote the set of processes defined by this language. We assume prefixing,  $\mu$ -, binds stronger than  $\vdash, |$ , which in turn bind stronger than  $+$ . We also omit most occurrences of  $\text{NIL}$ , for example, rendering  $a\text{NIL} + b\text{NIL}$  simply as  $a + b$ .

We now define a system of observations based on this language. The processes to be observed are all those definable in the language and we use exactly the same processes as observers. Observation is simply a form of communication. Consequently the definition of observation relies mainly on what we mean by communication. This is defined in Fig. 1:  $p \xrightarrow{x} q$  means that  $p$  may be transformed into  $q$  by performing an

$$\begin{array}{ll}
\text{(I)} & \tau p \xrightarrow{\tau} p \\
\text{(II)} & p \xrightarrow{a} p', q \xrightarrow{\bar{a}} q' \text{ implies } p|q \xrightarrow{\tau} p'|q' \\
\text{(III)} & ap \xrightarrow{\mu} p \\
\text{(IV)} & p \xrightarrow{\mu} p' \text{ implies } p + q \xrightarrow{\mu} p' \\
& q + p \xrightarrow{\mu} p' \\
\text{(V)} & p \xrightarrow{\mu} p' \text{ implies } p|q \xrightarrow{\mu} p'|q \\
& p \uparrow q \xrightarrow{\mu} p'|q \\
& q|p \xrightarrow{\mu} q|p'
\end{array}$$

FIG. 1. *Communication.*

internal communication. To make the language more expressive we have included a direct syntactic representation of internal actions and this accounts for rule (I). The next rule is the principal one: communication is the simultaneous occurrence of complementary actions.  $p \xrightarrow{a} p'$  means that  $p$  may perform the action  $a$  and thereby be transformed into  $p'$ . So  $p|q$  may perform a communication to be transformed into  $p'|q'$  whenever  $p \xrightarrow{a} p'$  and  $q \xrightarrow{\bar{a}} q'$ , i.e., whenever  $p$  and  $q$  can perform complementary actions. The remaining rules merely state how external actions are produced and how actions, both external and internal communications, factor through the operators.

The definition of the observation relation  $\rightarrow$  is given in Fig. 2. It is the least relation which satisfies rules (I), (II), and (III). The essential rule is (I) which states that the basic method of observation is by communication. Rule (II) states that simultaneous observations are allowed. The final rule states that observation is asynchronous; an observation is not a simple discrete step but an indeterminate sequence of interactions between observations.

*Examples.*

$$(1) \quad \langle \bar{a}b\bar{o}, a(c+bp) | eq \rangle \rightarrow \langle o, p | eq \rangle.$$

This observation comes about by virtue of two communications between the observer and the observed, the first via the action  $a$ , the second via  $b$ .

$$(2) \quad \langle \bar{a}(cbo_1 | (\bar{c}o_2 + d)), a(c+bp) | eq \rangle \rightarrow \langle o_1 | o_2, p | eq \rangle.$$

$$\begin{array}{ll}
\text{(I)} & o|p \xrightarrow{\tau} o'|p' \text{ implies } \langle o, p \rangle \rightarrow \langle o', p' \rangle \\
\text{(II)} & \langle o_1, p_1 \rangle \rightarrow \langle o'_1, p'_1 \rangle, \langle o_2, p_2 \rangle \rightarrow \langle o'_2, p'_2 \rangle \\
& \text{implies } \langle o_1 | o_2, p_1 | p_2 \rangle \rightarrow \langle o'_1 | o'_2, p'_1 | p'_2 \rangle \\
\text{(III)} & \text{(a) } c \rightarrow c \\
& \text{(b) } c \rightarrow c', c' \rightarrow c'' \text{ implies } c \rightarrow c'
\end{array}$$

FIG. 2. *Observations.*

Here there are also two interactions, but this time they are separated by a communication internal to the observer.

$$(3) \quad \langle ao_1 | (bo_2 + co_3), \bar{a}p_1 | \bar{b}p_2 \rangle \rightarrow \langle o_1 | o_2, p_1 | p_2 \rangle.$$

This involves two independent simultaneous observations of different subprocesses of the observed process, the observer  $ao_1$  observing the subprocess  $\bar{a}p$  and the observer  $bo_2 + co_3$  observing  $\bar{b}p_2$ .

$$(4) \quad \langle ao_1 | bo_2 + do_3, \bar{a}p_1 | c\bar{b}p_2 | \bar{c}p_3 \rangle \rightarrow \langle o_1 | o_2, p_1 | p_2 | p_3 \rangle.$$

In this case the two independent observations are separated by a computation internal to the observed process.  $\square$

It is straightforward to prove that this particular observation relation  $\rightarrow$  is symmetric. First we can prove  $p | q \xrightarrow{x} p' | q'$  implies  $q | p \xrightarrow{x} q' | p'$ , using induction on the length of the proof of  $p | q \xrightarrow{x} p' | q'$ . Using the same inductive proof method, we can then show  $\langle o, p \rangle \rightarrow \langle o', p' \rangle$  implies  $\langle p, o \rangle \rightarrow \langle p', o' \rangle$ .

Following the general definition we therefore have a generalised bisimulation equivalence  $\sim_O$  over  $P$ . Following the restricted definition we also have  $\sim_A$  over  $P$ , observational equivalence. Superficially this looks much weaker but in fact we have the following.

**THEOREM 1.2.1.**  $p \sim_O q$  if and only if  $p \sim_A q$ .  $\square$

In effect, the weakness of the observers is compensated for by the power of the observational method.

One direction of the theorem is very straightforward.

**LEMMA 1.2.2.**  $p \approx_O q$  implies  $p \sim_A q$ .

*Proof.* Let  $\sim'$  denote  $\sim_O \cap (A \times A)$ . Then referring to the definitions in the previous section, it is sufficient to show

$$\sim' \subseteq E_1(\sim', \sim_O), \quad \sim_O \subseteq E_2(\sim', \sim_O).$$

This is immediate because  $\sim_O$  satisfies

$$\sim_O = D_1(\sim_O, \sim_O), \quad \sim_O = D_2(\sim_O, \sim_O)$$

and the set of atomic observers is a subset of the complete set of observers.  $\square$

Note that to carry out this proof we did not need to know the atomic observers. They are essentially of the form  $NIL$  or  $aNIL$ . To prove the converse we introduce the set of *sequential observers*, observers of the form  $a_1 \cdots a_k NIL$ ,  $k \geq 0$ . We will assume that the distinguished action  $\tau$  does not appear in any sequential observer. The complementation notation is also extended to these terms in the natural way:  $\bar{a}_1 \cdots \bar{a}_k NIL$  is the complement of  $a_1 \cdots a_k NIL$ .

The first result we need is that these new observers are no more powerful than simple atomic ones.

**LEMMA 1.2.3.** If  $p \sim_A q$  and  $\langle s, p \rangle \rightarrow \langle NIL, p' \rangle$  where  $s$  is a sequential observer, then  $\langle s, q \rangle \rightarrow \langle NIL, q' \rangle$  for some  $q'$  such that  $p' \sim_A q'$ .

*Proof.* The proof is induction on the length of  $s$ . If this is zero or one, then the result follows by the definition of  $\sim_A$ . So we may assume  $s$  has the form  $at$ .

We can now prove by induction on the length of the proof of  $\langle at, p \rangle \rightarrow \langle NIL, p' \rangle$  that  $p \xrightarrow{x}^* q$  for some  $q$  such that  $q \xrightarrow{\bar{a}} r$  and  $\langle t, r \rangle \rightarrow \langle NIL, p' \rangle$ . (Here  $\xrightarrow{x}^*$  is the reflexive transitive closure of  $\xrightarrow{x}$ .) This proof uses in turn the fact that  $\langle at, p \rangle \rightarrow \langle t, x \rangle$  implies  $p \xrightarrow{x}^* q \xrightarrow{\bar{a}} r$  for some terms  $q$  and  $r$ . This means we have the atomic observation  $\langle aNIL, p \rangle \rightarrow \langle NIL, r \rangle$ . By the definition of  $\sim_A$ , we therefore have that  $\langle aNIL, q \rangle \rightarrow \langle NIL, r' \rangle$  for some  $r'$  such that  $r \sim_A r'$ . Now using the inductive hypothesis on  $\langle t, r \rangle \rightarrow \langle NIL, p' \rangle$ , we may assume  $\langle t, r' \rangle \rightarrow \langle NIL, q' \rangle$  for some  $q'$  such that  $p' \sim_A q'$ . The atomic

observation  $\langle aNIL, q \rangle \rightarrow \langle NIL, r' \rangle$  can be modified to give the observation  $\langle at, q \rangle \rightarrow \langle t, r' \rangle$ , which when combined with  $\langle t, r' \rangle \rightarrow \langle NIL, q' \rangle$  gives the required observation  $\langle at, q \rangle \rightarrow \langle NIL, q' \rangle$ .  $\square$

The essential part of the proof of the main result is that all observations can be sequentialised. This is formalised as follows.

LEMMA 1.2.4. *If  $\langle o, p \rangle \rightarrow \langle o', p' \rangle$ , then there exists a sequential observer  $s$  such that  $\langle s, o \rangle \rightarrow \langle NIL, o' \rangle$  and  $\langle \bar{s}, p \rangle \rightarrow \langle NIL, p' \rangle$ .*

*Proof.* The proof is by induction on the length of the derivation of  $\langle o, p \rangle \rightarrow \langle o', p' \rangle$ .

(i)  $o | p \xrightarrow{x} o' | p'$ . This case requires a separate proof by induction on the length of the derivation of  $o | p \xrightarrow{x} o' | p'$ . This is straightforward and is left to the reader. Note that the resulting  $s$  will be of length at most one.

(ii)  $\langle o_1 | o_2, p_1 | p_2 \rangle \rightarrow \langle o'_1 | o'_2, p'_1 | p'_2 \rangle$  because  $\langle o_1, p_1 \rangle \rightarrow \langle o'_1, p_1 \rangle$  and  $\langle o_2, p_2 \rangle \rightarrow \langle o'_2, p'_2 \rangle$ . By induction we have two sequences  $s_1, s_2$  such that

$$\begin{aligned} \langle s_1, o_1 \rangle &\rightarrow \langle NIL, o'_1 \rangle, & \langle \bar{s}_1, p_1 \rangle &\rightarrow \langle NIL, p'_1 \rangle, \\ \langle s_2, o_2 \rangle &\rightarrow \langle NIL, o'_2 \rangle, & \langle \bar{s}_2, p_2 \rangle &\rightarrow \langle NIL, p'_2 \rangle. \end{aligned}$$

Let  $s$  denote  $s_1 s_2$ . (The sequence  $s_2 s_1$  could also be used.) The observation  $\langle s_1 o_1 \rangle \rightarrow \langle NIL, o'_1 \rangle$  can be transformed into the observation  $\langle s, o_1 | o_2 \rangle \rightarrow \langle s_2, o'_1 | o_2 \rangle$  and the observation  $\langle s_2, o_2 \rangle \rightarrow \langle NIL, o'_2 \rangle$  can be transformed into  $\langle s_2, o'_1 | o_2 \rangle \rightarrow \langle NIL, o'_1 | o'_2 \rangle$ . Therefore,  $\langle s, o_1 | o_2 \rangle \rightarrow \langle NIL, o'_1 | o'_2 \rangle$ .

In a similar fashion we can derive  $\langle \bar{s}, p_1 | p_2 \rangle \rightarrow \langle NIL, p'_1 | p'_2 \rangle$ .

(iii) (a)  $\langle o, p \rangle \rightarrow \langle o, p \rangle$ . The required  $s$  is  $NIL$ .

(b)  $\langle o, p \rangle \rightarrow \langle o'', p'' \rangle, \langle o'', p'' \rangle \rightarrow \langle o', p' \rangle$ . In this case the proof is similar to part (ii).  $\square$

The converse of this lemma is also true.

LEMMA 1.2.5. *If  $\langle s, o \rangle \rightarrow \langle NIL, o' \rangle, \langle \bar{s}, p \rangle \rightarrow \langle NIL, p' \rangle$  where  $s$  is a sequential observer, then*

$$\langle o, p \rangle \rightarrow \langle o', p' \rangle.$$

*Proof.* The proof is by induction on the length of  $s$ .

(i) If  $s$  is  $NIL$ . Here we have that  $\langle NIL, o \rangle \rightarrow \langle NIL, o' \rangle$  and  $\langle NIL, p \rangle \rightarrow \langle NIL, p' \rangle$ . As in the previous lemma, these can be transformed to  $\langle o, p \rangle \rightarrow \langle o', p \rangle$  and  $\langle o', p \rangle \rightarrow \langle o', p' \rangle$  respectively, which leads to  $\langle o, p \rangle \rightarrow \langle o', p' \rangle$ .

(ii)  $s$  is  $aNIL$ . In this case we must use induction on the combined length of the derivations of  $\langle aNIL, o \rangle \rightarrow \langle NIL, o' \rangle$  and  $\langle \bar{a}NIL, p \rangle \rightarrow \langle NIL, p' \rangle$ . The base case is when  $o \xrightarrow{a} o'$  and  $p \xrightarrow{\bar{a}} p'$ . In this case  $o | p \xrightarrow{x} o' | p'$  and therefore  $\langle o, p \rangle \rightarrow \langle o', p' \rangle$ .

There are a large number of induction cases, all of which are straightforward. A typical example is when

$$\begin{aligned} \langle aNIL, o \rangle &\rightarrow \langle aNIL, o_1 \rangle \rightarrow \langle NIL, o' \rangle, \\ \langle aNIL, p \rangle &\rightarrow \langle NIL, p_1 \rangle \rightarrow \langle NIL, p' \rangle. \end{aligned}$$

The required observation is obtained in this case by combining the sequence of observations:

$$\langle o, p \rangle \rightarrow \langle o_1, p \rangle \rightarrow \langle o', p_1 \rangle \rightarrow \langle o', p' \rangle.$$

In this sequence the first observation is obtained by transforming  $\langle aNIL, o \rangle \rightarrow \langle aNIL, o_1 \rangle$ , the second by the inductive hypothesis and the third by transforming  $\langle NIL, p_1 \rangle \rightarrow \langle NIL, p' \rangle$ .

(iii) More generally, suppose  $s$  has the form  $at$ . In this case we can apply induction and the previous case.

If  $\langle at, o \rangle \rightarrow \langle NIL, o' \rangle$ , then there exists some  $o''$  such that  $\langle at, o \rangle \rightarrow \langle t, o'' \rangle$  and  $\langle t, o'' \rangle \rightarrow \langle NIL, o' \rangle$ . (This requires a separate proof.) The first derivation  $\langle at, o \rangle \rightarrow \langle t, o'' \rangle$  gives rise to a derivation  $\langle aNIL, o \rangle \rightarrow \langle NIL, o'' \rangle$ . Similarly, the derivation  $\langle \bar{a}\bar{t}, p \rangle \rightarrow \langle NIL, p' \rangle$  may be decomposed into  $\langle \bar{a}\bar{t}, p \rangle \rightarrow \langle \bar{t}, p'' \rangle$  and  $\langle \bar{t}, p'' \rangle \rightarrow \langle NIL, p' \rangle$ , with the former also giving rise to  $\langle \bar{a}NIL, p \rangle \rightarrow \langle NIL, p'' \rangle$ .

Now case (ii) may be applied to the pair  $\langle \bar{a}NIL, p \rangle \rightarrow \langle NIL, p'' \rangle$ ,  $\langle aNIL, o \rangle \rightarrow \langle NIL, o'' \rangle$  to obtain  $\langle o, p \rangle \rightarrow \langle o'', p'' \rangle$ . Induction may be applied to  $\langle t, o'' \rangle \rightarrow \langle NIL, o' \rangle$  and  $\langle \bar{t}, p'' \rangle \rightarrow \langle NIL, p' \rangle$  to obtain  $\langle o'', p'' \rangle \rightarrow \langle o', p' \rangle$ . These two observations may now be combined to obtain the required

$$\langle o, p \rangle \rightarrow \langle o', o' \rangle. \quad \square$$

The result is now obtained by combining these three lemmas.

*Proof of Theorem 1.2.1.* It is sufficient to show  $p \sim_A q$  implies  $p \sim_o q$  and to do so we prove

$$\sim_A \subseteq D_2(\sim_A, \sim_A).$$

Suppose  $p \sim_A q$  and  $\langle o, p \rangle \rightarrow \langle o', p' \rangle$  for an arbitrary observer  $o$ . From Lemma 1.2.4 there is a sequential observer  $s$  such that  $\langle s, o \rangle \rightarrow \langle NIL, o' \rangle$  and  $\langle s, p \rangle \rightarrow \langle NIL, p' \rangle$ . Applying Lemma 1.2.3 we obtain a  $q'$  such that  $p' \sim_A q'$  and  $\langle s, q \rangle \rightarrow \langle NIL, q' \rangle$ . From Lemma 1.2.5 it follows that  $\langle o, q \rangle \rightarrow \langle o', q' \rangle$ .  $\square$

There is a wide variety of modifications which might be applied to the definition of generalised observational equivalence. However, the same result will apply to each of these modifications. The essential reason is that all interactions between observers and processes can be sequentialised and therefore, by Lemma 1.2.3, any attempt to obtain an equivalence finer than  $\sim_A$  is doomed to failure.

We now proceed to show that  $\sim_A$  coincides with the standard definition of observational equivalence, as defined, for example in [Mil2].

**1.3. Reduction to observational equivalence.** In the last section we have seen that we may confine our attention to atomic observers. We may take atomic observers to be processes of the form  $NIL$  or  $aNIL$  for some observable action  $a$ . There are other atomic observers such as  $NIL | NIL$ ,  $aNIL | NIL$ ,  $\tau NIL$ ,  $a(NIL | NIL)$ , but every observer is equivalent to one in the basic form.

The effect of these atomic observers may be defined directly on processes. This gives operational semantics for our language in the usual style of [M1], [He], etc. It is in terms of next-action relations for every  $\mu \in \text{Act}$ . These are defined to be the least relations which satisfy the rules in Fig. 3. These are slightly different than the relations defined in Fig. 1. We can show that for terms not involving  $\uparrow$   $p \stackrel{\mu}{\Rightarrow} q$  if and only if  $p \xrightarrow{\tau} p' \xrightarrow{\mu} q' \xrightarrow{\tau} q$  for some  $p', q'$ ;  $\stackrel{\mu}{\Rightarrow}$  allows silent actions before and after the simple action  $\xrightarrow{\mu}$ .

LEMMA 1.3.1. (a) For  $a \neq \tau$ ,  $p \stackrel{a}{\Rightarrow} q$  if and only if  $\langle aNIL, p \rangle \rightarrow \langle NIL, q \rangle$ .

(b)  $p \stackrel{\tau}{\Rightarrow} q$  if and only if  $\langle NIL, p \rangle \rightarrow \langle NIL, q \rangle$ , where  $q$  is different from  $p$ .

*Proof.* Each direction is a proof by induction on the length of derivations.  $\square$

Using these action relations we can now define observational equivalence in the standard way. As is normal, the relation  $\stackrel{\tau}{\Rightarrow}$  is too discriminating as  $p \stackrel{\tau}{\Rightarrow} q$  means that  $p$  has definitely made some internal computation. Instead we use the weaker relation  $\stackrel{\varepsilon}{\Rightarrow}$ :  $p \stackrel{\varepsilon}{\Rightarrow} q$  means  $p$  can evolve to  $q$  by making zero or more internal communications. Formally  $p \stackrel{\varepsilon}{\Rightarrow} q$  if  $p$  is  $q$  or  $p \xrightarrow{\tau} q$ .

- (I)  $\mu p \stackrel{\mu}{\Rightarrow} p$
- (II)  $p \stackrel{\mu}{\Rightarrow} p'$  implies  $p + q \stackrel{\mu}{\Rightarrow} p'$   
 $q + p \stackrel{\mu}{\Rightarrow} p'$
- (III)  $p \stackrel{\mu}{\Rightarrow} p'$  implies  $p | q \stackrel{\mu}{\Rightarrow} p' | q$   
 $q | p \stackrel{\mu}{\Rightarrow} q | p'$   
 $p \upharpoonright q \stackrel{\mu}{\Rightarrow} p' | q$
- (IV)  $p \stackrel{\alpha}{\Rightarrow} p', q \stackrel{\alpha}{\Rightarrow} q'$  implies  $p | q \stackrel{\tau}{\Rightarrow} p' | q'$   
 $p \upharpoonright q \stackrel{\tau}{\Rightarrow} p' | q'$
- (V)  $p \stackrel{\tau}{\Rightarrow} p', p' \stackrel{\mu}{\Rightarrow} q$  implies  $p \stackrel{\mu}{\Rightarrow} q$   
 $p \stackrel{\mu}{\Rightarrow} p', p' \stackrel{\tau}{\Rightarrow} q$  implies  $p \stackrel{\mu}{\Rightarrow} q$

FIG. 3. Asynchronous actions.

DEFINITION 1.3.2. For  $R \subseteq P \times P$  let  $F(R) \subseteq P \times P$  be defined by

$$\langle p, p' \rangle \in F(R) \text{ if for every } \mu \text{ in } V\text{Act} \cup \{\varepsilon\} \text{ } p \stackrel{\mu}{\Rightarrow} q$$

implies  $p' \stackrel{\mu}{\Rightarrow} q'$  for some  $q'$  such that  $\langle p', q' \rangle \in R$ .  $\square$

Let  $\approx$  be the maximal solution to the equation

$$R = F(R).$$

THEOREM 1.3.3.  $p \sim_A q$  if and only if  $p \approx q$ .

*Proof.* It is sufficient to show

$$\begin{aligned} \sim_A &\subseteq F(\sim_A) \quad \text{and} \\ \approx' &\subseteq E_1(\approx', \approx), \\ \approx &\subseteq E_2(\approx', \approx) \quad \text{where } \approx' = \approx \cap A \times A. \end{aligned}$$

These follow in a straightforward manner from Lemma 1.3.1.  $\square$

We end this section with an equational characterisation of this relation. As usual  $\sim_A$  is not a congruence as it is not preserved by the operator  $+$ . Therefore we characterise the largest congruence contained in  $\sim_A, \sim_A^c$ . This is defined formally by

$$p \sim_A^c q \text{ if and only if for every context } C[\ ], C[p] \sim_A C[q].$$

THEOREM 1.3.4.  $\approx_A^c$  is the least congruence generated by the equations in Fig. 4.

There are many similar results in the literature. See for example [He], [DeN], [Mil3]. The only novelty here is the presence of the asynchronous parallel operator  $\upharpoonright$ .

A similar operator has been used in [Be] and related works. One advantage of this operator is that it gives a more powerful set of equations.

(A1)	$(x + y) + z = x + (y + z)$
(A2)	$x + y = y + x$
(A3)	$x + x = x$
(A4)	$x + NIL = x$
(B1)	$(x + y) \upharpoonright z = x \upharpoonright z + y \upharpoonright z$
(B2)	$(x \upharpoonright y) \upharpoonright z = x \upharpoonright (y \upharpoonright z)$
(B3)	$x \upharpoonright NIL = x$ $NIL \upharpoonright x = NIL$
(I1)	$x + \tau x = \tau x$
(I2)	$\mu \tau x = \mu x$ $\mu(x + \tau y) = \mu(x + y) + \mu y$
(X1)	$x   y = x \upharpoonright y + y \upharpoonright x$
(X2)	Let $y$ denote $\sum \{\lambda_j y_j, j \in J\}$ , $J$ a finite index set. $\mu x \upharpoonright y = \mu(x   y) + \sum \{\tau(x   y), \mu = \bar{\lambda}_j\}$

FIG. 4. A complete set of equations.

*Example 1.*  $(x | y) | z = x | (y | z)$ . This follows by repeated use of (B1), (B2):

$$\begin{aligned}
 (x | y) | z &= (x | y) \upharpoonright z + z \upharpoonright (x | y) \\
 &= (x \upharpoonright y + y \upharpoonright x) \upharpoonright z + z \upharpoonright (x | y) \\
 &= (x \upharpoonright y) \upharpoonright z + (y \upharpoonright x) \upharpoonright z + z \upharpoonright (x | y) \\
 &= x \upharpoonright (y | z) + y \upharpoonright (x | z) + z \upharpoonright (x | y) \\
 &= x \upharpoonright (y | z) + (y \upharpoonright z) \upharpoonright x + (z \upharpoonright y) \upharpoonright x \\
 &= x \upharpoonright (y | z) + (y \upharpoonright z + z \upharpoonright y) \upharpoonright x \\
 &= x \upharpoonright (y | z) + (y | z) \upharpoonright x \\
 &= x | (y | z).
 \end{aligned}$$

Note that this cannot be derived using the axioms in [He] even though such a particular instance is derivable.  $\square$

*Example 2.* The summand notation used in (X2) is justified by the equations (A1)–(A4): if  $J$  is the set  $\{1, \dots, n\}$ , then  $\sum \{p_j, j \in J\}$  stands for the term  $p_1 + \dots + p_n$ ; if  $J$  is empty it stands for  $NIL$ . The equation X2 gives rise to the more usual interleaving axiom from [Mil1], [He], which is also expressed using this summation notation:

$$\text{let } x, y \text{ denote } \sum \{\mu_i p_i, i \in I\}, \quad \sum \{\lambda_j q_j, j \in J\}$$

where both  $I$  and  $J$  are finite sets. Then

$$x | y = \sum \{\mu_i (p_i | y), i \in I\} + \sum \{\lambda_j (x | q_j), j \in J\} + \sum \{\tau(x_i | y_j), \mu_i = \bar{\lambda}_j\}.$$

This follows because

$$\begin{aligned}
 \text{(X1)} \quad x | y &= x \upharpoonright y + y \upharpoonright x \\
 \text{(B1)} \quad &= \sum \{\mu_i x_i \upharpoonright y, i \in I\} + \sum \{\lambda_j y_j \upharpoonright x, j \in J\} \\
 &= \sum \{\mu_i (x_i | y), i \in I\} + \sum \{\tau(x_i | y_j), \mu_i = \lambda_j\} \\
 &\quad + \sum \{\lambda_j (y_j | x), j \in J\} + \sum \{\tau(x_i | y_j), \mu_i = \lambda_j\}
 \end{aligned}$$

by (X2) and reorganisation.  $\square$

This example shows that the reduction of concurrency to nondeterminism occurs via the equation (X2). In the next section, where we develop a semantic equivalence for which this reduction no longer holds, this equation is no longer valid.

We will merely outline the proof of this characterisation theorem as the proofs of many similar theorems already appear in the literature. We use  $=_1$  to denote the least congruence generated by the equations.

(1) The first step is to obtain a simple characterisation of  $\sim_A^c$ .

Let  $p \approx^+ q$  if  $aNIL + p \approx aNIL + q$  for some  $a$  not appearing in  $p, q$ . Then we can show that  $p \sim_A^c q$  if and only if  $p \approx^+ q$ .

(2) *Soundness.* We can now show that each of the equations satisfies  $\approx^+$  from which it follows that  $p =_1 q$  implies  $p \sim_A^c q$ .

(3) *Derivation lemma.* If  $p \xrightarrow{\mu} q$  then  $p =_1 p + \mu q$ . This is proved by induction on the derivation of  $p \xrightarrow{\mu} q$ .

(4) The characterisation of  $\sim_A^c$  above makes it easy to show that  $p \sim_A q$  implies (i)  $p \sim_A^c q$ , or (ii)  $\tau p \sim_A^c q$ , or (iii)  $p \sim_A^c \tau q$ .

(5) *Completeness.* Using the equations (B1)-(B4), (X1) and (X2) all occurrences of  $|, \uparrow,$  may be eliminated from terms. Such terms are called *sumforms* and it is sufficient to show  $p \sim_A^c q$  implies  $p =_1 q$  when  $p$  and  $q$  are sumforms. Let  $q$  be of the form  $\sum \mu_i q_i$ . Then by symmetry it is sufficient to show  $p =_1 p + \mu_i q_i$  for each index  $i$ . Now  $q \xrightarrow{\mu_i} q_i$  and so  $p \xrightarrow{\mu_i} p'$  for some  $p'$  such that  $q_i \sim_A p'$ . From (4) and induction  $p' =_1 q'$ ,  $\tau p' =_1 q'$  or  $p' =_1 \tau q'$ . Applying Axiom (I2) if necessary, we have in each case that  $\mu p' =_1 \mu_i q_i$ . By (3) above it follows that  $p =_1 p + \mu_i q_i$ .  $\square$

**2. Adding time.** In this section we develop a version of bisimulation equivalence based on actions or communications which are not necessarily instantaneous. The result will be a semantic theory in which concurrency is not reducible to nondeterminism. The basic idea has already been outlined in the Introduction. Rather than assigning specific lengths of time to actions, we simply assume that there are observers which can detect the beginnings and endings of actions. We could use these basic observers to develop a general theory of observation as in § 1.1. However, a theorem similar to Theorem 1.2.1 would show once again that atomic observers suffice. Accordingly we develop the new equivalence along the lines of standard observational equivalence, as in § 1.3. This is the topic of § 2.1, where the main theorem is also stated. This is an equational characterisation of the new equivalence. The next section is devoted to the proof of this result. In the final section we discuss variations on the new definition of equivalence and address some natural questions which arise.

**2.1.** We now assume that the beginning of an action and the termination of an action are two distinct events. Moreover these distinct events may be observed. Let  $S(a)$  be an observer which can detect the start of the action  $a$  and  $F(a)$  one which can detect its termination. Referring to § 1.3 we can simply view  $S(a), F(a)$  as a new class of events and define an operational semantics in terms of next-event relations  $\xrightarrow{S(a)}, \xrightarrow{F(a)}$ . We will also continue to use complete actions, as given by  $\xrightarrow{\mu}$ .

When we assumed actions were instantaneous, the language used to describe processes was also sufficiently expressive to also describe possible states which a process might reach. This is no longer the case. For example, we cannot describe the state  $ap|bq$  attains when it initiates an  $a$  action but before the action terminates.

We propose a simple solution to this problem. For each visible action  $a$  we add a new prefixing operator  $a_s$ . For example,  $a_s p | b_s q$  denotes a process which has initiated an  $a$  action and a  $b$  action but has terminated neither. We will continue to use  $p, q$ , etc., to range over processes, terms in the basic language, and  $s, s'$ , etc., to range over the extended language of states. Let  $E = \{S(a), F(a), a \in VAct\} \cup Act$ ;  $E$  is the set of events. We use  $\forall E$  to denote the related set  $(E - \{\tau\}) \cup \{\varepsilon\}$ . The operational semantics is given in terms of a set of next-event relations  $\xrightarrow{e}$ , one for each  $e$  in  $E$ . They are



defined to be the least relations over states which satisfy the rules in Fig. 5. Rule (I) is the important new case. The process  $ap$  can initiate an  $a$  action and thereby be transformed into the intermediate state  $a_s p$ . This intermediate state can terminate the  $a$  action and thereby be transformed into  $p$ . The remaining rules are taken from Fig. 3.

A standard observational equivalence may now be defined using these event-relations. If  $R$  is a relation over states, define  $G(R)$  by:

$$\langle s, s' \rangle \in G(R) \text{ if for every } e \text{ in } VE \text{ } s \xrightarrow{e} t \text{ implies } s' \xrightarrow{e} t \text{ for some } t' \text{ such that } \langle t, t' \rangle \in R.$$

Let  $\approx_T$  be the maximal symmetric solution to the equation

$$R = G(R).$$

This equivalence relation we call *t-observational equivalence*. We are of course only interested in it as it applies to processes but, as we have seen, states are necessary to describe intermediate configurations.

We now give some examples of equivalent and inequivalent processes.

*Example 1.*  $aNIL|bNIL \not\approx_T abNIL + baNIL$ . This follows intuitively because  $aNIL|bNIL$  can initiate an  $a$  action and subsequently can initiate a  $b$  action without having terminated the  $a$  action; this is not possible of  $abNIL + baNIL$ . Formally  $aNIL|bNIL \xrightarrow{S(a)} a_s NIL|bNIL$  and  $abNIL + baNIL$  cannot perform a similar  $S(a)$ -move. The only possible  $S(a)$ -move is  $abNIL + baNIL \xrightarrow{S(a)} a_s bNIL$  and obviously  $a_s NIL|bNIL \not\approx_T a_s bNIL$ ; one can perform an  $S(b)$ -move which is impossible of the other.  $\square$

*Example 2.* Let  $p$  denote the process

$$p_1|(q_1 + q_2) + p_2|(q_1 + q_2) + (p_1 + p_2)|q_1 + (p_1 + p_2)|q_2$$

where  $p_1, p_2, q_1, q_2$  are arbitrary process then  $p \approx_T p + (p_1 + p_2)|(q_1 + q_2)$ .

- |       |  |
|-------|--|
| (I)   | $ap \xrightarrow{S(a)} a_s p$  |
|       | $a_s p \xrightarrow{F(a)} p$   |
| (II)  | $\mu p \xrightarrow{\mu} p$  |
| (III) | $s_1 \xrightarrow{e} s'_1 \text{ implies } s_1 + s_2 \xrightarrow{e} s'_1$   |
|       | $s_2 + s_1 \xrightarrow{e} s'_2$   |
| (IV)  | $s_1 \xrightarrow{e} s'_1 \text{ implies } s_1   s_2 \xrightarrow{e} s'_1   s_2$                                     |
|       | $s_2   s_1 \xrightarrow{e} s_2   s'_1$   |
|       | $s_1 \uparrow s_2 \xrightarrow{e} s'_1   s_2$  |
| (V)   | $s_1 \xrightarrow{a} s'_1, s_2 \xrightarrow{\bar{a}} s'_2 \text{ implies } s_1   s_2 \xrightarrow{\tau} s'_1   s'_2$ |
|       | $s_1 \uparrow s_2 \xrightarrow{\tau} s'_1   s'_2$  |
| (VI)  | (a) $s \xrightarrow{e} s$  |
|       | (b) $s_1 \xrightarrow{\tau} s'_1, s'_1 \xrightarrow{e} s_2 \text{ implies } s_1 \xrightarrow{e} s_2$                 |
|       | (c) $s_1 \xrightarrow{e} s'_1, s'_1 \xrightarrow{\tau} s_2 \text{ implies } s_1 \xrightarrow{e} s_2.$                |

FIG. 5. Events.

Note that this equality follows from the axioms (X1), (B2), (A1)–(A3).  $\square$

*Example 3.*  $abNIL + aNIL | bNIL \not\approx_{\tau} aNIL | bNIL$ . Intuitively the left-hand side can, by starting an  $a$ -action, get into a state in which it cannot perform a  $b$ -action without first terminating the  $a$ -action. This is impossible of the right-hand side.

*Example 4.*  $aNIL | (bNIL + cNIL) \not\approx_{\tau} aNIL | bNIL + aNIL | cNIL + a(bNIL + cNIL)$ . Intuitively this follows because in the right-hand side the decision to perform a  $b$ -action or a  $c$ -action must be taken either before the action  $a$  has commenced or after it has finished. This restriction does not apply to the left-hand side. Formally this can perform an  $S(a)$ -move to be transformed into the state  $a_sNIL | (bNIL + cNIL)$  and the right-hand side cannot be transformed into an equivalent state.  $\square$

The equivalence  $\approx_{\tau}$  is not a congruence for the usual reasons associated with the nondeterministic operator  $+$ :  $aNIL \approx_{\tau} \tau aNIL$  but  $bNIL + aNIL \not\approx_{\tau} bNIL + \tau aNIL$ . So when wishing to associate the new equivalence with equational theories we consider as usual the largest congruence contained in  $\approx_{\tau}$ , denoted  $\approx_{\tau}^c$ . This also has the following usual characterisation.

LEMMA 2.1.1.  $p \approx_{\tau}^c q$  if and only if  $p + aNIL \approx_{\tau} q + aNIL$  for some visible action  $a$  not occurring in  $p, q$ .  $\square$

A useful corollary to this characterisation is that if  $p \approx_{\tau}^c q$ , then both  $p$  and  $q$  must have matching  $\tau$ -moves:  $p \xrightarrow{\tau} p'$  implies  $q \xrightarrow{\tau} q'$  for some  $q'$  such that  $p' \approx_{\tau} q'$ , and vice versa for  $q$ . This will be useful in the sequel.

We now turn our attention to the problem of providing an equational characterisation of  $\approx_{\tau}^c$ . A natural place to start is with the equations in Fig. 4. As might be expected, (X2) is *not* satisfied by  $\approx_{\tau}^c$ . For example

$$aNIL \upharpoonright bNIL \approx_{\tau} a(NIL | bNIL)$$

as may be readily checked. Condition (X2) is the basis of “interleaving” or the reduction of concurrency to nondeterminism, associated with the standard observational equivalence. So its failure is not surprising. However, it remains true when  $\mu$  is  $\tau$ :

$$\tau x \upharpoonright y = \tau(x | y).$$

A related equation satisfied by  $\approx_{\tau}^c$  is

$$x \upharpoonright \tau y = x \upharpoonright y.$$

These two, together with (X1), imply the natural equation

$$\tau x | y = \tau(x | y).$$

All the other equations in Fig. 4 remain satisfied by  $\approx_{\tau}^c$  *except* (I3). For example,

$$a(bNIL + \tau cNIL) \not\approx_{\tau} a(bNIL + \tau cNIL) + acNIL.$$

The right-hand side can perform an  $S(a)$ -move to the state  $a_s cNIL$ ; no equivalent state can be reached by the left-hand side. Once more (I3) remains satisfied when  $\mu$  is  $\tau$ :

$$\tau(x + \tau y) = \tau(x + \tau y) + \tau y.$$

However this is already derivable from (I1), using (A1)–(A3):

$$\begin{aligned} \tau(x + \tau y) &= \tau(x + \tau y) + x + \tau y \\ &= \tau(x + \tau y) + x + \tau y + \tau y \\ &= \tau(x + \tau y) + \tau y. \end{aligned}$$

However there is an equation very similar in style to (I3) which is true of  $\approx_T^c$ :

$$x \upharpoonright (y + \tau z) = x \upharpoonright (y + \tau z) + x \upharpoonright z.$$

This revised set of equations is not complete for  $\approx_T^c$  as it ignores the possibility of communication between subprocesses. A natural additional equation would be

$$(*) \quad ax \mid \bar{a}y = ax \mid \bar{a}y + \tau(x \mid y).$$

Unfortunately, the addition of this equation will still not give a complete theory. As (X1) shows,  $\upharpoonright$  plays a more fundamental role than does  $\mid$ . What is required is a reformulation of (\*) in terms of  $\upharpoonright$ . The additional equation we use is

$$(C) \quad ax_1 \upharpoonright (\bar{a}x_2 \upharpoonright y + z) = ax_1 \upharpoonright (\bar{a}x_2 \upharpoonright y + z) + \tau(x_1 \mid x_2 \mid y).$$

We collect all the required equations in Fig. 6. Let  $\equiv$  denote the generated congruence. The main result of the paper may now be stated.

**THEOREM 2.1.2.**  *$P \approx_T^c q$  if and only if  $p \equiv q$ .*

The proof of this theorem is the subject of the next section.

**2.2. Proof of the equational characterisation.** We take for granted the alternative characterisation of  $\approx_T^c$  given in Lemma 2.1.1, although checking this would involve the reader in very tedious calculations. For example, it is necessary to check that  $\approx_T$  is preserved by all the operators except  $+$ . However, all of the calculations are straightforward and are simple variations on those found in references such as [He], [Mil1], [Mil2].

(A1)	$(x + y) + z = x + (y + z)$
(A2)	$x + y = y + x$
(A3)	$x + x = x$
(A4)	$x + NIL = x$
(B1)	$(x + y) \upharpoonright z = x \upharpoonright z + y \upharpoonright z$
(B2)	$(x \upharpoonright y) \upharpoonright z = x \upharpoonright (y \upharpoonright z)$
(B3)	$x \upharpoonright NIL = x$ $NIL \upharpoonright x = NIL$
(I1)	$x + \tau x = \tau x$
(I2)	$\mu \tau x = \mu x$
(NI3)	$x \upharpoonright (y + \tau z) = x \upharpoonright (y + \tau z) + x \upharpoonright z$
(X1)	$x \mid y = x \upharpoonright y + y \upharpoonright x$
(NX2)	$\tau x \upharpoonright y = \tau(x \mid y)$ $x \upharpoonright \tau y = x \upharpoonright y$
(C)	$ax_1 \upharpoonright (\bar{a}x_2 \upharpoonright y + z) = ax_1 \upharpoonright (ax_2 \upharpoonright y + z) + \tau(x_1 \mid x_2 \mid y)$

FIG. 6. Complete equations for  $\approx_T^c$ .

We also leave it to the reader to use this characterisation to check that all the equations are satisfied by  $\approx_T^c$ . Once more the calculations are straightforward. We therefore concentrate on completeness, showing  $p \approx_T^c q$  implies  $p \equiv q$ . This proof follows the general outline (given at the end of § 1.3) of the previous completeness proof but in this case the details are more difficult. However the following ingredient is still straightforward.

LEMMA 2.2.1.  $p \approx_T q$  implies

- (i)  $p \approx_T^c q$ , or
- (ii)  $\tau p \approx_T^c q$ , or
- (iii)  $p \approx_T^c \tau q$ .

*Proof.* Let us assume that  $p \approx_T q$  and let  $a$  be a visible action not occurring in  $p, q$ .

(a) Suppose that  $p \xrightarrow{\tau} p'$  for some  $p'$  such that  $p' \approx_T q$ . In this case we can show that  $a + p \approx_T a + q$ .

(b) By symmetry if  $q \xrightarrow{\tau} q'$  for some  $q'$  such that  $q' \approx_T p$ , we can show that  $a + \tau p \approx_T a + q$ .

(c) If neither (a) nor (b) is possible it follows directly that  $a + p \approx_T a + q$ .  $\square$

This result will be used in a manner similar to that of the completeness proof of § 1.3.

We also need a corresponding derivation lemma. However it will be more straightforward to establish this for only a subset of process terms. This subset of terms, called the simple forms, will play the role of the sumforms in the previous completeness proof. These are now explained.

The axiom (X1) enables us to eliminate all occurrences of the parallel operator  $|$  from terms, at the expense of introducing the asymmetric operator. This cannot be eliminated; otherwise we could once more reduce concurrency to nondeterminism. However we can always reduce terms to a particularly simple form.

DEFINITION 2.2.2. (i) *NIL* is a *simple form* (sf).

(ii) If each  $p_i, p'_i, p_j$  are sfs, then

$$\sum \{a_i p_i \upharpoonright p'_i, i \in I\} + \sum \{\tau p_j, j \in J\}$$

is also an sf where  $I, J$  are finite index sets.  $\square$

PROPOSITION 2.2.3. *Every process term can be reduced to a simple form, i.e., for every  $p$  there exists an sf  $\text{sf}(p)$  such that  $p \equiv \text{sf}(p)$ .*

*Proof.* The proof is by induction on the size of  $p$ . We proceed by case analysis on the structure of  $p$ .

(i)  $p$  is  $ap'$ . The required sf is  $ap' \upharpoonright \text{NIL}$ .

(ii)  $p$  is  $p' \upharpoonright p''$ . By induction suppose  $\text{sf}(p')$  is  $\sum a_i q_i \upharpoonright q'_i + \sum \tau q_j$ . Then

$$\begin{aligned} p &= {}_2 \sum (a_i q_i \upharpoonright q'_i) \upharpoonright p'' + \sum (\tau q_j) \upharpoonright p'' \quad \text{by repeated use of (B1)} \\ &= {}_2 \sum a_i q_i \upharpoonright (q'_i | p'') + \sum \tau (q_j | p'') \quad \text{by repeated use of (B2), (I4)}. \end{aligned}$$

By induction we may assume  $\text{sf}(q'_i | p'')$  and  $\text{sf}(q_j | p'')$  exists for each  $i, j$ . Therefore the required sf is

$$\sum a_i q_i \upharpoonright \text{sf}(q'_i | p'') + \sum \tau \text{sf}(q_j | p'').$$

(iii) The remaining cases are similar.  $\square$

We now proceed to state an appropriate version of the Derivation Lemma. In the actual completeness proof we are only interested in derivations with respect to  $\tau$ -actions and  $S(a)$ -actions; in the proof we will only need to absorb these kinds of derivatives. We therefore only need a derivation lemma for these kinds of actions. Moreover the definition of  $\xrightarrow{\tau}$  is completely independent of that of  $\xrightarrow{S(a)}$ . So we first prove the required result for  $\xrightarrow{\tau}$  and then we prove the corresponding result for  $\xrightarrow{S(a)}$ .

PROPOSITION 2.2.4 (Derivation Lemma). *If  $p$  is a simple form*

$$p \xrightarrow{\tau} p' \quad \text{implies } p \equiv p + \tau q.$$

*Proof.* The proof is by induction on the length of the derivation of  $p \xrightarrow{\tau} p'$ . Let us assume that  $p$  has the form  $\sum a_i p_i \upharpoonright p'_i + \sum \tau p'_j$ .

- (a)  $p'$  is  $p_j''$  for some  $j$ . Using the equations (A1)-(A4), the proof is immediate.  
 (b)  $p_j'' \xrightarrow{\tau} p'$ , for some  $j$ . By induction  $p_j'' \equiv p_j'' + \tau p'$ . It follows from (I1) that  $\tau p_j'' \equiv \tau p_j'' + \tau p'$  from which  $p \equiv p + \tau p'$  is immediate.  
 (c)  $a_i p_i \uparrow p_i' \xrightarrow{\tau} p'$  for some  $i$ . In this case  $p'$  has the form  $x|y$ , where  $p_i \xrightarrow{\varepsilon} x$  and  $p_i' \xrightarrow{\bar{a}_i} y$ .

There are two subcases, which we examine separately.

(i)  $x$  is  $p_i$ . Here we have a further inductive argument. This time we prove  $a_i p_i \uparrow p_i' \equiv a_i p_i \uparrow p_i' + \tau(x|y)$  by induction on the number of applications of rule (VI) in the derivation of  $p_i' \xrightarrow{\bar{a}_i} y$ . Let us assume  $p_i'$  has the form  $\sum b_j q_j \uparrow q_j' + \sum \tau q_k'$ .

(a) The number of applications is zero. In this case  $y$  is  $q_j|q_j'$  for some  $j$  such that  $b_j$  is  $\bar{a}_i$ . We may now apply the new equation (C) to obtain

$$a_i p_i \uparrow \bar{p}_i' \equiv a_i p_i \uparrow \bar{p}_i' + \tau(p_i|q_j|q_j').$$

(b) Otherwise  $q_k' \xrightarrow{\bar{a}_i} y$  for some  $k$  and also  $p_i' + q_k' \xrightarrow{\bar{a}_i} y$ . Moreover the number of applications of rule (VI) in this derivation is strictly less than in the derivation of  $p_i' \xrightarrow{\bar{a}_i} y$ . So we may apply induction to obtain

$$a_i p_i \uparrow (p_i' + q_k') \equiv a_i p_i \uparrow (p_i' + q_k') + \tau(x|y).$$

However  $p_i' \equiv p_i' + \tau q_k'$  from which it follows that  $p_i' \equiv p_i' + q_k'$  by (I1). The required result now follows.

(ii) The second case is when  $p_i \xrightarrow{\tau} x$ .

By case (i) we know

$$a_i p_i \uparrow p_i' \equiv a_i p_i \uparrow p_i' + \tau(p_i|y).$$

To complete the proof we now show

$$\tau(p_i|y) \equiv \tau(p_i|y) + \tau(x|y).$$

By induction we know  $p_i \equiv p_i + \tau x$ . Therefore

$$\begin{aligned} p_i|y &\equiv p_i|y + p_i \uparrow y && \text{by (X1)} \\ &\equiv p_i|y + (p_i + \tau x) \uparrow y \\ &\equiv p_i|y + (\tau x) \uparrow y && \text{by (B1)} \\ &\equiv p_i|y + \tau(x|y) && \text{by (NX2)} \end{aligned}$$

By applying (I1) we now obtain the required  $\tau(p_i|y) \equiv \tau(p_i|y) + \tau(x|y)$ .  $\square$

The proof of the corresponding result for  $\xrightarrow{S(a)}$  is much more straightforward.

**COROLLARY 2.2.5 (Derivation Lemma).** *If  $p$  is a simple form  $p \xrightarrow{S(a)} a_s p_1 | p_2$  implies  $p \equiv p + a p_1 \uparrow p_2$ .*

*Proof.* The proof is by induction on why  $p \xrightarrow{S(a)} a_s p_1 | p_2$ . Let us assume that  $p$  has the form  $\sum a_i p_i \uparrow p_i' + \sum \tau p_j$ .

(i) For some  $i$ ,  $a_i p_i \uparrow p_i' \xrightarrow{S(a)} a_s p_1 | p_2$ . Then  $a_i p_i$  is  $a p_1$  and  $p_i' \xrightarrow{\varepsilon} p_2$ . If  $p_i'$  is  $p_2$  we are finished, so assume  $p_i' \xrightarrow{\tau} p_2$ . By the previous result  $p_i' \equiv p_i' + \tau p_2$ . Therefore by (NI3)  $a_i p_i \uparrow p_i' \equiv a_i p_i \uparrow p_i' + a p_1 \uparrow p_2$ .

(ii) For some  $j$ ,  $p_j \xrightarrow{S(a)} a_s p_1 | p_2$ . By induction  $p_j \equiv p_j + a p_1 \uparrow p_2$  and from (I1) it follows that  $\tau p_j \equiv \tau p_j + a p_1 \uparrow p_2$ .  $\square$

We need one subsidiary, technical result before proving the Completeness Theorem. What we actually want to show is

$$(*) \quad a_s p | p' \approx_{\tau} a_s q | q' \text{ implies } p \approx_{\tau} q \text{ and } p' \approx_{\tau} q'.$$

However, in order to use induction, we will prove a stronger result. Its statement requires the introduction of some new notation. If  $X$  is a multi-set of terms  $\{t_1, \dots, t_k\}$ , we use  $\prod X$  to denote the product term  $t_1 | t_2 \cdots | t_k$ . Because of the commutativity and associativity of  $|$ , the exact enumeration of  $X$  does not matter. Also if  $k = 0$ , then  $\prod X$  denotes  $NIL$ . If  $X, Y$  denote multi-sets of the form  $\{a_{is}p_i, i \in I\}, \{a_{js}q_j, j \in J\}$ , respectively, we write  $X < Y$  to mean there is a one-to-one correspondence  $f: I \rightarrow J$  such that for every  $i$  in  $I$   $p_i \approx_{\tau} y_i$  for some  $i$  such that  $q_{f(i)} \xrightarrow{\varepsilon} y_i$ . We abbreviate  $\{t\} < \{s\}$  to  $t < s$  and it is simple to show that  $<$  is transitive on terms. This transitivity is used to prove the following general result about  $<$ .

LEMMA 2.2.6. *If  $X, Y$  are multi-sets, then  $Y < X$  and  $X \cup \{s\} < Y \cup \{t\}$  implies  $s < t$ .*

*Proof.* The proof is by induction on the size of  $X$  (and therefore also of  $y$ ). If  $X = \emptyset$ , then the result is immediate. If  $X = \{x\}$  then  $Y = \{y\}$  and  $y < x$ . There are two possible correspondences between  $\{x, s\}$  and  $\{y, t\}$ . One immediately gives  $s < t$ ; the other gives  $s < y, x < t$  and, by transitivity, we obtain  $s < t$ . Otherwise  $X$  contains at least two elements. Let  $y_i \rightarrow x_i$  be the correspondence from  $Y$  to  $X$ . Consider the correspondence from  $X \cup \{s\}$  to  $Y \cup \{t\}$ . If this maps  $s$  to  $t$  we immediately have  $s < t$ . Otherwise, by renaming, if necessary, we can assume  $s$  is mapped to  $y_1$ . If  $x_1$  is mapped to  $t$  we have  $s < t$ , again by transitivity. Otherwise assume  $x_1$  is mapped to  $y_2$ . Let  $X', Y'$  denote  $X - \{x_1, x_2\}, Y - \{y_1, y_2\}$ , respectively. Then  $X' \cup \{x_2\} < Y' \cup \{t\}$  and  $Y' < X'$ . By induction we have  $x_2 < t$  and therefore, once more by transitivity, we have  $s < t$ .  $\square$

LEMMA 2.2.7.  $\prod \{a_{is}p_i, i \in I\} | p \approx_{\tau}^c \prod \{b_{js}q_j, j \in J\} | q$  implies  $\{a_{is}p_i, i \in I\} < \{b_{js}q_j, j \in J\}$  and  $p \approx_{\tau} q$ .

*Proof.* The proof is by induction on the combined size of the terms. We use the alternative characterisation of  $\approx_{\tau}^c$  and the characterisation of  $\approx_{\tau}$  in Lemma 2.2.1. For convenience we denote the multi-sets  $\{a_{is}p_i, i \in I\}, \{b_{js}q_j, j \in J\}$  by  $X, Y$ , respectively, and the corresponding product terms by  $x, y$ .

(a)  $p \approx_{\tau}^c NIL$ . Then  $q \approx_{\tau} NIL$ , for otherwise the right-hand side could perform a complete action which could not be matched by the left-hand side. It follows that  $x \approx y$ . We must show  $X < Y$ .

Let  $a$  denote  $a_1$  for some  $1 \in I$ . Then  $x \xrightarrow{F(a)} p_1 | x'$ , where  $x'$  is  $\prod \{a_i, i \in I - \{1\}\}$ . So for some  $k \in J$ , such that  $b_k$  is  $a, y \xrightarrow{F(a)} q' | y'$ , where  $y'$  is defined in the obvious way from  $y$  and  $k, q_k \xrightarrow{\varepsilon} q'$  and  $p_1 | x' \approx q' | y'$ . Applying Lemma 2.2.1, we have three possibilities of which we will consider only one:  $p_1 | x' \approx_{\tau}^c \tau(q' | y')$ . Using the equation  $\tau(x | y) = x | \tau y$  this may be rewritten as  $p_1 | x' \approx_{\tau}^c \tau q' | y'$ . We may now apply induction because the combined size of the terms has decreased. We obtain  $X' < Y'$  and  $p_1 \approx \tau q'$ , where the sets  $X', Y'$  are defined in the obvious manner. Now  $p_1 \approx \tau q' \approx q'$  implies  $a_s p_1 < a_s q_k$  from which  $X < Y$  follows.

(b)  $p \not\approx_{\tau}^c NIL$ . Then for some action  $\mu, p \xrightarrow{\mu} p'$ . So  $x | p \xrightarrow{\mu} x | p'$ . Because  $x | p \approx_{\tau}^c y | q$ , it follows that  $p \xrightarrow{\mu} q'$  for some  $q'$  such that  $x | q' \approx y | p'$ . Applying Lemma 2.2.1 and induction we obtain, at least,  $X < Y$ . We must show  $p \approx q$ .

The argument we have just completed shows that every complete action of  $p$  (and by symmetry of  $q$ ) may be matched by a corresponding action of  $q$  (respectively  $p$ ). It remains to consider initial events. This argument may also be used, by considering a complete action from  $q$ , to show  $Y < X$ . In the final argument about initial events this fact is used. Suppose  $p \xrightarrow{S(a)} z$ . It is not too difficult to show  $z$  must be of the form  $a_s r | r'$ . Then  $x | p \xrightarrow{S(a)} x | a_s r | r'$ . So  $q \xrightarrow{S(a)} a_s w | w'$  such that  $y | a_s w | w' \approx_{\tau} x | a_s r | r'$ . We must show  $a_s w | w' \approx_{\tau} a_s r | r'$ . We do this by applying induction to  $y | a_s w | w' \approx_{\tau} x | a_s r | r'$ . Lemma 2.2.1 gives three possibilities of which we examine one. Suppose  $\tau(y | a_s w | w') \approx_{\tau}^c x | a_s r | r'$ , i.e.,  $y | a_s w | \tau w' \approx_{\tau}^c x | a_s r | r'$ . By induction  $\tau w' \approx_{\tau} r'$  and  $X \cup$

$\{a_s r\} < Y \cup \{a_s w\}$ . By the previous lemma  $a_s r < a_s w$ . However induction may also be applied to obtain  $Y \cup \{a_s w\} < X \cup \{a_s r\}$ . Since  $X < Y$  we also have  $a_s w < a_s r$ . The two facts  $a_s r < a_s w$  and  $a_s w < a_s r$  imply  $a_s w \approx_T a_s r$  and therefore  $a_s w | w' \approx_T a_s r | r'$ .

A symmetrical argument will show that every initial event of  $q$  may be matched by one from  $p$ . It follows that  $p \approx_T q$ .  $\square$

After this rather delicate proof the following is now straightforward.

**PROPOSITION 2.2.8.**  $a_s p | p' \approx_T a_s q | q'$  implies  $p \approx_T q$  and  $p' \approx_T q'$ .

*Proof.* The proof is by a direct application of the previous lemma, if we first apply Lemma 2.2.1. For example, this might give  $a_s p | p' \approx_T^c a_s q | \tau q'$  from which we obtain  $a_s p < a_s q$ ,  $a_s q < a_s p$ , and  $p' \approx_T \tau q'$ , i.e.,  $p' \approx_T q'$ . The first result implies  $q \xrightarrow{e} q''$  for some  $q''$  such that  $p \approx_T q''$  and the second implies  $p \xrightarrow{e} p''$  for some  $p''$  such that  $p'' \approx_T q$ . These two imply in turn that  $p \approx_T q$ .  $\square$

We are now ready for the completeness theorem.

**THEOREM 2.2.9 (Completeness).**  $P \approx_T^c q$  implies  $p \equiv q$ .

*Proof.* We may assume both  $p$  and  $q$  are simple forms and the proof is by induction on the combined size of the terms. We show  $p + q \equiv q$ . By symmetry  $p \equiv q$  will follow.

Suppose  $p$  has the form  $\sum a_i p_i | p'_i + \sum \tau p_j$ . We show  $q \equiv q + a_i p_i | p'_i$  and  $q \equiv q + \tau p_j$ .

(a)  $p \xrightarrow{S(a)} a_{is} p_i | p'_i$  so  $q \xrightarrow{S(a)} x$  for some  $x$  such that  $x \approx_T a_{is} p_i | p'_i$ . Using the operational semantics we can prove that  $x$  must have the form  $a_{is} q_1 | q_2$ . Applying the previous proposition we have  $p_i \approx_T q_1$  and  $p'_i \approx_T q_2$ . Now using Lemma 2.2.1, possibly (I2), and induction, we obtain  $a_i p_i \equiv a_i q_1$ . Applying Lemma 2.2.1 to  $p'_i \approx_T q_2$  one typical possibility is that  $\tau p'_i \approx_T q_2$ . Applying induction we obtain  $\tau p'_i \equiv q_2$  and by (NX2) we obtain  $a_i p_i | p'_i \equiv a_i q_1 | q_2$ .

It remains to show that  $q \equiv q + a_i q_1 | q_2$ . However, this follows by the second Derivation Lemma.

(b)  $p \xrightarrow{\tau} p_j$ . So  $q \xrightarrow{\tau} q'$  for some  $q'$  such that  $p_j \approx_T q$ . Again applying Lemma 2.2.1, induction, and possibly (I2), we have  $\tau p_j \equiv \tau q'$ . From the first Derivation Lemma we have  $q \equiv q + \tau q'$  and therefore  $q \equiv q + \tau p_j$ .  $\square$

**2.3. Remarks.** The equations in Fig. 6 are robust in the sense that if we make various changes to the notion of observation they remain complete. We look briefly at two possibilities. The observational equivalence  $\approx_T$  is based on the partitioning of time into three distinct phases with respect to a particular action: before the action, during the action, and after the action. This can be generalised to an arbitrary number of phases. In this generalisation we are allowed to examine a process an arbitrary number of times during the execution of an action. This is formalised by adding to the alphabet new prefixing operators  $a_n$ , for every visible action  $a$  and for every  $n > 0$ . The operational semantics is given by the rules

$$\begin{aligned} ap &\xrightarrow{S(a)} a_n p \quad \text{for any } n > 0 \\ a_1 p &\xrightarrow{F(a)} p \\ a_{n+2} p &\xrightarrow{D(a)} a_{n+1} p \end{aligned}$$

together with the usual rules for complete actions, communication and  $+$  and  $|$ . Let  $\sim_T$  be the observational equivalence generated in the usual way using this operational semantics. Then we can prove that for processes  $p, q$   $p \approx_T^c q$  if and only if  $p \sim_T q$ ; the extra increase in the power of observation does not lead to a finer congruence.

Another possibility is to allow observers to perform multiple experiments. This is formalised by defining relations  $\xrightarrow{M}$ , where  $M$  is a multiset of events. We omit the

details of the definition as they are rather complicated and tedious. However, the resulting observational congruence coincides once more with  $\approx_T^c$ .

The hypothesis that each action has a beginning and an end which are distinct from each other seems to be the simplest way of associating time with actions. These two examples indicate that this gives us considerable descriptive power, making redundant more detailed analysis of the time-dependent behaviour of processes.

For the time-based equivalence  $\approx_T$  the “interleaving axiom” no longer holds for processes. However, this was achieved by hypothesising observers which are more discriminating than processes. These observers are more discriminating in that they can find differences where processes cannot. At this more elementary level we can investigate observers observing observers, using instantaneous actions once more to define an equivalence, as in § 1. At this level “interleaving” will once more be valid. We can then consider processes as particular kinds of observers and at least for simple processes  $\approx_T$  can be derived from the more elementary interleaving equivalence on observers,  $\sim_A$ . The basic idea is to translate processes into observers by

$$\begin{aligned} \text{tr}: ap &\rightarrow S(a)F(a) \text{tr}(p) \\ p + p' &\rightarrow \text{tr}(p) + \text{tr}(p') \\ p | p' &\rightarrow \text{tr}(p) | \text{tr}(p'). \end{aligned}$$

In the special case where we do not allow communication we can then prove

$$p \approx_T q \text{ iff } \text{tr}(p) \sim_A \text{tr}(q).$$

It is not known if this is true when communication is allowed. However it seems that in this way “interleaving semantics” will often be sufficient to explain more complex notions of behaviour of processes.

Although it is sufficient it may be unnecessary and inconvenient. It would be preferable to be able to manipulate processes without having to examine their behaviour at more detailed levels. In the case examined in this paper we have shown that this is possible. The axioms of Fig. 6 are sufficient to derive all pairs of observationally congruent, ( $\approx_T^c$ ), processes. It is therefore unnecessary to translate them into the observer language and use the axioms at this more elementary level. Proofs using this method would in general be long since they involve expansion of terms using interleaving. Moreover each variation in the semantics, for example as outlined above, would necessitate new translations into new observer languages and new axioms for these underlying languages.

Finally, let us refer to some related work. Many models of concurrent processes have been proposed where concurrency is independent of nondeterminism. Perhaps the best known are Petri nets [Re]. However, this is a very operationally oriented model and it is difficult to impose on it an algebraic framework. Some attempts have been made (see [Go], for example), but the resulting equivalence on terms is very weak; although concurrency and nondeterminism are differentiated processes, such as  $aNIL + aNIL$  and  $aNIL$ , are also distinguished. The resulting model of processes is not very abstract. Similar remarks may be made of the work of Mazurkiewicz [Ma]. The related work on event structures [Wi] is very much concerned with developing algebraic models for concurrency but no attempt has yet been made to introduce into these structures a notion of abstract behaviour.

Recently there have been other attempts at generalising observational equivalence to take concurrency into account. The works known to the author in this direction are [De], [BO], [Ca], and [Gl]. In the first paper a general definition of observational



equivalence is given for nondeterministic measurement systems. This can be instantiated to give the usual observational equivalence on CCS or a more discriminating equivalence called partial-ordering observational equivalence. Partial orders are also used in [Bo] to generalise observational equivalence and the resulting equivalence is axiomatised. The basic technique used is to transfer much of the structure of processes onto the set of actions. Our equivalence can be shown to be strictly finer than theirs. In [Ca] observational equivalence is generalised by using the notion of locality; the resulting equivalence, called *distributed bisimulation*, is also axiomatised. Unlike  $\approx_{\tau}$  it satisfies

$$\mu(x + \tau y) = \mu(x + y) + \mu y.$$

Once more  $\approx_{\tau}$  is more discriminating. Finally, [G1] is a good survey of the area of “interleaving” versus “true concurrency” for processes. A variety of new equivalences are defined but, unlike the other papers, the algebraic language used is ACP, [Be].

**Acknowledgments.** The author thanks L. Mejia for many useful discussions at l’Ecole des Mines, Sophia-Antipolis, France. The author also thanks F. Williams for typing this difficult manuscript.

## REFERENCES

- [Be] J. BERGSTRA AND J. KLOP, *Algebra of communicating process with abstraction*, Theoret. Comput. Sci., 37 (1985), pp. 77–121.
- [Bo] G. BOUDOL AND I. CASTELLANI, *On the semantics of concurrency: partial orders and transition systems*, Rapport INRIA 550, INRIA, Rocquencourt, Le Chesnay, France.
- [Ca] I. CASTELLANI AND M. HENNESSY, *Distributed bisimulations*, Tech. Report, University of Sussex, 1987.
- [De] P. DEGANO, R. DENICOLA, AND U. MONTONARI, *Observational equivalences for concurrency models*, in Proc. Working Conference on Formal Description of Programming Concepts, Ebberup, Denmark, August 1986.
- [DeN] R. DENICOLA AND M. HENNESSY, *Testing equivalences for processes*, Theoret. Comput. Sci., 34 (1984), pp. 83–33.
- [Go] U. GOLTZ AND A. MYCROFT, *On the Relationship of CCS and Petri Nets*, Lecture Notes in Computer Science 179, Springer-Verlag, Berlin, New York, 1934.
- [He] M. HENNESSY AND R. MILNER, *Algebraic laws for nondeterminism and concurrency*, J. Assoc. Comput. Mach., 32 (1985), pp. 137–161.
- [Ma] A. MAZURKIEWICZ, *Concurrent program schemes and their interpretation*, in Proc. Aarhus Workshop on Verification of Parallel Programs, 1977.
- [Mil1] R. MILNER, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science 92, Springer-Verlag, Berlin, New York, 1980.
- [Mil2] ———, *Lectures on a calculus for communicating systems*, in Control Flow and Data Flow: Concepts of Distributed Programming, Proc. International Summer School directed by F. Bauer, F. Dijkstra, and C. Hoare, Springer Study Edition, 1986.
- [Mil3] ———, *A complete axiomatisation for observational congruence of finite-state behaviours*, Edinburgh, ECS-LLCS-86-8, University of Edinburgh, Edinburgh, 1986.
- [Re] W. REISIG, *Petri Nets: An Introduction*, Springer-Verlag, Berlin, New York, 1985.
- [G1] R. VAN GLABBEK AND F. VAANDRAGER, *Petri net models for algebraic concurrency*, Proc. Parle Conference; Lecture Notes in Computer Science, Springer-Verlag, Berlin, New York, 1987.
- [Wi] G. WINSKEL, *Categories of models for concurrency*, Tech. Report, University of Cambridge, 1984.

## A LINEAR ALGORITHM FOR TOPOLOGICAL BANDWIDTH IN DEGREE-THREE TREES\*

ZEVI MILLER†

**Abstract.** Let  $G$  be a graph, and  $f$  a one-to-one map of  $G$  into the positive integers. The *bandwidth* of  $G$  is  $B(G) = \min_f \max \{|f(x) - f(y)| : xy \in E(G)\}$ , where the max is taken over all edges  $xy$  in  $G$  and the min over all maps  $f$ .  $B(G)$  is related to the matrix bandwidth  $B(M)$  for a symmetric matrix  $M$ , and knowledge of the latter parameter is important for the efficient execution of certain matrix operations. The problem of determining  $B(G)$  for arbitrary  $G$  was shown by Papadimitriou to be NP-complete, and it was subsequently proved NP-complete even when  $G \in \Omega$ , where  $\Omega$  is the set of trees with maximum degree 3. Let  $B^*(G)$  be the minimum possible bandwidth of any subdivision of  $G$ , i.e., any graph obtained from  $G$  by inserting degree-2 points along edges of  $G$ . We present an  $O(n)$  algorithm for computing  $B^*(G)$  when  $G \in \Omega$ .

**Key words.** topological bandwidth

**AMS(MOS) subject classification.** 05C99

**1. Introduction.** Let  $G = (V, E)$  be a finite graph with no loops or multiple edges. A one-to-one map  $f: V(G) \rightarrow \mathbb{Z}$  of the vertex set of  $G$  into the integers will be called a *layout* of  $G$ . We let  $|f| = \max \{|f(v) - f(w)| : vw \in E(G)\}$ , and we define the *bandwidth* of  $G$ ,  $B(G)$ , to be  $B(G) = \min_f |f|$ . For convenience, we adopt the convention  $B(K_1) = 1$ , where  $K_1$  is the singleton graph.

The problem of determining  $B(G)$  for any  $G$  has importance in the theory of sparse matrices. Given an  $n \times n$  matrix  $A = (a_{ij})$ , define a graph  $G(A)$  by letting  $V(G(A)) = \{1, 2, \dots, n\}$  and declaring that  $ij$  is an edge if and only if  $a_{ij} \neq 0$  or  $a_{ji} \neq 0$ . Suppose we can find a symmetric permutation of the rows and columns of  $A$  so that all nonzero entries lie within the first  $k$  superdiagonals or the first  $k$  subdiagonals about the main diagonal. Then clearly  $B(G(A)) \leq k$ . Conversely if  $B(G(A)) \leq k$ , then the stated permutation exists. The problem of finding the permutation with the smallest  $k$ , equivalently, finding  $B(G(A))$ , is important in efficiently operating on and storing matrices, e.g., for Gaussian elimination, systolic processes, etc. Relevant work can be found in [1], [3], [9].

Recently the importance of bandwidth for complexity theory has been demonstrated in the work of Sudborough and Monien [1], [15]. The main kind of result has been that when certain well-known graph problems which are complete for some complexity class  $C$  are restricted to graphs with bounded bandwidth, the resulting problem is complete for space-bounded or simultaneous space- and time-bounded subclasses of  $C$ . Of course the bandwidth problem for arbitrary graphs [13], and even for trees of maximum degree 3 [6], is known to be NP-complete.

The problem of given a graph  $G$  to determine whether  $B(G) \leq k$  when  $k$  is fixed was proved polynomial time solvable in [14], with an improvement in [7].

The subject of this paper is the following topological version of bandwidth, attributed in [3] to R. Graham. For any graph  $G$ , let  $S(G)$  be the set of all subdivisions of  $G$ , i.e., the set of all graphs obtainable from  $G$  by finitely many compositions of the operation, called elementary subdivision, of replacing some edge  $xz$  by two edges  $xy$  and  $yz$ , where  $y$  is a vertex not previously in the graph. Obviously any  $H \in S(G)$

---

\* Received by the editors July 30, 1984; accepted for publication (in revised form) October 20, 1987. This work was supported in part by Office of Naval Research grant N0000 14-85-K-0621.

† Department of Mathematics and Statistics, Miami University, Oxford, Ohio 45056.

is homeomorphic with  $G$  and hence has the same underlying topology as  $G$ . The *topological bandwidth* of  $G$ ,  $B^*(G)$ , is defined by  $B^*(G) = \min \{B(H) : H \in S(G)\}$ .

Topological bandwidth is related to sparse matrices as follows. Given the system  $Ax = b$ , construct an equivalent system by replacing some term  $a_{ij}x_j$  by the new variable  $y$  and add a new equation  $a_{ij}x_j = y$ . This operation is equivalent to inserting a point of degree 2 in the edge  $ij$  of  $G(A)$ . Clearly  $B^*(G(A))$  is the smallest  $k$  such that for some system  $A'x = b'$  equivalent to  $Ax = b$  by a sequence of the above operations there is a permutation of the rows and columns which yields a  $2k + 1$  band form. The new system may be more efficient to work with if it is not too large, i.e., if the number of additional degree-2 vertices is not too large.

Another application is in regarding a graph  $G$  as a circuit with gates (corresponding to points of  $G$ ) laid out linearly for automation purposes [12], [16]. Inserting degree-2 points along edges of  $G$  corresponds to inserting drivers along wires of the circuit. Hence  $B^*(G)$  is the minimum length of the longest connection over all linear layouts allowing drivers. This minimum gives a lower bound on the time delay in transmission between elements of the circuit.

The cutwidth,  $cw(G)$ , of a graph  $G$  is defined as the minimum, over all layouts  $f$ , of  $\max_i |\{uv \in E(G) : f(u) \leq i < f(v)\}|$ . The relation between  $B^*(T)$  and  $cw(T)$  for trees  $T$  was explored by Chung in [5]. It was shown that  $cw(T)$  and  $B^*(T)$  can be arbitrarily far apart, but that  $cw(T) \leq B^*(T) - \log_2 B^*(T) + 2$ . Recently Yannakakis has found an  $O(n \log n)$  algorithm for  $cw(T)$  for arbitrary trees  $T$  [17].

The relation between  $cw(G)$  and  $B^*(G)$  was also investigated by Makedon, Papadimitriou, and Sudborough in [10]. They show that for any graph  $G$  we have  $B^*(G) \leq mcw(G) + 1$ , where  $mcw$  is the "modified cutwidth" considered in [10] and defined earlier by Lengauer in [18]. Since for any graph  $G$  we have  $mcw(G) \leq cw(G) - 1$ , it follows that  $B^*(G) \leq cw(G)$ . Makedon et al. also show that for any graph with maximum degree 3,  $B^*(G) = mcw(G) + 1$ . Thus an algorithm for topological bandwidth in trees of maximum degree 3 is also an algorithm for  $mcw$  in such trees.

The main result of this paper is an  $O(n)$  algorithm for computing  $B^*(T)$ , where  $T$  is any tree of maximum degree 3. First we develop an  $O(n \log n)$  algorithm which contains all the essential graph-theoretic ideas. Some additional data structure then leads to the  $O(n)$  solution. Our result is a contrast to the NP-completeness of bandwidth for trees of maximum degree 3, and hence shows that topological structure "alone" is not the reason for this NP-completeness.

We remark that an  $O(n \log n)$  algorithm for  $B^*(T)$  in degree-3 trees is obtained independently by Makedon, Papadimitriou, and Sudborough, in [10]. They show that the topological bandwidth problem for arbitrary graphs is NP-complete, they characterize graphs with topological bandwidth two, and they explore relations with pebbling and searching.

We follow standard graph-theoretic notation, as may be found in [2] or [8] for example. In particular, if  $G = (V, E)$  is a graph and  $H$  is a subgraph of  $G$ , then  $G - H$  is the graph obtained from  $G$  by deleting all points of  $H$  and all edges of  $G$  incident on those points.

**2. The main algorithm.** Let  $P$  be a path joining two end vertices of  $T$ . A subtree  $\alpha$  of  $T$  is called a *hanging from  $P$*  if it is one of the connected components of  $T - P$ . Thus each such  $\alpha$  contains a unique vertex  $x_\alpha$  such that  $x_\alpha y \in E(T)$  for some  $y \in P$ . We denote by  $Z(P)$  the set of all hangings from  $P$ . Let  $\|P\|$ , the *norm of  $P$  in  $T$* , be defined by  $\|P\| = 1 + \max \{B^*(\alpha) : \alpha \in Z(P)\}$ . To emphasize the role of  $T$ , we sometimes

write this as  $\|P\|_T$ . We let  $K_{1,2}$  (respectively,  $K_{1,3}$ ) be the tree with a central point joined to two (respectively, three) endpoints.

We begin with some lemmas.

LEMMA 1. *For any tree  $T$  there exists a layout of  $T$  satisfying  $|f| = B(T)$  such that  $f^{-1}(\min f(x))$  and  $f^{-1}(\max f(x))$  are both end vertices of  $T$ .*

*Proof.* Let  $g$  be a layout of  $T$  such that  $|g| = B(T)$ . Let  $p = g^{-1}(\min g(x))$  and  $q = g^{-1}(\max g(x))$ . We may suppose that at least one of  $p$  and  $q$  is not an end vertex (or leaf). Let  $P$  be the path in  $T$  joining  $p$  and  $q$ , and let  $Y_p$  (respectively,  $Y_q$ ) be the set of branches at  $p$  (respectively,  $q$ ) having empty intersection with  $P$ . Now define a new layout  $g_1$  of  $T$  as follows:

$$g_1(x) = \begin{cases} g(x), & x \in V(T) \setminus [Y_p \cup Y_q], \\ 2g(p) - g(x), & x \in Y_p, \\ 2g(q) - g(x), & x \in Y_q. \end{cases}$$

Notice that  $g_1$  just reflects  $Y_p$  and  $Y_q$  about  $p$  and  $q$ , respectively, and  $|g_1| = |g|$ . If  $g_1^{-1}(\min g_1(x))$  and  $g_1^{-1}(\max g_1(x))$  are both end vertices, then we are done. If not, then repeat the above process with  $g_1$  in place of  $g$ . Eventually we obtain a layout for  $T$  of the required kind.  $\square$

A path  $P$  joining two end vertices of the tree  $T$  is called a *backbone* of  $T$ . If  $f$  is a layout of  $T$  such that  $z_1 = f^{-1}(\min f(x))$  and  $z_2 = f^{-1}(\max f(x))$  are both end vertices of  $T$ , then the backbone  $P$  joining  $z_1$  and  $z_2$  is called a *backbone of  $f$* . We also say that  $f$  has backbone  $P$ .

Let  $T^* \in \mathcal{S}(T)$ , and let  $P$  be a backbone of  $T$ . A layout  $f$  of  $T^*$  will be called a *t-layout* of  $T$  ( $t$  for topological). We call  $P$  an *optimal backbone of  $T$*  if there exists such an  $f$  and  $T^*$  satisfying

- (i)  $|f| = B^*(T)$ , and
- (ii) the path joining  $f^{-1}(\min f(x))$  and  $f^{-1}(\max f(x))$  corresponds to  $P$  under the subdivision operation (so that  $f^{-1}(\min f(x))$  and  $f^{-1}(\max f(x))$  are endpoints of  $T^*$  corresponding to the endpoints of  $P$ ).

COROLLARY 1.1. *Every tree  $T$  has an optimal backbone.*

*Proof.* Let  $f$  be a  $t$ -layout of  $T$  with  $|f| = B^*(T)$ . By Lemma 1 we can find a  $t$ -layout  $g$  such that  $g^{-1}(\min g(x))$  and  $g^{-1}(\max g(x))$  are end vertices  $v_1$  and  $v_2$  (respectively) while  $|g| = B^*(T)$ . The path in  $T$  joining the vertices  $v_1$  and  $v_2$  is the required optimal backbone.  $\square$

Given a layout  $f$  of  $G$  and  $H \subseteq G$ , the layout of  $H$  induced by  $f$  is the layout  $f_H: V(H) \rightarrow \{1, 2, \dots, |V(H)|\}$  which maps the points of  $H$  in the same order as they are mapped by  $f$ .

LEMMA 2. *Let  $T$  be a nontrivial tree of maximum degree 3,  $P$  be an optimal backbone of  $T$ , and  $Z(P) = \{H_1, H_2, \dots, H_k\}$ . Then  $B^*(T) = \|P\| \equiv 1 + \max B^*(H_j)$ .*

*Proof.* We start with the lower bound  $B^*(T) \geq \|P\|$ . Consider a  $t$ -layout of  $T$ , say  $f: T^* \rightarrow \mathbb{Z}$  for some  $T^* \in \mathcal{S}(T)$ , such that  $|f| = B^*(T)$ . Let  $\bar{H}_i \in \mathcal{S}(H_i)$ ,  $1 \leq i \leq k$ , be the hanging from  $P^*$  (the subdivision of  $P$  in  $T^*$ ) corresponding to  $H_i$ . Let  $f_i$  be the layout of  $\bar{H}_i$  induced by  $f$ . Clearly  $|f_i| \geq B(\bar{H}_i) \geq B^*(H_i)$ , so there exist  $c, d \in V(H_i)$  (say with  $f(c) < f(d)$ ) such that  $f_i(d) - f_i(c) \geq B^*(H_i)$ . After translating  $f_i$  to correspond to the restriction of  $f$  to  $\bar{H}_i$ , we find that there is an edge  $ab \in E(P^*)$  such that without loss of generality either

- (i)  $f(a) < f_i(c)$  and  $f(b) > f_i(d)$ , or
- (ii)  $f_i(c) < f(a) < f_i(d)$ .

In either case we get  $|f| \geq 1 + B^*(H_i)$ , so by the arbitrariness of  $i$  we get  $|f| \geq \|P\|$ .

The bound  $B^*(T) \leq \|P\|$  follows from the layout indicated in Fig. 1. For each  $i$ ,  $1 \leq i \leq k$ , we take a layout  $f_i$  of some  $\bar{H} \in S(H_i)$  satisfying  $|f_i| = B^*(H_i)$ . Assuming without loss of generality that the  $H_i$  occur along  $P$  in indicial order, we place the layouts  $f_i$  along  $\mathbb{Z}$  in the same order so that  $f_i$  and  $f_{i+1}$  have no overlap for all  $i$ . We then lay out some  $P^* \in S(P)$  so that consecutive points on  $P^*$  are  $\|P\|$  units apart,  $\min\{P^*\} < \min f_1(x)$ , and  $\max\{P^*\} > \max f_k(x)$  (where we identify  $P^*$  with points of  $\mathbb{Z}$ ). Of course this requires us to “squeeze in” points of  $P^*$  into the range of the layouts  $f_i$ . We make room for this by iteratively translating values  $f_i(x)$  greater than the squeezed-in point one unit to the right. Since consecutive points on  $P^*$  are  $\|P\|$  units apart, and since  $B^*(H_i) < \|P\|$ , we will never have to stretch any pair  $f_i(x), f_i(y), xy \in E(\bar{H}_i)$ , more than  $\|P\|$  units apart. This defines a layout  $f: T^* \rightarrow \mathbb{Z}$  for some  $T^* \in S(T)$ . By construction we have  $|f(x) - f(y)| \leq \|P\|$  for any  $xy \in E(P^*) \cup (\cup E(\bar{H}_i))$ . Now for each  $i$  there is a unique edge  $w_i p_i \in E(T)$ ,  $w_i \in H_i, p_i \in P$ . We choose a  $w_i \in \bar{H}_i$  (to correspond to  $w_i$  and  $B_i$ ), and then find a  $p_i \in P^*$  (to correspond to  $p_i \in P$ ) that is at most distance  $\|P\|$  from  $w_i$ . Letting these  $w_i p_i$  be the remaining edges of  $T^*$ , we get  $|f| \leq \|P\|$  as required.  $\square$

In order to make use of the lemma in the algorithm we need some further notation.

Let  $T$  be a tree of maximum degree 3. If the vertices of  $T$  are numbered in the order of a depth-first search (DFS) starting at  $r$ , then the resulting numbers  $s$  can be used to define the usual relations on rooted trees: if  $x$  and  $y$  belong to the same branch at  $r$  and the path in  $T$  from  $r$  to  $y$  passes through  $x$ , then we write  $x \leq y$ . If  $x \leq y$  then we call  $x$  an *ancestor* of  $y$  and we call  $y$  a *descendant* of  $x$ . If  $y$  is a descendant of  $x$  and  $xy \in E(T)$ , then  $y$  is a *son* of  $x$  and  $x$  is a *father* of  $y$ . If  $Q$  is a subtree of  $T$ , then we define *head*( $Q$ ) to be the vertex  $x \in Q$  such that  $x \leq y$  for all  $y \in Q \setminus \{x\}$ . For  $x \in V(T)$ , let  $B_x(T)$  denote the subtree of  $T$  consisting of  $x$  and all descendants of  $x$  in  $T$ . We write  $B_x$  instead of  $B_x(T)$  when  $T$  is understood by context. For any subtree  $\gamma \subseteq T$  with *head*( $\gamma$ ) =  $s$ , we let  $(\gamma)_s$  be the tree with vertex set  $V(\gamma) \cup \{\bar{s}\}$ , where  $\bar{s}$  is a new symbol, and edge set  $E(\gamma) \cup \{s\bar{s}\}$ . Thus  $(\gamma)_s$  is obtained by attaching a new endpoint to  $s$ . Accordingly, if  $\beta$  is a backbone of  $\gamma$  which remains a backbone of  $(\gamma)_s$ , then we let  $\|\beta\|_s$  be the norm of  $\beta$  in  $(\gamma)_s$ . We say that  $B_x$  is *k-primitive* (or just primitive when  $k$  is understood) if  $B^*(B_x) = k$  and  $B^*(B_y) < k$  for all  $y \in B_x$ . Note that if  $B_x$  and  $B_z$  are distinct  $k$ -primitive branches, then  $B_x \cap B_z = \emptyset$ . If  $P$  is a path in  $T$ , we say that  $P$  *threads* the branch  $B_x$  if  $P$  passes through  $x$  and some end vertex of  $B_x$ .

COROLLARY 2.1. (a) Let  $B_x$  and  $B_y$  be branches of  $T$  such that  $B_x \cap B_y = \emptyset$  and  $B^*(B_x) = B^*(B_y) = B^*(T)$ . Then any optimal backbone of  $T$  threads both  $B_x$  and  $B_y$ .

(b) If  $T$  has three branches  $B_{x1}, B_{x2}, B_{x3}$  having pairwise empty intersection, then  $B^*(T) \geq 1 + \min_i B^*(B_{xi})$ .

*Proof.* (a) Let  $P$  be an optimal backbone of  $T$  violating the conclusion. Then at least one of  $B_x$  or  $B_y$  is contained in some hanging  $C$  of  $P$ . Thus by the lemma  $B^*(T) \geq 1 + B^*(C) \geq 1 + B^*(B_x) > B^*(T)$ , a contradiction.

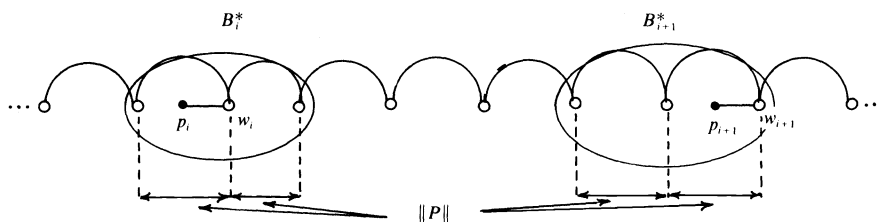


FIG. 1. The optimal  $t$ -layout of  $T$  from Lemma 2.

(b) By Corollary 1.1 we know that  $T$  must have an optimal backbone  $P$ . Now observe that for any optimal backbone  $P$  of  $T$  one of the  $B_{x_i}$ , say  $B_{x_1}$ , must be contained in a hanging from  $P$ . Hence  $B^*(T) \geq 1 + B^*(B_{x_1}) \geq 1 + \min B^*(B_{x_i})$ .  $\square$

**COROLLARY 2.2.** *For any backbone  $\beta$  of  $T$  we have  $B^*(T) \leq \|\beta\|$ . Hence by Corollary 1.1 and Lemma 2,  $B^*(T) = \min \{\|\beta\| : \beta \text{ a backbone of } T\}$ .*

*Proof.* Let  $Z(\beta) = \{B_j : 1 \leq j \leq k\}$ . Using the proof of the upper bound in the lemma we can construct a  $t$ -layout  $g$  of  $T$  such that  $|g| = \|\beta\|$ .  $\square$

Thus a backbone  $\beta$  of  $T$  is *optimal* (in the sense defined previously) if and only if  $\|\beta\|$  is a minimum among all backbones of  $T$ .

Before proceeding to the algorithm we introduce some notation. Let  $A$  and  $C$  be trees on which a DFS has been performed with roots  $x$  and  $y$ , respectively. Denote by  $A \vee C$  the tree with  $V(T) = V(A) \cup V(C) \cup \{r\}$ ,  $E(T) = E(A) \cup E(C) \cup \{rx, ry\}$ . The basic idea in our main procedure LABEL( $A, C, r$ ) is to use an inductive knowledge of  $B^*(A)$ ,  $B^*(C)$  and some additional information (described below) to find an optimal backbone  $\beta$  of  $A \vee C$  and compute  $B^*(A \vee C) = \|\beta\|$ .

Now let  $\tau$  be any tree of maximum degree 3 rooted at a point  $u$ . Define  $h(\tau)$  to be the point of  $\tau$  closest to  $u$  which can be on an optimal backbone of  $\tau$ . Also let  $\nu(\tau)$  be the number of  $B^*(\tau)$ -primitive branches of  $\tau$ . Thus  $\nu(\tau)$  equals 1 or 2 by Corollary 2.1(b).

**COROLLARY 2.3.** *Let  $\tau$  be a tree of maximum degree 3 rooted at  $u$ .*

(a) *If  $\nu(\tau) = 1$  then  $h(\tau) = u$ .*

(b) *Suppose  $\nu(\tau) = 2$ , and let  $B_x$  and  $B_y$  be the  $B^*(\tau)$ -primitive branches of  $\tau$ . Then  $h(\tau)$  is the point closest to  $u$  on the unique path in  $\tau$  joining  $x$  and  $y$ .*

*Proof.* (a) Let  $B_z$  be the unique  $B^*(\tau)$ -primitive branch of  $\tau$ . Clearly there exists a backbone  $\beta$  of  $\tau$  threading  $B_z$  and passing through  $u$ . Let  $\alpha$  be any hanging of  $\beta$ . If  $\alpha \subset B_z$ , then  $B^*(\alpha) \leq B^*(\tau) - 1$  by primitivity of  $B_z$ . If  $\alpha \subset \tau - B_z$  then  $B^*(\alpha) \leq B^*(\tau) - 1$  since otherwise  $B^*(\alpha) = B^*(\tau)$  and hence  $\alpha$  would contain a  $B^*(\tau)$ -primitive branch, contradicting the uniqueness of  $B_z$ . Thus  $\|\beta\| \leq B^*(\tau)$  so  $\beta$  is optimal and  $h(\tau) \leq \text{head}(\beta) = u$ . But  $h(\tau) \geq u$  since  $u$  is the root, so  $h(\tau) = u$ .

(b) By Corollary 2.1 any optimal backbone  $\beta$  must thread  $B_x$  and  $B_y$ . Hence  $\beta$  passes through  $x$  and  $y$  and the claim follows.  $\square$

We illustrate the conclusion of the lemma in Fig. 2. The curled line indicates an optimal backbone.

For any  $s \in \tau$ , let  $\tau/s$  be the tree  $\tau - B_s(\tau)$ . We will be interested in  $\tau/h(\tau)$ . We define the *signature* of  $\tau$ ,  $\text{sign}(\tau)$ , to be the 4-tuple  $(B^*(\tau), \nu(\tau), h(\tau), B^*((\tau)_u))$ .

We define a nested sequence Tree( $\tau$ ) of subtrees  $\{\tau_i\}$  of  $\tau$ , where  $u \in \tau_i$  for all  $i$ , as follows. Let  $\tau_1 = \tau$ . If  $\tau$  is a path, then  $\text{Tree}(\tau) = \{\tau_1\}$ . Otherwise assume inductively

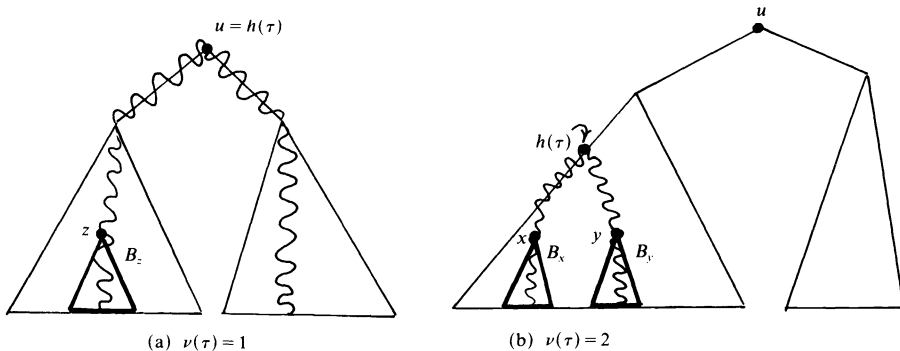


FIG. 2.  $h(\tau)$  depending on  $\nu(\tau)$ .

that  $\tau_1, \tau_2, \dots, \tau_i$  have been defined. If  $h(\tau_i) = u$ , then  $\text{Tree}(\tau) = \{\tau_1, \tau_2, \dots, \tau_i\}$ . Otherwise let  $\tau_{i+1} = \tau_i/h(\tau_i)$ . Note that if  $u$  has only one son, then  $\tau_{|\text{Tree}(\tau)|}$  is a path with endpoint  $u$ . We define  $\text{deck}(\tau)$  as the ordered list  $\text{deck}(\tau) = \{\text{sign}(\tau_1), \text{sign}(\tau_2), \dots, \text{sign}(\tau_{|\text{Tree}(\tau)|})\}$ , where  $\text{sign}(\tau_1)$  is the top and  $\text{sign}(\tau_{|\text{Tree}(\tau)|})$  is the bottom. If  $L = (l_1, l_2, \dots, l_m)$  and  $N = (n_1, n_2, \dots, n_j)$  are two ordered lists, we let  $L \oplus N$  be the ordered list  $(l_1, l_2, \dots, l_m, n_1, n_2, \dots, n_j)$ . Typically  $L$  and  $N$  will be of the form  $\text{deck}(\tau)$  for some  $\tau$ . If  $k \leq m$ , we let  $L_k = (l_1, l_2, \dots, l_k)$ , so  $\text{deck}(\tau)_k$  refers to the top  $k$  entries of  $\text{deck}(\tau)$ . We call these entries *frames*.

We now introduce the two procedures used in our algorithm. The main one LABEL  $(A, C, r)$  had  $\text{deck}(A)$  and  $\text{deck}(C)$  as input, where  $A$  and  $C$  are nonempty trees, and  $\text{deck}(A \vee C)$  as output. We use it in working our way up  $T$  in postorder, where  $A = B_s, C = B_t$  for  $s, t \in T$ , and  $A \vee C = B_u$ , where  $u$  is the father of  $s$  and  $t$ . The second, MAKEDECK  $(T, r)$  has  $\text{deck}(T)$  as input, where  $T$  is rooted  $r$ , and  $\text{deck}((T)_r)$  as output. It is called by LABEL in a certain case. The validity of LABEL will be proved in its description while that of MAKEDECK will follow from some later lemmas.

Procedure LABEL  $(A, C, r)$ .

*Input.*  $\text{deck}(A), \text{deck}(C)$       *Output.*  $\text{deck}(A \vee C)$

We let  $\text{Tree}(A) = \{\tau_1, \tau_2, \dots, \tau_k\}$ , and  $\text{Tree}(C) = \{\mu_1, \mu_2, \dots, \mu_l\}$ . Initially we scan  $\text{sign}(\tau_1) = \text{sign}(A)$  and  $\text{sign}(\mu_1) = \text{sign}(C)$  in  $\text{deck}(A)$  and  $\text{deck}(C)$ , respectively.

*Case 1.*  $B^*(A) = B^*(C) \equiv b$ .

Since  $\nu(A) + \nu(C) \geq 2$ , we have  $\nu(A) + \nu(C) \geq 3$  or  $\nu(A) = \nu(C) = 1$ . We consider these subcases in turn.

*Subcase 1(a).*  $\nu(A) + \nu(C) \geq 3$ .

By Corollary 2.1(b) it follows that  $B^*(A \vee C) \geq 1 + b$ . But any backbone  $\beta$  of  $A \vee C$  containing  $r$  satisfies  $\|\beta\| \leq 1 + b$  since any hanging  $\alpha$  of  $\beta$  is a subtree of  $A$  or  $C$  so by monotonicity of  $B^*$  satisfies  $B^*(\alpha) < b$ . Hence by Corollary 2.2 we have  $B^*(A \vee C) = 1 + b$ . Clearly  $\nu(A \vee C) = 1$  since  $A \vee C$  is its own  $(1 + b)$ -primitive branch, and  $h(A \vee C) = r$  since (as remarked above) any backbone containing  $r$  is optimal. The same  $\beta$  satisfies  $\|\beta\|_r \leq 1 + b$  since the only additional hanging it has in  $(A \vee C)_r$  is the single point  $\bar{r}$ . Thus  $B^*((A \vee C)_r) \leq 1 + b$ . The opposite inequality follows from  $(A \vee C)_r \supset A \vee C$  and monotonicity of  $B^*$ . The procedure thus ends with  $\text{deck}(A \vee C) = \{(1 + b, 1, r, 1 + b)\}$ .

*Subcase 1(b).*  $\nu(A) = \nu(C) = 1$ .

Let  $B_x \subset A$  and  $B_y \subset C$  be the unique  $b$ -primitive branches of  $A$  and  $C$ , respectively. Consider any backbone  $\beta$  of  $A \vee C$  which threads both  $B_x$  and  $B_y$ . Each of its hangings  $\alpha$  is either properly contained in  $B_x$  or  $B_y$ , or contains no  $b$ -primitive branches. Hence  $B^*(\alpha) \leq b - 1$ , so  $\|\beta\| \leq b$  and thus  $B^*(A \vee C) \leq b$  by Corollary 2.2. The opposite inequality is immediate by monotonicity of  $B^*$ . Thus  $B^*(A \vee C) = b$ . Obviously  $\nu(A \vee C) = 2$  since  $B_x$  and  $B_y$  are two  $b$ -primitive branches, and  $h(A \vee C) = r$  since any optimal backbone must thread  $B_x$  and  $B_y$  and hence contains  $r$ .

As for  $B^*((A \vee C)_r)$  we have two cases. If  $B^*(A) = 1$ , then the assumption  $\nu(A) = \nu(C) = 1$  implies that each of  $A$  and  $C$  has only one endpoint in  $A \vee C$ , and hence that  $A \vee C$  is a subdivision of a path. Thus  $(A \vee C)_r$  is a subdivision of  $K_{1,3}$ , so  $B^*((A \vee C)_r) = 2$ . We then return  $\text{deck}(A \vee C) = \{(1, 2, r, 2)\}$ . If  $B^*(A) \geq 2$ , then the same  $\beta$  above satisfies  $\|\beta\|_r \leq b$ , so  $B^*((A \vee C)_r) \leq b$ . The opposite inequality again follows from monotonicity of  $B^*$ . Hence we return  $\text{deck}(A \vee C) = \{(b, 2, r, b)\}$ .

Case 2.  $b = B^*(A) > B^*(C)$ .

Subcase 2(a).  $\nu(A) = 1$ .

Here we can find a backbone  $\beta$  of  $A \vee C$  which threads the unique  $b$ -primitive branch  $B$  of  $A$  and passes through  $r$ . Such a  $\beta$  satisfies  $\|\beta\| \leq b$ , so that  $B^*(A \vee C) = b$  and  $h(A \vee C) = r$ . Also  $\nu(A \vee C) = 1$  since  $B$  is unique. Again the same  $\beta$  satisfies  $\|\beta\|_r \leq b$  so  $B^*((A \vee C)_r) \leq b$  and by monotonicity  $B^*((A \vee C)_r) = b$ . We then return  $\text{deck}(A \vee C) = \{(b, 1, r, b)\}$ .

Subcase 2(b).  $\nu(A) = 2$ .

Let  $B_x$  and  $B_y$  be the two  $b$ -primitive branches of  $A$ . Clearly any backbone  $\beta$  of  $A \vee C$  satisfying  $\|\beta\| = b$  (if such exists) must thread both  $B_x$  and  $B_y$ , and  $\text{head}(\beta) = h(A)$ . Also, its hanging  $(A \vee C)/h(A)$  must have  $B^*$  value  $\leq b - 1$ .

It follows that if  $B^*((A \vee C)/h(A)) = b$  then  $B^*(A \vee C) \geq b + 1$ . Equality follows since for any backbone  $\delta$  of  $A \vee C$  containing  $r$  the hangings  $\alpha$  all satisfy  $B^*(\alpha) \leq \max\{B^*(A), B^*(C)\} = b$  so such a  $\delta$  satisfies  $\|\delta\| \leq b + 1$ . Also  $\|\delta\|_r \leq b + 1$  since the only additional hanging is a single point, and  $\nu(A \vee C) = 1$  since  $A \vee C$  is its own  $(b + 1)$ -primitive branch. Thus we output  $\text{deck}(A \vee C) = \{(b + 1, 1, r, b + 1)\}$ .

Suppose on the other hand that  $B^*((A \vee C)/h(A)) \leq b - 1$ . Take any backbone  $\beta$  of  $A \vee C$  threading the two  $b$ -primitive branches  $B_x$  and  $B_y$  of  $A$ . Then every hanging  $\alpha$  of it satisfies exactly one of the following:

- (a)  $\alpha \subset B_x$  or  $\alpha \subset B_y$  with proper containment.
- (b)  $\alpha$  contains no  $b$ -primitive branches.
- (c)  $\alpha = (A \vee C)/h(A)$ .

Each of these implies that  $B^*(\alpha) \leq b - 1$  so  $\|\beta\| \leq b$  and hence  $B^*(A \vee C) = b$ . Also  $\text{head}(\beta) = h(A)$  since any optimal backbone of  $A$  threads  $B_x$  and  $B_y$  and thus has a uniquely determined head which must therefore be the same as  $\text{head}(\beta)$ . Clearly  $\nu(A \vee C) = 2$ . As for  $B^*((A \vee C)_r)$ , the argument given above for  $A \vee C$  can be repeated for  $(A \vee C)_r$  to show that if  $B^*((A \vee C)_r/h(A)) \leq b - 1$ , then  $B^*((A \vee C)_r) = b$  while if  $B^*((A \vee C)_r/h(A)) = b$ , then  $B^*((A \vee C)_r) = b + 1$ . We conclude that  $\text{sign}(A \vee C) = (b, 2, h(A), z)$ , where  $z = B^*((A \vee C)_r)$  depends on  $B^*((A \vee C)_r/h(A))$  as indicated. Thus the first frame of  $\text{deck}(A \vee C)$  is  $\text{sign}(A \vee C)$  as given, so  $\text{deck}(A \vee C) = (b, 2, h(A), z) \oplus \text{deck}((A \vee C)/h(A))$ .

To summarize, we see that in Subcase 2(b) if  $B^*((A \vee C)/h(A)) = b$ , then  $\text{deck}(A \vee C) = \{(b + 1, 1, r, b + 1)\}$  while if  $B^*((A \vee C)/h(A)) \leq b - 1$ , then  $\text{deck}(A \vee C) = (b, 2, h(A), z) \oplus \text{deck}((A \vee C)/h(A))$ . Hence the output of LABEL( $A, C$ ) is given directly in the first case, while in the second the procedure must still produce  $\text{deck}((A \vee C)/h(A))$ . When this is done  $z$  is determined from the last entry, corresponding to  $B^*((A \vee C)_r/h(A))$ , of the top frame of  $\text{deck}((A \vee C)/h(A))$ , and the given  $\text{deck}(A \vee C)$  is returned.

Consider then the construction of  $\text{deck}((A \vee C)/h(A))$  which, since  $(A \vee C)/h(A) = (A/h(A)) \vee C$ , is  $\text{deck}((A/h(A)) \vee C)$ . If  $A/h(A) \neq \emptyset$ , then we do this by making a recursive call on LABEL( $A/h(A), C, r$ ) = LABEL( $\tau_2, \mu_1, r$ ) after popping  $\text{sign}(\tau_1) = \text{sign}(A)$  from the top of  $\text{deck}(A)$  so that  $\text{sign}(\tau_2) = \text{sign}(A/h(A))$  is available for scanning. We get as output the required  $\text{deck}((A/h(A)) \vee C)$ . If  $A/h(A) = \emptyset$  we cannot make this call and instead use procedure MAKEDECK as follows.

Suppose then that  $A/h(A) = \emptyset$ . Then  $(A/h(A)) \vee C = (C)_y$ , where  $y$  is the root of  $C$ , i.e., the son of  $r$  in  $C$ . Hence we get  $\text{deck}((A/h(A)) \vee C)$  by calling on procedure MAKEDECK( $C, y$ ).

This completes procedure LABEL( $A, C, r$ ).

We illustrate the Subcase 2(b) requiring recursion in Figs. 3 and 4. The pointers in Fig. 4 show the entries to be scanned in the indicated calls to LABEL.



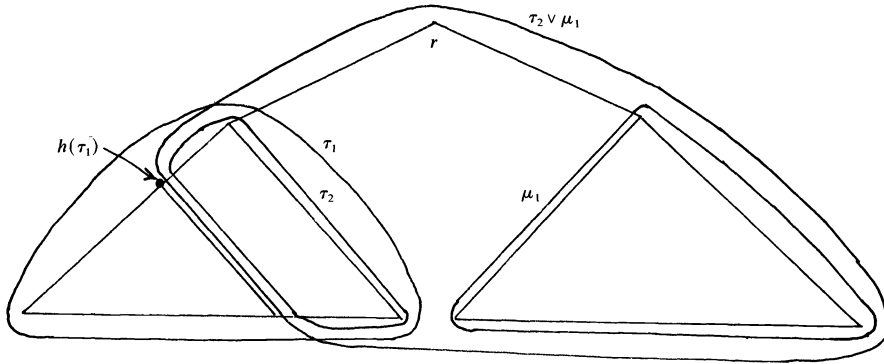


FIG. 3. Determining  $B^*(\tau_1 \vee \gamma_1)$  from  $B^*(\tau_2 \vee \gamma_1)$ .

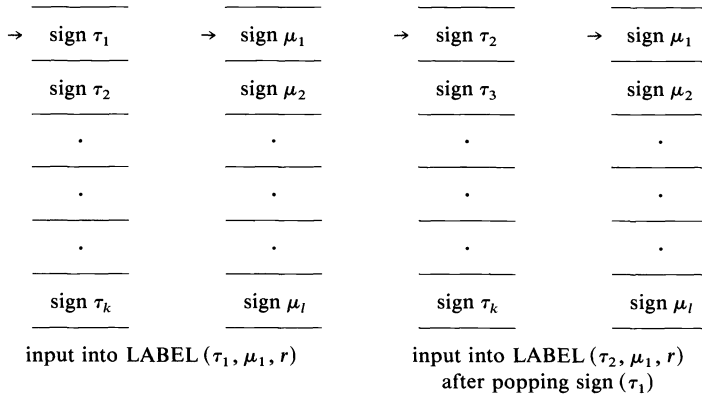


FIG. 4. The deck structure in recursion.

The procedure may continue calling itself through possibly  $k + l$  levels in recursion (after popping a sign ( $\tau_i$ ) or sign ( $\mu_j$ ) in preparation each time) until either Case 1 or Subcase 2(a) obtains, where of course  $A$  and  $C$  are replaced by some  $\tau_i$  or  $\mu_j$ . At this point the answer cascades back up as in the discussion of Subcase 2(b) to give the final output of LABEL ( $A, C, r$ ).

We must now develop the procedure MAKEDECK ( $T, r$ ) for an arbitrary tree  $T$  of maximum degree 3 rooted at a point  $r$  of degree 1 or 2. We begin with some lemmas.

A *pendant path* in  $T$  is a path  $P$  in  $T$  with endpoints  $x$  and  $y$  at least one of which is an endpoint in  $T$  and such that  $\deg(z, T) = 2$  for all  $z \in P \setminus \{x, y\}$ .

LEMMA 3. Let  $T$  be a tree. Let  $P$  be a pendant path in  $T$  from  $x$  to  $y$  such that  $\deg_T(x) = 1$ . Suppose  $T_p$  is a copy of  $T$  except that edges in  $P$  may be subdivided. Then  $B^*(T) = B^*(T_p)$ .

*Proof.* First note that  $B^*(T_p) \cong B^*(T)$  because any subdivision of  $T_p$  is also a subdivision of  $T$ . Thus we need only show the other direction, which we will do by induction on  $|T|$ . Let  $Q$  be an optimal backbone of  $T$ .

Case 1.  $P \cap Q = \emptyset$ . Then  $P$  is contained in  $H$ , a hanging of  $Q$ . By the induction hypothesis,  $B^*(H_p) = B^*(H)$  since  $H$  is smaller than  $T$ . Also the hangings of  $Q$  other than  $H$  do not contain  $P$ , so they have the same  $B^*$  values whether viewed as subtrees of  $T$  or  $T_p$ . Thus, all the hangings of  $Q$  in  $T_p$  have the same  $B^*$  values as the hangings of  $Q$  in  $T$ . Hence by Lemma 2 there is a  $t$ -layout of  $T_p$  having bandwidth  $\|Q\| = B^*(T)$ .

Case 2.  $P \cap Q = (y)$ . Let  $H = P' = P - y$ , so that  $H$  is a hanging of  $Q$  and  $P'$  is a pendant path in  $H$  (which happens to be all of  $H$ ). This gives the same situation as Case 1 with  $B^*(H) = B^*(H_p) = 1$ . The same argument as in Case 1 then gives  $B^*(T_p) \cong B^*(T)$ .

Case 3.  $P \subset Q$ . Then  $P$  is the tail of  $Q$  and it has no hangings, so by the construction of the optimal layout of  $T$  given by Lemma 2, the points on  $P$  will be numbered with integers spaced  $\|Q\|$  apart at the high end (or low end—it does not matter) of the range of the layout. But then it is easy to push out the high end of the range in order to fit in the additional points of  $T_p$ . This does not change the bandwidth.

Note that there can be no other way for  $P$  and  $Q$  to intersect because  $P$  is pendant. The proof is thereby completed.  $\square$

We now pass to a description of deck  $((T)_r)$  in terms of deck  $(T)$ . As notation we let  $\text{Tree}(T) = (\gamma_1, \gamma_2, \dots, \gamma_m)$  and  $\text{Tree}((T)_r) = (\gamma'_1, \gamma'_2, \dots, \gamma'_p)$ . We recall that  $\bar{r}$  is the endpoint not in  $T$  attached to  $r$ . We view it as the father of  $r$  in a depth-first search of  $(T)_r$ .

LEMMA 4. *Suppose  $T$  is rooted at a point  $r$  satisfying  $\text{deg}(r, T) = 1$ . Then  $\text{deck}((T)_r) = \text{deck}(T)_{m-1} \oplus (1, 1, \bar{r}, 1)$ .*

*Proof.* We begin by showing that  $\gamma'_i = (\gamma_i)_r$ ,  $1 \leq i \leq m$ , and that  $m = p$ . For the first claim we proceed by induction on  $i$ . It is trivially true for  $i = 1$ . Suppose it is true for all  $i \leq i_0$ , where  $i_0 < m$ . If  $h(\gamma_{i_0}) \neq r$ , then  $\gamma_{i_0}$  has two  $B^*(\gamma_{i_0})$ -primitive branches. Since  $\gamma'_{i_0} = (\gamma_{i_0})_r$  by induction, we have  $B^*(\gamma'_{i_0}) = B^*(\gamma_{i_0})$  by Lemma 3 and hence any optimal backbone of  $\gamma'_{i_0}$  must thread the same two branches. Thus  $h(\gamma'_{i_0}) = h(\gamma_{i_0})$  and hence  $\gamma'_{i_0+1} = (\gamma_{i_0+1})_r$ , so the induction is completed. The fact that  $m = p$  follows immediately since  $\gamma'_m = (\gamma_m)_r$  and  $h(\gamma_m) = r$  imply that  $h(\gamma'_m) = \bar{r}$ . Thus  $\gamma'_{m+1} = \emptyset$  so  $\gamma'_m$  is the last element of  $\text{Tree}((T)_r)$ .

We can now complete the proof. First we show that  $\text{sign}(\gamma'_j) = \text{sign}(\gamma_j)$  for  $j < m$ . The conditions  $\text{deg}(r, T) = 1$ ,  $\gamma'_j = (\gamma_j)_r$ , and Lemma 3 imply that  $B^*((\gamma'_j)_r) = B^*(\gamma'_j) = B^*((\gamma_j)_r) = B^*(\gamma_j)$ , and that the  $B^*(\gamma'_j)$ -primitive branches of  $\gamma'_j$  are identical to the  $B^*(\gamma_j)$ -primitive branches of  $\gamma_j$ . The condition  $\gamma'_j = (\gamma_j)_r$  for all  $j$  implies  $h(\gamma'_j) = h(\gamma_j)$  for  $j < m$ . It follows that  $\text{sign}(\gamma'_j) = \text{sign}(\gamma_j)$  for  $j < m$ .

For  $j = m$ , just observe that  $\gamma_m$  is a path with endpoint  $r$  so  $\gamma'_m = (\gamma_m)_r$  is a path with endpoint  $\bar{r}$ . Thus  $\text{sign}(\gamma'_m) = \{(1, 1, \bar{r}, 1)\}$ .

The lemma is thus proved.  $\square$

We now make some observations that lead to a description of deck  $((T)_r)$ , where  $r$  has degree 2.

LEMMA 5. *Suppose that for some integer  $j < m$  we have  $B^*(\gamma_q) = B^*((\gamma_q)_r)$  for  $1 \leq q \leq j$ . Then*

- (1)  $\gamma'_q = (\gamma_q)_r$  for  $1 \leq q \leq j + 1$ , and
- (2)  $\text{sign}(\gamma'_q) = \text{sign}(\gamma_q)$  for  $1 \leq q \leq j$ .

*Proof.* We start with (1). Proceeding by induction, suppose there is an integer  $q_0 \leq j$  for which (1) is true for all  $q \leq q_0$ , the case  $q_0 = 1$  being trivially true. Since  $q_0 < m$  we have  $h(\gamma_{q_0}) \neq r$ . Thus any optimal backbone of  $\gamma_{q_0}$  threads two  $B^*(\gamma_{q_0})$ -primitive branches of  $\gamma_{q_0}$ , and since  $B^*(\gamma_{q_0}) = B^*((\gamma_{q_0})_r)$ , any optimal backbone of  $(\gamma_{q_0})_r$  must thread these same two branches. Hence  $h(\gamma'_{q_0}) = h((\gamma_{q_0})_r) = h(\gamma_{q_0})$ , so  $\gamma'_{q_0+1} = (\gamma_{q_0+1})_r$  and the inductive step is done.

Now consider (2). We may suppose that  $m > 1$  since otherwise the statement is vacuously true. Take any  $q$  in the range  $1 \leq q \leq j$ . By hypothesis  $B^*(\gamma_q) = B^*((\gamma_q)_r)$  and by (1) we have  $B^*(\gamma'_q) = B^*((\gamma_q)_r)$ , so  $B^*(\gamma'_q) = B^*(\gamma_q)$ . By Lemma 3 we have  $B^*((\gamma'_q)_{\bar{r}}) = B^*(\gamma'_q) = B^*((\gamma_q)_r)$ . Hence the first and last entries of  $\text{sign}(\gamma'_q)$  are identical to those of  $\text{sign}(\gamma_q)$ . Since  $q < m$  we have  $h(\gamma_q) < m$ , and the argument in (1) can be

repeated to give  $h(\gamma'_q) = h(\gamma_q)$  and  $\nu(\gamma'_q) = \nu(\gamma_q) = 2$ . Hence  $\text{sign}(\gamma'_q) = \text{sign}(\gamma_q)$ . Part (2) is proved and we are done.  $\square$

LEMMA 6. *Suppose  $T$  is rooted at a point  $r$  satisfying  $\text{deg}(r, T) = 2$ .*

(1) *Let  $j$  be the first index, if it exists, such that  $B^*((\gamma_j)_r) \neq B^*(\gamma_j)$  (so that  $B^*((\gamma_j)_r) = B^*(\gamma_j) + 1$ ). Then*

$$\text{deck}((T)_r) = \text{deck}(T)_{j-1} \oplus (B^*(\gamma_j) + 1, 1, \bar{r}, B^*(\gamma_j) + 1).$$

(2) *If no such  $j$  exists, then*

(2a) *If  $\text{deg}(r, \gamma_m) = 1$ , then*

$$\text{deck}((T)_r) = \text{deck}(T)_{m-1} \oplus (B^*(\gamma_m), 1, \bar{r}, B^*(\gamma_m)).$$

(2b) *If  $\text{deg}(r, \gamma_m) = 2$ , then*

(2b') *If  $\nu(\gamma_m) = 1$ , then*

$$\text{deck}((T)_r) = \text{deck}(T)_{m-1} \oplus (B^*(\gamma_m), 1, \bar{r}, B^*(\gamma_m)).$$

(2b'') *If  $\nu(\gamma_m) = 2$ , then*

$$\text{deck}((T)_r) = \text{deck}(T) \oplus (1, 1, \bar{r}, 1).$$

*Proof.* (1) Suppose that the stated  $j$  exists. Then by Lemma 5  $\gamma'_q = (\gamma_q)_r$  for  $1 \leq q \leq j$  and  $\text{sign}(\gamma'_q) = \text{sign}(\gamma_q)$  for  $1 \leq q \leq j - 1$ . As for  $\text{sign}(\gamma'_j)$ , note that any backbone  $\beta$  of  $\gamma'_j = (\gamma_j)_r$  containing  $\bar{r}$  satisfies  $\|\beta\| \leq B^*(\gamma_j) + 1$  since all its hangings  $\alpha$  satisfy  $\alpha \subset \gamma_j$  and hence  $B^*(\alpha) \leq B(\gamma_j)$ . Thus  $h(\gamma'_j) = \bar{r}$ , so  $\text{sign}(\gamma'_j)$  is the last frame of  $\text{deck}((T)_r)$ . We have by Lemma 3 and hypothesis  $B^*((\gamma'_j)_{\bar{r}}) = B^*(\gamma'_j) = B^*((\gamma_j)_r) = B^*(\gamma_j) + 1$ . Finally  $\nu(\gamma'_j) = 1$  since  $\gamma'_j$  is its own  $B^*(\gamma_j)$ -primitive branch. Thus  $\text{sign}(\gamma'_j) = (B^*(\gamma_j) + 1, 1, \bar{r}, B^*(\gamma_j) + 1)$  and hence  $\text{deck}((T)_r) = \text{deck}(T)_{j-1} \oplus (B^*(\gamma_j) + 1, 1, \bar{r}, B^*(\gamma_j) + 1)$  as required.

(2) Suppose no such  $j$  exists. Since  $\gamma_m$  is the last tree in  $\text{Tree}(T)$ , it follows that  $h(\gamma_m) = r$ . Hence there is an optimal backbone  $\beta$  of  $\gamma_m$  containing  $r$ . We also have  $\text{sign}(\gamma'_q) = \text{sign}(\gamma_q)$  for  $q < m$  by Lemma 5.

Case 2(a). We can extend  $\beta$  to  $\bar{r}$  thereby obtaining a backbone  $\bar{\beta}$  of  $(\gamma_m)_r$ . Clearly  $\|\bar{\beta}\| = \|\beta\|$  since  $\bar{\beta}$  has the same hangings as  $\beta$ . Thus  $B^*((\gamma_m)_r) = B^*(\gamma_m)$ . By Lemma 5 we have  $\gamma'_m = (\gamma_m)_r$  so  $B^*(\gamma'_m) = B^*(\gamma_m)$ , and by Lemma 3,  $B^*((\gamma'_m)_{\bar{r}}) = B^*(\gamma'_m) = B^*(\gamma_m)$ . Finally  $h(\gamma'_m) = \text{head}(\bar{\beta}) = \bar{r}$ , and  $\nu(\gamma'_m) = 1$  since otherwise  $h(\gamma'_m) \neq \bar{r}$ . Thus  $\text{sign}(\gamma'_m) = (B^*(\gamma_m), 1, \bar{r}, B^*(\gamma_m))$ . Clearly  $\gamma'_m$  is the last element of  $\text{Tree}((T)_r)$ . Thus  $\text{deck}((T)_r) = \text{deck}(T)_{m-1} \oplus (B^*(\gamma_m), 1, \bar{r}, B^*(\gamma_m))$ .

Case 2(b). Assume first that  $\nu(\gamma_m) = 1$ . Then any backbone  $\bar{\beta}$  of  $\gamma'_m = (\gamma_m)_r$  which threads the unique  $B^*(\gamma_m)$ -primitive branch of  $\gamma_m$  satisfies  $\|\bar{\beta}\| = B^*(\gamma_m)$ . We can choose such a  $\bar{\beta}$  which contains  $\bar{r}$ . Hence we get  $B^*(\gamma'_m) = B^*(\gamma_m)$  and  $h(\gamma'_m) = \bar{r}$ . As above  $B^*((\gamma'_m)_{\bar{r}}) = B^*(\gamma'_m)$ , and  $\nu(\gamma'_m) = 1$  since  $\gamma'_m$  has the same unique  $B^*(\gamma_m)$ -primitive branch as  $\gamma_m$ . Thus  $\text{deck}((T)_r) = \text{deck}(T)_{m-1} \oplus (B^*(\gamma_m), 1, \bar{r}, B^*(\gamma_m))$ .

Now suppose  $\nu(\gamma_m) = 2$ . Since  $B^*(\gamma'_m) = B^*(\gamma_m)$  any optimal backbone  $\bar{\beta}$  of  $\gamma'_m$  must thread the same two  $B^*(\gamma_m)$ -primitive branches of  $\gamma_m$  as does  $\beta$ . Hence  $h(\gamma'_m) = \text{head}(\bar{\beta}) = \text{head}(\beta) = r$ . Clearly  $\nu(\gamma'_m) = \nu(\gamma_m) = 2$ , and  $B^*((\gamma'_m)_r) = B^*(\gamma'_m)$  as before. Thus  $\text{sign}(\gamma'_m) = \text{sign}(\gamma_m)$ . Also we have  $\gamma'_{m+1} = \{\bar{r}\}$ , so  $\text{sign}(\gamma'_{m+1}) = (1, 1, \bar{r}, 1)$ . Thus  $\text{deck}((T)_r) = \text{deck}(T) \oplus (1, 1, \bar{r}, 1)$ . The lemma is therefore proved.  $\square$

The procedure MAKEDECK  $(T, r)$  is now essentially given by Lemmas 4 and 6. For completeness we describe it explicitly.

Procedure MAKEDECK  $(T, r)$ .

*Input.*  $\text{deck}(T)$ , where  $T$  is a tree of maximum degree 3 rooted at  $r$ .

*Output.*  $\text{deck}((T)_r)$

(1) determine  $\text{deg}(r, T)$

(2) If  $\text{deg}(r, T) = 1$ , then

$$\text{deck}((T)_r) \leftarrow \text{deck}(T)_{m-1} \oplus (1, 1, \bar{r}, 1).$$

(3) If  $\text{deg}(r, T) = 2$ , then

(a) Search  $\text{deck}(T)$  from top to bottom until you find the minimum  $j$ , if it exists, such that  $B^*((\gamma_j)_r) \neq B^*(\gamma_j)$ .

(b) If such a  $j$  exists, then

$$\text{deck}((T)_r) \leftarrow \text{deck}(T)_{j-1} \oplus (B^*(\gamma_j) + 1, 1, \bar{r}, B^*(\gamma_j) + 1).$$

(c) If no such  $j$  exists, then determine  $\text{deg}(r, \gamma_m)$ . Depending on the answer, output  $\text{deck}((T)_r)$  according to the conditions defining cases (2a), (2b'), and (2b'') of Lemma 6.

This completes  $\text{MAKEDECK}(T, r)$ .

We remark that to determine  $\text{deg}(r, T)$  and  $\text{deg}(r, \gamma_m)$  we must maintain a new fifth entry, having value “1” or “2” to denote  $\text{deg}(r, \gamma_j)$  in  $\text{sign}(\gamma_j)$  for all  $\gamma_j \in \text{Tree}(T)$ . We describe the updating of  $\text{deg}(r, \gamma_j)$  here, omitting its formal inclusion in the lemmas and procedures to simplify their presentations.

We suppose then that in procedure  $\text{LABEL}(A, C, r)$  each frame  $\text{sign}(\tau_i)$  and  $\text{sign}(\mu_j)$  in  $\text{deck}(A)$  and  $\text{deck}(C)$ , respectively, has the additional bit  $\text{deg}(x, \tau_i)$  or  $\text{deg}(y, \mu_j)$  (respectively), where  $x$  and  $y$  are the roots of  $A$  and  $C$  (and thus the sons of  $r$  in  $T$ ). Our goal is to describe  $\text{deg}(r, \gamma_j)$  in all frames  $\text{sign}(\gamma_j)$  of  $\text{deck}(A \vee C)$ .

Suppose first that any of Subcases 1(a), 1(b), or 2(a) in  $\text{LABEL}(A, C, r)$  holds. Then the output  $\text{deck}(A \vee C)$  has a single frame. Since  $A$  and  $C$  are nonempty we have  $\text{deg}(r, A \vee C) = 2$ , so we insert “2” for  $\text{deg}(r, A \vee C)$  in the single frame.

Suppose then that Subcase 2(b) holds. The possible outputs are

(a)  $\text{deck}(A \vee C) = \{(b+1, 1, r, b+1)\}$ .

(b)  $\text{deck}(A \vee C) = (b, 2, h(A), B^*((A \vee C)_r)) \oplus \text{deck}((A \vee C)/h(A))$ .

In case (a) we again insert “2” for  $\text{deg}(r, A \vee C)$  as above. Suppose (b) holds. For the topmost frame we insert “2” again since  $A$  and  $C$  are nonempty so  $\text{deg}(r, A \vee C) = 2$ . The remaining frames (those in  $\text{deck}((A \vee C)/h(A))$ ) are handled as follows. If  $A/h(A) \neq \emptyset$ , then a call to  $\text{LABEL}(A/h(A) \vee C)$  was made and hence the degree entries in the frames of  $\text{deck}(A/h(A) \vee C) = \text{deck}((A \vee C)/h(A))$  have been inserted by recursion. If  $A/h(A) = \emptyset$ , then a call is made to  $\text{MAKEDECK}(C, y)$ . We then insert “1” in all frames of the resulting  $\text{deck}((C)_y)$ . The reason is that  $r$  (which plays the role of  $\bar{y}$  in  $\text{MAKEDECK}(C, y)$ ) is an endpoint of  $(C)_y$ , attached to  $y$  and hence has degree 1 in each element of  $\text{Tree}((C)_y)$ .

**3. Complexity.** We now discuss the complexity of procedures  $\text{LABEL}(A, C, r)$  and  $\text{MAKEDECK}(T, r)$  and the complexity of the resulting algorithm for  $B^*(T)$  in trees  $T$  of maximum degree 3.

Let  $T$  be any tree of maximum degree 3 rooted at a point  $r$ . We assume a DFS on  $T$  with associated branches  $B_x$  for all  $x \in T$ . Let  $|\text{deck}(T)|$  denote the number of frames (or signatures) in  $\text{deck}(T)$ , so  $|\text{deck}(T)| = |\text{Tree}(T)|$ . Our first goal is to estimate  $|\text{deck}(T)|$ .

For the next lemma we recall that a tree  $T$  is  $k$ -primitive if and only if it is its own (and hence unique)  $k$ -primitive branch, i.e.,  $B^*(T) = k$  and  $B^*(T') < k$  for any proper subtree  $T'$  with  $r \notin T'$ .

LEMMA 7. Let  $T$  be a  $k$ -primitive tree. Then  $|T| \geq 2^{k-1}$ .

*Proof.* Let  $f_k = \min \{p : |T| = p, T \text{ is } k\text{-primitive}\}$ . We show  $f_k \geq 2^{k-1}$  by induction on  $k$ .

For  $k = 1$ , the one-point tree is  $k$ -primitive by convention and has minimum order. Thus  $f_1 \geq 1 = 2^0$ . Now suppose  $f_j \geq 2^{j-1}$  for  $j < s$ .

Let  $T$  be  $s$ -primitive and have minimum possible order. We claim  $T$  must have at least two  $(s-1)$ -primitive branches. Suppose first that  $T$  has no  $(s-1)$ -primitive branches. Then any backbone  $\beta$  of  $T$  through  $r$  satisfies  $\|\beta\| \leq s-1$ , so  $B^*(A) \leq s-1$ , a contradiction. Similarly if  $T$  has only one  $(s-1)$ -primitive branch  $B_z$ , then any backbone  $\beta$  of  $T$  threading  $B_z$  and passing through  $r$  satisfies  $\|\beta\| \leq s-1$ , again a contradiction. Hence  $T$  has at least two  $(s-1)$ -primitive branches, so by induction  $|T| = f_s \geq 2f_{s-1} \geq 2^{s-1}$ .  $\square$

A sequence of trees  $T_1, T_2, \dots, T_k$  is called *monotone* if  $B^*(T_i) > B^*(T_{i+1}), i \geq 1$ .

LEMMA 8. For any tree  $T$ ,  $\text{Tree}(T)$  is *monotone*.

*Proof.* Let  $\text{Tree}(T) = \{\tau_1, \tau_2, \dots, \tau_l\}$ . Now by definition  $\tau_i/h(\tau_i) = \tau_{i+1}, 1 \leq i \leq l-1$ , and  $B^*(\tau_i) = \|\beta_i\|$  where  $\beta_i$  is a  $t$ -backbone of  $\tau_i$  with head  $(\beta_i) = h(\tau_i)$ . It follows that  $\tau_{i+1}$  is a hanging of  $\beta_i$  in  $\tau_i$ . Hence  $B^*(\tau_{i+1}) \leq \|\beta_i\| - 1 = B^*(\tau_i) - 1$ .  $\square$

COROLLARY 8.1. Let  $T$  be a tree on  $n$  points. Then  $|\text{deck}(T)| \leq B^*(T) \leq O(\log n)$ .

*Proof.* Since  $|\text{deck}(T)| = |\text{Tree}(T)|$  it suffices to prove the inequalities with  $|\text{deck}(T)|$  replaced by  $|\text{Tree}(T)|$ .

Let  $\text{Tree}(T) = \{\tau_1, \tau_2, \dots, \tau_l\}$ . Clearly  $B^*(\tau_1) = B^*(T)$ , so  $\tau_1$  contains a  $B^*(T)$ -primitive branch, and hence  $n = |\tau_1| \geq 2^{B^*(T)-1}$ . It follows that  $B^*(T) \leq O(\log n)$ . Also  $l < B^*(T)$  since  $\text{Tree}(T)$  is monotone.  $\square$

LEMMA 9. Let  $A$  and  $C$  be trees with  $|A| + |C| = n$ . Then  $\text{LABEL}(A, C, r)$  has complexity  $O(\log n)$ .

*Proof.* We may divide the running time of the algorithm into two parts. In the first part LABEL makes nested calls on itself of the form  $\text{LABEL}(\tau_i, \mu_j)$ . In the second part, which may not occur, LABEL calls on MAKEDECK  $(\mu_{j_0}, y)$  or MAKEDECK  $(\tau_{i_0}, x)$  for some  $j_0 \leq |\text{deck}(C)|$  or  $i_0 \leq |\text{deck}(A)|$ . Without loss of generality let this call be MAKEDECK  $(\mu_{j_0}, y)$ .

Consider the first part. Each call requires  $O(1)$  time since it consists in scanning  $\text{sign}(\tau_i)$  and  $\text{sign}(\mu_j)$  and then defining  $\text{deck}(\tau_i \vee \mu_j)$  as a single frame or the result of pushing a single frame onto the recursively computed  $\text{deck}(\tau_{i+1} \vee \mu_j)$  or  $\text{deck}(\tau_i \vee \mu_{j+1})$ . The call pops either  $\text{sign}(\tau_i)$  or  $\text{sign}(\mu_j)$  before the next call begins. The total number of calls is the number of times a frame is popped before the call to MAKEDECK  $(\mu_{j_0}, y)$ . Hence this number is  $|\text{deck}(A)| + j_0$ , so the time spent on the first part is  $O(|\text{deck}(A)| + j_0)$ .

In the second part MAKEDECK processes from top to bottom all the remaining  $|\text{deck}(C)| - j_0$  frames of  $\text{deck}(C)$ . It spends at most  $O(1)$  time on each frame since it only scans for the condition  $B^*((\gamma)_r) \neq B^*(\gamma_j)$  or inserts a new frame at the bottom of the ones it has already checked. The time for the second part is thus  $O(|\text{deck}(C)| - j_0)$ .

It follows that the total time of LABEL  $(A, C, r)$  is  $O(|\text{deck}(A)| + |\text{deck}(C)|) \leq O(\log |A| + \log |C|) \leq O(\log (|A||C|)) \leq O(\log(n^2)) = O(\log n)$ .  $\square$

LEMMA 10. Let  $A$  be a tree with  $|A| = n$  rooted at  $r$ . Then MAKEDECK  $(A, r)$  has complexity  $O(\log n)$ .

*Proof.* The proof is essentially the same as the second part of the proof of Lemma 9.  $\square$

We can now give an  $O(n \log n)$  algorithm for  $B^*(T)$ . Later we sketch how it can be refined to linear time.

THEOREM 1. Let  $T$  be a tree of maximum degree 3. Then  $B^*(T)$  can be computed

in  $O(n \log n)$  time.

*Proof.* Suppose first that  $t$  has a point  $r$  of degree 2.

Now view  $r$  as the root of  $T$ . Now order  $V(T)$  in reverse depth-first search from endpoints “up” to  $r$  (i.e., in postorder). If  $e$  is an endpoint, then  $\text{deck}(\{e\}) = \{(1, 1, e, 1)\}$ , (where we recall the convention  $B^*(\{e\}) = 1$ ). Now for some  $x \in T$ , assume inductively that  $\text{deck}(B_y)$  is known for each of (the at most two) sons of  $x$ . If  $x$  has only one son  $y$ , then  $\text{deck}(B_x)$  can be obtained from  $\text{deck}(B_y)$  by calling on MAKEDECK( $B_y, y$ ). Suppose then that  $x$  has sons  $y_1$  and  $y_2$ . We then apply LABEL( $B_{y_1}, B_{y_2}, x$ ) and get  $\text{deck}(B_x)$  as output.

Continuing to work our way up we eventually apply LABEL( $B_{s_1}, B_{s_2}, r$ ), where  $s_i$  are the sons of  $r$ . The output,  $\text{deck}(B_r) = \text{deck}(T)$ , has in its first entry the required  $B^*(T)$ . The algorithm then halts.

If  $T$  has no point of degree 2, then let  $r$  be a point of degree 1. We proceed as above, calling on either LABEL or MAKEDECK as we work our way up  $T$  (rooted at  $r$ ). Eventually we call on MAKEDECK( $B_s, s$ ), where  $s$  is the unique son of  $r$ . The output,  $\text{deck}((B_s)_s) = \text{deck}(T)$ , gives  $B^*(T)$  and the algorithm halts.

In either case there are a total of  $n$  calls of the form LABEL( $B_{y_1}, B_{y_2}, x$ ) or MAKEDECK( $B_y, y$ ), each requiring at most  $O(\log n)$  time by Lemmas 9 and 10. Hence the total time is bounded by  $O(n \log n)$ .  $\square$

**4. A  $O(n)$  time bound.** In this section we describe informally a way of lowering the complexity of our algorithm to  $O(n)$ .

Let  $\text{Tree}(A) = \{\tau_i : 1 \leq i \leq l\}$  and  $\text{Tree}(C) = \{\gamma_j : 1 \leq j \leq k\}$ . The main idea is to modify procedure LABEL( $A, C, r$ ). In the worst case it processes all  $|\text{deck}(A)| + |\text{deck}(C)| \leq \log |A| + \log |C|$  signatures in  $\text{deck}(A)$  and  $\text{deck}(C)$ , giving a time bound  $O(\log n)$ ,  $n = |A \vee C|$ . We indicate how the addition of new decks that point to gaps in the sequences  $\{B^*(\tau_i)\}$  and  $\{B^*(\gamma_j)\}$  permits an  $O(\min\{\log |A|, \log |C|\})$  implementation of the procedure.

As motivation, suppose that LABEL( $A, C, r$ ) performs in recursion LABEL( $\tau_c, \gamma_d, r$ ), say with  $B^*(\tau_c) > B^*(\gamma_d)$ . Suppose also that for some  $e > c$  we have  $B^*(\tau_e) = B^*(\gamma_d)$  and  $B^*(\tau_{i+1}) = B^*(\tau_i) - 1$  for  $c \leq i \leq e - 1$ . Then we have the equivalences  $B^*(\tau_c \vee \gamma_d) \leq B^*(\tau_c) \Leftrightarrow B^*(\tau_{c+1} \vee \gamma_d) \leq B^*(\tau_c) - 1 \Leftrightarrow \dots \Leftrightarrow B^*(\tau_e \vee \gamma_d) \leq B^*(\tau_c) - (e - c)$ . Of course the falsity of the above inequalities yields  $B^*(\tau_c \vee \gamma_d) = B^*(\tau_c) + 1$  in which case  $\text{deck}(\tau_c \vee \gamma_d)$  has only one frame. Thus the result of LABEL( $\tau_e, \gamma_d, r$ ) telescopes up to decide the result of LABEL( $\tau_c, \gamma_d, r$ ). As now constituted, LABEL( $A, C, r$ ) performs the  $O(e - c)$  operations for achieving the telescoping. Our goal is to do this in  $O(1)$  time by using a pair of successive pointers from a separate deck, one to  $B^*(\tau_c)$  and the other to  $B^*(\tau_e)$ . When the inequalities are false, we advance the pointer to  $B^*(\tau_c)$  one unit and delete the one to  $B^*(\tau_e)$ , and when they are true we leave both pointers stationary. In this way we can treat entire segments of  $\text{deck}(A)$  (or  $\text{deck}(C)$ ), i.e., a sequence of successive frames having successive  $B^*$  values, in  $O(1)$  time. The result will be that the time spent by LABEL in processing  $\text{deck}(A)$  and  $\text{deck}(C)$  becomes proportional to the number of gaps in the sequence of  $B^*$  values occurring in whichever of  $\text{deck}(A)$  or  $\text{deck}(C)$  has fewer frames.

To be precise, we define two additional decks, GAP( $A$ ) and GAP( $C$ ), as follows. Suppose  $R_{jm} = \{\tau_j, \tau_{j+1}, \dots, \tau_m\}$  is a subsequence of  $\text{Tree}(A)$  satisfying  $B^*(\tau_{i+1}) = B^*(\tau_i) - 1$ ,  $j \leq i \leq m - 1$ ,  $B^*(\tau_{j-1}) > 1 + B^*(\tau_j)$ , and  $B^*(\tau_{m+1}) < B^*(\tau_m) - 1$ . We call  $R_{jm}$  a *run* of  $\text{Tree}(A)$  (and also of  $\text{deck}(A)$  under the correspondence between  $\text{deck}(A)$  and  $\text{Tree}(A)$ ). For any  $\tau \in \text{Tree}(A)$ , let  $\text{run}(\tau)$  denote the run of  $\text{Tree}(A)$  to which  $\tau$  belongs. For each run  $R_{jm}$  there will be two consecutive elements  $b(R_{jm}), t(R_{jm})$  in

the list  $GAP(A)$ . We let  $t(jm)$  point to  $B^*(\tau_j)$  in  $sign(\tau_j)$  and  $b(jm)$  point to  $B^*(\tau_m)$  in  $sign(\tau_m)$ . If  $R_{jm}$  and  $R_{qr}$  are runs with  $m < q$ , then the pointers to  $R_{jm}$  come before (i.e., above) those to  $R_{qr}$ . The deck  $GAP(C)$  is constructed similarly.

Our new procedure  $\overline{LABEL}(A, C, r)$  can be described informally as follows. It has input deck  $(A)$ , deck  $(C)$ ,  $GAP(A)$ ,  $GAP(C)$ , and output deck  $(A \vee C)$  and  $GAP(A \vee C)$ . Without loss of generality we take  $B^*(A) \cong B^*(C)$ . For brevity we consider only the first case where  $\log|C| \leq \log|A|$ ; the case  $\log|C| > \log|A|$  is done with straightforward but tedious changes. We let  $bottomdeck(T)$  be the bottommost entry of deck  $(T)$  for a tree  $T$ .

(I) Work your way up deck  $(A)$  from  $bottomdeck(A)$  to  $sign(\tau_u)$ , where  $u = \min\{j: B^*(\tau_j) \cong B^*(\gamma_1)\}$  if  $B^*(\tau_l) \cong B^*(\gamma_1)$  and  $u = l$  otherwise. Also work up  $GAP(A)$  until you reach  $b(run(\tau_u))$ .

(II) Perform  $\overline{LABEL}(\tau_u, \gamma_1, r)$ .

(III) Construct deck  $((A \vee C)_r)$  and  $GAP((A \vee C)_r)$  as follows.

(A) Suppose  $B^*(\tau_u \vee \gamma_1) = 1 + B^*(\gamma_1)$ , so that deck  $(\tau_u \vee \gamma_1)$  is the single frame  $sign(\tau_u \vee \gamma_1) = (1 + B^*(\gamma_1), 1, r, B^*(\gamma_1))$ .

(1) Check if there is a run  $R_{ab}$  in  $Tree(A)$  one of whose members  $\tau$  satisfies  $B^*(\tau) = 1 + B^*(\gamma_1)$ . (Note. Such an  $R_{ab}$  must be  $run(\tau_u)$  or the run preceding  $run(\tau_u)$  in  $Tree(A)$ .)

(2) If there is, then move up to  $t(R_{ab})$  in  $GAP(A)$  (up  $\leq 2$  pointers) and follow  $t(R_{ab})$  to the top frame  $sign(\tau_a)$  of  $R_{ab}$ . Now do  $sign(\tau_a \vee \gamma_1) \leftarrow (1 + B^*(\gamma_1), 1, r, 1 + B^*(\gamma_1))$ . Now deck  $(A \vee C)$  is given by  $deck(A \vee C) = deck(A)_{a-1} \oplus sign(\tau_a \vee \gamma_1)$ . Note that the list  $deck(A)_{a-1}$  may now be regarded as  $\{sign(\tau_j \vee \gamma_1)\}$ .

(3) If there is no  $R_{ab}$ , then let  $deck(A \vee C) = deck(A)_{u-1} \oplus sign(\tau_u \vee \gamma_1)$ . Again, the entries  $\{sign(\tau_j)\}$ ,  $1 \leq j \leq u-1$ , may be regarded as  $\{sign(\tau_j \vee \gamma_1)\}$ .

(4)  $GAP(A \vee C)$  is obtained by making corresponding changes in  $GAP(A)$ . Suppose for example that (2) holds, and let  $run(\tau_a \vee \gamma_1) = R_{\alpha\beta}$  in deck  $(A \vee C)$ . If  $R_{\alpha\beta}$  is not a singleton, then  $R_{\alpha\beta}$  corresponds to a run  $R_{\alpha(\beta-1)}$  in deck  $(A)$  with  $\tau_a \vee \gamma_1$  appended at the end. Hence we let  $t(R_{\alpha\beta}) = t(R_{\alpha(\beta-1)})$  while  $b(R_{\alpha\beta})$  points to  $sign(\tau_a \vee \gamma_1)$ . Naturally  $b(R_{\alpha(\beta-1)})$  in  $GAP(A)$  is deleted. All elements of  $GAP(A)$  corresponding to runs about  $R_{\alpha\beta}$  are retained in  $GAP(A \vee C)$ . We omit here the changes appropriate to (3) since similar changes are described below.

(B) Suppose  $B^*(\tau_u \vee \gamma_1) = B^*(\gamma_1)$ . Then deck  $(A \vee C)$  is given by  $deck(A \vee C) = deck(A)_{u-1} \oplus deck(\tau_u \vee \gamma_1)$ . Now  $GAP((A \vee C)_r)$  is constructed as follows.

(1) Create  $GAP(\tau_u \vee \gamma_1)$  by processing deck  $(\tau_u \vee \gamma_1)$  top to bottom and forming the pointers appropriate to each run.

(2) Attach  $GAP(\tau_u \vee \gamma_1)$  to  $GAP(A)$  as follows.

Let  $GAP'(A)$  be the subdeck of  $GAP(A)$  of pointers corresponding to  $run(\tau_{u-1})$  and all runs above  $run(\tau_{u-1})$ .

(a) If  $B^*(\tau_{u-1}) > 1 + B^*(\gamma_1)$ , then just append  $GAP(\tau_u \vee \gamma_1)$  to the bottom of  $GAP'(A)$  and retain the pointer structure in each.

(b) If  $B^*(\tau_{u-1}) = 1 + B^*(\gamma_1)$ , then append as above but remove  $b(run(\tau_{u-1}))$  from  $GAP'(A)$  and  $t(run(\tau_u \vee \gamma_1))$  from  $GAP(\tau_u \vee \gamma_1)$ . (This has the effect of merging the runs  $run(\tau_{u-1})$  and  $run(\tau_u \vee \gamma_1)$  into one in  $GAP(\tau_u \vee \gamma_1)$ .)

We now examine the complexity of our procedure.

**THEOREM 2.**  $\overline{\text{LABEL}}(A, C, r)$  has complexity  $O(\min\{\log|C|, \log|A|\})$ .

*Proof.* For brevity we again restrict ourselves to the case  $B^*(A) \geq B^*(C)$  and  $\log|C| \leq \log|A|$  treated above.

First, (I) and (II) each require  $O(\log(C))$  time since each processes at most  $O(B^*(\gamma_1)) = O(\log|C|)$  frames. The construction of deck  $(A \vee C)$  takes  $O(1)$  time since it consists in just appending deck  $(\tau_u \vee \gamma_1)$  below a certain position in deck  $(A)$ . This position is accessed in  $O(1)$  time since after performing (I) and (II) we are just below the pointer in  $\text{GAP}(A)$  that leads to this position.

$\text{GAP}(A \vee C)$  is constructed in  $O(1)$  time in case (A), and in  $O(\log|C|)$  time in case (B). In (A) we simply adjust one or two pointers at the position of  $\text{GAP}(A)$  reached after (I) and (II). In (B) we first construct  $\text{GAP}(\tau_u \vee \gamma_1)$ . This takes  $O(\log|C|)$  time since we are just scanning frames  $\text{sign}(\tau_i \vee \gamma_j)$  of deck  $(\tau_u \vee \gamma_1)$  all with  $B^*$  values  $\leq B^*(\gamma_1)$ . The number of such frames is at most  $B^*(\gamma_1)$  by monotonicity of  $\text{Tree}(\tau_u \vee \gamma_1)$  from Lemma 8, and  $B^*(\gamma_1) \leq O(\log|\gamma_1|) = O(\log|C|)$  by Corollary 8.1. We then splice together  $\text{GAP}(\tau_u \vee \gamma_1)$  and  $\text{GAP}'(A)$  by at worst deleting two pointers (the position below which we have already accessed), and this is  $O(1)$  time. Hence the total time of  $\overline{\text{LABEL}}(A, C, r)$  is  $O(\log|C|)$ , as required.  $\square$

In a similar way we can define a new procedure  $\overline{\text{MAKEDECK}}(T, r)$  which does the same job as the old procedure  $\text{MAKEDECK}(T, r)$ ; that is, it constructs deck  $((T), r)$  from deck  $(T)$ . By creating additional pointer structure, we can make the new procedure run in time  $O(1)$  instead of the worst-case  $O(\log n)$  for the old procedure. A brief sketch of how this is done follows.

As motivation, recall that  $\text{MAKEDECK}(T, r)$  constructs deck  $((T), r)$  from deck  $(T)$  by doing one of the following operations:

- (op1) inserting a frame at the bottom of deck  $(T)$ ,
- (op2) replacing the bottom frame of deck  $(T)$  by a different frame, or
- (op3) finding the topmost frame of deck  $(T)$  in which  $B^*((\gamma_j), r) \neq B^*(\gamma_j)$  (if such a frame exists), and replacing it and the frames under it (i.e., those associated with  $\gamma_i$ , where  $i > j$ ) by the single frame  $(B^*(\gamma_j) + 1, 1, \bar{r}, B^*(\gamma_j) + 1)$ .

Notice that to do any one of these operations we need only access an “extreme” frame of some type: either the bottom frame in deck  $(T)$ , or the topmost relative to the property  $B^*((\gamma_j), r) \neq B^*(\gamma_j)$ .

This suggests the construction of a new deck; call it  $\text{FRAME}(T)$ . The elements of  $\text{FRAME}(T)$  will be pointers to those frames of deck  $(T)$  in which  $B^*((\gamma_j), r) \neq B^*(\gamma_j)$ , and, if not already present among the elements just mentioned, an additional pointer to the bottom frame of deck  $(T)$ . These elements will be ordered top to bottom in  $\text{FRAME}(T)$  in the same relative order as the frames of deck  $(T)$  to which they point. We can then access the required frame of deck  $(T)$  in each case by using either the topmost or bottommost element of  $\text{FRAME}(T)$ . Finding either of these elements takes  $O(1)$  time since they are at the top or bottom of  $\text{FRAME}(T)$ .

Now the procedure  $\overline{\text{MAKEDECK}}(T, r)$  operates by invoking  $\text{MAKEDECK}(T, r)$  and using  $\text{FRAME}(T)$  to access the frame in deck  $(T)$  under which the insertion specified by  $\text{MAKEDECK}$  will take place. The resulting construction of deck  $((T), r)$  takes  $O(1)$  time since it involves placing a single frame under or at the accessed frame of deck  $(T)$ .

Finally we sketch how the time required to maintain  $\text{FRAME}(T)$  is dominated by the complexity of procedures already in use, so that the total complexity of our algorithm is unaffected (up to a constant term) by the additional pointer structure. To update  $\text{FRAME}(T)$  we need to do the following:



(1) Construct  $\text{FRAME}((T)_r)$  from  $\text{FRAME}(T)$  (after  $\overline{\text{MAKEDECK}}(T, r)$  has constructed deck  $((T)_r)$  from deck  $(T)$ ).

(2) Construct  $\text{FRAME}(A \vee C)$  from  $\text{FRAME}(A)$  and  $\text{FRAME}(C)$  (after  $\overline{\text{LABEL}}(A, C, r)$  has constructed deck  $(A \vee C)$  from deck  $(A)$  and deck  $(C)$ ).

Upon starting task (1), note that we would have already performed one of the operations (op1), (op2), or (op3) during  $\overline{\text{MAKEDECK}}(T, r)$ . If (op2) was performed then  $\text{FRAME}((T)_r)$  is obtained from  $\text{FRAME}(T)$  by reorienting the bottom element of  $\text{FRAME}(T)$  to point to the bottom of deck  $((T)_r)$ . The remaining elements of  $\text{FRAME}(T)$  remain the same, except now regarded as members of  $\text{FRAME}((T)_r)$ . If (op3) was performed, then  $\text{FRAME}((T)_r)$  is just a single element pointing to the bottom frame of deck  $((T)_r)$ . If (op1) was performed, then the updating depends on whether

- (i) the bottom element of deck  $(T)$  has the property  $B^*((\gamma_m)_r) \neq B^*(\gamma_m)$ , or
- (ii) the bottom element of deck  $(T)$  does not have this property.

If (i) holds then  $\text{FRAME}((T)_r)$  is gotten by inserting a new element at the bottom of  $\text{FRAME}(T)$ , and pointing this element to the bottom of deck  $((T)_r)$ . If (ii) holds, then  $\text{FRAME}((T)_r)$  is obtained by reorienting the bottom element of  $\text{FRAME}(T)$  to point to the bottom of deck  $((T)_r)$ .

Notice that any of these updates requires at most  $O(1)$  time since it involves a  $O(1)$  manipulation (insertion and orientation of a pointer to the bottom of some deck) at or directly below an element of  $\text{FRAME}(T)$  already accessed in  $O(1)$  time. It is also easily checked that the deck  $\text{FRAME}((T)_r)$  so produced has the necessary properties, namely, elements pointing to the frames of deck  $((T)_r)$  in which  $B^*((\gamma_j)_r) \neq B^*(\gamma_j)$  and ordered in the natural way, and a bottom element (if not already present) pointing to the bottom of deck  $((T)_r)$ .

We can achieve (2) in a manner analogous to the construction of  $\text{GAP}(A \vee C)$  from  $\text{GAP}(A)$  and  $\text{GAP}(C)$  outlined previously. This requires time  $O(\min\{\log|A|, \log|C|\})$  which is already required by the procedure  $\overline{\text{LABEL}}$ . Details of the construction and time bound in (2) are omitted here since they are nearly identical with the above-mentioned construction and time analysis for  $\text{GAP}(A \vee C)$ .

Our procedures  $\overline{\text{LABEL}}$  and  $\overline{\text{MAKEDECK}}$  can now be used to give our linear algorithm.

**THEOREM 3.** *Let  $T$  be a tree of maximum degree 3 on  $n$  points. Then  $B^*(T)$  can be computed in  $O(n)$  time.*

*Proof.* We carry out our procedures in the order specified by the proof of Theorem 1. That is, we work our way up  $T$  from bottom to top in postorder. Thus we start by defining deck  $((e))$  for each endpoint  $e$  as before. Now as we meet each new point  $x \in T$  in postorder, our job is to compute deck  $(B_x)$  given deck  $(B_y)$  for each of the (at most) two children  $y$  of  $x$ . If  $x$  has two children  $y_1$  and  $y_2$ , then we apply  $\overline{\text{LABEL}}(B_{y_1}, B_{y_2}, x)$  to get deck  $(B_x)$  at a cost of  $\log(\min(|B_{y_1}|, |B_{y_2}|))$  time by Theorem 2. If  $x$  has one child  $y$ , then we apply  $\overline{\text{MAKEDECK}}(B_y, y)$  to get deck  $(B_x)$  at a cost of  $O(1)$  time by the above discussion. At the end we finally obtain deck  $(B_r) = \text{deck}(T)$ , where  $r$  is the root of  $T$  and from that we get  $B^*(T)$  by scanning the topmost frame of this deck.

Now let  $T(n)$  be the total time required by the algorithm described in the above paragraph. It follows from the description that  $T(n+1) \leq \max_{1 \leq k \leq n/2} \{T(k) + T(n-k) + \log(k)\}$ .

We now claim that  $T(n)$  satisfies

$$T(n) \leq An - B \log(n) + C$$

for some constants  $A$ ,  $B$ , and  $C$ . Of course, the theorem follows from this claim. Proceeding by induction, suppose the inequality holds for all  $n \leq p$ . Using the inductive assumption we then have

$$\begin{aligned}
 T(p+1) &\leq \max_{1 \leq k \leq p/2} \{T(k) + T(n-k) + \log(k)\} \\
 (*) \quad &\leq \max_{1 \leq k \leq p/2} \{Ak - B \log(k) + A(p-k) - B \log(p-k) + \log(k) + 2C\} \\
 &= \max_{1 \leq k \leq p/2} \{Ap - \log(k^{B-1}(p-k)^B) + 2C\}.
 \end{aligned}$$

We now show that there are constants  $B$  and  $C$  such that the inequality

$$(**) \quad \max_{1 \leq k \leq p/2} \{-\log(k^{B-1}(p-k)^B) + 2C\} \leq -B \log(p+1) + C$$

holds for  $p$  sufficiently large. Clearly (\*\*) is equivalent to  $\min_{1 \leq k \leq p/2} \{k^{B-1}(p-k)^B\} \geq (p+1)^{B-2} C$ . Choosing  $B=2$ , we find that for  $p \geq 4$  the minimum in the left-hand side is achieved at  $k=1$  and is therefore  $(p-1)^2$ . Now with  $C=-2$  we get (\*\*) to hold for  $p \geq 4$ . Thus (\*\*) holds with  $B=2$  and  $C=-2$  for all  $p \geq 4$ .

Now combining (\*) and (\*\*) and observing that  $Ap < A(p+1)$  we see that the inductive step is completed. It remains only to choose  $A$  sufficiently large relative to  $B=2$  and  $C=-2$  ( $A \geq 20$  will do) so that the claimed bound for  $T(n)$  holds for  $n \leq 3$ , thereby providing the base for the induction. The theorem is thus proved.  $\square$

**Acknowledgments.** The author thanks I. H. Sudborough and J. Ellis for discussions which stimulated the idea for the additional data structure used in the last section, and Robert Tarjan for suggesting the method of proving that the linear time bound follows from the recurrence satisfied by the time complexity. Thanks also go to Ruth Engel for executing the drawings.

#### REFERENCES

- [1] I. ARANY, L. SZODA, AND W. SMITH, *An improved method for reducing the bandwidth of sparse symmetric matrices*, Proc. 1971 IFIP Congress, pp. 1246-1250.
- [2] M. BEHZAD, G. CHARTRAND, AND L. LESNIAK-FOSTER, *Graphs and Digraphs*, Prindle, Weber, Schmidt, Boston, MA, 1979.
- [3] J. CHVATALOVA, *On the bandwidth problem for graphs*, Ph.D. thesis, Dept. of Combinatorics and Optimization, Univ. of Waterloo, Waterloo, Ontario, Canada, 1981.
- [4] P. CHINN, J. CHVATALOVA, A. K. DEWDNEY, AND N. E. GIBBS, *The bandwidth problem for graphs and matrices*, J. Graph Theory, 6 (1982), pp. 223-254.
- [5] F. R. K. CHUNG, *On the cutwidth and topological bandwidth of a tree*, SIAM J. Algebraic Discrete Methods, 6 (1985), pp. 268-277.
- [6] M. R. GAREY, R. L. GRAHAM, D. S. JOHNSON, AND D. KNUTH, *Complexity results for bandwidth minimization*, SIAM J. Appl. Math., 34 (1978), pp. 477-495.
- [7] E. GURARI AND I. H. SUDBOROUGH, *Improved dynamic programming algorithms for the bandwidth minimization problem and the min cut linear arrangement problem*, J. Algorithms, 5 (1984), pp. 531-546.
- [8] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.
- [9] W. LIU AND A. B. SHERMAN, *Comparative analysis of the Cuthill-McKee ordering algorithms for sparse matrices*, SIAM J. Numer. Anal., 13 (1976), pp. 198-213.
- [10] F. S. MAKEDON, C. H. PAPADIMITRIOU, AND I. H. SUDBOROUGH, *Topological bandwidth*, SIAM J. Algebraic Discrete Methods, 6 (1985), pp. 418-444.
- [11] B. MONIEN AND I. H. SUDBOROUGH, *Bandwidth constrained NP-complete problems*, Theoret. Comput. Sci., 41 (1985), pp. 141-167.
- [12] T. OHTSUKI, H. MORI, E. S. KUH, T. KASHIWABARA, AND T. FUJISAWA, *One-dimensional logic gate assignments and interval graphs*, IEEE Trans. Circuits and Systems, 26 (1979), pp. 675-684.

- [13] C. H. PAPANIMITRIOU, *The NP-completeness of the bandwidth minimization problem*, Computing, 16 (1976), pp. 237-267.
- [14] J. B. SAXE, *Dynamic programming algorithms for recognizing small bandwidth graphs in polynomial time*, SIAM J. Algebraic Discrete Methods, 1 (1980), pp. 363-369.
- [15] I. H. SUDBOROUGH, *Bandwidth constraints on problems complete for polynomial time*, Theoret. Comput. Sci., 26 (1983), pp. 25-52.
- [16] S. TRIMBERG, *Automated chip layout*, IEEE Spectrum, 19 (1982), pp. 38-45.
- [17] M. YANNAKAKIS, *A polynomial algorithm for the min cut linear arrangement of trees*, J. Assoc. Comput. Mach., 32 (1985), pp. 950-959.
- [18] T. LENGAUER, *Black-white pebbles and graph separation*, Acta Inform., 16 (1981), pp. 99-113.

## OPTIMAL BOUNDS FOR SOLVING TRIDIAGONAL SYSTEMS WITH PRECONDITIONING\*

PAOLO ZELLINI†

**Abstract.** Let (1)  $T\mathbf{x} = \mathbf{f}$  be a linear tridiagonal system of  $n$  equations in the unknowns  $x_1, \dots, x_n$ . It is proved that  $3n - 2$  (nonscalar) multiplications/divisions are necessary to solve (1) in a straight-line program excluding divisions by elements of  $\mathbf{f}$ . This bound is optimal if the cost of preconditioning of  $T$  is not counted. Analogous results are obtained in case (i)  $T$  is bidiagonal and (ii)  $T$  and  $\mathbf{f}$  are both centrosymmetric. The existence of parallel algorithms to solve (1) with preconditioning and with minimal multiplicative redundancy is also discussed.

**Key words.** multiplicative complexity, LU factorization, straight-line algorithms, parallel algorithms, tridiagonal systems, rational preconditioning

**AMS(MOS) subject classifications.** 68Q25, 65F05

**1. Introduction.** It is well known that the inverse  $S$  of a tridiagonal matrix  $T$  has a special structure which can be described in terms of a few vectors [2], [3], [7], [9], [24]. More precisely, if  $T$  has dimension  $n \times n$  and  $\det T \neq 0$ , then two  $n$ -vectors  $\mathbf{u} = [u_1, \dots, u_n]^T$  and  $\mathbf{v} = [v_1, \dots, v_n]^T$ , and one  $(n-1)$ -vector  $\mathbf{w} = [w_1, \dots, w_{n-1}]^T$  are sufficient to define  $S = T^{-1}$  (for more details see § 2). The amount of information to define  $T^{-1}$  can also be reduced in case  $T$  has a special structure: in particular if  $T$  is symmetric or centrosymmetric (see [7], [9], and § 2).

Observe that a general tridiagonal  $n \times n$  matrix  $T$  is a linear combination of  $3n - 2$  rank-one  $(0, 1)$  matrices. Thus  $3n - 2$  is the multiplicative complexity of a matrix-vector product  $T\mathbf{f}$ , even if all entries (indeterminates) in  $T$  and in  $\mathbf{f}$  commute with each other (this is a trivial consequence of the first results on the multiplicative complexity of a set of bilinear forms stated in [8], [10], [15], [20], [6]). Now, by the special structure of  $S = T^{-1}$ , we suspect that the same multiplicative complexity ( $3n - 2$  multiplicative operations) is in a special sense relevant to the "inverse" problem of computing the vector  $T^{-1}\mathbf{f} = S\mathbf{f}$ , or, equivalently, for solving a system of linear equations  $T\mathbf{x} = \mathbf{f}$ . More specifically, it will be proved in the next sections that  $3n - 2$  is an optimal lower bound to solve, with preconditioning, a general  $n \times n$  tridiagonal system  $T\mathbf{x} = \mathbf{f}$ . This bound holds for all straight-line programs, relative to a ground field  $G$  and to the set of indeterminates present in  $T$  and in  $\mathbf{f}$ ; the only restriction is excluding divisions by polynomials in the  $f_i$ 's. This last restriction does not seem to be serious since in all best known sequential and parallel algorithms for solving  $T\mathbf{x} = \mathbf{f}$  (LU decomposition, Cramer's rule, cyclic odd-even reduction, QR algorithms, methods using explicit formulas for the entries of  $T^{-1}$ ) there is no division by such polynomials. Analogous results can easily be stated for bidiagonal systems and for systems  $T\mathbf{x} = \mathbf{f}$ , where  $T$  is centrosymmetric and  $f_i = f_{n-i+1}$ .

**2. Preliminaries.** Let  $T = (t_{ij})$  be a general, i.e., with indeterminate entries,  $n \times n$  tridiagonal matrix,  $\det T \neq 0$ ,

$$(2.1) \quad t_{ij} = \begin{cases} a_i, & i = j, \\ b_i, & j - i = 1, \\ c_j, & i - j = 1, \\ 0 & \text{otherwise} \end{cases}$$

\* Received by the editors December 2, 1985; accepted for publication (in revised form) November 3, 1987.

† Dipartimento di Matematica e Informatica, Università di Udine, 6-33100 Udine, Italy.

and let  $S = (s_{ij})$  be a general matrix,  $\det S \neq 0$ , defined by

$$(2.2) \quad s_{ij} = \begin{cases} v_i u_j \prod_{q=i}^{j-1} w_q, & i < j, \\ u_i v_j, & i \cong j. \end{cases}$$

The relationship between (2.1) and (2.2) and an explicit formula for  $\det S$  are stated in the following proposition.

PROPOSITION 1 [9].

- (i)  $T^{-1}$  has the form (2.2).
- (ii)  $S^{-1}$  is tridiagonal.
- (iii)  $\det S = v_1 u_n \prod_{q=1}^{n-1} (u_q v_{q+1} - w_q u_{q+1} v_q)$ .

Observe that the  $3n - 2$  entries of  $T = S^{-1}$  are given, in explicit form, by [9]:

$$(2.3) \quad \begin{aligned} a_1 &= v_2 q_1 / v_1, & b_1 &= -w_1 q_1, \\ c_{i-1} &= -q_{i-1}, \\ a_i &= -p_i q_{i-1} q_i, \\ b_i &= -w_i q_i, & i &= 2, 3, \dots, n-1, \\ c_{n-1} &= -q_{n-1}, & a_n &= u_{n-1} q_{n-1} / u_n \end{aligned}$$

where

$$q_j = 1 / (u_j v_{j+1} - u_{j+1} v_j w_j), \quad p_j = u_{j+1} v_{j-1} w_{j-1} w_j - u_{j-1} v_{j+1}.$$

Observe that if  $w_q = 1$  in (2.2), then both  $S$  and  $S^{-1}$  are symmetric. Also, if  $w_q = 0$ , then  $S^{-1}$  is bidiagonal and the inverse of a bidiagonal  $n \times n$  has the form (2.2) with  $w_k = 0$ . The results in Proposition 1 are to be modified slightly in case  $T$  is centrosymmetric, i.e.,  $a_i = a_{n-i+1}$  and  $c_j = b_{n-j}$  in (2.1). More precisely, let  $M = (m_{ij})$  be a general centrosymmetric matrix ( $\det M \neq 0$ ) defined by

$$(2.4) \quad m_{ij} = \begin{cases} u_i v_{n-j+1}, & i \leq j, \\ u_{n-i+1} v_j, & i \geq j, \end{cases}$$

with  $u_i v_{n-i+1} \equiv u_{n-i+1} v_i$ .

Then we have the following proposition.

PROPOSITION 2.

- (i) If  $T$  in (2.1) is centrosymmetric, then  $T^{-1}$  has the form (2.4).
- (ii)  $M^{-1}$  is tridiagonal and centrosymmetric.
- (iii)  $\det M = u_1 v_1 \prod_{q=1}^{n-1} (u_{n-q+1} v_{q+1} - u_q v_{n-q})$ .

*Proof.* We can see Proposition 2 as a consequence of Proposition 1; in fact (2.2) and (2.4) can be made identical by reindexing the indeterminates  $u$  in (2.4) and by giving proper values to  $w_q$  in (2.2), i.e.,  $w_q = u_{n-q+1} u_{q+1}^{-1} v_{n-q} v_q^{-1}$ . The explicit formulas for the entries of  $M^{-1}$  are given by

$$(2.5) \quad \begin{aligned} a_1 &= -v_2 z_{n-1} / v_1, \\ b_i &= z_i, & i &= 1, 2, \dots, n-1, \\ a_i &= -t_i z_{i-1} z_i, & i &= 1, 2, \dots, \lfloor n/2 \rfloor, \\ a_m &= -(v_{m-1} + v_{m+1}) / u_m v_m (v_{m-1} - v_{m+1}) \quad (\text{for } n \text{ odd}), \end{aligned}$$

where  $m = \lfloor n/2 \rfloor + 1$  and

$$z_j = 1 / (u_{n-j} v_j - u_{j+1} v_{n-j+1}), \quad t_j = u_{n-j} v_{j-1} - u_{j+1} v_{n-i+2}.$$

Observe that the previous results can be extended to block tridiagonal matrices (though we do not develop this extension here in detail).

**3. Tridiagonal and bidiagonal systems.** Let

$$(3.1) \quad T\mathbf{x} = \mathbf{f}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}$$

be a system of linear equations where  $T$  is an  $n \times n$  general tridiagonal matrix of the form (2.1). In solving (3.1) we consider all (direct) straight-line algorithms in  $F[f_1, f_2, \dots, f_n]$  (thus excluding divisions by the  $f_i$ 's) where  $F = G(a_1, \dots, a_n, b_1, \dots, b_{n-1}, c_1, \dots, c_{n-1})$ ,  $G$  is a subfield of  $F$  and  $a_i, b_i, c_j, f_i$  are indeterminates. An operation  $\omega_1 \odot \omega_2$ , where  $\odot$  is a multiplication or a division, is said to be *active*, if none of the following conditions is satisfied:

- (i)  $\omega_2 \in G$ .
- (ii)  $\omega_1 \in G$  and  $\odot$  is a multiplication.
- (iii)  $\omega_1 \in F$  and  $\omega_2 \in F$ .

Thus the *inactive* operations are either scalar multiplications or involve only elements of  $G(\mathbf{a}, \mathbf{b}, \mathbf{c})$  [6]. The cost of the operations in  $G(\mathbf{a}, \mathbf{b}, \mathbf{c})$  is the cost of rational preconditioning of  $T$ .

Also in computing the vector

$$(3.2) \quad \mathbf{Sf} \quad \mathbf{S} \text{ defined by (2.2)}$$

with a straight-line algorithm in  $F[f_1, f_2, \dots, f_n]$ ,  $F = G(u_1, \dots, u_n, v_1, \dots, v_n, w_1, \dots, w_{n-1})$ ,  $G \subseteq F$ , an *active* operation (multiplication or division) is defined exactly as in the case of (3.1), except for replacing  $a_i, b_i, c_j$  by  $u_i, v_i, w_q$ .

The following result then holds.

**THEOREM 1.** *Every straight-line program solving (3.1) with preconditioning in  $F[f_1, f_2, \dots, f_n]$  uses at least  $3n - 2$  active operations. This bound is optimal.*

*Proof.* Solving (3.1) with the classical LU decomposition (where  $L$  and  $U$  are bidiagonal and the diagonal entries of either  $L$  or  $U$  are equal to 1) costs, exactly,  $3n - 2$  active operations in  $F[\mathbf{f}]$  [13]. Now let  $A'$  be an algorithm solving (3.1) with  $t$  active operations. Then we are able to define an algorithm  $A$  in  $F[\mathbf{f}]$ ,  $F = G(\mathbf{u}, \mathbf{v}, \mathbf{w})$ , computing  $\mathbf{Sf}$  in (3.2) with  $t$  active operations. In fact the two main steps of  $A$  are:

- (i) Compute  $T = S^{-1}$  according to (2.3) using only inactive operations in  $G(\mathbf{u}, \mathbf{v}, \mathbf{w})$ .
- (ii) Solve the system  $T\mathbf{x} = \mathbf{f}$  in  $F[f_1, f_2, \dots, f_n]$ , with  $F = G(\mathbf{u}, \mathbf{v}, \mathbf{w})$ , by exploiting the steps of  $A'$ .

Let  $G[[y_1, \dots, y_k]]$  denote the ring of power series in  $y_1, \dots, y_k$  with coefficients in  $G$  and let  $J(y_1, \dots, y_k)$ , or  $J(\mathbf{y})$ , be the ideal spanned by the third-order terms in  $y_1, \dots, y_k$ . Now every computation in  $F[\mathbf{f}]$ ,  $F = G(\mathbf{u}, \mathbf{v}, \mathbf{w})$  can be simulated by a computation in  $G[[\tilde{u}_1, \dots, \tilde{u}_n, \tilde{v}_1, \dots, \tilde{v}_n, \tilde{w}_1, \dots, \tilde{w}_{n-1}, f_1, \dots, f_n]]$ , without divisions, where

$$(3.3) \quad \begin{aligned} \tilde{u}_i &= u_i - k_i, \\ \tilde{v}_i &= v_i - h_i, \quad 1 \leq i \leq n, \\ \tilde{w}_q &= w_q - l_q, \quad 1 \leq q \leq n - 1, \end{aligned}$$

with  $k_i, h_i, l_q \in G$ . As usual [5], [6], [21], the change of indeterminates (3.3) is necessary to guarantee that only divisions by units in  $G[[\tilde{\mathbf{u}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}}, \mathbf{f}]]$  occur. Here we suppose also

that  $k_i, h_i, l_q$  are all different from zero. Call  $S'$  the matrix obtained from  $S$  by the substitution (3.3), so each entry of  $S'$  is a nonhomogeneous polynomial, including linear terms, in the new indeterminates  $\tilde{u}_i, \tilde{v}_i, \tilde{w}_q$ . Now reduce the computation in  $G[[\tilde{\mathbf{u}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}}, \mathbf{f}]]$  modulo  $J(\tilde{\mathbf{u}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}}, \mathbf{f})$  to obtain an algorithm  $B$  computing only the quadratic and linear part of  $S'\mathbf{f}$ . As each term in each entry of  $S'\mathbf{f}$  is linear in the  $f_i$ 's, the vector computed by  $B$  has no quadratic entry only in the indeterminates  $\tilde{u}_i, \tilde{v}_i, \tilde{w}_q$ . Thus  $B$  computes a vector

$$(3.4) \quad \tilde{S}\mathbf{f} + \mathbf{g}$$

where the elements of  $\tilde{S}$  are linear homogeneous functions in  $\tilde{u}_i, \tilde{v}_i, \tilde{w}_q$  and each entry of  $\mathbf{g}$  is linear in the  $f_i$ 's. It is well known [21], [5], [6], that the reduction mod  $J(\tilde{\mathbf{u}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}}, \mathbf{f})$  can be performed so that  $B$  uses only  $t$  active multiplications in  $G(\tilde{\mathbf{u}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}}, \mathbf{f})$ , whose factors are linear homogeneous functions in  $\tilde{u}_i, \tilde{v}_i, \tilde{w}_q, f_i$ . The explicit form of  $\tilde{S} = (\tilde{s}_{ij})$  is given by

$$(3.5) \quad \tilde{s}_{ij} = \begin{cases} h_i k_j \sum_{\mu=i}^{j-1} \xi_\mu \tilde{w}_\mu + \xi_{ij}(h_i \tilde{u}_j + k_j \tilde{v}_i), & i < j, \\ h_j \tilde{u}_i + k_i \tilde{v}_j, & i \geq j \end{cases}$$

where  $\xi_\mu = \prod_{\nu \neq \mu} l_\nu$ ,  $\xi_{ij} = \prod_{\nu=i}^{j-1} l_\nu$ . Now, according to the rank-immunity criterion [16], [14], [4], observe that the matrix  $S$  satisfies the following conditions: (i) replacing any subset  $\Gamma'$  of  $\Gamma = \{\tilde{v}_2, \dots, \tilde{v}_n, \tilde{w}_1, \dots, \tilde{w}_{n-1}\}$  by a  $G$ -linear combination of the remaining indeterminates of  $S$  cannot make any of the  $\{\tilde{v}_2, \dots, \tilde{v}_n, \tilde{w}_1, \dots, \tilde{w}_{n-1}\} \setminus \Gamma'$  disappear in  $\tilde{S}$ ; (ii) the row-rank of the matrix (3.5) is  $n$  even if each indeterminate in the set  $\Gamma$  is replaced by a  $G$ -linear combination of  $\tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_n$ . In other words,  $S$  is "n-immune" in  $\tilde{v}_2, \dots, \tilde{v}_n, \tilde{w}_1, \dots, \tilde{w}_{n-1}$ . (This is shown even more directly by setting  $v_1 = 1$  in  $S$ , without loss of generality, which forces the first column of  $\tilde{S}$  to become  $[\tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_n]^T$ .) As  $\Gamma$  has  $2(n-1)$  elements, we obtain  $t \geq n + 2(n-1) = 3n - 2$ .

A last remark is needed for the cardinality of  $G$ . We have tacitly assumed that the number of elements in  $G$  is infinite. But there is no loss of generality in this assumption: in fact, by introducing an extra indeterminate  $y$  and using  $F' = F(y)$  and  $G' = G(y)$  instead of  $F$  and  $G$ , respectively, we obtain that the ground field  $G'$  has infinitely many elements (essentially the same argument is used, for instance, in [23]). Thus the theorem holds for any field  $G$ .

*Remark 1.* Excluding divisions by polynomials in the  $f_i$ 's has the following justification: allowing such divisions forces us to extend to  $f_i$  the change of indeterminates (3.3). This modifies the task (3.4) by the addition of quadratic terms only in the  $\tilde{u}_i, \tilde{v}_i, \tilde{w}_q$ 's so that the rank-immunity criterion [15] can no longer be applied.

*Remark 2.* In the previous proof it is correct to assume that the leading principal minors of a general matrix  $T$  (or  $S^{-1}$ ) are nonzero (so the above-mentioned decomposition LU holds for  $T$ . Otherwise the degrees of freedom in  $T$  (or in  $S$ ) would be reduced and Theorem 1 would not be true (for example, five active operations would suffice to solve (3.1) with  $n = 3$  and  $a_1 a_2 - b_1 c_1 = 0$ ).

An analogous result can be stated if  $T$  is bidiagonal (say,  $b_i = 0$  in (3.1)). In this case a lower bound of  $2n - 1$  operations is obtained which is optimal for all straight-line algorithms solving  $T\mathbf{x} = \mathbf{f}$  in  $F[\mathbf{f}]$ , with  $F = G(\mathbf{a}, \mathbf{c})$ .

**THEOREM 2.** *Every straight-line algorithm solving an  $n \times n$  bidiagonal system  $T\mathbf{x} = \mathbf{f}$  in  $F[\mathbf{f}]$ ,  $F = G(\mathbf{a}, \mathbf{c})$ , requires  $2n - 1$  active operations. This bound is optimal.*

*Proof.* The proof is analogous to that of Theorem 1. The matrix  $\tilde{S}$  in (3.4), with  $v_1 = 1$  (without loss of generality) and  $w_j = 0$  in (2.2), has elements  $\tilde{s}_{ij}$  defined by

$$\tilde{s}_{ij} = \begin{cases} \tilde{u}_i, & j = 1, \\ h_j \tilde{u}_i + k_i \tilde{v}_j, & i \geq j > 1, \\ 0 & \text{elsewhere.} \end{cases}$$

Clearly this matrix is  $n$ -immune in  $\tilde{v}_2, \tilde{v}_3, \dots, \tilde{v}_n$ . The trivial algorithm by forward-substitution employs exactly  $2n - 1$  (active) operations.

The LU decomposition is not the unique best preconditioning. In fact, for a given tridiagonal  $T \ n \times n$ , we can easily calculate three diagonal matrices  $D_1 = \text{diag}(\alpha_1, \alpha_2, \dots, \alpha_n)$ ,  $D_2 = \text{diag}(\beta_1, \beta_2, \dots, \beta_n)$ , and  $D = \text{diag}(d_1, d_2, \dots, d_n)$  such that

$$(3.6) \quad D_1 T D_2 = R D R^T$$

where  $R = (r_{i,j})$  is defined by

$$(3.7) \quad r_{ij} = \begin{cases} 1, & i = j, \\ -1, & i - j = 1, \\ 0 & \text{elsewhere.} \end{cases}$$

It is easy to see that  $\alpha_i, \beta_i$ , which are supposed to be different from zero, satisfy the second-order linear recurrences (for  $b_i, c_i \neq 0$ )

$$(3.8) \quad \alpha_i = -\frac{a_{i-1}\alpha_{i-1} + b_{i-2}\alpha_{i-2}}{c_{i-1}}, \quad i = 2, 3, \dots, n$$

$$\beta_i = -\frac{a_{i-1}\beta_{i-1} + c_{i-2}\beta_{i-2}}{b_{i-1}},$$

where  $\alpha_1$  and  $\beta_1$  are arbitrary ( $\alpha_1, \beta_1 \neq 0$ ) and  $\alpha_2 = -a_1\alpha_1/c_1, \beta_2 = -a_1\beta_1/b_1$ .

We can see that  $\alpha_i, \beta_i$  are proportional to the leading principal minors of  $T$ , so  $\alpha_i, \beta_i \neq 0$  in case these minors are nonzero or  $T$  is diagonally dominant.

By (3.8) we can solve (3.1) with  $3n - 2$  active operations by the sequence of matrix-vector products indicated by the formula

$$(3.9) \quad \mathbf{x} = D_2(R^T)^{-1}D^{-1}R^{-1}D_1\mathbf{f}$$

where  $R^{-1}$  is the lower-triangular matrix whose nonzero entries are all equal to 1.

Observe that (3.8) and (3.9) produce a parallel algorithm  $P$  in an SIMD model to solve (3.1) in  $O(\log n)$  steps with  $O(n)$  processors. (Recall that SIMD is the abbreviation for "Single Instruction Multiple Data," which means roughly that all processors interpret the same instructions and execute them on different data.) The number of active operations in this algorithm is still  $3n - 2$ , so  $P$  has minimal multiplicative redundancy if the cost of preconditioning is not counted.  $P$  should be compared with other known parallel algorithms: the modified LU method of Stone [19] employs  $O(n \log n)$  active operations to solve two bidiagonal systems. The methods of odd-even reduction [12], [17] and Swarztrauber's algorithm [22] have low redundancy but employ more than  $3n - 2$  active operations. The cyclic odd-even reduction also extends to block tridiagonal matrices, but the blocks have to be commuting [11]. The formulas (3.6)–(3.9) extend to block matrices on condition that the nondiagonal square blocks of  $T$  and the leading block submatrices of both  $T$  and its (block) transpose are nonsingular. In other words the previous algorithm is block-extensible if all the blocks



of  $T$  are nonsingular (not necessarily commuting) and both  $T$  and its (block) transpose are diagonally dominant [11]. Thus the algorithm  $P$  could be applied, in principle, to a wide class of linear (tridiagonal, block-tridiagonal or band) systems arising in the discretization of ordinary differential equations, elliptic partial differential problems, or spline interpolation.

**4. The case of centrosymmetric tridiagonal matrices.** By Theorem 1, LU decomposition defines an optimal preconditioning to solve a general tridiagonal system  $T\mathbf{x} = \mathbf{f}$ . But the LU method is insensitive to some special properties which may belong, in many applications, to  $T$  or to  $\mathbf{f}$ . In problems concerning the cubic splines or the discretization of ordinary differential equations,  $T$  is more often symmetric or centrosymmetric (or differs from a symmetric or centrosymmetric matrix by a low-rank correction). Moreover, in these problems, we may have a centrosymmetry property for  $\mathbf{f}$ , i.e.,  $f_i = f_{n-i+1}$ . Also, if  $T$  is centrosymmetric, a centrosymmetry of  $\mathbf{f}$  could be assumed for the approximation, through the power method, of the (possible) dominant eigenvalue of  $T^{-1}$  (that gives the minimal eigenvalue of  $T$ ): in fact, for any eigenvector  $\mathbf{y} = [y_1, y_2, \dots, y_n]^T$  of a centrosymmetric matrix the entries  $y_i$  of  $\mathbf{y}$  satisfy the equality  $y_i = y_{n-i+1}$  [1].

**THEOREM 3.** Any algorithm solving  $T\mathbf{x} = \mathbf{f}$ , where  $T$  is a tridiagonal, centrosymmetric matrix of dimension  $n \times n$  and  $f_i = f_{n-i+1}$ , requires  $3\lfloor n/2 \rfloor - 2$  active operations in  $F[\mathbf{f}]$ ,  $F = G(a, b)$ . This bound is optimal.

*Proof.* To prove that  $3\lfloor n/2 \rfloor - 2$  active operations are necessary, follow the same argument in the proof of Theorem 1. Observe that in (2.4) there are only  $\lfloor n/2 \rfloor + n$  free indeterminates because  $u_i v_{n-i+1} \equiv u_{n-i+1} v_i$ . Assume these indeterminates are  $u_1, u_2, \dots, u_{\lfloor n/2 \rfloor}, v_1, \dots, v_n$ . Now put  $u_i = u_{n-i+1} v_i v_{n-i+1}^{-1}$  in (2.4) and let  $A$  be an algorithm solving the system  $T\mathbf{x} = \mathbf{f}$  with  $t$  active operations in  $F[\mathbf{f}]$ . Then we have implicitly defined a bilinear program that computes a vector  $\tilde{M}\mathbf{f}_1 + \mathbf{g}_1$ , where  $\mathbf{f}_1^T = [f_1, f_2, \dots, f_{\lfloor n/2 \rfloor}]$ ,  $\mathbf{g}_1$  is an  $\lfloor n/2 \rfloor$ -vector whose entries are linear functions of the  $f_i$ 's and the elements of the  $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$  matrix  $\tilde{M}$  are linear functions of new indeterminates  $\tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_{\lfloor n/2 \rfloor}, \tilde{v}_1, \dots, \tilde{v}_n$ .

In particular, for  $u_1 = 1$  (without loss of generality) and for proper choice of nonzero  $G$ -coefficients  $k$  and  $h$ , the last column of  $\tilde{M}$  is an  $\lfloor n/2 \rfloor$ -vector whose  $i$ th entry is  $k_i \tilde{v}_{\lfloor n/2 \rfloor + 1} + h_{\lfloor n/2 \rfloor} \tilde{u}_i$  for  $1 < i \leq \lfloor n/2 \rfloor$  and  $\tilde{v}_{\lfloor n/2 \rfloor}$  for  $i = 1$ . For  $n = 7$ ,  $\tilde{M}$  has the explicit form

$$\begin{bmatrix} \tilde{v}_7 + \tilde{v}_1 & \tilde{v}_6 + \tilde{v}_2 & \tilde{v}_5 + \tilde{v}_3 & \tilde{v}_4 \\ \theta_1 \tilde{v}_1 + \theta_2 \tilde{v}_2 + & k_2 \tilde{v}_6 + h_6 \tilde{u}_2 + & k_2 \tilde{v}_5 + h_5 \tilde{u}_2 + & k_2 \tilde{v}_4 + h_4 \tilde{u}_2 \\ \theta_3 \tilde{v}_6 + \theta_4 \tilde{u}_2 & k_2 \tilde{v}_2 + h_2 \tilde{u}_2 & k_2 \tilde{v}_3 + h_3 \tilde{u}_2 & \\ \varepsilon_1 \tilde{v}_1 + \varepsilon_2 \tilde{v}_3 + & \xi_1 \tilde{v}_2 + \xi_2 \tilde{v}_3 + & k_3 \tilde{v}_5 + h_5 \tilde{u}_3 + & k_3 \tilde{v}_4 + h_4 \tilde{u}_3 \\ \varepsilon_2 \tilde{v}_5 + \varepsilon_4 \tilde{u}_3 & \xi_4 \tilde{v}_5 + \xi_5 \tilde{u}_3 & k_3 \tilde{v}_3 + h_3 \tilde{u}_3 & \\ 2(h_1 \tilde{u}_4 + k_4 \tilde{v}_1) & 2(k_4 \tilde{v}_2 + h_2 \tilde{u}_4) & 2(k_4 \tilde{v}_3 + h_3 \tilde{u}_4) & k_4 \tilde{v}_4 + h_4 \tilde{u}_4 \end{bmatrix}$$

where the coefficients  $k_i, h_i$  are assumed to be nonzero and the coefficients  $\theta, \varepsilon, \xi$  depend on  $k_i, h_i$ . In general, for  $n$  odd (the case  $n$  even is left to the reader),  $u_1 = 1$ ,  $\tilde{M}$  is  $\lfloor n/2 \rfloor$ -immune in  $\tilde{v}_1, \tilde{v}_2, \dots, \tilde{v}_{\lfloor n/2 \rfloor - 1}, \tilde{v}_{\lfloor n/2 \rfloor + 1}, \dots, \tilde{v}_n$ . Then  $\lfloor n/2 \rfloor + n - 1$  (active) multiplications are necessary to compute  $\tilde{M}\mathbf{f}_1 + \mathbf{g}_1$  and  $t \geq 3\lfloor n/2 \rfloor - 2$ .

Now observe that neither LU decomposition nor trivial halving of the dimension of the system and using Sherman-Morrison formula for inverting "modified" matrices [18] define an algorithm with  $3\lfloor n/2 \rfloor - 2$  active operations. To reach this bound,

consider all nonsingular centrosymmetric  $n \times n$  matrices of the form

$$(4.1) \quad \begin{bmatrix} R & 0 \\ 0 & R^T \end{bmatrix} \text{diag}(x_1, \dots, x_{n/2}, x_{n/2}, \dots, x_1) \begin{bmatrix} R^T & 0 \\ 0 & R \end{bmatrix} \quad (n \text{ even}), \quad \text{or} \\ \begin{bmatrix} R & \beta & 0 \\ \alpha^T & 1 & \delta^T \\ 0 & \gamma & R^T \end{bmatrix} \text{diag}(x_1, \dots, x_{\lfloor n/2 \rfloor}, \dots, x_1) \begin{bmatrix} R^T & \alpha & 0 \\ \beta^T & 1 & \gamma^T \\ 0 & \delta & R \end{bmatrix} \quad (n \text{ odd})$$

where  $\alpha^T = [0, \dots, 0, -1]$ ,  $\delta^T = [-1, 0, \dots, 0]$ ,  $\beta = \mathbf{0}$ ,  $\gamma = \mathbf{0}$ , and  $R \equiv (r_{ij})$  is defined as in (3.7) for  $i, j = 1, \dots, \lfloor n/2 \rfloor$ . Now compute diagonal, centrosymmetric matrices  $D_1 = \text{diag}(\alpha_1, \dots, \alpha_{\lfloor n/2 \rfloor}, \dots, \alpha_1)$ ,  $D_2 = \text{diag}(\beta_1, \beta_{\lfloor n/2 \rfloor}, \dots, \beta_1)$ , and  $D = \text{diag}(d_1, \dots, d_{\lfloor n/2 \rfloor}, \dots, d_1)$  such that  $D_1 T D_2$  has the form (4.1) ( $\alpha_i$  and  $\beta_i$  are computable through second-order linear recurrences, as in (3.8)). As  $\alpha_i, \beta_i$  are arbitrary, this preconditioning produces an algorithm of  $3 \lfloor n/2 \rfloor - 2$  active operations to solve  $T\mathbf{x} = \mathbf{f}$ . Obviously,  $\alpha_i$  and  $\beta_i$  must be different from zero, which is verified if the leading principal submatrices of  $T$  are nonsingular (or if  $T$  is diagonally dominant).

The above preconditioning suggests a parallel algorithm that works in  $O(\log n)$  steps with  $O(n)$  processors and has (apart from the cost of preconditioning) minimal multiplicative redundancy. This algorithm extends to block-tridiagonal matrices, where the nondiagonal blocks of  $T$  and the leading block submatrices of both  $T$  and its (block) transpose are nonsingular.

**Acknowledgment.** The author thanks the referee for his useful suggestions.

#### REFERENCES

- [1] A. L. ANDREW, *Eigenvectors of certain matrices*, Linear Algebra Appl., 7 (1973), pp. 151-162.
- [2] W. W. BARRETT, *A theorem on inverses of tridiagonal matrices*, Linear Algebra Appl., 27 (1979), pp. 211-217.
- [3] R. BEVILACQUA AND M. CAPOVANI, *Proprietà delle matrici a banda ad elementi ad a blocchi*, Boll. Un. Mat. Ital. B(6), 13 (1976), pp. 844-861.
- [4] D. BINI, *On commutativity and approximation*, Theoret. Comput. Sci., 28 (1984), pp. 135-150.
- [5] D. BINI, M. CAPOVANI, G. LOTTI, AND F. ROMANI, *Complessità Numerica*, Boringhieri, Torino, 1981.
- [6] A. BORODIN AND I. MUNRO, *The Computational Complexity of Algebraic and Numeric Problems*, Elsevier, New York, 1975.
- [7] B. BUKHBERGER AND G. A. EMELIANENKO, *Methods of inverting tridiagonal matrices*, U.S.S.R. Comput. Math. and Math. Phys., 13 (1973), pp. 10-20.
- [8] R. W. BROCKETT AND D. DOBKIN, *On the optimal evaluation of a set of bilinear forms*, in Proc. 5th Annual ACM Symposium on Theory of Computing, 1973, pp. 88-95.
- [9] M. CAPOVANI, *Su alcune proprietà delle matrici tridiagonali e pentadiagonali*, Calcolo, 8 (1971) pp. 149-159.
- [10] C. M. FIDUCCIA, *On obtaining upper-bounds on the complexity of matrix multiplication*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972.
- [11] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computation*, The Johns Hopkins University Press, Baltimore, MD, 1983.
- [12] D. HELLER, *A survey of parallel algorithms in numerical linear algebra*, SIAM Rev., 20 (1978), pp. 740-777.
- [13] E. ISAACSON AND H. B. KELLER, *Analysis of Numerical Methods*, John Wiley, New York, 1966.
- [14] J. B. KRUSKAL, *Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistic*, Linear Algebra Appl., 18 (1977), pp. 95-138.
- [15] J. C. LAFON, *Optimum computation of  $p$  bilinear forms*, Linear Algebra Appl., 10 (1975), pp. 225-240.
- [16] J. VAN LEEUWEN AND P. VAN EMDE BOAS, *Some elementary proofs of lower bounds in complexity theory*, Linear Algebra Appl., 19 (1978), pp. 63-80.
- [17] A. H. SAMEH, *Numerical Parallel Algorithms—a Survey*, in High Speed Computer and Algorithm Organization, D. J. Kick, D. H. Lawrie, and A. H. Sameh, eds., Academic Press, New York, 1977.

- [18] J. SHERMAN AND W. MORRISON, *Adjustment of an inverse matrix corresponding to a change in one element*, *Annals Math. Statist.*, 21 (1950), pp. 124–127.
- [19] H. S. STONE, *An efficient parallel algorithm for the solution of a tridiagonal linear system of equations*, *J. Assoc. Comput. Mach.*, 20 (1973), pp. 27–38.
- [20] V. STRASSEN, *Evaluation of rational functions*, in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 1–10.
- [21] V. STRASSEN, *Vermeidung von Divisionen*, *J. Reine Angew. Math.*, 264 (1973), pp. 184–202.
- [22] P. N. SWARZTRAUBER, *A parallel algorithm for solving general tridiagonal equations*, *Math. Comp.*, 33 (1979), pp. 185–199.
- [23] S. WINOGRAD, *On the number of multiplications necessary to compute certain functions*, *Commun. Pure Appl. Math.*, 23 (1970), pp. 165–179.
- [24] T. YAMAMOTO AND Y. IKEBE, *Inversion of band matrices*, *Linear Algebra Appl.*, 24 (1979), pp. 105–111.

## AVERAGE-CASE LOWER BOUNDS FOR SEARCHING\*

COLIN MCDIARMID†

**Abstract.** Lower bounds are given for certain average search times in a set with a random linear order about which there is partial information. These bounds extend various recent worst-case and average-case results, in particular those of Alt and Mehlhorn, Borodin et al., and Mairson concerning searching “semi-sorted” tables and trade-offs between presorting time and search time. We make fuller use of the framework of information theory than have previous investigations.

**Key words.** searching, sorting, average case, lower bound, trade-off, information, uncertainty

**AMS(MOS) subject classifications.** 86P10, 60C05

**1. Introduction.** *Searching.* We consider problems related to “searching semi-sorted tables” [2] and “data location for partially ordered structures” [5], [8].

Suppose that a set  $X$  of  $n$  ( $\geq 2$ ) elements has an unknown linear order  $\pi$  about which we may have partial information. We let  $\pi$  be given as a bijection from  $X$  to  $N = \{1, \dots, n\}$ . Also there is an order-preserving injection  $s$  from  $X$  into the real numbers  $R$ ; that is,  $s(x) < s(x')$  if  $\pi(x) < \pi(x')$ . We think of the number  $s(x)$  being stored at the location  $x$  in our partly ordered data structure. For an (unrevealed) query  $y \in R$  we must determine if  $y$  is one of the numbers stored, and if so at which location. We adopt a “decision-tree” approach. The basic steps allowed are comparisons  $y: s(x)$  for  $x \in X$  and comparisons  $s(x): s(x')$  for  $x, x' \in X$ . Observe that if we have determined that  $y$  is one of the numbers stored, then we must actually have found the location  $x$  in  $X$  such that  $y = s(x)$ .

We shall investigate average-case lower bounds on the number of comparisons required. Note that given the linear order  $\pi$  the actual mapping  $s$  is not important; and we may as well assume that  $s = \pi$ . Thus, we may simplify the statement of the problem as follows: for an unrevealed linear order  $\pi$  on  $X$  and query  $y \in R$  we must determine if  $y \in N$  by making comparisons  $y: \pi(x)$  or  $\pi(x): \pi(x')$ . Given some decision-tree algorithm as above the search time  $S = S(\pi, y)$  taken on the (unrevealed) input  $(\pi, y)$  is defined to be the number of comparisons required.

If we know the linear order  $\pi$ , then binary search yields a worst-case time of about  $\log n$ , and this is best possible. (Throughout the paper  $\log$  denotes logarithm to the base 2.) At the other extreme, if any linear order  $\pi$  is possible, then clearly we need worst-case time equal to  $n$ .

We shall consider both “unsuccessful” search, when the query  $y \notin N$ , and the more interesting case of “successful” search, when  $y \in N$ . Unsuccessful search is the easier to analyse. When  $y \notin N$  we shall actually give lower bounds for  $S'(\pi, y)$ , which we define to be the smallest possible number of questions as above to prove that  $y \notin N$ . Of course we always have  $S(\pi, y) \geq S'(\pi, y)$ . Equivalently, for each pair  $(\pi, y)$  we could define  $S'(\pi, y)$  to be the minimum over all allowed decision-trees of the search time  $S(\pi, y)$ . Results concerning  $S'$  may also be phrased in terms of “nondeterministic” search algorithms [2]. For successful search we shall consider also a hybrid quantity

\* Received by the editors May 5, 1986; accepted for publication (in revised form) July 8, 1987.

† Computer Science Department, Rutgers University, New Brunswick, New Jersey 08903 and Institute of Economics and Statistics, Oxford University, Oxford, England.

$S^*$ , where for  $y \in N$  we define  $S^*(y)$  to be the maximum value of  $S(\pi, y)$  over all possible linear orders  $\pi$ .

We shall assume that a random linear order  $\pi$  on  $X$  and an independent random query  $y \in R$  are input according to certain distributions of interest, and give lower bounds concerning the random search times  $S$ ,  $S^*$ , and  $S'$ . When analysing a successful search we assume that the random query  $y$  is uniformly distributed over the set  $N = \{1, 2, \dots, n\}$ . When analysing an unsuccessful search we assume that the random query  $y$  is uniformly distributed over the set  $N' = N - \frac{1}{2} = \{\frac{1}{2}, \frac{3}{2}, \dots, n - \frac{1}{2}\}$ . Our investigation falls into two parts: "restricted search" and "general search."

In restricted search only comparisons involving the query  $y$  are allowed, as in [2], [5]. Let us consider an arbitrary probability distribution  $\mathbf{p} = (p_\pi: \pi \in \Pi)$  on a set  $\Pi$  of linear orders on  $X$ . Recall that the *uncertainty*  $H = H(\mathbf{p})$  of the random linear order  $\pi$  is defined to be  $-\sum_{\pi \in \Pi} p_\pi \log p_\pi$ . Also, if each member of  $\Pi$  is equally likely then  $H = \log |\Pi|$ . We give lower bounds concerning the random search times  $S$ ,  $S^*$ , and  $S'$  in terms of the quantity  $H$ . The results for  $S^*$  and  $S'$  extend the work of Alt and Mehlhorn [2] on searching semi-sorted tables. As noted in [2] the lower bounds are sometimes, but not always, sharp. For tight bounds of a different nature see [5].

In general search we also allow comparisons  $\pi(x): \pi(x')$  for  $x, x' \in X$ . We now must restrict the probability distributions considered for the random linear order  $\pi$ . We shall assume that  $\Pi$  is the set of linear extensions of a partial order, and that  $p_\pi > 0$  for each  $\pi \in \Pi$ . In order to investigate successful general search we also analyse the problem of searching for the minimum element in a random linear extension of a partial order.

The two combinations for which we here give good lower bounds on search times are thus (a) restricted search with arbitrary distributions on linear orders, and (b) general search with certain special distributions. Feldman (see [10]) has very recently given an elegant construction for a large set of linear orders that can be searched quickly when we allow comparisons  $\pi(x): \pi(x')$ . This example shows that there are no corresponding lower bounds for general search with arbitrary distributions of linear orders, and throws our results here into sharp relief.

For both restricted and general search there are interesting "trade-off" lower bounds concerning sorting or preprocessing times and searching times.

*Sorting or preprocessing.* Suppose that the set  $X$  with its unknown linear order  $\pi$  is to be partially sorted or preprocessed so that subsequent searches as above can be performed quickly. In the preprocessing stage only (binary) questions concerning  $\pi$  may be asked. If only comparisons  $\pi(x): \pi(x')$  for  $x, x' \in X$  are allowed we call this *restricted sorting*. We use the phrase general sorting when arbitrary binary questions are allowed. Denote by  $P = P(\pi)$  the time taken, that is the number of questions asked, on input  $\pi$ . With each leaf of the decision tree corresponding to the preprocessing stage we associate a search algorithm as above.

*Trade-offs.* Let us consider restricted sorting and general searching. There are similar results below concerning general sorting and restricted searching.

If we do no preprocessing we require worst-case search time  $n$ . At the other extreme it takes worst-case time about  $n \log n$  to sort completely, and then search requires worst-case time about  $\log n$ . As an intermediate strategy we can make  $n/k$  lists of length about  $k$  in worst-case time about  $n \log k$ , and then search in worst-case time about  $(n/k) \log k$ . There is clearly a trade-off between worst-case preprocessing or sorting times and searching times.

**THEOREM 1.1** [4]. *For any restricted sorting and general searching algorithms as above, the maximum over all linear orders  $\pi$  of  $X$  and all (successful) queries  $y \in N$  of the quantity  $P(\pi) + n \log S(\pi, y)$  is at least  $n \log n + O(n)$ .*

We shall give related trade-offs in the average case which extend this worst-case result and extend also the results stated in Mairson [6], [7].

*Plan of the paper.* In the next section, we present our main results. Then in § 3 we give a brief discussion on uncertainty and random linear orders. Section 4 contains the main proofs, apart from that of the theorem on searching for the minimum which is postponed to the last section, § 5.

## 2. Statement of results.

**2.1. Restricted search.** In this section we allow the search algorithms only to make comparisons involving the unknown query  $y$ , as in [2], [5], that is, to ask questions “ $y: \pi(x)?$ ” for  $x \in X$ . The random linear order  $\pi$  on  $X$  may have an arbitrary probability distribution. Let  $\Pi$  be the set of linear orders that occur with positive probability. Denote the uncertainty associated with  $\pi$  by  $H = H(\pi)$ . The main result on restricted search is the following theorem.

**THEOREM 2.1.** *For any restricted search algorithm*

- (i)  $E[n \log S^*] \geq \log |\Pi|$ ,
- (ii)  $E'[n \log S'] \geq H$ ,
- (iii)  $E[n \log S] \geq H - n \log \log n$ .

Here we use  $E\{E'\}$  to denote expectation over the random linear order  $\pi$  and the independent random query  $y$  uniformly distributed over  $N\{N'\}$ .

**COROLLARY 2.2.** *If each linear order  $\pi$  in  $\Pi$  is equally likely, then*

- (i)  $E[S^*] \geq |\Pi|^{1/n}$ ,
- (ii)  $E'[S'] \geq |\Pi|^{1/n}$ ,
- (iii)  $E[S] \geq |\Pi|^{1/n} / \log n$ .

Parts (i) and (ii) of Corollary 2.2 contain the theorems of Alt and Mehlhorn [2]. Now let us consider “trade-off” results. Recall that  $P = P(\pi)$  denotes time spent in the sorting or preprocessing stage.

**COROLLARY 2.3.** *For any general sorting and restricted searching algorithms*

- (i)  $E[P + n \log S^*] \geq H$ ,
- (ii)  $E'[P + n \log S'] \geq H$ ,
- (iii)  $E[P + n \log S] \geq H - n \log \log n$ .

**2.2. General search.** Suppose now that as well as comparisons involving the query  $y$  we may make comparisons  $\pi(x): \pi(x')$ , where  $x, x' \in X$ . We shall consider only “well-behaved” distributions for the random linear order  $\pi$  on  $X$ . Let  $\Pi$  denote the set of linear orders that occur with positive probability. We shall always assume when considering general search that  $\Pi$  is the set of linear extensions of some partial order, and sometimes we shall further assume that each linear order  $\pi$  in  $\Pi$  is equally likely. We consider first the quantities  $S^*$  and  $S'$  and postpone briefly the more difficult full average-case analysis for successful search.

**THEOREM 2.4.** *If  $\Pi$  is the set of linear extensions of a partial order, then for any general searching algorithm*

- (i)  $E[n \log S^*] \geq \log |\Pi|$ ,
- (ii)  $E'[n \log S'] \geq H$ .

**COROLLARY 2.5.** *If  $\Pi$  is the set of linear extensions of a partial order, then for any restricted sorting algorithm and general searching algorithms*

- (i)  $E[P + n \log S^*] \geq H$ ,
- (ii)  $E'[P + n \log S'] \geq H$ .

**COROLLARY 2.6.** *If all  $n!$  linear orders on  $X$  are equally likely, then both  $E[P + n \log S^*]$  and  $E'[P + n \log S']$  are at least  $n \log n + O(n)$ .*

Both of the inequalities in Corollary 2.6 extend the worst-case result Theorem 1.1 of Borodin et al. [4]. This is of course immediate for the mixed worst-case average-case

result concerning  $S^*$ , and it will be shown easily for the result concerning unsuccessful search.

All the results above follow naturally from the general discussion in § 3 below on uncertainty and random linear orders. However, we cannot prove such strong average-case results for successful search, when we also average over the random linear order  $\pi$ , and we must work harder. In order to prove Theorem 2.7 below we shall investigate carefully the problem of searching for the minimum element in a random linear extension of a partial order (see § 2.3).

**THEOREM 2.7.** *Let  $\Pi$  be the set of linear extensions of a partial order, and suppose that each linear order  $\pi$  in  $\Pi$  is equally likely. Then for any general search algorithm*

$$E[S] \geq (1/2n)|\Pi|^{2/n}.$$

This theorem gives weaker bounds than Theorem 2.4, and we can deduce correspondingly weaker trade-off results. We give two forms of trade-off result.

**COROLLARY 2.8.** *Let  $\Pi$  be the set of linear extensions of a partial order, and suppose that each linear order  $\pi$  in  $\Pi$  is equally likely. Then for any restricted sorting algorithm and general searching algorithms,*

- (i)  $2E[P] + n \log E[S] \geq 2 \log |\Pi| - n \log 2n,$
- (ii)  $E[4^{P/n}S] \geq |\Pi|^{2/n}/2n.$

The most interesting case of Corollary 2.8 is when  $\Pi$  is the set of all linear orders on  $X$ .

**COROLLARY 2.9.** *Suppose that all  $n!$  linear orders on  $X$  are equally likely. Then for any restricted sorting algorithm and general searching algorithms,*

- (i)  $2E[P] + n \log E[S] \geq n \log n - n \log 2e^2,$
- (ii)  $E[4^{P/n}S] \geq n/2e^2.$

Part (i) of this result is perhaps the more attractive. It extends a result proposed and partially proved by Mairson [6], [7] (who has  $3E[P]$  instead of  $2E[P]$ ). Indeed the present investigation was inspired by hearing Mairson present this result. However, this part has the unsatisfactory feature that it does not seem to yield any worst-case result, whereas part (ii) at least yields half the lower bound in Thm. 1.1. Corollary 2.9 represents only partial progress towards investigating the following conjecture.

**Conjecture 2.10.** With premises as in Corollary 2.9

$$E[P + n \log S] \geq (1 + o(1))n \log n.$$

**2.3. Searching for the minimum.** Let  $Q$  be a partial order on the set  $X$  of  $n \geq 2$  elements and let  $\Pi$  be the set  $L(Q)$  of linear extensions of  $Q$ . Suppose that these linear orders  $\pi$  occur randomly with each equally likely, and we wish to find the minimum element  $\pi^{-1}(1)$  quickly. The basic steps allowed are questions “Is  $x$  the minimum?” where  $x \in X$  and comparisons  $\pi(x) : \pi(x')$  where  $x, x' \in X$ . We may stop only when we receive a positive answer to a question of the former type. Given a decision-tree search algorithm as described, let  $T(\pi)$  be the time taken (number of questions) on input  $\pi \in \Pi$ . The following result is a main step in proving Theorem 2.7 above.

**THEOREM 2.11.** *If the partial order  $Q$  has  $m$  minimal elements, then*

$$E[T] \geq 1 + m(m - 1)/2n.$$

It is straightforward to check that this result is the best possible, in the sense that for any  $1 \leq m \leq n$  there is a suitable partial order  $Q$  and search algorithm such that the expected search time  $E[T]$  equals  $1 + m(m - 1)/2n$ . Indeed we may let  $Q$  consist of a set  $Z$  of  $n - m$  elements with any partial order, an element  $x_1$  less than each  $z \in Z$ , and  $m - 1$  “independent” elements  $x_2, \dots, x_m$ ; and then search for the minimum by testing in the fixed order  $x_1, x_2, \dots, x_m$ .

To see that  $E[T]$  is as stated, let  $\Pi_i$  denote the set of linear extensions  $\pi$  in  $\Pi$  with  $x_i$  the minimum, for each  $i = 1, \dots, m$ . Then by considering last where to put  $x_i$  we see that  $|\Pi_i| = |\Pi|/n$  for each  $i = 2, \dots, m$ . Hence

$$\begin{aligned} E[T] &= (1/|\Pi|) \sum_{j=1}^m j|\Pi_j| \\ &= 1 + m(m-1)/2n. \end{aligned}$$

Theorem 2.11 above is related to the ‘‘accounting game’’ claim proposed by Mairson [6], [7].

**3. Uncertainty and random linear orders.** We start with a brief general discussion on uncertainty (see for example Ash [3]).

We consider only random variables taking finitely many values. Let the random variable  $Y$  take values in the set  $\{x_1, \dots, x_n\}$ , with  $P\{Y = x_i\} = p_i$  for  $i = 1, \dots, n$  and  $\mathbf{p} = (p_1, \dots, p_n)$ . The *uncertainty* (or *entropy*)  $H(Y)$  or  $H(\mathbf{p})$  is defined to be  $-\sum_{i=1}^n p_i \log p_i$ . (We take  $t \log t = 0$  if  $t = 0$ .) Thus  $H(\mathbf{p}) \geq 0$ , with equality if and only if we have the trivial case that some  $p_i = 1$ .

We note first a very useful inequality. Let  $q_1, \dots, q_n > 0$  with  $\sum_{i=1}^n q_i = 1$ . Then

$$(3.1) \quad -\sum_{i=1}^n p_i \log p_i \leq -\sum_{i=1}^n p_i \log q_i$$

with equality if and only if  $p_i = q_i$  for each  $i$ .

To prove (3.1) note that for  $t > 0$ ,  $\ln t \leq t - 1$  with equality if and only if  $t = 1$ . Here  $\ln$  denotes natural logarithm. Thus

$$\sum_{i=1}^n p_i \ln (q_i/p_i) \leq \sum_{i=1}^n (q_i - p_i) = 0,$$

and (3.1) follows easily. (Lemma 4.1 of [5] is essentially the above inequality again.)

If we set  $q_i = 1/n$  in (3.1) we find that

$$(3.2) \quad H(\mathbf{p}) \leq \log n, \text{ with equality if and only if the } p_i \text{ are all equal.}$$

For a pair  $(Y, Z)$  of random variables let us write  $H(Y, Z)$  for  $H((Y, Z))$ . Thus if  $P\{(Y, Z) = (y, z)\} = p(y, z)$ , then the uncertainty is

$$H(Y, Z) = -\sum_y \sum_z p(y, z) \log p(y, z).$$

Denote the conditional probabilities for  $Z = z$  given  $Y = y$  by  $p(z|y)$ . Then the conditional uncertainty of  $Z$  given that  $Y = y$  is

$$H(Z | Y = y) = -\sum_z p(z|y) \log p(z|y).$$

The *conditional uncertainty of  $Z$  given  $Y$* ,  $H(Z | Y)$ , is the appropriate weighted average of the above quantities, which yields

$$H(Z | Y) = -\sum_y \sum_z p(y, z) \log p(z|y).$$

From this definition we easily find the basic property

$$(3.3) \quad H(Y, Z) = H(Y) + H(Z | Y).$$

(3.4) It follows from (3.3) that the average number of binary questions required to determine the value of a random variable  $Y$  is at least the uncertainty  $H(Y)$ .



One way to prove this is by induction on the depth of the corresponding decision tree. The result is clear if we need no questions, for then  $H(Y) = 0$ . Consider a nontrivial decision tree such that the result holds for any shallower tree. Let the binary random variable  $Z$  specify the answer to the first question. Then  $H(Z) \leq 1$  by (3.2) and so by the induction hypothesis the expected number of steps to determine the value of  $Y$  is at least

$$1 + H(Y|Z) = 1 + H(Y, Z) - H(Z) \geq H(Y).$$

Given a random variable  $Y$  we define a distribution for a random variable  $\tilde{T} = \tilde{T}(Y)$  as follows. List the quantities  $P\{Y = x\}$  in nonincreasing order, as  $p'_1 \geq p'_2 \geq \dots \geq p'_n$ , say. Then let  $P\{\tilde{T} = i\} = p'_i$  for each  $i = 1, \dots, n$ .

Suppose that we wish to determine the value of  $Y$  by asking questions of the form "Is  $Y = x$ ?", until we receive the answer "yes." Given an appropriate list of questions let  $T$  be the random number of questions required. Clearly for any  $t = 1, 2, \dots, n$

$$P\{T \leq t\} \leq \sum_{i=1}^t p'_i.$$

Thus

$$(3.5) \quad T \geq \tilde{T} \text{ in distribution.}$$

Now let  $g(n) = 1 + 1/2 + \dots + 1/n$ . Then

$$\begin{aligned} H(Y) &= - \sum_{i=1}^n p'_i \log p'_i \\ &\leq - \sum_{i=1}^n p'_i \log (ig(n))^{-1} \quad \text{by the inequality (3.1)} \\ &= E[\log \tilde{T}] + \log g(n) \\ &\leq E[\log T] + \log g(n) \quad \text{by (3.5).} \end{aligned}$$

Hence we have

$$(3.6) \quad E[\log T] \geq H(Y) - \log g(n).$$

Now let us consider the uncertainty of a random vector  $\mathbf{Y} = (Y_1, \dots, Y_n)$ . By (3.3),

$$H(\mathbf{Y}) = H(Y_1) + H(Y_2, \dots, Y_n | Y_1),$$

and thus we obtain

$$(3.7) \quad H(\mathbf{Y}) = \sum_{k=1}^n H(Y_k | Y_l \text{ for each } l < k).$$

Denote by  $A$  the set of vectors  $\mathbf{y} = (y_1, \dots, y_n)$  such that  $P\{\mathbf{Y} = \mathbf{y}\} > 0$ . For each  $\mathbf{y} \in A$  and  $k \in N$  let

$$h(\mathbf{y}, k) = H(Y_k | Y_l = y_l \text{ for each } l < k).$$

Then averaging over  $\mathbf{Y}$  we have that for each  $k \in N$

$$E[h(\mathbf{Y}, k)] = H(Y_k | Y_l \text{ for each } l < k)$$

and so by (3.7)

$$(3.8) \quad \sum_{k=1}^n E[h(\mathbf{Y}, k)] = H(\mathbf{Y}).$$

Now for each  $y \in A$  and  $k \in N$  let us also define  $m(y, k)$  to be the number of different possible values for  $Y_k$  given that  $Y_l = y_l$  for each  $l < k$ . Then by (3.2)

$$h(y, k) \leq \log m(y, k).$$

Hence by (3.8)

$$(3.9) \quad \sum_{k=1}^n E[\log m(Y, k)] \geq H(Y).$$

Suppose that we have a random linear order  $\pi$  on a set  $X$  of  $n$  elements. Recall that we think of  $\pi$  as a bijection from  $X$  to  $N = \{1, \dots, n\}$ . Thus the inverse  $\pi^{-1}$  may be thought of as a random vector  $(\pi^{-1}(1), \dots, \pi^{-1}(n))$ . We shall use (3.9) with  $Y$  replaced by  $\pi^{-1}$ . Note that  $H(\pi^{-1}) = H(\pi)$ . Note also that if  $\pi_0$  is a possible linear order and  $k \in N$ , then  $m(\pi_0^{-1}, k)$  is the number of different possible  $k$ th elements in linear orders  $\pi$  that agree with  $\pi_0$  in the first  $k - 1$  elements. Let us state this particular case of (3.9) as

$$(3.10) \quad \sum_{k=1}^n E[\log m(\pi^{-1}, k)] \geq H(\pi).$$

We now note one easy corollary of the result (3.10). Suppose that each linear order  $\pi$  in a set  $\Pi$  is equally likely, so that by (3.2)  $H(\pi) = \log |\Pi|$ . Now recall that since  $\log(x)$  is concave, for any random variable  $Z \geq 1$  we have Jensen's inequality that

$$(3.11) \quad \log E[Z] \geq E[\log Z].$$

Hence

$$\begin{aligned} \log \left\{ (1/n) \sum_{k=1}^n E[m(\pi^{-1}, k)] \right\} &\geq (1/n) \sum_{k=1}^n \log E[m(\pi^{-1}, k)] \\ &\geq (1/n) \sum_{k=1}^n E[\log m(\pi^{-1}, k)] \\ &\geq (1/n) \log |\Pi| \quad \text{by (3.10).} \end{aligned}$$

Thus

$$(3.12) \quad (1/n) \sum_{k=1}^n E[m(\pi^{-1}, k)] \geq |\Pi|^{1/n}.$$

These last results (3.10) and its corollary (3.12) may be visualised in terms of the "permutation trees" of Alt and Mehlhorn [2]. We have used elementary information theory to replace the detailed inductive proof in [2] essentially of (3.12) (see also Theorem 3.7.1 of Mairson [6], [7]).

**4. Main proofs.** In this section, we use the general results on uncertainty given in the last section to deduce the results of restricted and unrestricted search given in § 2. For the proof of Theorem 2.7 on unrestricted search we shall also need Theorem 2.11 on searching for the minimum. This last result is proved in the next (and last) section.

*Proof of Theorem (2.1).* Let us start with some notation. Given  $\pi_0 \in \Pi$  and  $k_0 \in N$  let

$$\Pi_0 = \Pi_0(\pi_0, k_0) = \{\pi \in \Pi: \pi^{-1}(l) = \pi_0^{-1}(l) \text{ for each } l < k_0\},$$

$$A_0 = A_0(\pi_0, k_0) = \{x \in X: \pi_0(x) \geq k_0\},$$

$$M_0 = M_0(\pi_0, k_0) = \{x \in X: \pi(x) = k_0 \text{ for some } \pi \in \Pi_0\}.$$

Thus  $M_0 \subseteq A_0$  and  $|M_0| = m(\pi_0^{-1}, k_0)$  in the notation of (3.10).

Let us prove part (i) first. Let  $k_0 \in N$  and suppose that  $y = k_0$ . Reveal  $k_0$  and that  $y = k_0$  or  $k_0 - \frac{1}{2}$ . Consider any  $\pi_0 \in \Pi$ , and restrict  $\pi$  to  $\Pi_0$ . Of course

$$S^*(k_0) \geq \max \{S(\pi, k_0) : \pi \in \Pi_0\}.$$

Now suppose that we reveal  $\Pi_0$  and that  $\pi \in \Pi_0$ . Then we still have

$$(4.1) \quad \max \{S(\pi, k_0) : \pi \in \Pi_0\} \geq |M_0|.$$

For clearly we can restrict attention to questions of the form “Is  $\pi(x) = y$ ?” for  $x \in M_0$ ; and given any sequence of less than  $|M_0|$  such questions an adversary could answer “no” each time, and we would not know whether  $y = k_0$  or  $k_0 - \frac{1}{2}$ . Thus we have shown that for any  $\pi_0 \in \Pi$

$$(4.2) \quad S^*(k_0) \geq m(\pi_0^{-1}, k_0).$$

Hence

$$\begin{aligned} E[n \log S^*] &= \sum_{k_0=1}^n \log S^*(k_0) \\ &\geq \sum_{k_0=1}^n E[\log m(\pi^{-1}, k_0)] \\ &\geq H \quad \text{by (3.10).} \end{aligned}$$

But as far as  $S^*$  is concerned the precise distribution of the random linear order  $\pi$  is not relevant, only the set  $\Pi$ . Thus by (3.2) we may replace the last  $H$  by  $\log |\Pi|$ .

Now consider part (ii) of Theorem 2.1. Let  $k_0 \in N$  and  $\pi_0 \in \Pi$ , and suppose that  $y = k_0 - \frac{1}{2}$  and  $\pi = \pi_0$ . Reveal  $k_0$  and that  $y = k_0$  or  $k_0 - \frac{1}{2}$ ; and reveal also  $\Pi_0$  and that  $\pi \in \Pi_0$ . Then still

$$(4.3) \quad S'(\pi_0, k_0 - \frac{1}{2}) \geq m(\pi_0^{-1}, k_0).$$

For as above we may restrict attention to questions of the form “Is  $\pi(x) = y$ ?” for  $x \in M_0$ . The answer to each such question of course will be “no,” but we cannot know that  $y \notin N$  until we have asked such a question for each  $x \in M_0$ .

Hence by (4.3)

$$\begin{aligned} E'[n \log S'] &= \sum_{k_0=1}^n E[\log S'(\pi, k_0 - \frac{1}{2})] \\ &\geq \sum_{k_0=1}^n E[\log m(\pi^{-1}, k_0)] \\ &\geq H \quad \text{by (3.10).} \end{aligned}$$

Finally let us prove part (iii) of Theorem 2.1. Let  $k_0 \in N$  and  $\pi_0 \in \Pi$ , and suppose that  $y = k_0$  and  $\pi = \pi_0$ . Reveal  $k_0$  and that  $y = k_0$  or  $k_0 - \frac{1}{2}$ ; and reveal  $\Pi_0$  and that  $\pi \in \Pi_0$ . Again we may restrict attention to questions of the form “Is  $\pi(x) = y$ ?” for  $x \in M_0$ , but now we must continue until we receive the answer “yes.” Since here  $\pi(x) = y$  if and only if  $\pi^{-1}(k_0) = x$  we are identifying the value of the random variable  $Y = \pi^{-1}(k_0)$  as considered in the discussion preceding the inequalities (3.5) and (3.6). Here  $Y$  ranges over the set  $M_0$ , and  $|M_0| \leq n - k_0 + 1$ . Hence by (3.6)

$$\begin{aligned} E[\log S(\pi, k_0) | \pi \in \Pi_0] &\geq H(\pi^{-1}(k_0) | \pi \in \Pi_0) - \log g(n - k + 1) \\ &= h(\pi_0^{-1}, k_0) - \log g(n - k + 1) \end{aligned}$$

in the notation of (3.8).

Now keep  $k_0$  fixed but consider different possible  $\pi_0$ . Then

$$\begin{aligned} E[\log S(\pi, k_0)] &= \sum_{\pi_0 \in \Pi} E[\log S(\pi, k_0) | \pi \in \Pi_0(\pi_0, k_0)] P(\pi = \pi_0) \\ &\geq \sum_{\pi_0 \in \Pi} h(\pi_0^{-1}, k_0) P(\pi = \pi_0) - \log g(n - k_0 + 1) \\ &= E[h(\pi^{-1}, k_0)] - \log g(n - k_0 + 1). \end{aligned}$$

Hence

$$\begin{aligned} E[n \log S] &= \sum_{k_0=1}^n E[\log S(\pi, k_0)] \\ &\geq \sum_{k_0=1}^n E[h(\pi^{-1}, k_0)] - \sum_{k=1}^n \log g(n - k + 1) \\ &= H - \sum_{k=1}^n \log g(k) \quad \text{by (3.8)}. \end{aligned}$$

We shall see below that

$$(4.4) \quad \sum_{k=1}^n \log g(k) \leq n \log \log n \quad \text{for } n \geq 3.$$

This will then establish part (iii) of the theorem for  $n \geq 3$ . The case  $n = 2$  is easy to check separately, since then

$$E[n \log S] \geq 1 \geq H.$$

It remains then only to prove (4.4). Now for each  $k \geq 1$

$$g(k) \leq 1 + \int_1^k (1/x) dx = 1 + \ln k.$$

Hence since  $\ln(x)$  is concave

$$(1/n) \sum_{k=1}^n g(k) \leq 1 + (1/n) \sum_{k=1}^n \ln k \leq 1 + \ln \frac{n+1}{2}.$$

But the function

$$f(x) = \log x - \left( 1 + \ln \frac{x+1}{2} \right) \quad \text{for } x > 0$$

is increasing, and  $f(4) > 0$ . Hence

$$1 + \ln \frac{n+1}{2} < \log n \quad \text{for } n \geq 4.$$

Thus

$$(1/n) \sum_{k=1}^n g(k) < \log n \quad \text{for } n \geq 4,$$

and it is easy to check separately that this inequality holds also for  $n = 3$ . Hence, since  $\log(x)$  is concave,

$$\begin{aligned} (1/n) \sum_{k=1}^n \log g(k) &\leq \log \left\{ (1/n) \sum_{k=1}^n g(k) \right\} \\ &< \log \log n \quad \text{for } n \geq 3. \end{aligned}$$

This completes the proof of (4.4), and thus of Theorem 2.1.  $\square$

Corollary 2.2 follows easily from Theorem 2.1 using (3.2) and Jensen’s inequality (3.11).

In each of the three parts of Corollary 2.3, the case when there is no preprocessing is just the corresponding part of Theorem 2.1 (where we have replaced the  $\log |\Pi|$  by  $H$  again): and the result may now be proved exactly as was (3.4) by induction on the depth of the preprocessing decision tree.

*Proof of Theorem 2.4.* Let us use notation from the proof of Theorem 2.1. Consider the following condition on a set  $\Pi$  of linear orders on  $X$ .

- (4.5) Given  $\pi_0 \in \Pi$ ,  $k_0 \in N$ ,  $x_0 \in M_0$ , and a partial order  $Q$  on  $A_0$  with  $x_0$  minimal, if some  $\pi \in \Pi_0$  has restriction  $\pi|_{A_0}$  in the set  $L(Q)$  of linear extensions of  $Q$  then some  $\pi \in \Pi_0$  has  $\pi|_{A_0}$  in  $L(Q)$  and has  $\pi(x_0) = k_0$ .

Clearly the set of linear extensions of a partial order satisfies this condition. For given  $\pi \in \Pi_0$  with  $\pi|_{A_0}$  in  $L(Q)$ , if the rank  $\pi(x_0) \neq k_0$  then  $x_0$  has rank  $> k_0$ , and we may simply move  $x_0$  down to rank  $k_0$  and “shuffle up” other elements as required. (This also holds for any set  $\Pi$  consisting of those linear extensions of a partial order which satisfy further conditions of the form  $\pi(x) = k$  or  $\pi(x) \geq k$ .)

We shall see that Theorem 2.4 in fact holds for any set  $\Pi$  of linear orders which satisfies the condition (4.5), by showing that both the inequalities (4.2) and (4.3) from the proof of Theorem 2.1 still hold.

In order to prove that (4.2) still holds we shall show that (with the information revealed as before) the inequality (4.1) still holds. Now we may restrict attention to questions of the form “Is  $\pi(x) = y$ ?” for  $x \in M_0$  and of the form “ $\pi(x) : \pi(x')$ ?” for  $x, x' \in A_0$ .

Consider a sequence of  $q < |M_0|$  such questions. An adversary could say “no” to each question of the former kind, and respond to questions of the latter kind by picking an arbitrary  $\hat{\pi} \in \Pi_0$  and answering according to  $\hat{\pi}$ . Let  $B$  be the set of elements  $x \in M_0$  mentioned in questions of the former kind, and let  $C$  be the set of elements  $x \in M_0$  for which there is a question “ $\pi(x) : \pi(x')$ ?” which received the answer “ $\pi(x) > \pi(x')$ .” Then  $|B| + |C| = q < |M_0|$  and there is some  $x_0 \in M_0 \setminus (B \cup C)$ . Now the answers to the questions of the latter kind yield a partial order  $Q$  on  $A_0$ , and the linear order  $\hat{\pi}$  in  $\Pi_0$  is such that  $\hat{\pi}|_{A_0}$  is in  $L(Q)$ . Hence by the condition (4.5) there is some  $\pi \in \Pi_0$  with  $\pi(x_0) = k_0$  which yields all the answers given. Thus it is still possible that  $y = k_0$ , and thus also possible that  $y = k_0 - \frac{1}{2}$ .

This completes the proof that the inequality (4.1) still holds and thus also the inequality (4.2).

Finally we must show that the inequality (4.3) still holds. But consider any sequence of  $q < |M_0|$  questions as above, and let  $B$  and  $C$  be as defined there. Chance now takes the role of the adversary. Just as above there is some  $x_0 \in M_0 \setminus (B \cup C)$  and some  $\pi \in \Pi_0$  with  $\pi(x_0) = k_0$  which would yield all the answers given. Hence it is still possible that  $y \in N$ , and the inequality (4.3) follows.  $\square$

Corollary 2.5 follows from Theorem 2.4 just as Corollary 2.3 followed from Theorem 2.1. Also, Corollary 2.6 follows immediately from Corollary 2.5 and (3.2).

To see that the result in Corollary 2.6 concerning unsuccessful search yields the worst-case result Theorem 1.1 note that we may assume that for each leaf  $v$  of the sorting decision tree the corresponding search algorithm  $S_v$  is minimal in the following sense. Let  $\Pi_v$  denote the set of linear orders  $\pi$  that reach leaf  $v$ . If a node in the decision tree for  $S_v$  cannot be reached by any pair  $(\pi, y)$ , where  $\pi \in \Pi_v$  and  $y \in N$  then that node must be a leaf, labelled “ $y \notin N$ .” But now consider any pair  $(\pi_1, y_1)$ , where  $\pi_1 \in \Pi_v$  and  $y_1 \in N'$ . Some pair  $(\pi_2, y_2)$ , where  $\pi_2 \in \Pi_v$  and  $y_2 \in N$  must reach the father

of the leaf reached by  $(\pi_1, y_1)$ , and must then reach a different son of that father. Thus  $S_v(\pi_1, y_1) \cong S_v(\pi_2, y_2)$ . Now it is clear that the maximum search time over  $\pi \in \Pi_v, y \in N'$  is at most the maximum search time over  $\pi \in \Pi_v, y \in N$  (and it is not hard to see that in fact equality holds).

In order to prove Theorem 2.7 we shall use Theorem 2.11, which is proved in the next section.

*Proof of Theorem 2.7.* Let us again use the notation from the proof of Theorem 2.1. Let  $k_0 \in N$  and  $\pi_0 \in \Pi$ , and suppose that  $y = k_0$  and  $\pi = \pi_0$ . Reveal  $k_0$  and that  $y = k_0$  or  $k_0 - \frac{1}{2}$ ; and reveal also  $\Pi_0$  and that  $\pi \in \Pi_0$ . By Theorem 2.11

$$E[S(\pi, k_0) | \pi \in \Pi_0] \cong m(\pi_0^{-1}, k_0)^2 / 2n.$$

But by (3.12)

$$E[m^2] \cong E[m]^2 \cong |\Pi|^{2/n}.$$

Hence

$$E[S] \cong E[m^2] / 2n \cong |\Pi|^{2/n} / 2n. \quad \square$$

*Proof of Corollary 2.8.* Both of these inequalities may be proved from Theorem 2.7 along the lines of the proof of (3.4), by induction on the depth of the preprocessing decision tree. We spell out proofs below (though no new ideas are involved).

If there is no preprocessing, each inequality reduces to Theorem 2.7. For each inequality we then consider a nontrivial decision tree such that the result holds for any shallower tree. Suppose that the first question asked is the comparison  $\pi(x) : \pi(x')$ , where  $x, x' \in X, x \neq x'$ . Let  $\Pi_1, \Pi_2$  be the sets of linear orders  $\pi$  in  $\Pi$  with  $\pi(x) < \pi(x')$ ,  $\pi(x) > \pi(x')$ , respectively. For  $i = 1, 2$  let  $t_i = |\Pi_i| / |\Pi|$ , and let  $P_i, S_i$  refer to  $\Pi_i$ .

Now consider the inequality (i). We have

$$\begin{aligned} 2E[P] + n \log E[S] &= 2(1 + t_1 E(P_1) + t_2 E(P_2)) + n \log (t_1 E[S_1] + t_2 E[S_2]) \\ &\cong 2 + \sum_{i=1}^2 t_i (2E(P_i) + n \log E[S_i]) \\ &\qquad\qquad\qquad \text{by the concavity of the log function} \\ &\cong 2 + \sum_{i=1}^2 t_i 2 \log (t_i |\Pi|) - n \log 2n \\ &\qquad\qquad\qquad \text{by the induction hypothesis} \\ &= 2 \log |\Pi| - n \log 2n + 2(1 + t_1 \log t_1 + t_2 \log t_2) \\ &\cong 2 \log |\Pi| - n \log 2n \quad \text{by (3.2).} \end{aligned}$$

This completes the proof of inequality (i).

Now consider the inequality (ii). We have

$$\begin{aligned} E[4^{P/n} S] &= 4^{1/n} \left( \sum_{i=1}^2 t_i E[4^{P_i/n} S_i] \right) \\ &\cong 4^{1/n} \left( \sum_{i=1}^2 t_i (t_i |\Pi|)^{2/n} / 2n \right) \\ &\qquad\qquad\qquad \text{by the induction hypothesis} \\ &= (|\Pi|^{2/n} / 2n) \{ 4^{1/n} (t_1^{1+2/n} + t_2^{1+2/n}) \} \\ &\cong |\Pi|^{2/n} / 2n. \end{aligned}$$

To see the last inequality note that if  $\delta \geq 0$  the function  $f(x) = x^{1+\delta}$  is convex for  $x > 0$ .  $\square$

Corollary 2.9 of course follows immediately from Corollaries 2.8 and 3.2.

Since the proofs of our results on successful general search rest on Theorem 2.11 and thus on Lemma 5.6 below it will be seen that these results actually hold for more general models of computation (see for example Yao [9]).

**5. Searching for the minimum.** In this section, we prove Theorem 2.11 on searching for the minimum element in a random linear extension of a partial order. We make a careful investigation which may also be of help in resolving Conjecture 2.10. We shall need to consider more general sets of linear orders, and we shall also “assist” the search algorithms.

Given a set  $\Pi$  of linear orders on  $X$ , a partial order  $Q$  on  $X$  and a subset  $Y$  of the minimal elements in  $Q$  let us call  $\Pi$  *good* (with respect to  $Q$  and  $Y$ ) if for any fixed  $\tilde{\pi}$  in  $\Pi$

$$\Pi = \{ \pi \in L(Q) : \pi^{-1}(1) \in Y, \pi(x) = \tilde{\pi}(x) \text{ for each } x \in X \setminus Y \}.$$

Recall that  $L(Q)$  denotes the set of linear extensions of  $Q$ . Here we shall always deal with good sets of linear orders.

We shall assist the search algorithm by giving away certain information for free (and thus assist our analysis). Consider a comparison “ $\pi(x) : \pi(x')$ ?”: if say the result is “ $\pi(x) > \pi(x')$ ” (so that certainly  $x$  is not the minimum) then the rank  $\pi(x)$  of  $x$  is revealed for free. Similarly, the question “Is  $x$  the minimum?” is answered by revealing  $\pi(x)$ . (See also the comment following the proof of the lemma below.) Note that if  $\Pi$  is good with respect to  $Q$  and  $Y$  then the only questions an assisted algorithm need ever use are of the form “Is  $x$  the minimum?” for  $x \in Y$ , and “ $\pi(x) : \pi(x')$ ?” for  $x, x' \in Y$ .

LEMMA 5.1. *Suppose that the set  $\Pi$  of linear extensions is good with respect to  $Q$  and  $Y$ , and that each linear order  $\pi \in \Pi$  is equally likely. Then for any assisted search algorithm as above the random time  $T$  taken satisfies  $T \geq \tilde{T}$  in distribution, where  $\tilde{T} = \tilde{T}(\pi^{-1}(1))$  is as defined in § 3.*

This result says that it is optimal to order the elements of  $X$  by decreasing size of  $|\Pi(x)|$  and repeatedly to ask questions “Is  $x$  the minimum?” in this fixed order, ignoring any extra information gained and never making any comparisons (see (3.5)). Here  $\Pi(x)$  is the set of linear orders  $\pi \in \Pi$  with  $\pi(x) = 1$ .

*Proof.* Let  $Z = X \setminus Y$ . For each  $y \in Y$  let  $b(y)$  be the least  $k \in \mathbb{N}$  such that there is a  $z \in Z$  with  $y < z$  in  $Q$  and rank  $\pi(z) = k$  for some (or each)  $\pi \in \Pi$ . If there is no such  $z$  then set  $b(y) = \infty$ . The key observation is that, with the natural conventions regarding  $\infty$

$$(5.2) \quad \text{if } y, y' \in Y \text{ and } b(y) \geq b(y') \text{ then } |\Pi(y)| \leq |\Pi(y')|.$$

For the map  $\pi \rightarrow \pi' = (y, y')\pi$  gives an injection from  $\Pi(y)$  into  $\Pi(y')$ . To see this, note that if  $\pi \in \Pi(y)$  then  $\pi' \in L(Q)$  and so  $\pi' \in \Pi$ .

Let us order the elements of  $Y$  as  $x_1, \dots, x_m$ , say, so that  $b(x_1) \leq \dots \leq b(x_m)$ .

We shall prove the lemma by induction on the depth  $d$  of the decision tree. The case  $d = 1$  (one question) is trivial, so consider an instance with  $d > 1$ , and such that the lemma holds for any shallower decision tree. Clearly we may assume that  $m = |Y| > 1$ .

Let  $t \in \{1, \dots, m\}$ , and let  $R_t = R_t(\Pi)$  denote the set of linear orders  $\pi \in \Pi$  which reach leaves of the decision tree at depth at most  $t$ . Then  $P\{T \leq t\} = |R_t|/|\Pi|$ , and we must show that

$$(5.3) \quad |R_t| \leq \sum_{k=1}^t |\Pi(x_k)|.$$

This is clearly true if  $t = 1$ , so suppose  $t \geq 2$ . There are two cases to consider.

*Case (a).* The first question is “Is  $x$  the minimum?” for some  $x \in Y$  (or equivalently, “What is  $\pi(x)$ ?”). The root of the decision tree has one son which is a leaf and is reached by the linear orders  $\pi$  in  $\Pi(x)$  (if  $\Pi(x)$  is not empty). It also has a nonleaf son  $s$  for each possible value of  $\pi(x) \neq 1$ . For each such  $s$  let  $\Pi^s$  be the set of linear orders  $\pi \in \Pi$  that reach it. Then  $\Pi^s$  is good with respect to  $Q$  and  $Y \setminus \{x\}$ . Hence we may use the induction hypothesis to bound  $|R_t \cap \Pi^s|$ .

Note that by the key observation (5.2) the sets  $\Pi^s(x_1), \Pi^s(x_2), \dots$ , other than  $\Pi^s(x)$  are nonincreasing in size (and perhaps are empty from some point on). There are two subcases, depending on whether or not  $x$  is in the set  $Y_t = \{x_1, \dots, x_{t-1}\}$ .

(a1) Suppose that  $x \in Y_t$ . Then for each nonleaf son  $s$

$$|R_t \cap \Pi^s| \leq \sum \{|\Pi^s(x_k)| : k = 1, \dots, t, x_k \neq x\}.$$

Hence

$$\begin{aligned} |R_t| &\leq |\Pi(x)| + \sum \{|\Pi(x_k)| : k = 1, \dots, t, x_k \neq x\} \\ &= \sum_{k=1}^t |\Pi(x_k)|. \end{aligned}$$

(a2) Suppose that  $x \in Y \setminus Y_t$ . Then for each nonleaf son  $s$ ,

$$|R_t \cap \Pi^s| \leq \sum_{k=1}^{t-1} |\Pi^s(x_k)|,$$

and so

$$\begin{aligned} |R_t| &\leq |\Pi(x)| + \sum_{k=1}^{t-1} |\Pi(x_k)| \\ &\leq \sum_{k=1}^t |\Pi(x_k)|. \end{aligned}$$

*Case (b).* The first question is a comparison  $\pi(x) : \pi(x')$ , where  $x, x' \in Y$ . Let

$$\Pi_l = \{\pi \in \Pi : \pi(x) < \pi(x')\}, \quad \Pi_r = \Pi \setminus \Pi_l.$$

Let the partial orders  $Q_l, Q_r$  be obtained from  $Q$  by adding the inequalities  $x < x', x > x'$ , respectively.

The root of the decision tree has a collection of left sons corresponding to  $\Pi_l$  and right sons corresponding to  $\Pi_r$ . Each left son  $s$  corresponds to a possible value for  $\pi(x')$ . Consider such a left son  $s$ . Let  $\Pi_l^s$  be the set of linear orders  $\pi \in \Pi_l$  that reach it. Then  $\Pi_l^s$  is good with respect to  $Q_l$  and  $Y \setminus \{x'\}$ . Also note that by the key observation (5.2) the sets  $\Pi_l^s(x_1), \Pi_l^s(x_2), \dots$ , other than  $\Pi_l^s(x)$  and  $\Pi_l^s(x') (= \emptyset)$  are nonincreasing in size. Similar comments hold for each right son  $s$ , with corresponding set  $\Pi_r^s$  of linear orders. Thus we may use the induction hypothesis much as in Case (a).

Note that if  $x$  or  $x'$  is in  $Y_t$ , then by (5.2) for each left son  $s$   $|\Pi_l^s(x)| \leq |\Pi_l^s(x_t)|$ , and similarly for each right son  $s$   $|\Pi_r^s(x')| \leq |\Pi_r^s(x_t)|$ . There are now three subcases.



(b1) Suppose that  $x, x' \in Y_t$ . For each left son  $s$

$$|R_t \cap \Pi_l^s| \leq \sum_{k=1}^t |\Pi_l^s(x_k)|,$$

since the  $t$  sets  $\Pi_l^s(x_1), \dots, \Pi_l^s(x_t)$  contain  $(t-1)$  largest sets of the form  $\Pi_l^s(x_i)$  (together with the set  $\Pi_l^s(x')$  which is empty). Hence

$$|R_t \cap \Pi_l| \leq \sum_{k=1}^t |\Pi_l(x_k)|.$$

Similarly

$$|R_t \cap \Pi_r| \leq \sum_{k=1}^t |\Pi_r(x_k)|,$$

and so (5.3) holds.

(b2) Suppose that  $x \in Y_t, x' \in Y \setminus Y_t$ . For each left son  $s$ ,

$$|R_t \cap \Pi_l^s| \leq \sum_{k=1}^{t-1} |\Pi_l^s(x_k)|$$

since  $\Pi_l^s(x_1), \dots, \Pi_l^s(x_{t-1})$  are  $(t-1)$  largest sets of the form  $\Pi_l^s(x_i)$ ; and so

$$|R_t \cap \Pi_l| \leq \sum_{k=1}^{t-1} |\Pi_l(x_k)|.$$

Also, for each right son  $s$

$$|R_t \cap \Pi_r^s| \leq \sum_{k=1}^{t-1} |\Pi_r^s(x_k)| + |\Pi_r^s(x')|$$

since  $|\Pi_r^s(x')| \geq |\Pi_r^s(x_t)|$  and so the  $t$  sets  $\Pi_r^s(x_1), \dots, \Pi_r^s(x_{t-1}), \Pi_r^s(x')$  contain  $(t-1)$  largest sets of the form  $\Pi_r^s(x_i)$  (together with the set  $\Pi_r^s(x)$  which is empty). Hence

$$|R_t \cap \Pi_r| \leq \sum_{k=1}^{t-1} |\Pi_r(x_k)| + |\Pi_r(x')|.$$

Thus

$$\begin{aligned} |R_t| &\leq \sum_{k=1}^{t-1} (|\Pi_l(x_k)| + |\Pi_r(x')|) \\ &\leq \sum_{k=1}^t |\Pi_l(x_k)| \quad \text{as required for (5.3).} \end{aligned}$$

(b3) Suppose that  $x, x' \in Y \setminus Y_t$ . This is the most interesting subcase. Let  $b = b(x_{t-1})$ . Consider a left son  $s$ . By the key observation (5.2), if  $\pi(x') \geq b$  (for each  $\pi \in \Pi_l^s$ ) then  $\Pi_l^s(x_1), \dots, \Pi_l^s(x_{t-1})$  are  $(t-1)$  largest sets of the form  $\Pi_l^s(x_i)$ ; and if  $\pi(x') < b$ , then  $\Pi_l^s(x_1), \dots, \Pi_l^s(x_{t-2}), \Pi_l^s(x)$  are  $(t-1)$  largest sets of this form. Hence

$$|R_t \cap \Pi_l^s| \leq \begin{cases} \sum_{k=1}^{t-1} |\Pi_l^s(x_k)| & \text{if } \pi(x') \geq b, \\ \sum_{k=1}^{t-2} |\Pi_l^s(x_k)| + |\Pi_l^s(x)| & \text{if } \pi(x') < b. \end{cases}$$

Thus

$$\begin{aligned} |R_t \cap \Pi_l| &\leq \sum_{k=1}^{t-2} |\Pi_l(x_k)| + |\{\pi \in \Pi_l(x_{t-1}): \pi(x') \geq b\}| \\ &\quad + |\{\pi \in \Pi_l(x): \pi(x') < b\}|. \end{aligned}$$

Similarly

$$|R_t \cap \Pi_r| \leq \sum_{k=1}^{t-2} |\Pi_r(x_k)| + |\{\pi \in \Pi_r(x_{t-1}): \pi(x) \geq b\}| + |\{\pi \in \Pi_r(x'): \pi(x) < b\}|.$$

Hence

$$|R_t| \leq \sum_{k=1}^{t-2} |\Pi(x_k)| + |\{\pi \in \Pi(x_{t-1}): \max(\pi(x), \pi(x')) \geq b\}| + |\{\pi \in \Pi(x): \pi(x') < b\}| + |\{\pi \in \Pi(x'): \pi(x) < b\}|.$$

Now  $b = b(x_{t-1}) \leq b(x)$ . Let  $\pi \in \Pi(x)$ , and let  $\pi' = (x, x_{t-1})\pi$ . Then  $\pi' \in L(Q)$  since  $x_{t-1}$  is minimal in  $Q$ , and if  $x < z$  in  $Q$ , then  $\pi'(z) = \pi(z) \geq b(x) \geq b > \pi(x_{t-1}) = \pi'(x)$ . It follows that  $\pi' \in \Pi$ . Thus the map  $\pi \rightarrow \pi'$  yields an injection from  $\{\pi \in \Pi(x): \pi(x') < b\}$  into  $\{\pi \in \Pi(x_{t-1}): \pi(x), \pi(x') < b\}$ . Also by (5.2)

$$|\{\pi \in \Pi(x'): \pi(x) < b\}| \leq |\Pi(x')| \leq |\Pi(x_t)|.$$

Hence

$$\begin{aligned} |R_t| &\leq \sum_{k=1}^{t-2} |\Pi(x_k)| + |\{\pi \in \Pi(x_{t-1}): \max\{\pi(x), \pi(x')\} \geq b\}| \\ &\quad + |\{\pi \in \Pi(x_{t-1}): \max\{\pi(x), \pi(x')\} < b\}| + |\Pi(x_t)| \\ &= \sum_{k=1}^t |\Pi(x_k)|. \end{aligned}$$

This completes the proof of Lemma 5.1.  $\square$

Suppose that in case (a) of the proof above the set  $\Pi(x)$  is empty. Then arguing as above, we see that

$$|R_t| \leq \sum_{k=1}^{t-1} |\Pi(x_k)|.$$

Hence the lemma will still hold if we do not charge for questions “Is  $x$  the minimum?” when  $\Pi(x)$  is empty. Thus the assistance given to the search algorithm may now take the simple form: whenever it is not possible for an element  $x$  to be the minimum, then its rank  $\pi(x)$  is revealed for free (and this process may be repeated).

Let  $Q$  be a partial order on  $X$ , with set  $Y$  of minimal elements. For each  $y \in Y$  let the “above-set”  $A(y)$  be the set of  $z \in X$  such that  $y < z$  in  $Q$ . We shall be interested in the case that the sets  $A(y)$  for  $y \in Y$  are “nested,” that is, we can list  $Y$  as  $x_1, \dots, x_m$ , say, so that  $A(x_1) \supseteq \dots \supseteq A(x_m)$ . This is certainly the case if the set  $Z = X \setminus Y$  is linearly ordered by  $Q$ .

**LEMMA 5.4.** *Let  $\Pi$  be the set of linear extensions of a partial order  $Q$  on  $X$  such that the above-sets  $A(y)$  of the minimal elements  $y$  are nested. Suppose that each linear order  $\pi$  in  $\Pi$  is equally likely. Then for any assisted search algorithm as above the random search time  $T$  satisfies  $T \geq \tilde{T}$  in distribution, where  $\tilde{T} = \tilde{T}(\pi^{-1}(1))$ .*

*Proof.* Suppose that revealing the rank  $\pi(z)$  of the nonminimal elements  $z$  partitions  $\Pi$  into nonempty sets  $\Pi^s$  of linear orders. Then each set  $\Pi^s$  is good with respect to  $Q$  and  $Y$ . Also by the key observation (5.2),  $|\Pi^s(x_1)| \geq \dots \geq |\Pi^s(x_m)|$ . Hence by Lemma 5.1, for any  $1 \leq t \leq m$

$$|R_t(\Pi^s)| \leq \sum_{k=1}^t |\Pi^s(x_k)|,$$

and so

$$|R_t(\Pi)| = \sum_s |R_t(\Pi^s)| \leq \sum_{k=1}^t |\Pi(x_k)|. \quad \square$$

*Proof of Theorem 2.11.* Note first that it is sufficient to prove the theorem for the case when the set  $X \setminus Y$  is linearly ordered by  $Q$ . For we could assist the search algorithm by first revealing the linear order on this set; then for each subproblem the expected search time would be at least  $1 + m(m - 1)/2n$ . And so, the overall expected (assisted) search time would be at least this value.

Let us assume then that the set  $X \setminus Y$  is linearly ordered by  $Q$ . The key observation now is that for each  $x \in Y$

$$(5.5) \quad |\Pi(x)| \geq |\Pi|/n.$$

To see this, define a function  $\phi$  from  $\Pi$  into  $\Pi(x)$  as follows. For each linear order  $\pi \in \Pi$  let  $\phi(\pi)$  be the linear order in  $\Pi$  obtained by moving  $x$  to be the minimum. Then for each  $\pi_0 \in \Pi(x)$  we have  $|\{\pi \in \Pi: \phi(\pi) = \pi_0\}| \leq n$ . Hence  $|\Pi| \leq n|\Pi(x)|$ , as required. (See also Theorem 3.3.4 of Mairson [6], [7].)

Now by Lemma 5.4  $E[T] \geq E[\tilde{T}]$ . But  $E[\tilde{T}] = \sum_j j p_j$ , where the summation is over  $j = 1, \dots, m$  and  $p_1, \dots, p_m$  are the values  $|\Pi(x)|/|\Pi|$  arranged in nonincreasing order. Also the minimum value of the linear program  $\min \sum_j j p_j$  subject to  $\sum_j p_j = 1, p_j \geq 1/n$  ( $j = 1, \dots, m$ ) clearly occurs at  $p_2 = \dots = p_m = 1/n, p_1 = 1 - (m - 1)/n$ , and the value equals

$$1 - (m - 1)/n + (1/n) \sum_{j=2}^m j = 1 + m(m - 1)/2n.$$

Hence we have shown that

$$E[T] \geq E[\tilde{T}] \geq 1 + m(m - 1)/2n$$

as required.  $\square$

*Remark.* Lemma 5.4 does *not* hold for an arbitrary partial order  $Q$ .

*Example 5.6.* Let  $X = \{x_1, \dots, x_6\}$  and let the partial order  $Q$  consist of the relations  $x_1 < x_3, x_2 < x_3, x_4 < x_6, x_5 < x_6$ , as shown in Fig. 1.

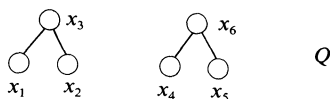


FIG. 1

A decision tree yielding the random search time  $\tilde{T}$  is shown in Fig. 2. Here the notation “ $x_i?$ ” means “Is  $x_i$  the minimum?”, and  $y, n$  stand for yes and no. The numbers at the leaves are the numbers of linear extensions of  $Q$  reaching them. Note that

$$P\{\tilde{T} \leq 3\} = 60/80.$$

A decision tree with initial query the comparison  $\pi(x_1): \pi(x_4)$  is shown in Fig. 3. Note that here

$$P\{T \leq 3\} = 62/80.$$

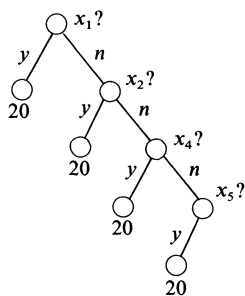


FIG. 2

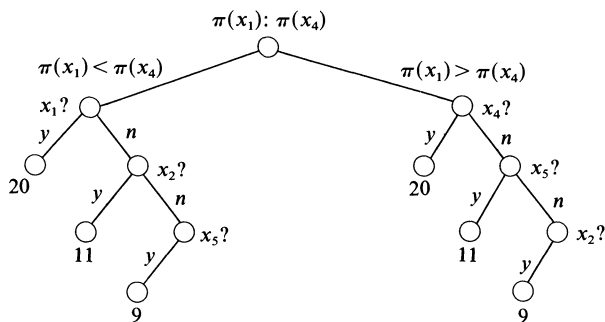


FIG. 3

Perhaps for any partial order  $Q$  we still have  $E[\log T] \cong E[\log \tilde{T}]$ . If this is so, then we could deduce as for Theorem 2.1 (iii) that

$$E[n \log S] \cong \log |\Pi| - n \log \log n,$$

and Conjecture 2.10 would follow.

**Acknowledgments.** My thanks to Allan Borodin and to a referee for bringing to my attention the recent reference [10].

REFERENCES

[1] H. ALT, K. MEHLHORN, AND J. I. MUNRO, *Partial match retrieval in implicit data structures*, Inform. Process. Lett., 19 (1984), pp. 61–65.  
 [2] H. ALT AND K. MEHLHORN, *Searching semisorted tables*, SIAM J. Comput., 14 (1985), pp. 840–848.  
 [3] R. ASH, *Information Theory*, John Wiley, New York, 1967.  
 [4] A. BORODIN, L. J. GUIBAS, N. A. LYNCH, AND A. C. YAO, *Efficient searching using partial ordering*, Inform. Process. Lett., 12 (1981), pp. 71–75.  
 [5] N. LINIAL AND M. SAKS, *Every poset has a central element*, J. Combin. Theory Ser. A, 40 (1985), pp. 195–210.  
 [6] H. G. MAIRSON, *Average case lower bounds on the construction and searching of partial orders*, Rapports de Recherche 415, INRIA, June 1985.  
 [7] ———, *Average case lower bounds on the construction and searching of partial orders*, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 303–311.  
 [8] M. SAKS, *The information theoretic bound for problems on ordered sets and graphs*, in Graphs and Orders, I. Rival, ed., D. Reidel, Dordrecht, the Netherlands, 1985, pp. 137–168.  
 [9] A. C. YAO, *Should tables be sorted?*, J. Assoc. Comput. Mach., 28 (1981), pp. 615–628.  
 [10] A. BORODIN, F. E. FICH, F. MEYER AUF DER HEIDE, E. UPFAL, AND A. WIGDERSON, *Tradeoff between search and update time for the implicit dictionary problem*, Proc. ICALP 86, Lecture Notes in Computer Science, Springer-Verlag, New York, Heidelberg, Berlin, 1986, pp. 50–59.

**ERRATUM:**  
**An  $O(n \log \log n)$ -Time Algorithm for  
Triangulating a Simple Polygon\***

ROBERT E. TARJAN<sup>†‡</sup> AND CHRISTOPHER J. VAN WYK<sup>†</sup>

Figures 22 and 23 were erroneously transposed on pages 175 and 176 (*SIAM J. Comput.*, 17(1988), pp. 143-178). They should have appeared as follows:

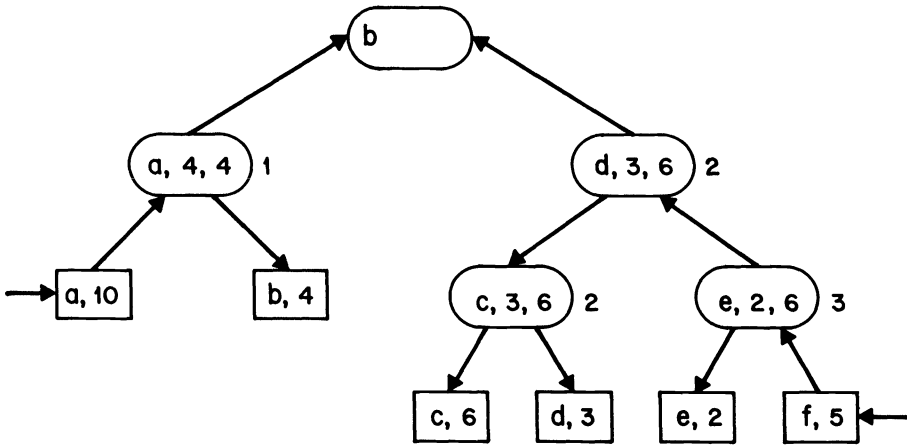


FIG. 22. A heterogeneous red-black finger search tree. The colors of nodes are not shown. The items are the letters *a* through *f*. The numbers in external nodes are secondary values. The numbers in internal nodes are the minimum and maximum secondary values reachable from the nodes. The numbers outside the nodes are the number of external nodes reachable from them.

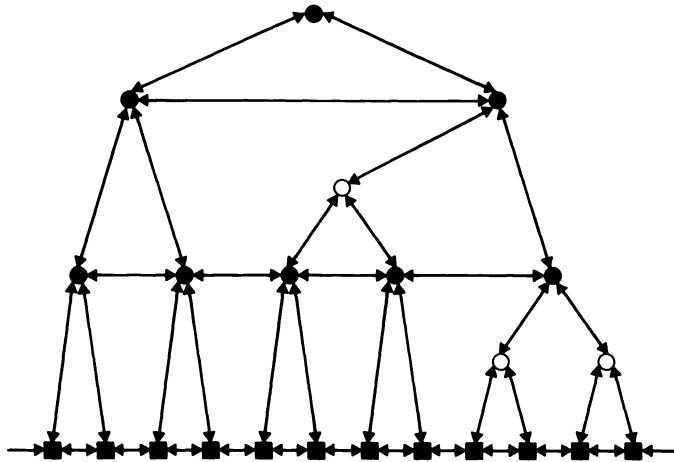


FIG. 23. The pointers in a homogeneous red-black finger search tree.

SIAM regrets this error.

\*Received by the editors July 13, 1988; accepted for publication July 25, 1988.

<sup>†</sup>AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

<sup>‡</sup>Department of Computer Science, Princeton University, Princeton, New Jersey 08544. The work of this author was partially supported by National Science Foundation grant DCR-8605962.

## EFFICIENT SOLUTION OF CONNECTIVITY PROBLEMS ON HIERARCHICALLY DEFINED GRAPHS\*

THOMAS LENGAUER† AND EGON WANKE†

**Abstract.** Using hierarchical definitions we can describe very large graphs in small spaces. In this paper we discuss how such succinct graph descriptions can be used to speed up the solution of graph problems. We present the bottom-up method that solves graph problems without expanding the hierarchical description. This allows solutions that are efficient in terms of the hierarchical graph description instead of the size of the expanded graph. We exemplify the method by giving efficient solutions to connectivity problems on hierarchical graphs. Our results have applications in computer-aided design for integrated circuit design and other engineering problems.

**Key words.** connectivity, biconnectivity, hierarchical graphs, succinct graph descriptions, query

**AMS(MOS) subject of classification.** 05C40

**1. Introduction.** Graph algorithms have many applications in engineering, e.g., in the areas of mechanical and electrical engineering. As the technology in these areas advances, the sizes of graphs to be processed increases dramatically. Perhaps the best example for this development is the area of very large-scale integration (VLSI) design. Today's integrated circuits have up to several hundred thousand transistors. Processing such designs requires the internal representation of graphs with up to over a million vertices.

On the other hand large engineering designs always have some regular structure. This is because they are developed with relatively small resources, e.g., a few people in a relatively short time. Therefore most computer-aided design (CAD) systems for engineering design allow for some mechanism of succinct design description. This enables the designer to describe a large design in a small space. The complete explicit design description is generated by the computer in a data-expansion step.

Traditionally this data expansion was the first action taken by the CAD system for design processing. It involves the generation of large amounts of data that with today's applications generally do not fit into the address space of the computer. Thus processing the expanded data is both complicated (since it involves secondary storage management) and inefficient (because of large amounts of input/output on secondary storage). Even if the expanded design fits into the address space of the computer and secondary storage management can be avoided, page swapping is usually extensive. The resulting performance problems often are the critical obstacle to the acceptance of CAD tools.

In this paper we will present a method that circumvents these problems, when design processing is done on the basis of graph algorithms. The regularity of an engineering design is reflected by the regularity of the graph representing it. Thus such a graph can also be described succinctly. We will present a method that allows us to process succinctly described graphs *before* they are expanded. Thus the data-expansion step may only be needed to generate the final output of the processing. As a consequence, working storage requirements for design processing are dramatically reduced, and secondary storage management as well as expensive page-swapping overhead can be avoided. However, this is only possible if we choose an appropriate model for succinct graph description.

---

\* Received by the editors August 27, 1985; accepted for publication (in revised form) November 9, 1987.

† Universität-Gesamthochschule Paderborn, Fachbereich 17—Mathematik/Informatik, Paderborn, Federal Republic of Germany.

The remainder of the paper is organized as follows: Section 2 gives a short summary of models for succinct graph description presented in the literature, and introduces the graph model we use in this paper. Section 3 introduces the *bottom-up method* for hierarchical graph processing and applies it to the connectivity problem of undirected graphs. Sections 4 and 5 present the hierarchical solution of the biconnectivity, respectively, strong connectivity problem. Section 6 gives conclusions.

**2. The graph model.** Several approaches towards succinct graph description have been pursued in the literature. In [GW83] graphs are succinctly described by Boolean circuits computing their adjacency matrix. Thus a graph  $G = (V, E)$  with  $n$  vertices ( $V = \{0, \dots, N-1\}$ ) is defined by a Boolean circuit computing a predicate  $\phi(i, j) \in \{0, 1\}$ ,  $0 \leq i, j \leq N-1$ , where  $\phi(i, j) = 1$  if and only if  $\{i, j\} \in E$ . For example, the square grid of  $N \times N$  nodes is defined by the predicate

$$\phi(i, j) \Leftrightarrow (|x_i - x_j| = 1 \text{ and } y_i = y_j) \text{ or } (|y_i - y_j| = 1 \text{ and } x_i = x_j), i, j \in \{0, N^2 - 1\}$$

where  $x_i = i \bmod N$ ,  $y_i = i \text{ div } N$ . (For simplicity assume that  $n$  is a power of 2.) Note that the Boolean circuit size of  $\phi(i, j)$  is  $O(\log N)$  in this case. Unfortunately, [GW83] shows that this data compression cannot be significantly exploited when processing the graph: Even the simplest graph problems, e.g., connectivity, minimum or maximum degree, etc. become NP-hard when the graph is defined in this form.

Reference [BOW83] introduces an explicit hierarchical specification for sets of iso-oriented rectangles in the Euclidean plane. Their notion of hierarchy is taken from mask data formats for VLSI layouts (see [MC80]). Reference [BOW83] shows that many simple problems on rectangle sets (e.g., intersection questions) become NP-complete if the set is defined hierarchically. Reference [Wa84] translates the results of [BOW83] into graph theory. Bentley et al. hierarchically describe graphs whose vertices are points in  $d$ -dimensional Euclidean space. Here a graph is built up out of primitive components (edges, denoted by sets of pairs of vertices) and copies of previously defined subgraphs (cells).

A cell can be copied by translating it along a specified vector. The copy of the cell is specified by an identifying number of the cell and the point in space defining the vector of translation. Thus the  $N \times N$  square grid that is placed into the plane such that all edges have unit length and the lower left corner is placed at the origin can be defined by the following sequence of subcells. The cell  $G_k$  defines a  $(2^k + 1) \times (2^k + 1)$  grid.

$G_0$ :	primitive edges:	$\{(0, 0)(0, 1)\}, \{(0, 0)(1, 0)\},$ $\{(1, 0)(1, 1)\}, \{(0, 1)(1, 1)\}$
	no subcells	
$G_k$ ( $k > 0$ ):	no primitive edges	
	subcells	$(G_{k-1}, (0, 0)), (G_{k-1}, (0, 2^{k-1})),$ $(G_{k-1}, (2^{k-1}, 0)),$ $(G_{k-1}, (2^{k-1}, 2^{k-1}))$

(The fact that each interior edge of the grid is represented twice here is technical and insignificant for our discussion.) In this way we can describe an  $N \times N$  grid with a specification of length  $O((\log N)^2)$  (logarithmic cost measure).

Reference [Wa84] shows that in this model even the simplest graph problems become NP-hard. The intuitive reason is that two subcells can be placed such that they overlap each other. This placement can be described in space  $O(1)$ , but it can make a large number of "implicit" connections in the region of overlap. Thus in order to make the graph model amenable to hierarchical processing we have to explicitly define a "boundary" of a subcell, i.e., a set of vertices at which the environment can connect to the cell. We call such vertices *pins*.

Reference [S182] shows that if connections to subcells are restricted to pins, and if there are only a few pins in a subcell, then certain problems become easy. Thus our graph model will be a certain restricted type of context-free graph grammar.

DEFINITION 1. A hierarchical (un-) directed graph  $\Gamma = (G_1, \dots, G_k)$  is a finite sequence of (un-) directed simple graphs  $G_i$  called cells. Here the graph  $G_i$  has  $n_i$  vertices and  $m_i$  edges.  $p_i$  of the vertices are distinguished and called pins. The other  $n_i - p_i$  vertices are called inner vertices.  $r_i$  of the inner vertices are distinguished and called nonterminals. The other  $n_i - r_i$  vertices are called terminals or proper vertices.

Each pin has a unique label, its name. Without loss of generality we can assume that the pins are named with numbers between 1 and  $p_i$ . Each nonterminal has two labels, a name and a type. The type is a symbol from  $\{G_1, \dots, G_{i-1}\}$ . If a nonterminal  $v$  has type  $G_j$  then  $v$  has degree  $p_j$  and each proper vertex that is a neighbor of  $v$  has a label  $(v l)$  such that  $1 \leq l \leq p_j$ . We say that the neighbor of  $v$  labeled  $(v, l)$  matches the  $l$ th pin of  $G_j$ . (All neighbors of a nonterminal must be terminals.)

We assume that  $\Gamma$  is not redundant in the sense that each  $G_i$  is the type of some nonterminal  $\nu$  in some  $G_j, j > i$ . The size of  $\Gamma$  is  $n := \sum_{1 \leq i \leq k} n_i$ , the edge number is  $m := \sum_{1 \leq i \leq k} m_i$ . Note that with  $\Gamma = (G_1, \dots, G_k)$  also each prefix  $\Gamma_i = (G_1, \dots, G_i), i < k$  is a hierarchical graph.

Definition 1 essentially describes a context-free graph grammar with axiom  $G_k$ , where each nonterminal only has one production. Thus the language of the graph grammar has one word. This word is called the expansion of  $\Gamma$ .

DEFINITION 2. The *expansion*  $E(\Gamma)$  of the hierarchical graph  $\Gamma$  is obtained as follows:

$k = 1$ :  $E(\Gamma) = G_1$ . The pins of  $E(\Gamma)$  are the pins of  $G_1$ .

$k > 1$ : Repeat the following step for each nonterminal  $\nu$  of  $G_k$ , say, of type  $G_j$ :  
Delete  $\nu$  and its incident edges. Insert a copy of  $E(\Gamma_j)$  by identifying the  $l$ th pin of  $E(\Gamma_j)$  with the node in  $G_k$  that is labeled  $(\nu, l)$ .

The size of  $E(\Gamma)$ , i.e., its number of vertices, is denoted by  $N$ . The number of edges of  $E(\Gamma)$  is denoted by  $M$ .

The expansion of  $\Gamma$  naturally has a tree structure, namely the parse tree for the word  $E(\Gamma)$  in the graph grammar  $\Gamma$ .

DEFINITION 3. With  $\Gamma$  and  $E(\Gamma)$  we associate the so-called *hierarchy tree*  $T$  that is defined as follows. Each node in  $T$  has two labels, a name and a type. (The root of  $T$  has no name.) A node  $x$  of type  $G_i$  in  $T$  has  $r_i$  sons. Each son of  $x$  has the name and type of one of the nonterminals in  $G_i$ .  $T$  is defined inductively:

$k = 1$ :  $T$  has one node with type  $G_1$ .

$k > 1$ :  $T$  has a root of type  $G_k$ . The son of the root that has the name and type of the nonterminal  $\nu$  of type  $G_j$  in  $G_k$  is the root of a copy of the hierarchy tree for  $\Gamma_j = (G_1, \dots, G_j)$ .

$T$  is the parse tree of the unique word generated by the context-free graph grammar. Clearly each node of type  $G_i$  in the hierarchy tree  $T$  corresponds to a copy of  $E(\Gamma_i)$  in  $E(\Gamma)$ . (In the context of  $T$  we will use exclusively the term node. In the context of  $\Gamma$  or  $E(\Gamma)$  we will use exclusively the term vertex.)

The terminology in the above three definitions is taken partly from the area of graph grammars (nonterminals) and partly from the area of CAD for VLSI design (cells, pins).

Let us define the  $N \times N$  grid hierarchically using the above definition. This time we choose  $N = 2^k$ , again for technical reasons.  $\Gamma = (G_1, \dots, G_k)$ , where  $\Gamma_i$  defines the  $2^i \times 2^i$  grid. We define  $G_i$  pictorially. Hereby we denote pins by squares, nonterminals by circles labeled with the type of subcells they represent, and terminals that are not



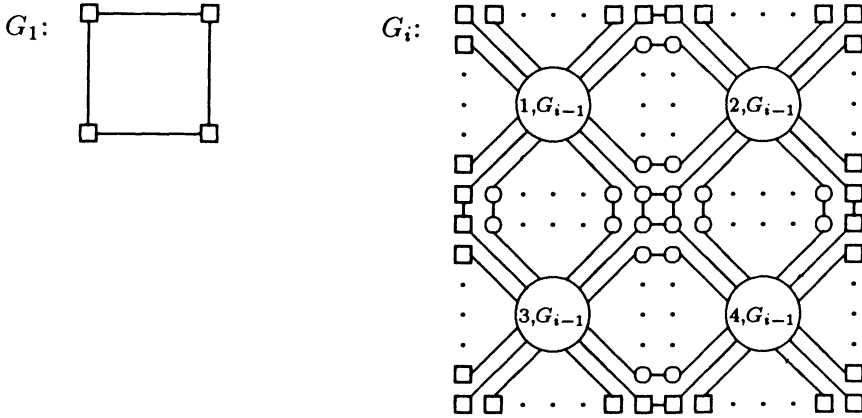


FIG. 1

pins by dots.  $\Gamma$  is defined as in Fig. 1. The labels on the neighbors of nonterminals are omitted for the sake of readability of Fig. 1. The correspondence is clear from the position of the vertices in the figure.

Note that  $\Gamma$  has length  $O(N)$ , as opposed to  $O((\log N)^2)$  in the model of [Wa84]. This is because the boundary of a grid grows as the square root of the size of the grid. There are hierarchical graphs such that  $n = O(\text{polylog}(N))$ , but in general this model is weaker than the model of [Wa84] in that hierarchical descriptions can be longer. However, the model presented here allows us to solve a large number of interesting and important graph problems before data expansion. Furthermore the model is quite natural and corresponds to the actual representation of data in a number of CAD systems.<sup>1</sup>

The following example will accompany us through the remainder of the paper.

*Example 1.* Let  $\Gamma = (G_1, G_2, G_3, G_4)$  be the hierarchical graph shown in Fig. 2. The expansion  $E(\Gamma)$  is shown in Fig. 3; the hierarchy tree is shown in Fig. 4.

In Fig. 4, we again omitted labels on the vertices. The correspondence between pins of  $G_j$  and neighbors of a nonterminal of type  $G_j$  in cell  $G_i$  is clear by the positions of the vertices in the figure. Each edge in  $E(\Gamma)$  belongs to a unique node  $x$  in  $T$ . For example, the edge  $e_1$  in Fig. 3 belongs to the unique node in  $T$  that is of type  $G_3$ . Similarly a vertex in  $E(\Gamma)$  belongs to a unique node  $x$  in  $T$ , where that vertex appears as a terminal that is not a pin. For example, the vertex  $v_1$  belongs to the same node  $x$  in  $T$  as the edge  $e_1$ . (This is even though the vertex  $v_1$  is also represented by pins in three descendants of  $x$  in  $T$ .) Thus an edge (vertex) in  $E(\Gamma)$  can be identified by providing the path from the root of  $T$  to the node  $x$  in  $T$  to which the edge (vertex) belongs, and then identifying the edge (vertex) inside  $x$ . In general, a pathname is a sequence of nonterminals  $v_1, \dots, v_{i-1}$  followed by a terminal node  $v$  or an edge  $e$ . Here,  $v_1$  is a nonterminal in  $G_k$  and if  $v_k$  is a nonterminal of type  $G_j$ , then  $v_{k+1}$  is a nonterminal in  $G_j$ . For technical reasons we assume pathnames start with a dummy nonterminal of type  $G_k$ . Note that  $E(\Gamma)$  can have multiple edges, in general, even though  $G_i$  has no multiple edges, for all  $i$ . However, the results in this paper also generalize to the case that  $G_i$  may be a multigraph.

<sup>1</sup> Reference [Ga82] presents a hierarchical graph model that is based on the abutment of subcells at lists of pins. This model allows some degree of hierarchical processing but in general some expansion is necessary, because pin lists can also be composed hierarchically.

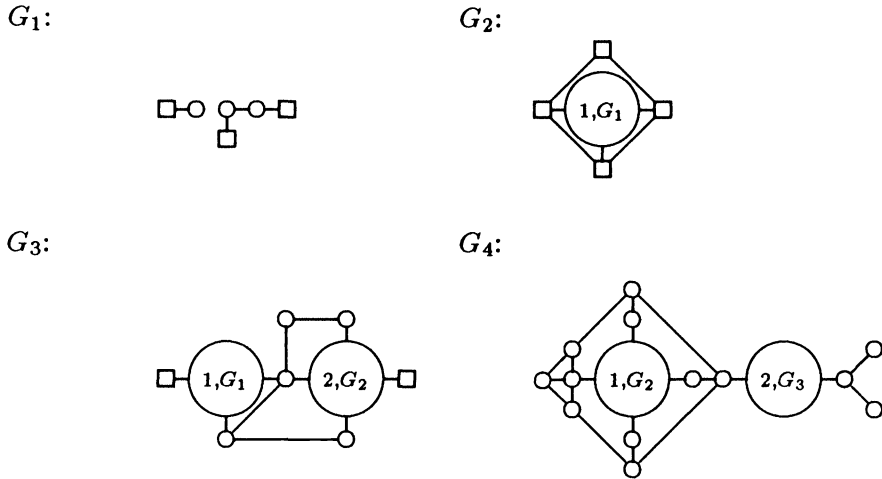


FIG. 2

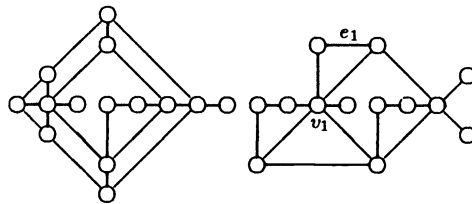


FIG. 3

**3. The bottom-up method.** For certain graph problems the solution on a subgraph can in some sense be done independently of the environment in which the subgraph is placed. Such problems can be solved hierarchically in a bottom-up fashion.

The bottom-up method is based on the concept of replaceability. Let  $P$  be a binary question that can be asked about graphs, e.g., the question “Is  $E(T)$  connected?”

DEFINITION 4. Let  $G, G'$  be two cells with pins  $p_1, \dots, p_n$  and no nonterminal vertices. Let  $H$  be a cell with no pins and exactly one nonterminal vertex  $v$ . Vertex  $v$  has degree  $n$ . Let the neighbors of  $v$  be marked as defined in Definition 1.  $G, G'$  are *replaceable* with respect to  $P$  if for each such  $H$  the answer to  $P$  is the same no matter whether we substitute  $G$  or  $G'$  for  $v$  in  $H$ .

The aim of the bottom-up method is to find small graphs  $G_i^b$  that are replaceable with  $E(\Gamma_i)$  and to find such graphs fast. In order to do so, the bottom-up method fills a table of graphs, the so-called BU-table (see Fig. 5 for the BU-table for the connectivity of Example 1.) Row  $i$  of the BU-table corresponds to cell  $G_i$ . The BU-table has three columns. (We will discuss a fourth column that may be added to the BU-table later.) The first column contains the cells  $G_1, \dots, G_k$ . All other columns are empty initially.

The bottom-up method fills the second and third columns of the table row-wise from top to bottom with graphs  $\tilde{G}_i$ , respectively,  $G_i^b$  that are replaceable with  $E(\Gamma_i)$  but smaller than  $E(\Gamma_i)$ .  $\tilde{G}_i$  (in column 2) is obtained by substituting  $G_j^b$  instead of  $E(\Gamma_j)$  for all nonterminals of type  $G_j$  in  $G_i$ . Since the  $G_j^b$  and  $E(\Gamma_j)$  are replaceable by assumption,  $\tilde{G}_i$  and  $E(\Gamma_i)$  are also replaceable (with respect to question  $P$ ). We call  $\tilde{G}_i$  the *fooled* graph.  $\tilde{G}_i$  is then further reduced in size by a shrinking procedure,

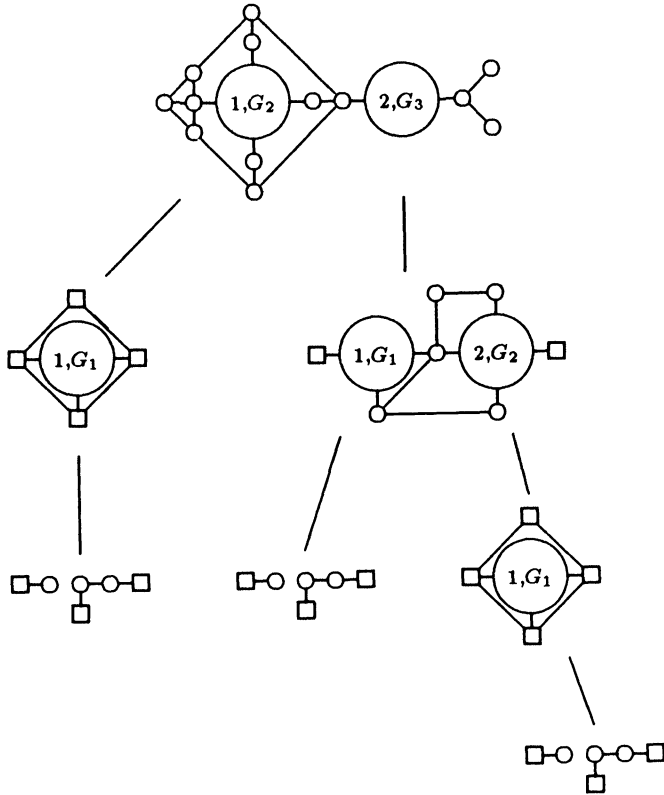


FIG. 4

called *burning*, that depends on the question  $P$ . The result  $G_i^b$  is put in column 3 and called the *burnt graph*.

The correctness of the bottom-up method is ensured if the burning process produces replaceable graphs (with respect to question  $P$ ). This is so because if we attach replaceable graphs to the same environment we get replaceable graphs again. The efficiency of the bottom-up method depends on two things. First, the burning process must be efficient. Second, the burning process must yield small graphs, such that the graphs in the BU-table do not become too large. In particular, if every  $G_i^b$  has at most  $O(p_i)$  vertices and edges, then every  $\tilde{G}_i$  has at most  $O(m_i + n_i)$  vertices and edges. In this case, we can fill the BU-table with a linear time burner in linear time in the size of the hierarchical description.

After the BU-table has been filled the answer to the question can then usually be obtained by a quick inspection of its contents.

In order to apply the bottom-up method to the question  $P$ : “*Is  $E(\Gamma)$  connected?*”, we only have to find an appropriate burning procedure. For connectivity this is especially simple: To burn  $\tilde{G}_i$  down to  $G_i^b$  we find the connected components of  $\tilde{G}_i$ . We reserve a vertex  $c$  for each connected component that contains a pin. If there are connected components without pins we also reserve an additional vertex representing all of them. (If  $\tilde{G}_i$  does not have pins and is not connected we need two such vertices.) These vertices and the pins of  $\tilde{G}_i$  are the vertices of  $G_i^b$ . The edges in  $G_i^b$  connect each pin with the vertex representing the connected component containing this pin. The proof of the following lemma is straightforward.

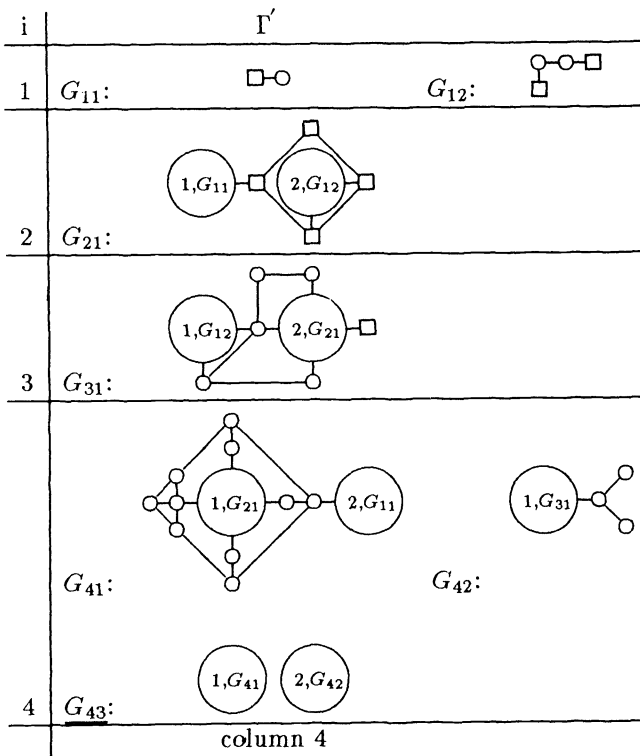
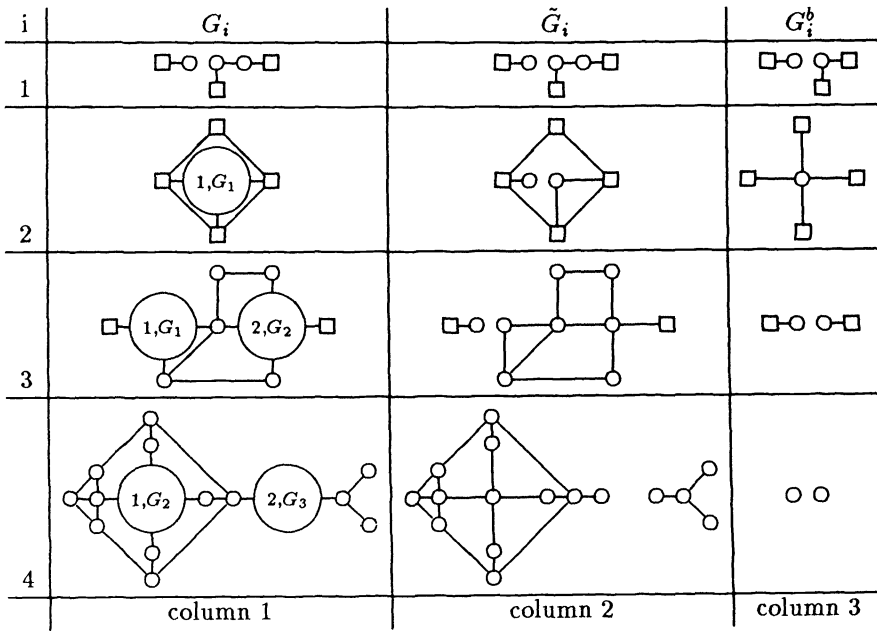


FIG. 5

LEMMA 1.  $\tilde{G}_i$  is replaceable with  $G_i^b$  with respect to connectivity.

An inductive application of Lemma 1 yields Theorem 1.

THEOREM 1.  $E(\Gamma)$  is connected if and only if  $\tilde{G}_k$  is connected.

Since the size of  $G_i^b$  is  $O(p_i)$  and  $G_i^b$  can be constructed in linear time in the size of  $\tilde{G}_i$ , connectivity of  $E(\Gamma)$  can be decided in time  $O(m+n)$ . Figure 5 gives the bottom-up table for the connectivity of Example 1.

The burning process above answer the decision problem: “Is  $E(\Gamma)$  connected?” By augmenting the BU-table appropriately we can also solve other problems pertaining to connectivity. The augmentation adds a fourth column that contains a hierarchical graph  $\Gamma'$  such that  $E(\Gamma') = E(\Gamma)$ , but  $\Gamma'$  explicitly describes the connected components of  $E(\Gamma)$ . Figure 5 shows  $\Gamma'$  for Example 1. Figure 6 gives the hierarchy tree  $T'$  or  $\Gamma'$ . In order to find  $\Gamma'$  we scan through column 2 of the BU-table. When processing row  $i$  we make a new *component* cell out of each connected component of  $\tilde{G}_i$ . Nonterminals whose types are these component cells replace the vertices of  $G_i^b$  representing the connected components later on in the hierarchy. The component cell  $G_{il}$  for the  $l$ th component of  $\tilde{G}_i$  consists of the graph of this component, where all vertices representing connected components in some copy of  $G_j^b$  inside  $\tilde{G}_i$  are replaced with nonterminals of the appropriate component cell types. A vertex that represents several isolated components of  $\tilde{G}_j$  without pins inside a copy of  $G_j^b$  is replaced with a nonterminal, whose type is a cell that consists just of isolated nonterminals representing each of the components. Such a cell is called *disconnected*. All other cells are called *connected*. The names of disconnected cells are underscored in Figs. 5 and 6. Finally, if  $G_k^b$  has several components we create a disconnected root cell joining all of them. Components containing only a nonterminal and its pins with no additional vertex or edges cause no creation of component cells. Rather, the component cell type of the nonterminal is reused higher up in the hierarchy. (In Fig. 5  $G_{11}$  is reused for a component of  $G_3$  inside  $G_{41}$  in this way.)

THEOREM 2. (a)  $E(\Gamma') = E(\Gamma)$

(b)  $m', n' = O(m+n)$ . The depth of the hierarchy tree  $T'$  of  $\Gamma'$  is at most  $k+1$ .

(c) A cell in  $\Gamma'$  is connected exactly if its expansion is connected.

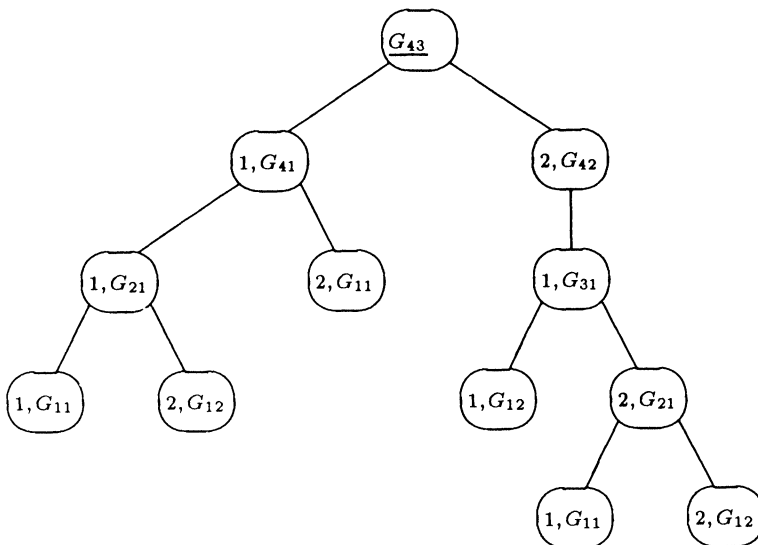


FIG. 6

(d) *The hierarchy tree  $T'$  of  $\Gamma'$  has a top part consisting exclusively of nodes whose type is a disconnected cell. The bottom part of  $T'$  consists exclusively of nodes whose type is a connected cell. The connected components of  $\Gamma$  are represented by the maximal subtrees of  $T'$  that are rooted at nodes whose type is a connected cell.*

*Proof.* The proof of part (a) is obvious.

(b)  $m', n' = O(m+n)$  since  $\tilde{G}_i$  has size  $O(m_i + n_i)$ . To see that  $T'$  has depth at most  $k+1$ , note that as we scan a path from a node  $x$  to the root in  $T'$  we move downward in column 4 of the BU-table. Here we scan at most one cell per row of the BU-table, except when we encounter the first disconnected cell. There we scan two cells out of the same row. (In the example this row is row 4.)

(c) The proof is by induction on the number of the row containing the cell in column 4 that contains the cell.

(d) The proof follows from (c).  $\square$

Because of Theorem 2(d) the pathnames in  $T'$  start with a segment of nonterminals whose type is disconnected, then continue with a segment of nonterminals whose type is connected, and finish with a terminal vertex or edge. Pathnames in  $\Gamma'$ , respectively,  $\Gamma$  can be translated into each other by scanning the pathname backwards and substituting between corresponding cells in column 1, respectively 4, of the BU-table.

We can solve a large variety of problems pertaining to connectivity on  $\Gamma'$ . For example, consider the problem of producing the adjacency structure for the connected component of  $E(\Gamma)$  containing vertex  $v$  given by its pathname in  $\Gamma$ . The solution to this problem may be a large data structure; thus the running time will be large. We call such problems construction problems. But we can produce the output using only as much working storage as needed by the BU-table. We just translate the pathname of  $v$  in  $\Gamma$  into its pathname in  $\Gamma'$  and expand the leftmost cell  $G_j$  in the pathname that is connected. If a hierarchical description of the connected component suffices we only select the cells of  $\Gamma'$  that occur inside the hierarchy tree rooted at  $G_j$ . These cells can be identified in time  $O(n)$  by scanning through the BU-table from bottom to top.

In many applications we need more information than that given by the decision problem but less information than is provided by the construction problem. Especially in interactive CAD tools we want to selectively query critical parts of the design. This corresponds, for instance, to the following query problem: Q1 ( $v, w$ ): *Given two vertices  $v, w$  in  $E(\Gamma)$ , are they connected?* Such a query can also be answered easily on  $\Gamma'$ . We just translate the pathnames of  $v, w$  in  $\Gamma$  into pathnames in  $\Gamma'$  and ask whether in both pathnames the beginning segment up to the first nonterminal whose type is connected is identical. This takes time  $O(m+n)$  for preprocessing and time  $O(k)$  for answering each query, by Theorem 2(a).

**4. Biconnectivity.** In this section we will show how to decide biconnectivity of a hierarchical graph in time  $O(m+n)$ . We will also solve construction and query problems pertaining to biconnectivity. We will not make the assumption that  $E(\Gamma)$  is connected, and answer the question  $P$ : *“Is  $E(\Gamma)$  connected and biconnected?”*

The biconnectivity burner starts on  $\Gamma$  with some graph  $\tilde{G}_i$ , and manipulates the following kind of tree structures (see [Ha69] for the basic definitions of biconnectivity).

**DEFINITION 5.** The *biconnectivity forest* for a graph  $G$  is a bipartite forest  $T^b$ , with two kinds of labeled nodes, a-nodes and b-nodes. Each a-node  $a$  is labeled with an articulation point  $v_b$  of  $G$ . Each b-node  $b$  is labeled with a block  $C_b$  of  $G$ . There is an edge between  $a$  and  $b$  if  $v_a \in C_b$ . The biconnectivity forest  $T$  is said to *represent*  $G$ .

In order to keep the biconnectivity burner simple we will sometimes mark a graph  $G_i^b$  after burning. Marking  $G_i^b$  will mean that  $G_i^b$  is not biconnected and can never

be biconnected in any environment. When a marked graph  $G_i^b$  is substituted into some  $G_i$  to yield  $\tilde{G}_i$ ,  $\tilde{G}_i$  is marked too. When  $\tilde{G}_i$  is marked,  $G_i^b$  is marked automatically. Replaceability will be shown with respect to the question  $P'$ : “Is  $G$  connected and biconnected, or is  $G$  marked?” The biconnectivity burner is defined as follows.

- (1) *Construct biconnectivity forest.* Construct the biconnectivity forest  $T^b$  for the resulting graph  $G$ .
- (2) *Delete leafblocks not containing pins.* Delete a leaf  $b$  in a tree of  $T^b$  representing a connected component of  $G$  such that  $b$  is a b-node, and the only pin of  $C_b$ , if any, is the unique relevant articulation point in  $C_b$  (if any). If now an a-node becomes a leaf, delete the a-node as well. Repeat this step until no more deletions are possible. (Do not delete the last b-node of  $T^b$ .)
- (3) *Collapse long paths.* Let  $a_1, b_1, \dots, a_{n-1}, b_{n-1}, a_k (k > 3)$  be a path of maximal length in  $T^b$  with the following properties:
  - (a) The  $a_i$  are a-nodes, the  $b_i$  are b-nodes;
  - (b) All nodes on the path have degree two;
  - (c) No labels of nodes on the path are pins or contain pins.
 Replace the path  $b_1, a_2, \dots, b_{k-1}$  with a new b-node  $b'$  in  $T^b$ . Connect  $b'$  to  $a_1$  and  $a_k$ . Label  $b'$  with the graph consisting just of  $v_{a_1}$  and  $v_{a_2}$  and the edge between them. (We call this edge a *collapsed edge*.) Repeat this step until no more changes are possible.
- (4) *Reduce blocks.* Consider all b-nodes  $b$  in  $T$  that are still labeled with blocks of  $\tilde{G}_i$ . Replace  $C_b$  with the following graph  $C'_b$ , which is chosen via a case distinction with respect to the size of the set  $S_b$  of pins and relevant articulation points inside  $C_b$  after Step (2).
  - Case 1.*  $|S_b| \geq 3$ .  $C'_b$  is a simple cycle containing all vertices in  $S_b$ .
  - Case 2.*  $|S_b| = 2$ .  $C'_b$  is the graph containing the two vertices in  $S_b$  and an edge between them.
  - Case 3.*  $|S_b| = 1$ . In this case  $S_b$  contains one pin. (All other parts were deleted in Step (2).)
  - Case 3.1.* If  $C_b$  contains at least one edge then  $C'_b$  is the graph consisting of the vertex in  $S_b$ , an additional new vertex, and an edge between them.
  - Case 3.2.* Otherwise  $C_b = C'_b$ , i.e.,  $C_b$  consists of only the pin in  $S_b$ .
  - Case 4.*  $|S_b| = 0$ .  $C'_b$  consists of a single vertex with no edges.
 All edges created in this step are called *cycle edges*.
- (5) *Mark  $G_i^b$ .* If any deletions have been made in Step (2) or if  $\tilde{G}_i$  was marked, mark  $G_i^b$ .

The tree  $T^b$  resulting after completion of the reduction represents  $G_i^b$ . Figure 7 shows an example of an application of the biconnectivity burner. Note that all reduction steps yield valid biconnectivity forests for graphs of decreasing size.

Figure 8 shows the BU-table for biconnectivity for Example 1.

Let us first bound the size of  $G_i^b$ .

LEMMA 2.  $G_i^b$  has size  $O(p_i)$ .

*Proof.* In Step (2) the graph  $\tilde{G}_i$  is reduced such that each leaf of  $T^b$  is a bold node. After Step (3) we meet a bold node at least every fourth node along a path of degree-2 nodes in  $T^b$ . Since each pin labels only one bold node, the size of  $T^b$  is  $O(p_i)$  after Step (3). Thus  $G_i^b$  has size  $O(p_i)$ , as well.  $\square$

Since the biconnectivity forest of  $\tilde{G}_i$  can be generated in linear time, generating the BU-table takes time  $O(m + n)$ . In order to show the correctness of the biconnectivity burner we prove the following theorem.

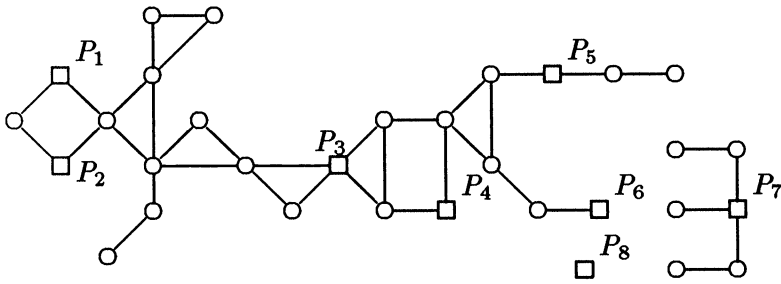


FIG. 7(a). A graph  $\tilde{G}_i$  with eight pins.

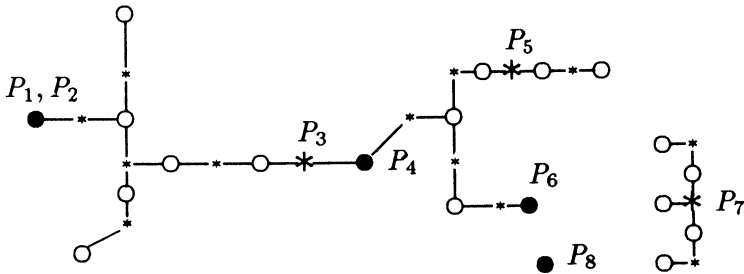


FIG. 7(b). The biconnectivity forest  $T^b$  for  $G$  constructed in Step (1). In Figs. 7(b), (c), (d): \*—a-node that does not represent a pin; \*—a-node that represents a pin;  $\bigcirc$ —b-node representing a block that has no pins that are not articulation points in  $T^b$ ;  $\bullet$ —b-node representing a block that has pins that are not articulation points in  $T^b$ .

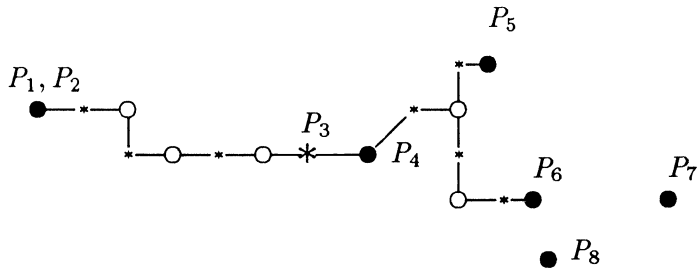


FIG. 7(c). The forest  $T^b$  after Step (2).

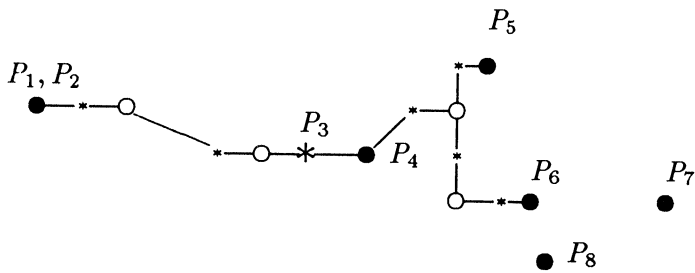


FIG. 7(d). The forest  $T^b$  after Step (3).



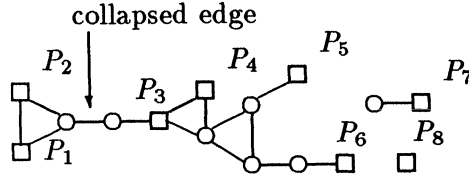


FIG. 7(e). The graph  $G_i^b$ .

$i$	$G_i$	$\tilde{G}_i$	$G_i^b$
1			
2			
3			
4			
	column 1	column 2	column 3

FIG. 8. Marks in the lower right corner of an entry show that a graph is marked.

THEOREM 3. (a)  $\tilde{G}_i$  and  $G_i^b$  are replaceable with respect to  $P'$ .

(b)  $E(\Gamma)$  is connected and biconnected if and only if  $G_k^b$  is not marked.

*Proof.* (a) We show that each step of the reduction transforms a graph  $G$  into a new graph  $G'$  that is replaceable with  $G$  with respect to  $P'$ . Step (1) does not change  $G$ . Step (2) deletes parts of  $G$  without pins that are attached at some articulation point and thus can never be biconnected in any environment. The rest of  $G$  is not changed. Marking  $G'$  if any deletions occur ensures replaceability. Steps (3) and (4) yield replaceable graphs since paths through  $G$  can be simulated by paths through  $G'$  and vice versa, even if a given vertex has to be circumvented. Case 3.1 in Step (4) ensures that the single pin  $p$  in  $S_b$  remains an articulation point in  $G'$  for any environment  $H$  that has an edge incident to  $p$ .

(b) By Step (5) of the burning process  $G_i^b$  is marked only if some deletion of material from  $\tilde{G}_i$  occurred that could never be biconnected in any environment. Since  $G_k^b$  always is a single vertex ( $G_k$  has no pins), (b) follows from part (a).  $\square$

As in § 3 we augment the BU-table with a fourth column containing a hierarchical graph  $\Gamma'$  such that  $E(\Gamma') = E(\Gamma)$  but the blocks of  $E(\Gamma)$  are represented explicitly in

$\Gamma'$ . We construct the portion of  $\Gamma'$  belonging to row  $i$  when burning  $\tilde{G}_i$  down to  $G_i^b$ . Parts cut off during Step (2) of the burning process are represented by so-called *branch* cells. Paths collapsed in Step (3) are represented by *path* cells. Blocks shrunk in Step (4) are represented by *block* cells. The details of this process are as follows.

In Step (2) we create a block cell for each deleted b-node. This is done as described for Step (4) below. The pins of a block cell are the relevant articulation points of the block. Finally, for each connected part cut off from the tree we create a branch cell recreating it. This cell has nonterminals whose types are the block cells representing cut off blocks in the component. These nonterminals are connected as the cutoff part of the tree prescribes. Each branch cell has one pin, namely the articulation point at which the part was cut off. There is an additional branch cell representing all connected components of the biconnectivity tree that have no pins. An identical branch cell already existing can be reused, but a block cell must not be reused as a branch cell.

During Step (3) we create a new block cell for each block on the collapsed path as described below (Step (4)). Then a new path cell is created that recreates the collapsed path and the parts of  $\tilde{G}_i$  attached to it that were deleted in Step (2). To this end, nonterminals whose types are the block, path, and branch cells representing the parts of the collapsed path are created and connected appropriately. The resulting path cell represents the collapsed edge in  $G_i^b$ .

During Step (4) we create for each reduced block a block cell. This block cell has the same general structure as the block in  $\tilde{G}_i$  but parts from some copy of  $G_j^b$  are represented by their corresponding block cells. If a block consists of a single edge that is a path edge, no block cell is created for this block, but the corresponding path cell is reused higher up in the hierarchy. The pins of a block cell are all pins and all relevant articulation points of the corresponding block.

The result of this process is shown in Fig. 9 for Example 1. Figure 10 shows the hierarchy tree  $T'$  of  $\Gamma'$ . Composed cells are underlined in Fig. 10.

THEOREM 4. (a)  $E(\Gamma) = E(\Gamma')$ .

(b)  $m', n' = O(m + n)$ . The depth of the hierarchy tree  $T'$  of  $\Gamma'$  is  $O(k)$ .

(c) Each block in  $E(\Gamma)$  is represented by a subtree of  $T'$  with the following properties:

(c.1) The type of the root  $x$  of the subtree is a block cell, and when going from  $x$  towards the root of  $T'$  we meet first (perhaps) a few path cells and then a branch cell.

(c.2) From the subtree rooted at  $x$  in  $T'$  all subtrees are deleted that are rooted at branch cells.

In Fig. 10 the subtrees of  $T'$  that represent blocks of  $E(\Gamma)$  are circled with dotted lines.

*Proof.* (a) The proof is by induction on the cell number in  $\Gamma'$ .

(b) Since  $\tilde{G}_i$  has size  $O(m_i + n_i)$ ,  $m', n' = O(m + n)$ .  $T'$  has depth  $O(k)$  since on each path from a node  $x$  in  $T'$  to the root at most four cells in each row entry of column 4 of the BU-table can occur, namely two block cells, one branch cell, and one path cell.

(c) The proof holds because of the following facts. First, when cutting off parts of  $E(\Gamma)$  in Step (2) we cut off blocks that can never be enlarged in any environment. A branch cell represents the part of  $E(\Gamma)$  being cut off. Second, branch cells and path cells represent compositions of blocks, in which each block is also represented by a separate block cell. Third, because path cells are reused, cells representing chains of blocks collapsed in Step (3) remain path cells, until the chain of blocks is absorbed into a single block by biconnecting it higher up in the hierarchy.  $\square$

Theorem 4 enables us to use  $\Gamma'$  for solving a variety of problems pertaining to biconnectivity.



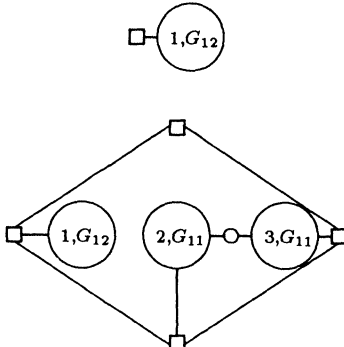
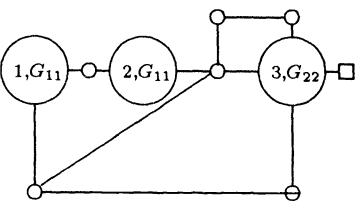
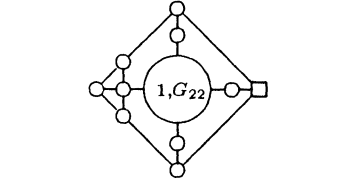
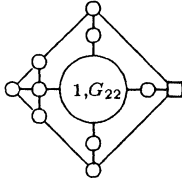
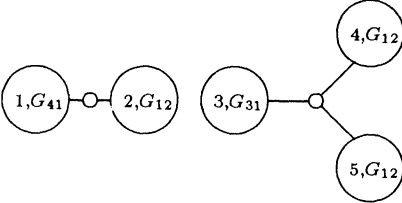
i	$\Gamma'$	cell created in step no.
1	$G_{11}$ : 	4
	$G_{12}$ : 	
2	<u><math>G_{21}</math></u> : 	4
	$G_{22}$ : 	
3	<u><math>G_{31}</math></u> : 	2
	$G_{41}$ : 	
4	$G_{42}$ : 	2
	column 4	

FIG. 9. Branch cells are underscored; there are no path cells.

Consider the following construction problem: *Given an edge  $e$  in  $E(\Gamma)$ , produce its biconnected component.* We can solve this problem by first translating the pathname of  $e$  into  $\Gamma'$ . Then we look for the root of the block containing  $e$ . To this end we scan the pathname of  $e$  right-to-left until we find a branch cell. Then we go left-to-right in the pathname until we find a cell that is a block cell. The resulting node  $x$  is the root of the block containing  $e$ . To produce the adjacency structure we just expand the subtree  $T_x$ , but whenever we find a nonterminal whose type is a branch cell we do not expand it. This procedure may take a lot of time if much output has to be produced, but it only needs working space  $O(m+n)$ , because of Theorem 4(b). A hierarchical representation of the block containing  $e$  can be obtained by first finding the root of the block as above and then scanning bottom-up through the BU-table selecting the

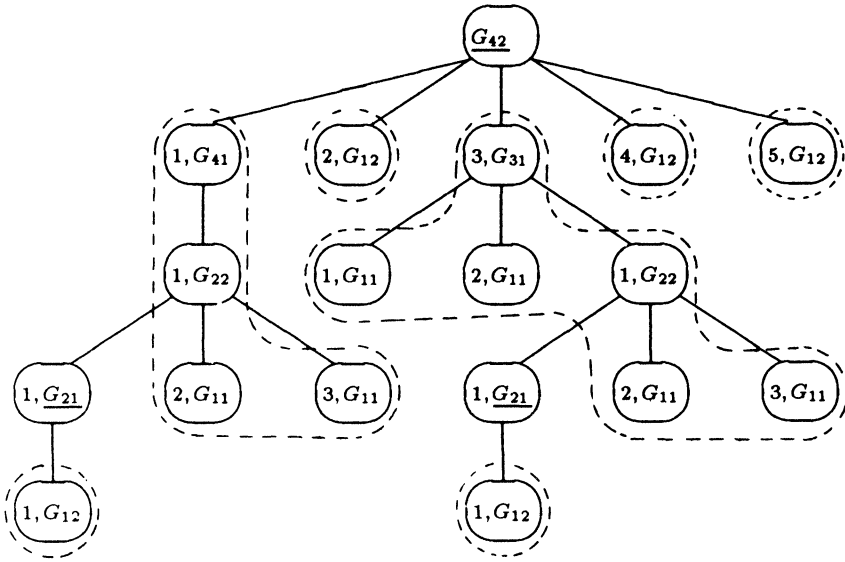


FIG. 10

rows of the table describing cells in the subtree of  $T'$  rooted at  $x$ . Here we cut off nodes representing branch cells. The whole process takes time  $O(m + n)$ . Other construction problems can be solved similarly.

The query Q2 ( $e_1, e_2$ ): *Given two edges  $e_1 \neq e_2$  in  $E(\Gamma)$  are they on a common simple cycle?* can be answered simply by comparing the pathnames for the roots of the blocks containing  $e_1$  and  $e_2$ . If they are identical the answer to the query is yes.

In order to answer the query Q3 ( $v$ ): *Given a vertex  $v$  in  $E(\Gamma)$ , is  $v$  an articulation point?* we augment  $\Gamma'$  some more. We mark a vertex in  $G_i$ , respectively,  $\tilde{G}_i, G_i^b$  if some deletions occur at this vertex during some execution of Step (2). Vertices retain their marks when some copy of  $G_j^b$  is substituted into some  $\tilde{G}_i$ . Now, obviously a vertex is an articulation point if the vertex denoted by the last symbol in the pathname of  $v$  is marked.

The only other vertices in  $E(\Gamma)$  that can be articulation points are vertices that are relevant articulation points on some path just before collapsing it in some  $G_i$ , such that the path does not get biconnected higher up in the hierarchy tree.

The pathnames of such vertices have the property that as we scan the pathname backwards we find a sequence of path cells followed by a branch cell. Thus query Q3 ( $v$ ) can also be answered in time  $O(k)$ .

Other query problems can be solved in a similar manner. It should be noted that an appropriate modification of the above construction yields a modified hierarchy on which connectivity and biconnectivity problems can be answered efficiently.

**5. Strong connectivity.** In this section we remark on how to hierarchically answer the question: *“Is  $E(\Gamma)$  strongly connected?”* given a hierarchical directed graph  $\Gamma$ .

The burning process is quite straightforward.  $G_i^b$  is just the reduction of the transitive closure of  $\tilde{G}_i$  to the pins of  $G_i$ . If  $\tilde{G}_i$  will never yield a strong connected result in any environment, we add an isolated vertex to  $G_i^b$ . (If  $\tilde{G}_i$  does not have pins we need two such vertices.) There are two problems with this burning process. First, it does not take linear time, but in general it takes time  $O(n^{2.39} \dots)$  (as of today

[CW86]). Second it may produce dense graphs with  $O(p_i^2)$  edges. Thus the substitution of the  $G_j^b$  into  $G_i$  to yield  $\tilde{G}_i$  may take  $\Omega(n_i^3)$  time. The resulting graph (after deleting multiple edges) may have size  $\Omega(n_i^2)$ . Thus filling the BU-table in general takes time  $O(\sum_{i=1}^k n_i^3)$ . If the hierarchy is *efficient* in the sense that there are many small cells, this can still be close to linear. In general, we do not know how to achieve a linear time hierarchical strong connectivity test. Indeed the following argument shows that we will probably not achieve such a test with the bottom-up method.

**THEOREM 5.** *There are at least  $2^{p^{2/4}}$  different equivalence classes of replaceable graph with  $p$  pins with respect to strong connectivity.*

*Proof.* The  $2^{p^{2/4}}$  different burnt graphs that can be produced by the above strong connectivity burner are not replaceable pairwise.  $\square$

Thus any strong connectivity burner has to produce at least  $2^{p^{2/4}}$  different burnt graphs with  $p$  pins. Since each graph is unambiguously described by its adjacency structure, and this structure has size  $O((m+n) \log n)$  for a graph with  $m$  edges and  $n$  vertices in the logarithmic cost measure, we must have  $m+n = \Omega(p^2/\log p)$ . This can only yield strong connectivity tests with time  $O(\sum_{i=1}^k (n_i^3/\log n_i))$ . We could circumvent this argument by labeling graphs appropriately and thus increasing their variety in an efficient way. However, we know of no helpful scheme for doing so.

There is another problem with strong connectivity. We do not know how to construct a hierarchical graph  $\Gamma'$  explicitly representing the strong components of  $\Gamma$ . We can identify the strong components in  $\tilde{G}_i$  during the burning process and provide special cells for them. But  $G_i^b$  does not specify how to use these cells later on. On the other hand, any burnt graph that might provide explicit locations for nonterminals whose types are these cells seems to become very large. Having no  $\Gamma'$  makes it more difficult to solve query problems pertaining to strong connectivity. We can still efficiently solve construction problems such as the following: *For each vertex in  $E(\Gamma)$  give the number of its strong component.* This is done by first generating the BU-table and then performing a preorder traversal on the hierarchy tree. When visiting the root all component numbers for vertices belonging to the root are output. Then the sons of the root are processed recursively. When a node  $x$ , say, of type  $G_i$ , in  $T$  is processed the component numbers for its pins are known. We connect all pins in the same component up into a circuit in  $\tilde{G}_i$ . Then we find the strong components resulting graph. The vertices belonging to  $x$  that are in a component with pins receive the number of that component. The other components are numbered with new numbers. The traversal takes a long time but only uses space  $O(m+n)$ .

The query Q4 ( $v, w$ ): *Given two vertices  $v, w$  in  $T$ , are  $v, w$  strongly connected?* can be answered by doing the above preorder traversal just on the subtree of  $T$  consisting of the paths from the two nodes to which  $v, w$  belong to the root. This takes time  $O(\sum_{i=1}^k n_i^2)$  since a strong connectivity check on a potentially dense graph must be performed on each node in the subtree.

How to improve on these algorithms is an open question.

**6. Conclusions, further results, and open problems.** We introduced a model for the succinct hierarchical representation of graphs that is widely applicable in areas of engineering, e.g., in CAD for integrated circuits. The model is a strongly restricted version of a context-free grammar that generates exactly one word. We efficiently solved connectivity problems on this model. For decision problems we achieved solutions with linear (connectivity, biconnectivity), respectively, cubic (strong connectivity) running time in the size of the hierarchical description. For construction and query problems we achieved large savings of working space and preprocessing time.

We also generated short hierarchical descriptions for the output of construction problems.

The graph problems we discussed are all relevant to integrated circuit design. Connectivity tests are used in electrical checking; biconnectivity tests occur in testability analysis. The results on strong connectivity generalize to the single source shortest path problem which is at the basis of many one-dimensional compaction algorithms [Le82], [U184] and also occurs in wire-routing applications. The hierarchical compaction algorithm has been implemented [Ki84]. Reference [Le82] contains some results showing that it is difficult in general to construct short hierarchical descriptions of the shortest path tree.

Reference [HS84] presents a hierarchical compaction algorithm with low page fault complexity. This algorithm processes the hierarchical description of the graph top-down, and uses  $O(N)$  space. The definition of hierarchy is a little different in [HS84]. There a graph is only partitioned into a collection of cells without the notion of a hierarchy tree. Here we exploit the presence of a hierarchy tree with many identical subtrees to reduce the space requirement of the hierarchical compaction.

Another application of the hierarchical strong connectivity algorithm is the solution of hierarchical linear systems, as occur in finite element modeling of mechanical structures [LW87].

A hierarchical solution to the minimum spanning forest problem in almost-linear time is given in [Le87]. The minimum spanning forest problem is relevant in wire routing for integrated circuit layouts. Reference [Le87] solves construction and query problems without resorting to a short hierarchical description of the minimum spanning tree. But such a description along the lines presented here is also possible in that case. The complexities of the solutions to the construction and query problems are the same as those given in §§ 3 and 4 of this paper. Reference [Le86b] discusses a linear time hierarchical planarity test. Reference [Le86a] gives an overview of the application of hierarchical processing in CAD for VLSI design.

There are also problems that cannot be solved hierarchically using this model. A thorough discussion is given in [LW87]. There it is also shown that the complexity of the hierarchical version of a graph problem cannot in general be deduced from the complexity of its nonhierarchical version. Two examples of hard problems are the circuit value problem and the network flow problem. Both become PSPACE complete in their hierarchical version. This is unfortunate as the circuit value problem is at the basis of most simulation algorithms for integrated circuits.

All hierarchical solutions to graph problems mentioned in this paper have the property that they use a nonhierarchical algorithm for the same graph problem as a subroutine. Thus they do not depend on a specific nonhierarchical solution method; any nonhierarchical algorithm will do. This is important for practicality, since often asymptotically worse algorithms are superior in practice for small graphs, and the subcells can be expected to be small. For this reason the BU-method is not only a tool for proving asymptotic results on hierarchical graphs, but it should serve as a powerful basis for bringing hierarchical graph algorithms on the computer.

We believe that the bottom-up method introduced in this paper is a powerful method for speeding up the processing of hierarchically defined graphs such as arise in many areas of application, notably in VLSI design. The hierarchical graph model is still rather restricted, and extensions should be possible that are useful in several areas of application while preserving the efficient solvability of a large number of important graph problems. The hierarchical graph model discussed in this paper is used in the HILL (Hierarchical Layout Language) system [LM84].

## REFERENCES

- [BOW83] J. L. BENTLEY, T. OTTMANN, AND P. WIDMAYER, *The complexity of manipulating hierarchically defined sets of rectangles*, Adv. in Comput. Res., 1 (1983), pp. 127–158.
- [CW86] D. COPPERSMITH AND S. WINOGRAD, *Matrix multiplication via arithmetic progressions*, in Proc. 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 1–6.
- [Ga82] H. GALPERIN, *Succinct representation of graphs*, Ph.D. thesis, Department of Electrical Engrg. and Computer Sci., Princeton University, Princeton, NJ, 1982.
- [GJ79] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA, 1979.
- [GW83] H. GALPERIN AND A. WIGDERSON, *Succinct representations of graphs*, Inform. and Control, 56 (1983), pp. 183–198.
- [Ha69] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.
- [HS84] M. A. HUANG AND K. STEIGLITZ, *A hierarchical compaction algorithm with low page fault complexity*, in Proc. MIT-Conference on Advanced Research in VLSI, 1984, pp. 203–212.
- [Ki84] C. KINGSLEY, *A hierarchical error-tolerant compacter*, in Proc. 21st Design Automation Conference, 1984, pp. 126–132.
- [Le82] T. LENGAUER, *The complexity of compacting hierarchically specified layouts of integrated circuits*, in Proc. 23rd Annual IEEE Symposium on the Foundation of Computer Science, 1982, pp. 358–368.
- [LM84] T. LENGAUER AND K. MEHLHORN, *The HILL system: A design environment for the hierarchical specification, compaction, and simulation of integrated circuit layouts*, in Proc. MIT-Conference on Advanced Research in VLSI, P. Penfield Jr., ed., Artech House Company, 1984, pp. 139–149.
- [Le86a] T. LENGAUER, *Exploiting hierarchy in VLSI design*, in Proc. Aegean Workshop on Computing, F. Makedon et al., eds., Lecture Notes in Computer Science 227, Springer-Verlag, Berlin, New York, 180–193.
- [Le86b] ———, *Hierarchical planarity testing algorithms*, in Proc. ICALP 86, L. Kott, ed., Lecture Notes in Computer Science 226, Springer-Verlag, Berlin, New York, 1986, pp. 215–225.
- [Le87] ———, *Efficient algorithms for finding minimum spanning forests in hierarchically defined graphs*, J. Algorithms, 8 (1987), pp. 260–284.
- [LW87] T. LENGAUER AND K. WAGNER, *The correlation between the complexities of the non-hierarchical and hierarchical versions of graph problems*, in Proc. STACS 87, F. J. Brandenburg, G. Vidal-Naquet, and M. Wirsing, eds., Lecture Notes in Computer Science 247, Springer-Verlag, Berlin, New York, 1987, pp. 100–113.
- [MC80] C. A. MEAD AND L. CONWAY, *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA, 1980.
- [SI82] A. O. SLISENKO, *Context-free grammars as a tool for describing polynomial-time subclasses of hard problems*, Inform. Process. Lett., 14 (1982), pp. 52–56.
- [Ta72] R. E. TARJAN, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.
- [Ul84] J. D. ULLMAN, *Algorithm aspects of VLSI*, Computer Science Press, Rockville, MD, 1984.
- [Wa84] K. WAGNER, *The complexity of problems concerning graphs with regularities*, Tech. Report 84/52, Sektion Mathematik, Universität Jena, German Democratic Republic, 1984.

## A FAST PARALLEL ALGORITHM FOR DETERMINING ALL ROOTS OF A POLYNOMIAL WITH REAL ROOTS\*

MICHAEL BEN-OR<sup>†</sup>, EPHRAIM FEIG<sup>‡</sup>, DEXTER KOZEN<sup>§</sup>, AND PRASOON TIWARI<sup>¶</sup>

**Abstract.** Given a polynomial  $p(z)$  of degree  $n$  with  $m$  bit integer coefficients and an integer  $\mu$ , the problem of determining all its roots with error less than  $2^{-\mu}$  is considered. It is shown that this problem is in the class NC if  $p(z)$  has all real roots. Some very interesting properties of a Sturm sequence of a polynomial with distinct real roots are proved and used in the design of a fast parallel algorithm for this problem. Using Newton identities and a novel numerical integration scheme for evaluating a contour integral to high precision, this algorithm determines good approximations to the linear factors of  $p(z)$ .

**Key words.** parallel algorithms, polynomial, roots, Sturm sequences, numerical integration, root separation, zeros

**AMS(MOS) subject classifications.** 12D10, 68Q99

**1. Introduction.** Determining the set of all roots of a polynomial is a classical problem with applications in many branches of engineering. Although many sequential algorithms have been designed for this problem (see, for example, [H74] and [S82]), to date no fast parallel algorithm is known. In this paper we show that this problem is in NC if all roots of the given polynomial are real. Moreover, the problem of determining if all the roots of a given polynomial are real is also in NC. The best previous result for this root-finding problem appears in [P85] and has worst-case running time  $O(n \log n)$  using  $n$  processors. Here  $n$  is the degree of the given polynomial and each processor is capable of performing arithmetic operations on operands having polynomially many bits in one step. Our results achieve considerable improvement in the parallel running time at the expense of polynomially many processors.

We use an exclusive read exclusive write PRAM (parallel random access machine) as the model of parallel computation [FW78]. In addition, we assume that a processor can perform arithmetic operations  $+$ ,  $-$ ,  $\times$ , and  $/$  in time polylog in the length (of the binary representation) of the operands.

The Sturm sequences of a polynomial were defined by C. Sturm in 1829 and have been studied for over 150 years [BP60]. We prove some very interesting properties of a Sturm sequence of a polynomial with distinct real roots and use these properties in the design of our algorithm. In particular, these properties imply that we can quickly compute a point on the real line such that at least a fraction of the roots of the polynomial lie on either side of the point.

We factor the given polynomial into its approximate linear factors, and hence approximately determine all its roots. This approximate factorization is achieved by recursively factoring the given polynomial into two approximate factors of almost

---

\* Received by the editors February 10, 1986; accepted for publication (in revised form) November 18, 1987. This work was begun in the summer of 1985 while the authors were at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York. A preliminary version of this paper appeared in the Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, Berkeley, California, May 28-30, 1986, © 1986 Association for Computing Machinery, Inc.

<sup>†</sup> Hebrew University, Jerusalem, Israel.

<sup>‡</sup> IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.

<sup>§</sup> Department of Computer Science, Cornell University, Ithaca, New York 14853.

<sup>¶</sup> Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801. Present address, IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598. The work of this author was supported by an IBM Graduate Fellowship and by the Office of Naval Research under contract N00014-85-K-0570.



equal degree. These two factors are in turn obtained by numerically evaluating a contour integral and then using the Newton identities.

Finally, in the last section, we also prove that the problem of determining all real roots in the presence of complex roots is in NC if and only if the problem of determining all roots is in NC.

The numerical integration scheme is based on a suggestion by Franco Preparata. This work has been inspired by the work of Schönhage [S82], and complements recent work of the first and the third authors on algebraic cell decomposition of  $\mathbf{R}^d$  [BKR84], [KY85]. The algorithm presented here also appears in [T86].

**2. The basic strategy.** In this section we give an overview of our algorithm for simultaneously determining all roots of a polynomial  $h(z)$ , which has distinct real roots. The major steps of the algorithm are given below.

#### ALGORITHM ROOTS.

**Input.** A polynomial  $h(z)$  of degree  $n$  having  $m$  bit integer coefficients, distinct real roots, and an error tolerance  $\mu$ .

**Output.** Approximations  $\tilde{z}_1, \tilde{z}_2, \dots, \tilde{z}_n$  to the roots  $z_1, z_2, \dots, z_n$  of  $h(z)$  such that  $|z_i - \tilde{z}_i| < 2^{-\mu}$ .

#### Method.

*Step 1.* Divide  $h(z)$  by its leading coefficient. Let  $p(z)$  be the resulting polynomial. The leading coefficient of  $p(z)$  is 1.

*Step 2.* Factor  $p(z)$  recursively in the following manner until all monic linear factors are found.

*Step 2(a).* Find a point  $w$  that separates the roots of  $p(z)$  into two sets  $L$  and  $R$ , those to the left and to the right of  $w$ , respectively, each containing between  $\frac{1}{4}$  and  $\frac{3}{4}$  of all roots of  $p(z)$ . In addition,  $w$  is not too close to any root of  $p(z)$ .

*Step 2(b).* Using a numerically evaluated contour integral and the Newton identities, determine approximations to the two monic factors  $p_1(z)$  and  $p_2(z)$  of  $p(z)$  with roots  $L$  and  $R$ , respectively.

In case the given polynomial has repeated roots, well-known methods (see, for example, [vG83a]) can be used to reduce the problem to that of determining the roots of a polynomial with distinct roots. An alternate method of performing this reduction appears in [T86]. This method has the advantage that it only uses integer addition and multiplication.

In order to show that ROOTS can be implemented in  $\log^{O(1)}(m+n+\mu)$  steps using  $(m+n+\mu)^{O(1)}$  processors, we must show that: (i) ROOTS can be implemented in  $\log^{O(1)}(m+n+\mu)$  steps using  $(m+n+\mu)^{O(1)}$  processors on a PRAM, where each processor is also capable of performing a real arithmetic operation in one step; and (ii) the operands of each (real) arithmetic operation can be expressed using  $(m+n+\mu)^{O(1)}$  bits.

It is not difficult to check that all real numbers computed by our algorithm are bounded in absolute value by  $2^{(m+n+\mu)^{O(1)}}$ . Therefore, in order to prove assertion (ii) of the last paragraph, we show that all these real numbers need not be computed exactly, but that we can approximate them to within  $2^{-(m+n+\mu)^{O(1)}}$  of their correct value and still ensure a correct output.

First, following Schönhage [S82], we show that it is sufficient to compute the coefficients of all polynomials constructed during an execution of ROOTS to within  $2^{-(m+n+\mu)^{O(1)}}$  of their correct value. The following theorem will be useful in proving this fact.

**THEOREM 2.1** [C29], [H70]. *Let  $h(z)$  be a polynomial of degree  $n$  with  $m$  bit integer coefficients. Then all roots of  $h(z)$  are less than  $2^m$  in absolute value.*

Since the roots of  $p(z)$  are the same as the roots of  $h(z)$ , all roots of  $p(z)$  are also bounded in absolute value by  $2^m$ . If  $h(z) = \sum_{i=0}^n h_i z^i$  has distinct roots  $z_1, z_2, \dots, z_n$ , then define the *height* of  $h(z)$ , denoted  $|h|$ , to be the sum  $\sum_{i=0}^n |h_i|$ . Also define the *minimum root separation*  $\Delta(h) = \min_{i \neq j} |z_i - z_j|$ . Unless stated otherwise, logarithms in this paper are to base 2.

**THEOREM 2.2** [M64]. *If  $h(z)$  is a univariate polynomial with integer coefficients, of degree  $n \geq 2$ , and distinct roots, then the minimum root separation  $\Delta(h)$  is at least  $\sqrt{3} n^{-(n+2)/2} |h|^{-(n-1)}$ .*

**COROLLARY 2.3.** *Let  $h(z)$  be a polynomial with  $m$  bit integer coefficients, degree  $n \geq 2$ , and distinct roots  $z_1, z_2, \dots, z_n$ , then  $\Delta = \min_{i \neq j} |z_i - z_j| > 2^{-\sigma}$ , where  $\sigma = 2n \log n + mn$ .*

We also need a bound on perturbation induced on the zeros of a polynomial as a result of perturbation of its coefficients. We will frequently use a perturbation bound which is a corollary to the following well-known theorem of Ostrowski.

**THEOREM 2.4** [H70]. *Let  $f(z) = z^n + a_1 z^{n-1} + \dots + a_n$  and  $g(z) = z^n + b_1 z^{n-1} + \dots + b_n$ . Let  $\gamma = 2 \max(|a_j|^{1/j}, |b_j|^{1/j})$ , and define  $\epsilon > 0$  by  $\epsilon^n = \sum_{j=1}^n |b_j - a_j| \gamma^{n-j}$ . Then the zeros  $z_j$  and  $w_k$  of  $f$  and  $g$ , respectively, can be ordered so that  $|z_j - w_j| < 2n\epsilon$ .*

**COROLLARY 2.5.** *Consider the polynomial  $p(z)$  of degree  $n$  with real coefficients, leading coefficient 1, and roots  $z_1, z_2, \dots, z_n$  such that  $|z_i| < 2^m$ . Let  $\tilde{p}(z)$  be an approximation to  $p(z)$  such that the corresponding coefficients of  $\tilde{p}(z)$  and  $p(z)$  differ by at most  $2^{-\pi}$ . Then there is an ordering of the roots  $\tilde{z}_i$  of  $\tilde{p}(z)$  such that  $|z_i - \tilde{z}_i| < n^2 2^{m+4-\pi/n}$ , for all  $i$ .*

**COROLLARY 2.6.** *If  $\pi > n(m + \mu + 2 \log n + 4)$ , then the roots of  $\tilde{p}_0(z)$  approximate the roots of  $p(z)$  with error less than  $2^{-\mu}$ .*

As a consequence of this corollary, all roots of  $\tilde{p}(z)$  are bounded in absolute value by  $2^{m+1}$  whenever  $\pi$  is chosen appropriately. The following corollary is important for establishing the correctness of our algorithm.

**COROLLARY 2.7.** *If  $p(z)$  has distinct real roots,  $\Delta(p) > 2^{-\sigma}$ ,  $\mu \geq \sigma + 2$ ,  $\pi > n(m + \mu + 2 \log n + 4)$ , and all coefficients of  $\tilde{p}(z)$  are real, then  $\tilde{p}(z)$  has distinct real roots.*

*Proof.* Since all coefficients of  $\tilde{p}(z)$  are real, its roots are either real or appear in complex conjugate pairs. Suppose  $\tilde{z}_i$  and  $\tilde{z}_j$  is a pair of complex conjugate roots of  $\tilde{p}(z)$ . By Corollary 2.6, we have  $|z_i - \tilde{z}_i| < 2^{-\mu}$ ,  $|z_j - \tilde{z}_j| < 2^{-\mu}$ , and therefore the imaginary part of each of  $\tilde{z}_i$  and  $\tilde{z}_j$  is less than  $2^{-\mu}$ . This implies that  $|\tilde{z}_i - \tilde{z}_j| < 2^{-\mu+1}$ . But,  $|z_i - z_j| > 2^{-\sigma}$  implies  $|\tilde{z}_i - \tilde{z}_j| > |z_i - z_j| - |\tilde{z}_i - z_i| - |z_j - \tilde{z}_j| > 2^{-\sigma} - 2^{-\mu+1}$ . For  $\mu \geq \sigma + 2$ , we have  $|\tilde{z}_i - \tilde{z}_j| > 2^{-\mu+1}$ —a contradiction. Hence, all roots  $\tilde{p}(z)$  are real. The fact that they are distinct can also be proved using a similar argument.  $\square$

In the following discussion we will assume that  $\mu \geq \sigma + 2$ . Also in view of Corollary 2.6, let  $\pi = n(m + \mu + 2 \log n + 4) + 1$ . We wish to determine approximate linear factors  $L_1, L_2, \dots, L_n$  of  $p(z)$  such that

$$(2.1) \quad |p(z) - L_1 L_2 \dots L_n| < 2^{-\pi}.$$

Then Corollary 2.6 would imply that the easily available zeros of  $L_i$  are sufficiently good approximation to the zeros of  $p(z)$ . However, in order to achieve  $2^{-\pi}$  precision in (2.1), we may need precision of more than  $\pi$  bits beyond the binary point in all intermediate computations.

Next we determine the precision required during the factorization in Step 2 in order to satisfy the above requirements. Suppose that the precision of  $\lambda$  bits beyond

the binary point is sufficient for all intermediate computations in order to finally compute the approximate linear factors which satisfy (2.1). We approximate the coefficients of  $p(z)$  to within  $2^{-\lambda}$  in order to obtain an approximation  $p_0(z)$  to  $p(z)$ . Then we factor  $p_0(z)$  recursively to obtain all linear factors  $L_i$ , always computing the factors to  $\lambda$  bit precision beyond the binary point. Choice of the precision  $\lambda$  ensures that at any stage of the recursive factorization of  $p(z)$ , we have a set of polynomials  $p_1(z), p_2(z), \dots, p_l(z)$  whose product is a good approximation to  $p(z)$ .

**LEMMA 2.8.** *Given a monic polynomial  $p(z)$  of degree  $n$ , all of whose roots are bounded in absolute value by  $2^m$ , suppose  $p_1(z), p_2(z), \dots, p_l(z)$  are monic polynomials such that  $g(z) = p(z) - p_1(z)p_2(z) \cdots p_l(z)$  is of degree at most  $n - 1$  and  $|g(z)| < 2^{-\pi}$ . Then each coefficient of  $p_i(z)$ ,  $1 \leq i \leq l$ , is less than  $2^{2n+mn}$  in absolute value and therefore  $|p_i| < (n + 1)2^{2n+nm}$ .*

*Proof.* Let  $\tilde{p}(z) = p_1(z)p_2(z) \cdots p_l(z)$ . By the comment after Corollary 2.6 and the choice of  $\pi$ , each root of  $\tilde{p}(z)$ , and hence each root of  $p_i(z)$  for  $1 \leq i \leq l$ , is less than  $2^{m+1}$ . Therefore, each coefficient of  $p_i(z)$  is less than  $2^n(2^{m+1})^n$ .  $\square$

Suppose  $p_i(z)$  is monic,  $\deg(p_i) > 1$ , and let  $p_j(z)$  and  $p_k(z)$  be the two monic factors of  $p_i(z)$  computed during its factorization. Recall that the coefficients of these factors are truncated beyond the  $\lambda$ th bit after the binary point. Then, we have the following.

**LEMMA 2.9.**  $|p_i(z) - p_j(z)p_k(z)| < (n + 1)^2 2^{1+2n+nm-\lambda}$ .

*Proof.* Observe that  $p_i(z) = (p_j(z) + \gamma(z))(p_k(z) + \xi(z))$  where each coefficient of  $\gamma(z)$  and  $\xi(z)$  is less than  $2^{-\lambda}$ . Then,  $|\gamma(z)|, |\xi(z)| < n2^{-\lambda}$ , and hence  $|p_i - p_jp_k| = |p_i\xi + p_k\gamma + \gamma\xi|$ , and the assertion in the statement of the lemma follows from Lemma 2.8.  $\square$

A corollary to the following theorem specifies the precision  $\lambda$  sufficient for the recursive factorization routine.

**THEOREM 2.10.** *Let  $p_1(z), p_2(z), \dots, p_l(z)$  be the known factors of  $p(z)$  at an intermediate stage such that  $|p - p_1p_2 \cdots p_l| < \theta$ ,  $\theta \leq 2^{-\pi}$ , and  $\deg(p_1) > 1$ . Suppose  $p_1(z)$  is approximately factored into  $p_j(z)$  and  $p_k(z)$  whose coefficients are determined only to  $\lambda$  bits beyond the binary point. Then,  $|p - p_jp_kp_2p_3 \cdots p_l| < \theta + (n + 1)^3 2^{1+4n+2mn-\lambda}$ .*

*Proof.*  $|p - p_jp_kp_2p_3 \cdots p_l| = |p - p_1p_2 \cdots p_l + p_1p_2 \cdots p_l - p_jp_kp_2p_3 \cdots p_l| < \theta + |p_2p_3 \cdots p_l| |p_1 - p_jp_k|$ . Application of Lemmas 2.8 and 2.9 proves the theorem.  $\square$

**COROLLARY 2.11.** *Let  $L_1(z), L_2(z), \dots, L_n(z)$  be the approximate factors of  $p(z)$  obtained by the recursive factorization using precision of  $\lambda \geq \pi + \log n$  bits beyond the binary point. Then,  $|p(z) - L_1L_2 \cdots L_n| < n2^{-\lambda} + (n + 1)^4 2^{1+4n+2mn-\lambda}$ .*

*Proof.* For the purpose of establishing this lemma, we can assume that our algorithm works as follows. To begin, we have a set of polynomials  $F = \{p_0(z)\}$  such that  $|p(z) - p_0(z)| < n2^{-\lambda}$ . Our algorithm can be viewed as selecting a polynomial of degree at least two, and replacing it with its approximate factors. By Theorem 2.10, each such operation increases the error between  $p(z)$  and the product of polynomials by at most  $(n + 1)^3 2^{1+4n+2mn-\lambda}$ .  $\square$

Equation (2.1) and Corollary 2.11 imply that  $\lambda = \pi + 2 + 4 \log(n + 1) + 4n + 2mn$  is sufficient precision in order to obtain the desired approximations to the roots of  $p(z)$ . Substituting for  $\pi$ , we get  $\lambda = O(n(\log n + m + \mu))$ .

**3. Computing the factors using the Newton identity.** Consider a polynomial  $p(z)$  of degree  $n$  having distinct real roots, leading coefficient 1, and all other coefficients having at most  $\lambda$  bits beyond the binary point. Let  $z_1 < z_2 < \dots < z_n$  be the distinct real zeros of  $p(z)$ ,  $|z_i| < 2^{m+1}$ ; and let  $p_1(z) = \sum_{i=0}^k c_i z^i$  be the factor of  $p(z)$  with roots

$z_1, z_2, \dots, z_k$ . In this section, we present a fast parallel algorithm for determining good approximations to coefficients  $c_i$ 's using good approximations to  $s_j = \sum_{i=1}^k z_i^j$ , for  $j=1, \dots, k$ . More precisely, we show that  $c_i$ 's can be computed correct to  $\lambda$  bits beyond the binary point, if approximations  $\tilde{s}_j$  to  $s_j$  are given such that  $|\tilde{s}_j - s_j| < 2^{-\tau}$ , where  $\tau$  is polynomially bounded. In the next section, we will present a fast parallel algorithm for computing  $\tilde{s}_j$  to the required precision.

We will use the well-known relation between the  $s_i$ 's and the  $c_i$ 's given by the Newton identity:

$$(3.1) \quad \begin{bmatrix} 1 & & & & & \\ s_1 & 2 & & & & \\ s_2 & s_1 & 3 & & & \\ s_3 & s_2 & s_1 & 4 & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ s_{k-1} & s_{k-2} & s_{k-3} & s_{k-4} & \dots & k \end{bmatrix} \begin{bmatrix} c_{k-1} \\ c_{k-2} \\ c_{k-3} \\ \vdots \\ c_0 \end{bmatrix} = - \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_k \end{bmatrix}.$$

Let  $\mathbf{M}$  denote this  $k \times k$  lower triangular matrix and let us denote this equation by  $\mathbf{M}\mathbf{c} = -\mathbf{s}$ . We compute  $\mathbf{M}^{-1}$  using Csanky's adaptation of Leverrier's method [C76]:

$$(3.2) \quad \mathbf{M}^{-1} = - \frac{\mathbf{M}^{k-1} + d_1\mathbf{M}^{k-2} + \dots + d_{k-1}\mathbf{I}}{d_k},$$

where  $d_i$  is the coefficient of  $x^{k-i}$  in  $\prod_{i=0}^k (x - i)$ . Since we use approximations  $\tilde{s}_i$  to  $s_i$  in these computations, the result may not be the exact value of  $c_j$  but an approximation  $\tilde{c}_j$ . In the rest of this section, our aim is to determine a reasonable upper bound on the deviation  $|c_j - \tilde{c}_j|$ . The following sequence of claims leads to the error estimate of Theorem 3.3 in the entries of the computed inverse in terms of the approximation error in the entries of the original matrix.

**LEMMA 3.1.** *Let  $A = (a_{ij})$  be a  $k \times k$  matrix, and suppose that  $\tilde{A}$  is given such that  $\tilde{A} = A + E$ , where  $E = (e_{ij})$  and  $|e_{ij}| \leq \varepsilon$ . Let  $A^m = (a_{ij}^{(m)})$ , and  $\alpha = \max\{1, \max_{ij}\{a_{ij}\}\}$ . Suppose  $\tilde{A}^m = (\tilde{a}_{ij}^{(m)})$  is the  $m$ th power of  $\tilde{A}$  obtained when all intermediate computations are carried out with error less than  $2^{-\kappa}$ . If  $2^{-\kappa+1} + 4k^2\varepsilon^2 < \varepsilon\alpha$  then  $\tilde{A}^m$ , for  $1 \leq m \leq k$ , can be computed such that  $|a_{ij}^{(m)} - \tilde{a}_{ij}^{(m)}| \leq (2^{q+1} - 1)(\alpha k)^{2^q - 1} \varepsilon$  where  $q = \lceil \log m \rceil$ .*

*Proof.* Let us arrange the computation of  $A^m$  as a binary tree  $T$  of height  $\lceil \log m \rceil$ . Each vertex of  $T$  is associated with a result matrix. The result associated with each leaf of  $T$  is  $A$ . The result associated with any other vertex is the product, computed using real arithmetic, of matrices associated with all its children. Observe that any vertex of height  $i$  is labeled with  $A^l$ , where  $l \leq 2^i$ . In this manner, the root of  $T$  is labeled with  $A^m$ .

We will bound the change in each of the entries of the result matrices if we replace  $A$  by  $\tilde{A}$  at the leaves, and use arithmetic operations with error at most  $2^{-\kappa}$  instead of real arithmetic. Observe that in order to derive a worst-case estimate, we can assume that all errors are additive, all entries of  $A$  are nonnegative, and that  $\tilde{A} = A + \varepsilon J$ , where  $J$  is a  $k \times k$  matrix of all 1's. Since our error estimate is a monotonically increasing function of  $m$ , we can bound the error in the entries of all result matrices at height  $q$  by the error bound for the corresponding entries of  $A^{2^q}$  and  $\tilde{A}^{2^q}$ . We will also use the fact that  $|a_{ij}^{(l)}| \leq \alpha^l k^{l-1}$ .

It is not difficult to check that  $|a_{ij}^{(2)} - \tilde{a}_{ij}^{(2)}| < k[2\varepsilon\alpha + \varepsilon^2 + 2^{-\kappa}] + k2^{-\kappa}$ . But  $2^{-\kappa+1} < \varepsilon[\alpha - 4k^2\varepsilon]$  implies that  $k[2\varepsilon\alpha + \varepsilon^2 + 2^{-\kappa+1}] < k[3\varepsilon\alpha]$ . Inductively, if  $|a_{ij}^{(2^{q-1})} - \tilde{a}_{ij}^{(2^{q-1})}| < k^{2^{q-1}-1}[(2^q - 1)\varepsilon\alpha^{2^{q-1}-1}]$ , then  $|a_{ij}^{(2^q)} - \tilde{a}_{ij}^{(2^q)}| < k^{2^q-1}[2(2^q - 1)\varepsilon\alpha^{2^q-1} + (2^q - 1)^2\varepsilon^2\alpha^{2^q-2} + 2^{-\kappa}] + k2^{-\kappa}$ . But  $2^{-\kappa+1} + 4k^2\varepsilon^2 < \varepsilon\alpha$  implies that  $[(2^q - 1)^2\varepsilon^2\alpha^{2^q-2} + 2^{-\kappa+1}] < \varepsilon\alpha^{2^q-1}$ , and the lemma follows.  $\square$

Returning to the problem of solving (3.1), we can state the following.

**COROLLARY 3.2.** *Given approximations  $\tilde{s}_i$  to  $s_i$  such that  $|s_i - \tilde{s}_i| < 2^{-\tau}$ ,  $\tau \geq 1$ , an approximation  $\tilde{\mathbf{R}} = (\tilde{r}_{ij})$  to  $\mathbf{R} = (r_{ij}) = \mathbf{M}^{-1}$  can be computed such that  $|r_{ij} - \tilde{r}_{ij}| \leq (2k)^{k+2} \{4k(\alpha k)^{2k} 2^{-\tau}\}$ , where  $\alpha = k2^{k+mk}$ .*

*Proof.* In (3.2),  $1 \leq |d_i| \leq (2k)^k$  and can be computed exactly. Let  $\mathbf{B} = (b_{ij})$  be the numerator of the right side of (3.2). Since  $|s_i| < k2^{k+mk} = \alpha$ ,  $|b_{ij}| \leq k(2k)^k (\alpha k)^k$ . Compute the approximations to the powers of  $\mathbf{M}$  using the algorithm suggested in Lemma 3.1, with  $\alpha = k2^{k+mk}$  and  $\kappa = \tau$ . These are then used in computing an approximation  $\tilde{\mathbf{B}} = (\tilde{b}_{ij})$  to  $\mathbf{B}$ . Then  $|b_{ij} - \tilde{b}_{ij}| \leq (2k)^{k+1} \{4k(\alpha k)^{2k} 2^{-\tau}\}$ , provided the scalar multiplication and addition in (3.2) are carried out to reasonable precision. Next, compute  $1/d_k$  to  $\tau$  bits beyond the binary point and then compute the product  $\tilde{\mathbf{R}} = (1/d_k)\tilde{\mathbf{B}}$ . Then  $|r_{ij} - \tilde{r}_{ij}| \leq (2k)^{k+2} \{4k(\alpha k)^{2k} 2^{-\tau}\}$ .  $\square$

**THEOREM 3.3.** *Given approximations  $\tilde{s}_i$  to  $s_i$  such that  $|s_i - \tilde{s}_i| < 2^{-\tau}$ , approximations  $\tilde{c}_i$ 's to the coefficients  $c_i$  can be correctly computed to  $\tau - 17mk^2$  bits.*

*Proof.* Approximate the vector  $\mathbf{c}$  by the product  $\tilde{\mathbf{R}}\tilde{\mathbf{s}}$ , using the matrix  $\tilde{\mathbf{R}}$  computed in Corollary 3.2.  $\square$

**COROLLARY 3.4.** *In order to compute the coefficients  $c_i$  correct to the  $\lambda$  bits beyond the binary point, it is sufficient to compute  $s_i$  correct to  $\tau = \lambda + 17mk^2$  bits.*

Substituting for  $\lambda$  from § 2, we get  $\tau = O(mn^2 + n\mu)$ . In the next section we describe a way to compute the  $s_i$ 's to sufficient accuracy by numerically evaluating a contour integral.

**4. Evaluating a contour integral to high precision.** Let  $p(z) = \sum_{i=0}^n p_i z^i$ ,  $p_n = 1$  be a polynomial of degree  $n$  with simple real roots  $z_1 < z_2 < z_3 < \dots < z_n$  such that  $|z_i| < 2^m + 1$ . Then  $|p_i| < 2^{2n+mn}$ . Suppose that at most  $\lambda$  bits are given for any  $p_i$  after the binary point. In addition, a point  $W$  with coordinates  $(w, 0)$  is given such that  $|w - z_j| > 2^{-\beta} = d$  for all  $1 \leq j \leq n$  and  $z_k < w < z_{k+1}$ . In order to compute  $s_l$  with error less than  $2^{-\tau}$  (see § 3), we wish to evaluate the following integral to  $\tau$  significant bits beyond the binary point:

$$\sum_{i=1}^k z_i^l = s_l = \frac{1}{2\pi i} \int_{\Gamma} \frac{z^l p'(z) dz}{p(z)}$$

Here  $\Gamma$  can be chosen as the boundary of the rectangle with vertices  $A, B, C$ , and  $D$ , traversed in the counterclockwise direction, where  $A, B, C$ , and  $D$  are points in the complex plane with coordinates  $(w, -2^{m+1}), (w, 2^{m+1}), (-2^{m+1}, 2^{m+1})$ , and  $(-2^{m+1}, -2^{m+1})$ , respectively (see Fig. 1).

For convenience, the contour  $\Gamma$  is partitioned into four segments  $AB, BC, CD$ , and  $DA$ . This gives the following expression for the contour integral:

$$(4.1) \quad \int_{\Gamma} = \int_A^B + \int_B^C + \int_C^D + \int_D^A$$

We will only discuss a scheme for computing the first term on the right-hand side of (4.1). A similar scheme can be used to compute the third integral on the right-hand side of (4.1). The second and the fourth integrals are much simpler to compute. These are discussed towards the end of this section.

The integral  $\int_A^B$  can be further partitioned into two symmetric parts about the point  $W$  with coordinates  $(w, 0)$  in the complex plane, i.e.,  $\int_A^B = \int_A^W + \int_W^B$ . Let  $I_{WB}$  denote the integral  $\int_W^B (z^l p'(z) dz / p(z))$  and let  $\tilde{I}_{WB}$  denote the computed approximation to  $I_{WB}$ . We require that the error  $|I_{WB} - \tilde{I}_{WB}|$  be bounded by  $2^{-(\tau+3)}$  so that the total error in the right-hand side of (4.1) does not exceed  $2^{-\tau}$ . In [S82], Schönhage uses a numerical

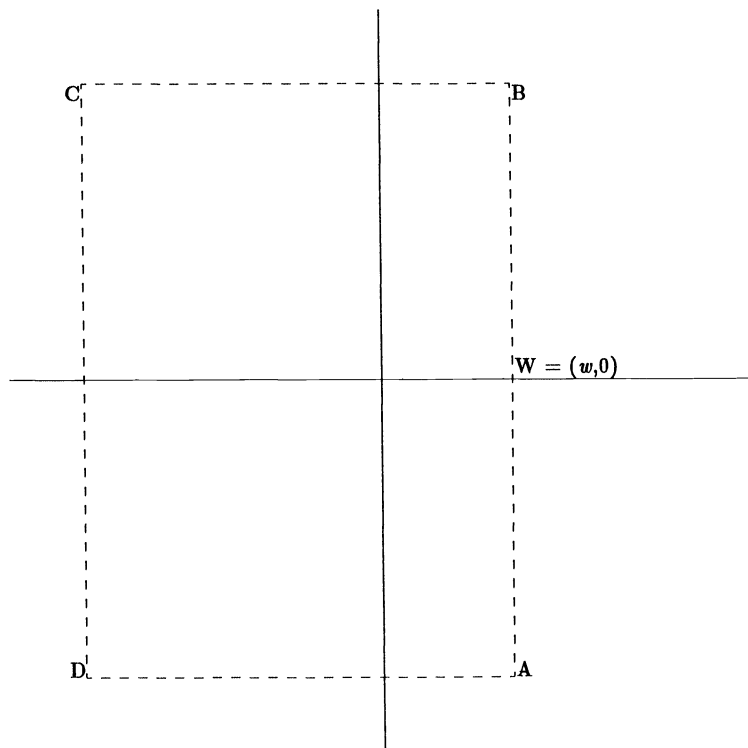


FIG. 1. The contour of integration.

integration scheme which is based on properties of the sums of the roots of unity and requires that all poles of the integrand be relatively far from the contour of integration (which is a circle). However, we cannot use Schönhage’s scheme because some poles of the integrand under consideration may be too close to the contour of integration  $\Gamma$ . The following scheme for evaluating  $I_{WB}$  is based on a suggestion by Franco Preparata to “increase the spacing of samples on  $C$  geometrically as one goes away from the real axis.”

Define two sequences  $y_j$  and  $a_j$  for  $j = -1, 0, 1, \dots, q$ , where  $q = \lceil \log_{3/2} 2^{m+1} - \log_{3/2} (d/2) \rceil$ , as follows:  $y_{-1} = 0, y_0 = (d/2), y_1 = (3/2)(d/2), \dots, y_j = (3/2)^j (d/2), \dots, y_{q-1} = (3/2)^{q-1} (d/2)$ ; and  $a_j = w + iy_j$  for all  $-1 \leq j \leq q-1$  (here  $i = \sqrt{-1}$ ). In addition, let  $y_q = 2^{m+1}$  and  $a_q = w + iy_q$ . Observe that, in the complex plane, the points  $a_j$  are all on the segment from  $W$  to  $B$ .

Let  $f(z) = z^l p'(z)/p(z)$ . The Taylor series of  $f(z)$  about  $a_j$  is given by

$$f(z) = f(a_j) + f'(a_j)(z - a_j) + \dots + \frac{f^{(t)}(a_j)}{t!} (z - a_j)^t + R_t(z),$$

where

$$|R_t(z)| \leq \frac{M y_j}{y_j - r} \left( \frac{r}{y_j} \right)^{t+1}, \quad r = |z - a_j| < y_j$$

[C48], and  $M$  is the maximum value of  $f(z)$  in the closed disc of radius  $y_j$  centered

at  $a_j$ . Observe that the radius of convergence of the above series is greater than  $y_j = (3/2)^j(d/2)$ , for  $j \geq 0$  and is  $> d$  for  $j = -1$ . Let  $T_j(z)$  be the series consisting of the first  $t + 1$  terms of the Taylor series of  $f(z)$  about  $a_j$ . We would approximate the integral  $I_{WB}$  as follows:

$$I_{WB} = \int_{a_{-1}}^{a_q} f(z) dz = \sum_{j=-1}^{q-1} \int_{a_j}^{a_{j+1}} f(z) dz \cong \sum_{j=-1}^{q-1} \int_{a_j}^{a_{j+1}} T_j(z) dz = \tilde{I}_{WB}.$$

Then,

$$(4.2) \quad \tilde{I}_{WB} = \sum_{j=-1}^{q-1} \sum_{u=0}^t \frac{\alpha_{uj}}{u+1} (a_{j+1}^{u+1} - a_j^{u+1}),$$

where  $\alpha_{uj}$  is the coefficient of  $z^u$  in  $T_j(z)$ . In order to estimate the error  $|I_{WB} - \tilde{I}_{WB}|$ , we need to find an upper estimate on  $M$  in the vicinity of the segment  $WB$ :

$$\begin{aligned} M &= \max_j \max_{z: |z-a_j| \leq y_j} \left| \frac{z'p'(z)}{p(z)} \right| \leq n \max_{z, z_i} \left| \frac{z'}{z-z_i} \right| \\ &\leq \frac{n2^{2(m+1)l}}{d} < 2^{2ml+2l+\beta+\log n}. \end{aligned}$$

Consequently, for  $z$  belonging on the segment from  $a_j$  to  $a_{j+1}$

$$|R_t(z)| \leq \frac{My_j}{y_j - r} \left(\frac{r}{y_j}\right)^{t+1} < 2^{2ml+2l+\beta+\log n-t}.$$

Suppose the powers of  $a_j$  are computed exactly (note that this requires only polynomial precision) while evaluating the right-hand side of (4.2), and approximations  $\tilde{\alpha}_{uj}$  to  $\alpha_{uj}$ , such that  $|\alpha_{uj} - \tilde{\alpha}_{uj}| < 2^{-\nu}$ , are used in computing  $\tilde{I}_{WB}$ . Then,

$$|I_{WB} - \tilde{I}_{WB}| \leq \int_W^B R_t(z) dz + 2^{-\nu} \sum_{i=1}^{t+1} 2^{i(m+1)} < 2^{2ml+2l+\beta+\log n+m+1-t} + 2^{(m+1)(t+2)-\nu}.$$

In order to ensure  $|I_{WB} - \tilde{I}_{WB}| \leq 2^{-(\tau+3)}$ , it is sufficient to require that  $2^{2ml+2l+\beta+\log n+m+1-t} < 2^{-(\tau+4)}$  and  $2^{(m+1)(t+2)-\nu} < 2^{-(\tau+4)}$ . Hence  $t = \tau + O(mn + \beta)$  and  $\nu = O(m(\tau + mn + \beta))$ . Substituting for  $\tau$ , we get  $t = O(mn^2 \log n + n\mu + \beta)$  and  $\nu = O(m^2 n^2 \log n + mn\mu + m\beta)$ .

The integral  $\int_A^W$  can also be evaluated in a similar manner. This completes the evaluation of the integral  $\int_A^B$ . The integral  $\int_C^D$  in (4.1) can also be evaluated in a similar manner. The remaining two integrals  $\int_B^C$  and  $\int_D^A$  of (4.1) can also be computed using the above method. However, in case of these integrals, we can get away with using only five sample points. For example, in order to compute  $\int_B^C$ , we need to consider only the five sampling points  $(w, 2^{m+1})$ ,  $(2^m, 2^{m+1})$ ,  $(0, 2^{m+1})$ ,  $(-2^m, 2^{m+1})$ , and  $(-2^{m+1}, 2^{m+1})$ , if  $w > 2^m$ .

The Taylor coefficients required by this integration scheme can be determined quickly using an algorithm due to von zur Gathen [vG83b]. An alternate method of computing these coefficients based on Kung's [K74] power series inversion method, with complete error analysis, appears in [T86].

**5. Separating the roots.** The integration scheme described in the last section requires a point  $w$  such that  $|w - z_i| > 2^{-\beta}$ , where  $z_i, i = 1, 2, \dots, n$  are the distinct real roots of  $p(z)$ , and  $z_l < w < z_{l+1}$ . Moreover, for algorithm ROOTS to have only  $O(\log n)$  depth of recursion, we require that  $n/4 \leq l \leq 3n/4$ . In the following paragraphs, we describe a fast algorithm to determine one such point  $w$ .

Given the polynomial  $p(z)$ , let us define the sequence of polynomials  $f_0(z), f_1(z), \dots, f_n(z)$  such that  $f_0(z) = p(z), f_1(z) = p'(z)$ , and  $f_{i+2}(z)$  is defined as the negative of the remainder obtained on dividing  $f_i(z)$  by  $f_{i+1}(z)$ :

$$f_i = q_i(z)f_{i+1}(z) - f_{i+2}(z).$$

It is well known (see, for example, [H71]) that  $f_i$ 's form a Sturm sequence for the polynomial  $p(z)$  over the interval  $(-\infty, \infty)$ . Fast parallel algorithms for computing  $f_i$  are implicit in [C66] and [C67]. We will show that if  $p(z)$  has distinct real roots, then  $f_i(z)$ 's have degree  $n - i$ , and therefore  $q_i(z)$ 's are linear in  $z$ . Let  $r_i$  be the real root of  $q_i$ . In this section we prove that  $w_i$  can be chosen close to one of the  $r_i$ 's. A fast parallel algorithm for determining  $r_i$ 's appears in [vG83a].

Let  $z_{ij}, j = 1, 2, \dots$ , be the roots of  $f_i$  such that  $z_{ij} \leq z_{i,j+1}$ . Let  $R_i$  be the set of roots of the polynomial  $f_i$  and let  $S_i, i > 1$ , be the (multi) set  $R_i \cup \{r_{i-2}\}$ . If  $A$  and  $B$  are two subsets of the set of real numbers, then we say that  $A$  (strictly) interleaves  $B$  if the (open) closed interval defined by any two points of  $B$  contains a point  $A$ . The following properties of the sets  $R_i$  and  $S_i$  play an important role in locating  $w$ .

**LEMMA 5.1.** *If  $f_0$  has distinct real roots, then  $R_{i+1}$  strictly interleaves  $R_i$ , for  $i = 0, 1, \dots, n - 2$ .*

*Proof.* We prove the assertion in the lemma by induction on  $i$ .

*Induction Hypothesis.*  $R_{i+1}$  strictly interleaves  $R_i$ , and  $R_i$  has  $n - i$  distinct elements.

*Basis.* For  $i = 0$ , the hypothesis reduces to Rolle's theorem [M66]. In case the polynomial has distinct real roots, Rolle's theorem implies that roots of the derivative strictly interleave the roots of the polynomial.

*Induction Step.* Suppose the hypothesis holds for  $0 \leq i \leq k - 1$ . Then  $z_{k-1,j}$  and  $z_{k,j}$  are all real and distinct. Let  $z_{k-1,1} < z_{k,1} < z_{k-1,2} < z_{k,2} < \dots < z_{k-1,n-k} < z_{k,n-k} < z_{k-1,n-k+1}$ . From the defining equation:

$$f_{k-1} = q_{k-1}f_k - f_{k+1}$$

we conclude that  $f_{k+1}(z_{kj}) = -f_{k-1}(z_{kj})$ , for  $1 \leq j \leq n - k$ . But since  $R_k$  strictly interleave  $R_{k-1}$ ,  $\text{sign}\{f_{k-1}(z_{kj})\} = -\text{sign}\{f_{k-1}(z_{k,j+1})\}$ . Therefore,  $\text{sign}\{f_{k+1}(z_{kj})\} = -\text{sign}\{f_{k+1}(z_{k,j+1})\}$  and  $f_{k+1}$  must have a zero between  $z_{kj}$  and  $z_{k,j+1}$ . This argument, along with the degree constraint on  $f_{k+1}(z)$ , proves the assertion for  $i = k$ .  $\square$

This lemma implies that  $\text{deg}(f_i(z)) = n - i$ . Therefore, it also implies that each  $q_i(z)$  is linear in  $z$ .

**LEMMA 5.2.** *If  $f_0$  has distinct real roots and  $z, z'$  are two adjacent roots of  $f_{i-1}, 1 \leq i < n$ , then either  $r_{i-1} \in [z, z']$  or there is a root of  $f_{i+1}$  in the open interval  $(z, z')$ .*

*Proof.* Consider the equation

$$(5.1) \quad \frac{f_{i+1}}{f_{i-1}} = q_{i-1} \frac{f_i}{f_{i-1}} - 1.$$

By Lemma 5.1,  $f_i$  has exactly one root in the interval  $[z, z']$  which is also in the interval  $(z, z')$ . The shape of the plot of the rational function  $f_i/f_{i-1}$  is one of those shown in



Fig. 2. If  $r_{i-1} \notin [z, z']$ , then  $q_{i-1}$  does not change sign on the interval  $[z, z']$ . By (5.1), we conclude that the plot of  $f_{i+1}/f_{i-1}$  also has one of the shapes shown in Fig. 2. Hence  $f_{i+1}(z)$  must have a zero in the interval  $(z, z')$ .  $\square$

**COROLLARY 5.3.** *If  $f_0$  has distinct real roots, then  $S_{i+1}$  interleaves  $R_{i-1}$ , for  $i = 1, 2, \dots, n - 1$ .*

*Proof.* As in the proof of Lemma 5.1, let  $z_{k-11} < z_{k1} < z_{k-12} < z_{k2} < \dots < z_{k-1n-k} < z_{kn-k} < z_{k-1n-k+1}$ . Recall that  $r_{k-1}$  is the only zero of  $q_{k-1}$ . If  $r_{k-1}$  is not in the interval  $[z_{k-11}, z_{k-1n-k+1}]$ , then by Lemma 5.2,  $f_{k+1}(z)$  must have at least  $n - k$  zeros, one in each of the  $n - k$  intervals  $(z_{k-1j}, z_{k-1j+1})$ ,  $1 \leq j \leq n - k$ . But this contradicts the fact that  $\deg(f_{k+1}) = n - k - 1$ . Therefore  $r_{k-1}$  must lie in the interval  $[z_{k-11}, z_{k-1n-k+1}]$ .

If  $r_{k-1}$  equals either  $z_{k-11}$  or  $z_{k-1n-k+1}$  then  $r_{k-1}$  is also a root of  $f_{k+1}$ . This contradicts the fact that  $R_{k+1}$  strictly interleaves  $R_k$ . Therefore,  $r_{k-1}$  lies in the open interval  $(z_{k-11}, z_{k-1n-k+1})$ .

Next, suppose that  $r_{k-1}$  coincides with one of the other roots of  $f_{k-1}$ , say  $r_{k-1} = z_{k-1\alpha}$ ,  $1 < \alpha < n - k + 1$ . Then  $z_{k-1,\alpha}$  is also a root of  $f_{k+1}(z)$ . By Lemma 5.2, each of the intervals  $(z_{k-1j}, z_{k-1j+1})$ ,  $1 \leq j < \alpha - 1$  and  $\alpha < j < n - k + 1$ , contains one root of  $f_{k+1}(z)$ . This accounts for all the  $n - k - 1$  roots of  $f_{k+1}(z)$ . Therefore  $z_{k-11} < z_{k+11} < z_{k-12} < z_{k+12} < \dots < z_{k-1\alpha-1} < z_{k+1\alpha-1} = z_{k-1\alpha} = r_{k-1} < z_{k-1\alpha+1} < z_{k+1\alpha} < \dots < z_{k-1n-k} < z_{k+1n-k-1} < z_{k-1n-k+1}$ . Hence the assertion in the corollary holds in this case.

In case  $r_{k-1}$  does not coincide with any root of  $f_{k-1}(z)$ , the corollary immediately follows from Lemma 5.2. Hence  $S_{k+1}$  interleaves  $R_{k-1}$ .  $\square$

The main result of this section follows as a corollary to the following theorem.

**THEOREM 5.4.** *There exists a  $k$ ,  $0 \leq k \leq n - 2$ , such that  $r_k$  splits the real line into two semi-infinite intervals,  $I_1 = (-\infty, r_k)$  and  $I_2 = [r_k, \infty)$ , with the property that  $|R_0 \cap I_i|$  is in the range  $[n/4, 3n/4]$ , for  $i = 1, 2$ .*

In order to prove this theorem, we need the following lemma:

**LEMMA 5.5.** *Any subinterval  $I$  of the real line which contains  $l > \lfloor (n + 1)/2 \rfloor$  roots of  $f_0$  contains at least one  $r_k$  for some  $k$ :  $0 \leq k \leq n - 2$ .*

*Proof.* The proof of the lemma is by contradiction. Suppose an interval  $I$  contains  $l > \lfloor (n + 1)/2 \rfloor$  roots of  $f_0$  and none of the  $r_k$ 's. In order to prove the lemma, we first prove by induction that  $I$  contains at least  $l - \lfloor (m + 1)/2 \rfloor$  roots of  $f_m$ .

*Basis.* Trivially true for  $f_0$  and  $f_1$ .

*Induction Step.* Suppose  $I$  contains  $l - \lfloor (m - 1)/2 \rfloor$  roots of  $f_{m-2}$ . Since  $r_{m-2}$  is not in  $I$ , by Corollary 5.3, the roots of  $f_m$  in  $I$  interleave the roots of  $f_{m-2}$  in  $I$ . Hence  $f_m$  has at least  $l - \lfloor (m - 1)/2 \rfloor - 1 = l - \lfloor (m + 1)/2 \rfloor$  roots in the interval  $I$ .

But  $f_n$  is of degree 0 and therefore has no root in  $I$ —a contradiction.  $\square$

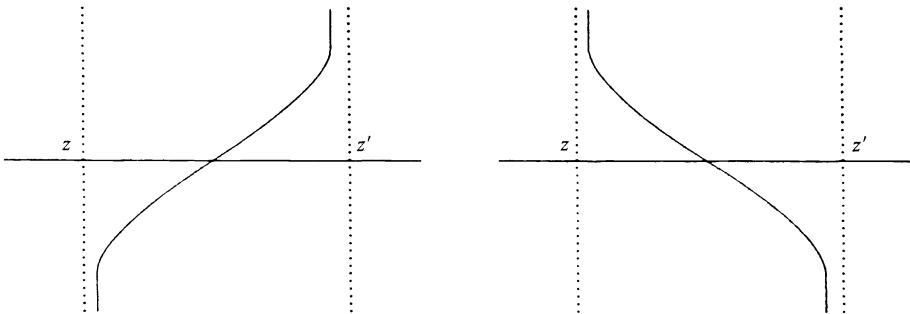


FIG. 2. Possible shapes of the plots of  $f_i/f_{i-1}$  and  $f_{i+1}/f_{i-1}$ .

*Proof of Theorem 5.4.* Consider the interval  $I$  containing the middle  $\lfloor (n+3)/2 \rfloor$  roots of  $f_0$ . By Lemma 5.5,  $I$  contains some  $r_k$ .  $\square$

The above theorem proves that there exists a  $k$  such that at least  $n/4$  roots of  $p(z)$  are on either side of  $r_k$  on the real line. For convenience, let us denote this distinguished  $r_k$  by  $v$ . There may be a root of  $f_0$  which is very close to  $v$ . Therefore, the choice of  $v$  as the point  $w$  in the description of the numerical integration scheme is not suitable. We get around this problem by choosing  $w$  very close to  $v$  as follows:

$$w = \begin{cases} v & \text{if } s_- = s_+ = s_0, \\ v - 2^{-(\sigma+2)} & \text{if } s_+ \neq s_0 \neq 0, \\ v + 2^{-(\sigma+2)} & \text{if } s_- \neq s_0 \neq 0, \\ v + 2^{-(\sigma+1)} & \text{if } s_0 = 0, \end{cases}$$

where  $\sigma$  is as defined in Corollary 2.3 and  $s_-$ ,  $s_0$ , and  $s_+ \in \{-, 0, +\}$  are the signs of  $p(v - 2^{-(\sigma+2)})$ ,  $p(v)$ , and  $p(v + 2^{-(\sigma+2)})$ , respectively.

**6. Determining real roots in the presence of complex roots is as hard as determining all roots.** In this section, we reduce the problem of determining all roots of a polynomial to the problem of determining all real roots of two related polynomials. The reduction uses some well-known techniques from the *Theory of Equations* (see, for example, [U48]). The size of the two related polynomials is polynomially bounded by the size of the initial polynomial and the reduction is in NC. This establishes that the problem of determining all real roots of a polynomial is in NC if and only if the problem of determining all roots (real and complex) of a polynomial is in NC.

Suppose we wish to determine all roots of a polynomial  $h(z)$  of degree  $n$  with  $m$  bit integer coefficients. Replace  $z$  by  $x + iy$ ,  $x, y$  real, and  $i = \sqrt{-1}$ . Also write  $h(x + iy) = f(x, y) + ig(x, y)$ , where  $f(x, y)$  and  $g(x, y)$  are the real and imaginary parts of  $h(x + iy)$ , respectively. Let  $f(x, y) = \sum_{j=0}^n \psi_j(x)y^j = \sum_{j=0}^n \omega_j(y)x^j$  and  $g(x, y) = \sum_{j=0}^n \phi_j(x)y^j = \sum_{j=0}^n \chi_j(y)x^j$ . Define  $X$  to be the  $2n \times 2n$  matrix as follows:

$$\begin{aligned} X(i, j) &= \psi_{n+i-j}(x), & 1 \leq i \leq n, \\ X(i, j) &= \phi_{i-j}(x), & n \leq i \leq 2n. \end{aligned}$$

Similarly, define a  $2n \times 2n$  matrix  $Y$  using  $\omega_i(y)$  and  $\chi_i(y)$  instead of  $\psi_i(x)$  and  $\phi_i(x)$ , respectively. Then  $\det(X)$  and  $\det(Y)$  are polynomials of degree at most  $n^2$ . The real and imaginary parts of any root of  $h(z)$  are real roots of  $\det(X)$  and  $\det(Y)$ , respectively. Thus, by finding all real roots of  $\det(X)$  and  $\det(Y)$ , we can determine all the roots of  $h(z)$ .

**Acknowledgments.** We would like to thank Franco Preparata for suggesting the “geometric spacing” of samples in order to evaluate various contour integrals to high precision and Don Coppersmith for making many useful suggestions during the course of this work. We would also like to thank Alok Aggarwal, Ashok Chandra, Ming-Deh Huang, Barry Trager, and Victor Miller for helpful discussions, and a referee for suggestions that have improved this manuscript. The referee also pointed out an error in the proof of Lemma 3.1 in the original version of this manuscript. In addition, the last author wishes to thank Tamer Basar, Dan Grayson, Michael Loui, Kameshwar Poolla, Sanjeev Rao, and Vasant Rao for helpful discussions on the general root-finding problem.

The IBM Thomas J. Watson Research Center, Yorktown Heights, provided the creative atmosphere in which this work began.

## REFERENCES

- [BKR84] M. BEN-OR, D. KOZEN, AND J. REIF, *The complexity of elementary algebra and geometry*, 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 457–464; in *J. Comput. System Sci.*, to appear.
- [B84] S. J. BERKOWITZ, *On computing the determinant in small parallel time using a small number of processors*, *Inform. Process. Lett.*, 18 (1984), pp. 147–150.
- [BGH82] A. BORODIN, J. VON ZUR GATHEN, AND J. HOPCROFT, *Fast parallel matrix and GCD computations*, *Inform. and Control*, 52 (1982), pp. 241–256.
- [BP60] W. S. BURNSIDE AND A. W. PANTON, *The Theory of Equations*, Vols. 1, 2, Dover, New York, 1960.
- [C29] A. CAUCHY, *Sur la résolution des équations numériques et sur la théorie d'élimination*, *Oeuvres*, 19 (1829), pp. 87–161.
- [C66] G. E. COLLINS, *Polynomial remainder sequences and determinants*, *Amer. Math. Monthly*, 73 (1966), pp. 708–712.
- [C67] ———, *Subresultants and reduced polynomial remainder sequences*, *J. Assoc. Comput. Mach.*, 14 (1967), pp. 128–142.
- [C76] L. CSANKY, *Fast parallel matrix inversion algorithms*, *SIAM J. Comput.*, 5 (1976), pp. 618–623.
- [C48] D. R. CURTIS, *Analytic Functions of a Complex Variable*, Mathematical Association of America, Washington, D.C., 1948, p. 104.
- [FW78] S. FORTUNE AND J. WYLLIE, *Parallelism in random access machines*, 10th Annual ACM Symposium on Theory of Computing, 1978, pp. 114–118.
- [vG83a] J. VON ZUR GATHEN, *Parallel algorithms for algebraic problems*, *SIAM J. Comput.*, 13 (1984), pp. 802–824.
- [vG83b] ———, *Representations of rational functions*, *SIAM J. Comput.*, 15 (1986), pp. 432–452.
- [H71] L. E. HEINDEL, *Integer arithmetic algorithms for polynomial real zero determination*, *J. Assoc. Comput. Mach.*, 18 (1971), pp. 533–548.
- [H74] P. HENRICI, *Applied and Computational Complex Analysis*, Vol. 1, John Wiley, New York, 1974.
- [H70] A. S. HOUSEHOLDER, *The Numerical Treatment of Single Nonlinear Equations*, McGraw-Hill, New York, 1970.
- [KY85] D. KOZEN AND C.-K. YAP, *Algebraic cell decomposition in NC*, 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 515–521.
- [K74] H. T. KUNG, *On computing reciprocals of power series*, *Numer. Math.*, 22 (1974), pp. 341–348.
- [M64] K. MAHLER, *An inequality for the discriminant of a polynomial*, *Michigan Math. J.*, 11 (1964), pp. 257–262.
- [M66] M. MARDEN, *Geometry of Polynomials*, Math. Surveys 3, American Mathematical Society, Providence, RI, 1966.
- [P85] V. Y. PAN, *Fast and efficient algorithms for sequential and parallel evaluation of polynomial zeros and of matrix polynomials*, 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 522–531.
- [S82] A. SCHÖNHAGE, *The fundamental theorem of algebra in terms of computational complexity*, unpublished manuscript, 1982.
- [T86] P. TIWARI, *The communication complexity of distributed computing and a parallel algorithm for polynomial roots*, Ph.D. thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 1986.
- [U48] J. V. USPENSKY, *Theory of Equations*, McGraw-Hill, New York, 1948.

## A LOWER BOUND ON THE COMPLEXITY OF THE UNION-SPLIT-FIND PROBLEM\*

KURT MEHLHORN<sup>†</sup>, STEFAN NÄHER<sup>‡</sup>, AND HELMUT ALT<sup>‡</sup>

**Abstract.** We prove a  $\Theta(\log \log n)$  (i.e., matching upper and lower) bound on the complexity of the Union-Split-Find problem, a variant of the Union-Find problem. Our lower bound holds for **all** pointer machine algorithms and does not require the separation assumption used in the lower-bound arguments of Tarjan [*J. Comput. Systems Sci.*, 18 (1979), pp. 110–127] and Blum [*SIAM J. Comput.*, 15 (1986), pp. 1021–1024]. We complement this with a  $\Theta(\log n)$  bound for the Split-Find problem under the separation assumption. This shows that the separation assumption can imply an exponential loss in efficiency.

**Key words.** complexity, interval splitting, lower bound, pointer machine, priority queue, union

**AMS(MOS) subject classification.** 68

**1. Introduction.** We consider the following three operations on a linear list  $x_1, x_2, \dots, x_n$  of items, some of which are marked:

- UNION ( $x_i$ ): given a pointer to the marked item  $x_i$  unmark this item;
- SPLIT ( $x_i$ ): given a pointer to the unmarked item  $x_i$  mark this item;
- FIND ( $x_i$ ): given a pointer to the item  $x_i$  return a pointer to  $x_j$ ,  
where  $j = \min \{l \mid l \geq i \text{ and } x_l \text{ is marked}\}$ .

Note that the marked items partition the linear list  $x_1, \dots, x_n$  into intervals of unmarked items. Then FIND ( $x_i$ ) returns (a pointer to) the right endpoint of the interval containing  $x_i$ , SPLIT ( $x_i$ ) splits the interval containing  $x_i$ , and UNION ( $x_i$ ) joins the two intervals having  $x_i$  as a common endpoint. We call the problem above the **Union-Split-Find problem**; P. v. Emde Boas et al. [EKZ77] called it a priority queue problem. They referred to the three operations as Insert, Delete, and Successor, and exhibited an  $O(\log \log n)$  solution for it. We will also consider the **Split-Find problem** and the **Union-Find problem** (only operations split, find and union, find, respectively). Note that our Union-Find problem is a restriction of the usual Union-Find problem (here called the general Union-Find problem) because we allow only adjacent intervals to be joined. The Union-Split-Find problem is important for a number of applications, e.g., dynamic fractional cascading [MN86] and computing shortest paths [M84b, p. 47].

We study the complexity of the Union-Split-Find problem in the pointer machine model of computation (Kolmogorov [Ko53], Knuth [Kn68], Schönhage [S73], Tarjan [T79]). A pointer machine captures the list-processing capabilities of computers; its storage consists of records connected by pointers. Previously, lower bounds in a restricted pointer machine model (the term “restricted” is explained below) were obtained by Tarjan [T79] and Blum [B86]. Tarjan proved a  $\Theta(\alpha(n))$  bound on the amortized cost of the general Union-Find problem and Blum proved a  $\Theta(\log n / \log \log n)$  bound on the worst-case cost. Tarjan’s and Blum’s lower bounds rely heavily on the following **separation assumption** (quoted from Tarjan [T79]):

At any time during the computation, the contents of the memory can be partitioned into collections of records such that each collection corresponds to a currently existing

\* Received by the editors December 31, 1986; accepted for publication (in revised form) December 9, 1987. This research was supported by the Deutsche Forschungsgemeinschaft, grant SPP ME 620/6-1.

<sup>†</sup> FB10, Informatik Universität des Saarlandes, D-6600 Saarbrücken, Federal Republic of Germany.

<sup>‡</sup> FB Mathematik, WE3, Freie Universität Berlin, D-1000 Berlin 33, Federal Republic of Germany.

set,  $\dots$  and no record in one collection contains a pointer to a record in another collection.

Because of this assumption we call their model **restricted**. If we view pointers as undirected edges in a graph, then the separation assumption states that every currently existing set corresponds to a component of the graph.

The main results of this paper are as follows.

(1) The complexity of the Union-Split-Find problem in the pointer machine model is  $\Theta(\log \log n)$ . Here, the upper bound is on the worst-case cost of the three operations and the lower bound is on the amortized cost of the three operations, i.e., there are arbitrarily large  $m$  and sequences of  $m$  SPLIT, FIND, and UNION operations having a total cost of  $\Omega(m \log \log n)$ .

The upper bound can be found in [EKZ77], [Ka84], and [MN86]. The solution of [MN86] supports two additional operations ADD and ERASE which allow us to modify the underlying linear list; the solution does not satisfy the separation assumption. The lower bound will be proved in § 2 of this paper.

(2) In the restricted pointer machine model the worst-case complexity of the Split-Find problem is  $\Theta(\log n)$  and the amortized complexity of the Union-Split-Find problem is  $\Theta(\log n)$ .

This will be shown in § 3.

**2. The lower bound.** In this section we will show that each solution for the Union-Split-Find problem on a pointer machine requires  $\Omega(\log \log n)$  computational steps, even in the amortized sense. More precisely, we show that there are arbitrarily large  $m$  and sequences of  $m$  UNION, FIND, SPLIT operations having a total cost of  $\Omega(m \log \log n)$ .

Our machine model is a pointer machine as described in [T79]. Its memory  $M$  consists of an unbounded collection of records, each containing two pointers to other records and an arbitrary amount of additional information. Thus  $M$  can be regarded as a directed graph with outdegree 2. We assume that the set of items  $S$  is realized by two sets of records in  $M$ , a set of input records  $I = \{x_1^*, x_2^*, \dots, x_n^*\}$ , and a set of output records  $O = \{y_1^*, y_2^*, \dots, y_n^*\}$ . (The distinction between input and output records is merely a notational convenience.)

FIND( $x$ ) is executed as follows. The machine starts with a pointer to input Record  $x^*$  (corresponding to item  $x \in S$ ) in some register  $r$ . It stops with a pointer to the output record  $y^*$  in  $r$  which corresponds to item  $y = \text{FIND}(x)$ . Since it can access records in  $M$  only by pointers contained in one of its registers, there must be a path  $p$  formed by records and pointers in  $M$  starting in  $x^*$  and ending in  $y^*$  such that pointers to all records on  $p$  have been loaded into a register during the execution of the FIND. In addition to traversing a path from  $x^*$  to  $y^*$  the FIND operation may change some number of pointers. The cost of the FIND operation is certainly bounded from below by the length of a shortest path from  $x^*$  to  $y^*$  in  $M$  plus the number of pointers changed.

When executing the operation SPLIT( $y$ ) the machine starts with a pointer to record  $y^*$  in some register. After the operation record  $y^*$  is reachable via pointers in  $M$  for all records  $x^*$  corresponding to items  $x$  with  $\text{FIND}(x) = y$ . To achieve this the machine changes certain pointers in  $M$ . The number of pointers changed is the cost of SPLIT( $y$ ). The cost of UNION( $y$ ) is defined similarly.

In order to model the actions of the pointer machine on its memory  $M$  in a more abstract way we consider FIND and SPLIT as operations on a directed graph  $G$  as follows:

Let  $G = (V, E)$  be a directed graph with the following:

- (1) For all  $v \in V$ :  $\text{outdegree}(v) = 2$ .
- (2) There is a set of  $n$  input nodes  $I = \{x_1, x_2, \dots, x_n\} \subseteq V$ .
- (3) There is a set of  $n$  output nodes  $O = \{y_1, y_2, \dots, y_n\} \subseteq V - I$  which may be marked (we call  $y_i$  the output node corresponding to input node  $x_i$ ,  $1 \leq i \leq n$ ).

FIND ( $x_i$ ),  $x_i \in I$ , returns output node  $y_j$  such that  $j \geq i$ , and  $j$  is minimal with  $y_j$  marked. In addition, it may replace some edges of  $G$  by new edges. The complexity of FIND ( $x_i$ ) is the length of the shortest path from  $x_i$  to  $y_j$  in  $G$  plus the number of edges replaced.

SPLIT ( $y$ ),  $y \in O$ , marks output node  $y$  and replaces some edges of  $G$  by new edges such that every marked output node  $y$  is reachable from all input nodes  $x$  with  $\text{FIND}(x) = y$ . The complexity of SPLIT ( $y$ ) is the number of edges replaced.

UNION ( $y$ ),  $y \in O$ , unmarks output node  $y$  and replaces some edges of  $G$  with new edges. The complexity of UNION ( $y$ ) is the number of edges replaced.

LEMMA 1. *Let  $k$  be any integer, let  $L = 2^{2k+1} + 1$ , let  $n \geq 2^{(5k)2^k}$ , and let  $G_0 = (V, E)$  be a directed graph with all output nodes unmarked. Then there is a sequence of  $L$  SPLIT operations followed by  $L$  FIND operations, followed by  $L$  UNION operations such that we have the following:*

- (1) *The total cost of the sequence is at least*

$$\min \left( k \cdot L, \frac{1}{2^{2k+4}} n^{1/2^k} \right).$$

- (2) *All output nodes are unmarked after executing the sequence.*

*Proof.* We call a graph  $G$  a  $k$ -structure if and only if for every input node  $x$  there is a path of length at most  $k$  from  $x$  to the output node  $y = \text{FIND}(x)$ .

The idea of the proof is as follows. We first execute a sequence of  $L$  SPLIT instructions (this sequence is constructed below) which leaves us with a data structure  $G_1$ . We next execute a hardest FIND on  $G_1$ , i.e., an operation  $\text{FIND}(x)$  where the shortest path from  $x$  to  $\text{FIND}(x)$  has maximal length. Execution of this FIND instruction yields the data structure  $G_2$ . Again we perform a hardest FIND,  $\dots$ . In this way we perform a total of  $L$  FINDs, each FIND being a hardest FIND in the present data structure. Finally, we perform a sequence of UNIONS which undo all the SPLITS. This yields a data structure  $G'_0$  (generally different from  $G_0$ ) in which all output nodes are unmarked.

In order to estimate the cost of this sequence of instructions we distinguish two cases. Assume first that no  $G_i$ ,  $1 \leq i \leq L$ , is a  $k$ -structure. Then each FIND costs at least  $k$  time units for a total cost of  $k \cdot L$  time units. Assume next that some  $G_i$  is a  $k$ -structure. We will show that it takes at least  $(1/2^{2k+4}) \cdot n^{1/2^k}$  edge changes to construct  $G_i$  from  $G_0$ ; more precisely, we show the following claim.

CLAIM. *Let  $G_0$  be an arbitrary data structure with all output nodes unmarked. Then there is a set  $\{z_1, \dots, z_L\} \subseteq O$  of output nodes such that no  $k$ -structure  $G'$  with exactly  $z_1, \dots, z_L$  marked can be constructed from  $G_0$  with fewer than  $(1/2^{2k+4}) \cdot n^{1/2^k}$  edge replacements.*

*Proof.* For all  $v \in V$  and  $l \geq 1$  let  $R_l(v)$  denote the set of all input nodes  $w$  such that  $v$  is reachable from  $w$  on a path of length at most  $l$ .

Let  $c_0, c_1, \dots, c_{2^k}$  be a sequence of positive constants with

$$n = c_0 \geq c_1 \geq \dots \geq c_{2^k-1} \geq c_{2^k} = 1.$$

Then the following lemma holds for  $G_0$ .

LEMMA 2. *There is an  $l$ ,  $0 \leq l \leq 2^k - 1$ , and  $l$  distinct nodes  $v_1, \dots, v_l$  such that we have the following:*

- (1) *Either  $l = 0$ , or  $|\bigcap_{1 \leq i \leq l} R_{k-1}(v_i)| \geq c_l$ .*

(2) For each  $w \in V - \{v_1, \dots, v_l\}$ :  $|I^* \cap R_{k-1}(w)| < c_{l+1}$ , where  $I^* = \bigcap_{1 \leq i \leq l} R_{k-1}(v_i)$  ( $I^* = I$  if  $l=0$ ).

*Proof.* Let  $l_0$  be maximal such that there are distinct nodes  $v_1, \dots, v_{l_0}$  with  $|\bigcap_{1 \leq i \leq l_0} R_{k-1}(v_i)| > c_0$ . We only have to show that  $l_0 < 2^k$ . Assume  $l_0 \geq 2^k$ . Then there are  $2^k$  distinct nodes  $v_1, \dots, v_{2^k}$  with  $\bigcap_{1 \leq i \leq 2^k} R_{k-1}(v_i) \neq \emptyset$ . But for every  $x \in I$  there are at most  $2^k - 1$  nodes  $v \in V$  with  $x \in R_{k-1}(v)$ , a contradiction.  $\square$

Now let  $l \geq 0$  and let  $v_1, v_2, \dots, v_l$  be nodes satisfying the conditions of Lemma 2.. We give a sequence of  $2^{2^{k+1}} + 1$  SPLIT operations which require at least  $(c_l/2L - 2^{k+1})/c_{l+1}$  edge replacements. First we divide  $I^* = \bigcap_{1 \leq i \leq l} R_{k-1}(v_i)$  into  $2L$  intervals  $A_1, A_2, \dots, A_{2L}$  of about equal size.  $A_1$  consists of the  $\lfloor I^*/2L \rfloor$  smallest indexed items in  $I^*$ ,  $A_2$  consists of the next  $\lfloor I^*/2L \rfloor$  smallest indexed items in  $I^*$ , and so forth. Clearly,

$$|A_i| \geq \frac{c_l}{2L} - 1 \quad \text{for all } i.$$

Let  $A'_1, A'_2, \dots, A'_{2L}$  be the corresponding intervals of output nodes.

LEMMA 3. In every  $A'_j$ ,  $j \in \{2, 4, 6, \dots, 2L\}$ , there is an output node  $y_j$  that is reachable in  $G_0$  from at most  $2^{k+1}$  input nodes in  $A_{j-1}$  on a path of length  $\leq k$ .

*Proof.* For all  $x \in A_{j-1}$  we have  $|\{v \in V \mid x \in R_k(v)\}| \leq 2^{k+1} - 1$ . Thus we conclude

$$\begin{aligned} \sum_{x \in A_{j-1}} |\{v \in V \mid x \in R_k(v)\}| &\leq |A_{j-1}| \cdot (2^{k+1} - 1) \\ &= |A'_j| \cdot (2^{k+1} - 1). \end{aligned} \quad \square$$

Let  $y_j \in A'_j$ ,  $j \in \{2, 4, \dots, 2L\}$  be defined as in Lemma 3. Then there exists for each  $j$  a set  $B_{j-1} \subseteq A_{j-1}$  such that  $|B_{j-1}| \geq |A_{j-1}| - 2^{k+1}$  and such that for each  $x \in B_{j-1}$  there is no path of length at most  $k$  from  $x$  to  $y_j$  in  $G_0$ . Next consider any  $k$ -structure  $G'$  in which exactly the  $y_j$ 's defined above are marked. In  $G'$  there must be a path of length at most  $k$  from any  $x \in B_{j-1}$  to  $y_j$ . Let  $p(x)$  be any such path from  $x$  to  $y_j$  and let  $e(x)$  denote the first new edge on  $p(x)$ . Let the edge  $e(x)$  start in vertex  $v(x)$ . We have the following lemma.

LEMMA 4. (1)  $|\{x \in B_{j-1} \mid v(x) = w\}| \leq c_{l+1}$  for every node  $w \notin \{v_1, v_2, \dots, v_l\}$  and every  $j$ .

(2) There is a  $j \in \{2, 4, 6, \dots, 2L\}$  such that  $v(x) \notin \{v_1, v_2, \dots, v_l\}$  for all  $x \in B_{j-1}$ .

*Proof.* (1) If  $v(x) = w$  then  $w$  was reachable from  $x$  in the data structure  $G_0$  in at most  $k - 1$  steps. Thus the claim follows immediately from the definition of  $v_1, \dots, v_l$ .

(2) For every  $v \in \{v_1, v_2, \dots, v_l\}$  we have  $|\{y \in O \mid v \in R_k(y)\}| \leq 2^{k+1}$  and thus

$$\begin{aligned} \sum_{v \in \{v_1, \dots, v_l\}} |\{y \in O \mid v \in R_k(y)\}| &\leq l 2^{k+1} \\ &\leq 2^{2^{k+1}} \quad \text{since } l \leq 2^k. \end{aligned}$$

That is, all paths of length at most  $k$  using nodes from  $\{v_1, \dots, v_l\}$  cannot lead to more than  $2^{2^{k+1}} = L - 1$  output nodes. Since we marked  $L$  nodes in  $O$  there must be one  $y_j$  such that  $v(x) \notin \{v_1, \dots, v_l\}$  for all  $x \in B_{j-1}$ .  $\square$

Now consider any  $j \in \{2, 4, \dots, 2L\}$  satisfying condition (2) of Lemma 4; i.e.,  $v(x) \notin \{v_1, \dots, v_l\}$  for all  $x \in B_{j-1}$ . Next observe that by (1) of Lemma 4 for any node  $w \notin \{v_1, \dots, v_l\}$  there are at most  $c_{l+1}$  nodes  $x \in B_{j-1}$  with  $v(x) = w$ . Thus

$$\frac{|B_{j-1}|}{c_{l+1}} = \frac{|A_{j-1}| - 2^{k+1}}{c_{l+1}} \geq \frac{c_l/2L - 1 - 2^{k+1}}{c_{l+1}} \geq \frac{c_l/(2^{2^{k+2}} + 2) - 1 - 2^{k+1}}{c_{l+1}}$$

edges must be changed to obtain  $G'$  from  $G_0$ . With

$$c_i = n^{1-i/2^k}, \quad 0 \leq i \leq 2^k$$

we conclude further that

$$\begin{aligned} \frac{c_l / (2^{2k+2} + 2) - (2^{k+1} + 1)}{c_{l+1}} &\cong \frac{n^{1/2^k}}{2^{2k+2} + 2} - (2^{k+1} + 1) \\ &\cong \frac{n^{1/2^k}}{2 \cdot (2^{2k+2} + 2)} \quad \text{since } n \cong 2^{(5k)2^k} \\ &\cong \frac{n^{1/2^k}}{2^{2k+4}}. \end{aligned}$$

This completes the proof of the claim and of Lemma 1.  $\square$

**THEOREM 1.** (a) *The single operation worst-case complexity of the Split-Find problem is  $\Omega(\log \log n)$ .*

(b) *The amortized complexity of the Union-Split-Find problem is  $\Omega(\log \log n)$ ; i.e., there are arbitrarily large  $m$  and sequences of  $m$  UNION, SPLIT, and FIND operations having a total cost of  $\Omega(m \log \log n)$ .*

*Proof.* (a) Let  $k$  be maximal such that  $n \cong 2^{(5k)2^k}$ . Then  $k = \Omega(\log \log n)$  and

$$\begin{aligned} \frac{1}{2^{2k+1} + 1} \cdot \frac{1}{2^{2k+4}} n^{1/2^k} &\cong \frac{2^{5k}}{2^{2k+4}(2^{2k+1} + 1)} \\ &= \Omega(2^k) \\ &= \Omega(\log n) \\ &= \Omega(\log \log n). \end{aligned}$$

Consider the sequence of  $2L$  SPLITs and FINDs constructed in the proof of Lemma 1. The cost of this sequence is

$$\Omega\left(\min\left(k \cdot L, \frac{1}{2^{2k+4}} \cdot n^{1/2^k}\right)\right) = \Omega(L \cdot \log \log n),$$

and hence at least one operation of the sequence has cost  $\Omega(\log \log n)$ .

(b) Let  $k$  be maximal such that  $n \cong 2^{(5k)2^k}$  and let  $m = 3 \cdot L \cdot s$  for some  $s$ . Let  $G_0$  be any data structure with all output nodes unmarked. By part (a) and Lemma 1 there is a sequence of  $3 \cdot L$  instructions such that

- (1) the total cost of the sequence is  $\Omega(L \log \log n)$ ;
- (2) all output nodes are unmarked after executing the sequence.

Thus again by part (a) and Lemma 1 there is another sequence of  $3 \cdot L$  instructions  $\dots$ . Using this argument  $s$  times yields a sequence of  $m$  instructions with total cost  $\Omega(m \log \log n)$ .  $\square$

**3. A lower bound in the restricted model.** In this section we prove a  $\Omega(\log n)$  lower bound for the worst-case complexity of the Split-Find problem and a  $\Omega(\log n)$  lower bound for the amortized complexity of the Union-Split-Find problem in the restricted pointer machine model. We consider pointer machine algorithms satisfying the following **separation condition**:



The memory can be partitioned into subgraphs such that each subgraph corresponds exactly to a current interval. There exists no edge from a node in such a subgraph to a node outside the subgraph [B86], [T79].

As in the previous section we define a  $k$ -structure.

DEFINITION. Let  $G = (V, E)$  be a directed graph with input nodes  $I = \{x_1, \dots, x_n\}$  and output nodes  $O = \{y_1, \dots, y_n\}$  some of which may be marked.  $G$  is called a  **$k$ -structure** if and only if for every  $x \in I$  there is a path of length at most  $k$  to  $\text{FIND}(x)$  and  $G$  fulfills the separation condition; i.e., for every  $x_i, x_j \in I$  with  $\text{FIND}(x_i) \neq \text{FIND}(x_j)$  there is no  $v \in V$  that is reachable from both  $x_i$  and  $x_j$ .  $\square$

LEMMA 5. Let  $n \geq (4k)^k/k!$ . In any  $k$ -structure with output nodes  $y_1, \dots, y_{n-1}$  unmarked and  $y_n$  marked satisfying the separation condition there is a SPLIT operation requiring at least  $k/12 \cdot n^{1/k}$  edge replacements.

*Proof.* Since  $y_n$  is the only marked output node there is, for each input node  $x$ , a path of length at most  $k$  to  $y_n$ . Let  $T$  be the subgraph of  $G$  formed by all shortest paths from input nodes to  $y_n$ . Then  $T$  is a tree with root  $y_n$ , leaves  $\{x_1, \dots, x_n\}$ , and height at most  $k$ . For any node  $v$  in  $T$  let  $I(v)$  denote the set of input nodes which are the leaves of the subtree rooted at  $v$ . At each internal node  $v$  of  $T$  the incoming edges can be ordered  $(w_1, v), (w_2, v), \dots, (w_m, v)$  such that the minimal index of any input node in  $I(w_i)$  is smaller than the minimal index of any input node in  $I(w_j)$  for all  $1 \leq i < j \leq m$ . (We call  $w_i$  the  $i$ th child of  $v$ .)

Now assume that SPLIT( $x$ ) is executed for some  $x \in I$ . Let

$$p: x = v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_l = y_n$$

be the path from  $x$  to the root  $y_n$  in  $T$ . For any  $v_i, 1 \leq i \leq l$ , on this path let  $w_1^i, \dots, w_j^i = v_{i-1}, \dots, w_m^i$  be the children of  $v_i$  in the order defined above. Before the split operation  $v_i$  is reachable from all input nodes of both of the following sets:

$$L_i = \{u \mid u \text{ is the node with minimal index in } I(w_k^i), 1 \leq k \leq j-1\},$$

$$R_i = \{u \mid u \text{ is the node with minimal index in } I(w_k^i), j+1 \leq k \leq m\}.$$

After the split  $\text{FIND}(a) \neq \text{FIND}(b)$  for all  $a \in L_i, b \in R_i$  and by the separation condition  $v_i$  is not reachable from any node of at least one of the sets  $L_i$  or  $R_i$ . Thus the execution of SPLIT( $x$ ) requires at least

$$\min(|L_i|, |R_i|) = \min\{j-1, \text{indegree}(v_i) - j \mid v_{i-1} \text{ is the } j\text{th child of } v_i\}$$

edge replacements for each node  $v_i, 1 \leq i \leq l$ , on path  $p$ .

The total cost for SPLIT( $x$ ) is at least

$$c(p) = \sum_{i=1}^l \min\{j-1, \text{indegree}(v_i) - j \mid v_{i-1} \text{ is the } j\text{th child of } v_i\}.$$

We prove a lower bound for  $c(p)$ .

LEMMA 6. Let  $T$  be any tree of height  $k$  with  $n$  leaves and root  $r$ . Then there is a leaf  $x$  in  $T$  such that the path  $p$  from  $x$  to  $r$  has cost

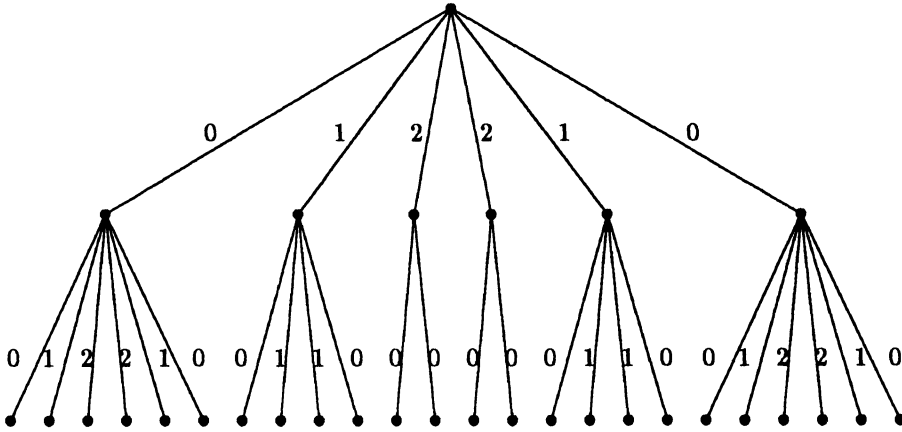
$$c(p) \geq \frac{k}{12} \cdot n^{1/k}.$$

*Proof.* Define  $l(k, j)$  as the maximal number of leaves in any tree  $T$  of height  $k$ , such that  $c(p) \leq j$  for every path  $p$  from a leaf to the root of  $T$ .

Then we have

$$l(0, j) = 1 \quad \text{for } j \geq 0$$





$$\ell(2, 2) = 24$$

FIG. 2

The term  $2^k$  accounts for the fact that in every vertex each label can be used twice:

$$\begin{aligned} l(k, j) &\leq 2^k \cdot \binom{k+j-1}{k-1} \\ &\leq 2^k \cdot \frac{(k+j-1)^{k-1}}{(k-1)!} \\ &\leq 2^k \cdot \frac{(k+j-1)^k}{k!} \\ &\leq 2^k \cdot \frac{(k+j)^k}{k!}. \end{aligned}$$

We conclude that in any tree of height  $k$  with  $n$  leaves there exists a path  $p$  from some leaf to the root such that

$$\begin{aligned} n &\leq 2^k \cdot \frac{(c(p) + k)^k}{k!} \\ &\leq 2^k \cdot \frac{(2 \cdot c(p))^k}{k!} \quad \text{since } n \geq \frac{(4k)^k}{k!} \text{ and hence } c(p) \geq k, \end{aligned}$$

and finally

$$\begin{aligned} c(p) &\geq \frac{1}{2} \left( \frac{n \cdot k!}{2^k} \right)^{1/k} \\ &\geq \frac{k}{12} \cdot n^{1/k} \quad \text{since } (k!)^{1/k} \geq \frac{k}{3} \text{ if } k \geq 6. \end{aligned}$$

This completes the proof of Lemma 5.  $\square$

**THEOREM 2.** *In the restricted pointer machine model we have the following:*

- (a) *The single-operation worst-case complexity of the Split-Find problem is  $\Omega(\log n)$ .*
- (b) *The amortized complexity for the Union-Split-Find problem is  $\Omega(\log n)$ ; i.e., there are arbitrarily large  $m$  and sequences of  $m$  UNION, SPLIT, FIND operations having a total cost of  $\Omega(m \log n)$ .*

*Proof.* (a) Let  $k$  be maximal such that  $n \geq (4k)^k/k!$ . Then  $k = \Omega(\log n)$ . Let  $G$  be any data structure with all output nodes except  $y_n$  unmarked. By Lemma 5 there is either a FIND operation which costs more than  $k$  time units or there is a SPLIT operation having a cost of

$$\frac{k}{12} \cdot n^{1/k} \geq \frac{\log n}{12} \cdot n^{1/\log n} = \frac{\log n}{6} = \Omega(\log n).$$

(b) Let  $G$  be any structure with all output nodes except  $y_n$  unmarked. Define  $k$  as in part (a). If  $G$  is not a  $k$ -structure then we perform a hardest FIND in  $G$ ; this FIND has cost  $k = \Omega(\log n)$  and leaves us with a structure  $G'$  where all output nodes except  $y_n$  are unmarked. If  $G$  is a  $k$ -structure then there is a SPLIT operation of cost  $k/12 \cdot n^{1/k} = \Omega(\log n)$ . We perform it and immediately undo it by the corresponding UNION operation. This leaves us with a structure  $G'$ , where all output nodes except  $y_n$  are unmarked. At this point we are in the initial situation and we have forced the algorithm to spend  $\Omega(\log n)$  time units on at most two operations. Part (b) follows.  $\square$

We want to point out that  $O(\log n)$  is clearly also an upper bound for the complexity of the Union-Split-Find problem in the restricted model. We only have to represent each interval by a balanced tree (cf., e.g., [M84a, § III.5.3.1]).

We close this section with the following corollary.

**COROLLARY 1.** *Pointer machines obeying the separation assumption are exponentially weaker than pointer machines without the separation assumption. This is true for the worst-case complexity and for the amortized complexity.*

*Proof.* This follows immediately from Theorems 1 and 2.  $\square$

**4. Conclusions and open problems.** In this paper we proved several new lower bounds for the Union-Split-Find problem. In particular, we presented an  $\Omega(\log \log n)$  lower bound on the amortized complexity of the Union-Split-Find problem valid for **all** pointer machine algorithms and an  $\Omega(\log n)$  lower bound for restricted (in the sense of Tarjan [T79] and Blum [B86]) pointer machine algorithms. Our lower bounds match known upper bounds. Thus our results reveal that the separation assumption of Tarjan and Blum can imply an exponential loss in efficiency.

Table 2 summarizes all known bounds for the complexity of pointer machine algorithms for the Union-Split-Find problem on intervals including the results of this paper.

There are still several open problems in both models. We have no lower bounds on the amortized complexity of Union-Find and Split-Find and for the worst-case

TABLE 2

Problem	General model		Restricted model	
Union-Find worst case amortized	$O(\log \log n)$ —	[EKZ77]	$O(\log n/\log \log n)$ $\Theta(\alpha(n))$	[B86] [T79]
Split-Find worst case amortized	$\Theta(\log \log n)$ —	<b>new</b>	$\Theta(\log n)$ $O(\log^* n)$	<b>new</b> [HU73]
Union-Split-Find worst case amortized	$\Theta(\log \log n)$ $\Theta(\log \log n)$	<b>new</b> <b>new</b>	$\Theta(\log n)$ $\Theta(\log n)$	<b>new</b> <b>new</b>

complexity of Union-Find in the general model. In the restricted model it remains an open problem to determine whether the Split-Find algorithm of Hopcroft and Ullman [HU73], whose amortized running time is  $O(\log^* n)$ , is optimal. We suppose that Blum's lower bound proof for the general Union-Find problem cannot be modified to work also for the interval problem.

We want to point out that the Union-Find problem and the Split-Find problem considered in this paper have amortized complexity  $\Theta(1)$  on random access machines. This was shown by Gabow and Tarjan [GT83] for the Union-Find problem and by Imai and Asano [IA84] for the Split-Find problem.

## REFERENCES

- [B86] N. BLUM, *On the single-operation worst-case time complexity of the disjoint set union problem*, SIAM J. Comput., 15 (1986), pp. 1021–1024.
- [EKZ77] P. V. EMDE BOAS, R. KAAS, AND E. ZIJLSTRA, *Design and implementation of an efficient priority queue*, Math. Systems Theory, 10 (1977), pp. 99–127.
- [GT83] H. N. GABOW AND R. E. TARJAN, *A linear-time algorithm for a special case of disjoint set union*, in Proc. 15th Annual SIGACT Symposium, 1983, pp. 246–251.
- [IA84] T. IMAI AND T. ASANO, *Dynamic segment intersection with applications*, in Proc. 25th Annual IEEE Symposium on the Foundations of Computer Science, Singer Island, FL, 1984, pp. 393–402.
- [HU73] J. E. HOPCROFT AND J. D. ULLMAN, *Set merging algorithms*, SIAM J. Comput., 2 (1973), pp. 294–304.
- [Ka84] R. G. KARLSSON, *Algorithms in a restricted universe*, Report CS-84-50, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1984.
- [Kn68] D. E. KNUTH, *The Art of Computer Programming*, Vol. 3, Fundamental Algorithms, Addison-Wesley, Reading, MA, 1968.
- [Ko53] A. N. KOLMOGOROV, *On the notion of algorithm*, Uspekhi Mat. Nauk., 8 (1953), pp. 175–176.
- [M84a] K. MEHLHORN, *Data Structures and Algorithms. Vol. 1. Sorting and Searching*, Springer-Verlag, Berlin, New York, 1984.
- [M84b] ———, *Data Structure and Algorithms. Vol. 2. Graph-Algorithms and NP-Completeness*, Springer-Verlag, Berlin, New York, 1984.
- [M84c] ———, *Data Structures and Algorithms. Vol. 3. Multidimensional Searching and Computational Geometry*, Springer-Verlag, Berlin, New York, 1984.
- [MN86] K. MEHLHORN AND S. NÄHER, *Dynamic fractional cascading*, TR 06/1986, FB10, Universität des Saarlandes, Saarbrücken, Federal Republic of Germany, 1986.
- [S73] A. SCHÖNHAGE, *Storage modification machines*, SIAM J. Comput., 9 (1980), pp. 490–508.
- [T79] R. E. TARJAN, *A class of algorithms which require nonlinear time to maintain disjoint sets*, J. Comput. Systems Sci., 18 (1979), pp. 110–127.

## A SHORT-TERM NEURAL NETWORK MEMORY\*

ROBERT J. T. MORRIS† AND WING SUNG WONG†

**Abstract.** Neural network memories with storage prescriptions based on Hebb's rule are known to collapse as more words are stored. By requiring that the most recently stored word be remembered precisely, a new simple short-term neural network memory is obtained and its steady state capacity analyzed and simulated. Comparisons are drawn with Hopfield's method, the delta method of Widrow and Hoff, and the revised marginalist model of Mezard, Nadal, and Toulouse.

**Key words.** neural networks, associative memory, short-term memory, machine learning, adaptive systems

**AMS(MOS) subject classifications.** 68Q05, 68T05

**1. Introduction.** In a paper [1] which attracted considerable attention, Hopfield proposed a neural network model consisting of  $N$  neurons taking values  $e^1, \dots, e^N \in \{-1, 1\}$ , and interconnected by a synaptic matrix  $T$ , where  $T_{ij}$  represents the synaptic efficacy by which neuron  $j$  influences neuron  $i$ . The collective behavior of the neurons is described by the equation

$$(1.1) \quad e^i \leftarrow \operatorname{sgn} \left( \sum_{j=1}^N T_{ij} e^j - u^i \right), \quad i = 1, \dots, N,$$

where  $\operatorname{sgn}(x) = x/|x|$ ,  $x \neq 0$ , and from now on we will assume  $u^i = 0$ . Equation (1.1) is a simplified, discrete version of the McCullough-Pitts physiological model. Hopfield required that the neurons' values be updated *randomly* and *asynchronously*. He showed that this model had the *autoassociative* property, as defined by Kohonen [2], i.e., it was able to retrieve a previously stored word from part of its specification. The prescription for storage of words  $e_s = (e_s^1, \dots, e_s^N)$ ,  $s = 1, 2, \dots, m$  was obtained from the equation

$$(1.2) \quad T_{ij} = \begin{cases} \sum_{s=1}^m e_s^i e_s^j, & i \neq j, \\ 0, & i = j. \end{cases}$$

This equation has frequently been called Hebb's rule and is simple and attractive, partly because of its property of *locality*; i.e., the synapse  $T_{ij}$  is updated with the storage of new word  $e_s$  according to a simple function of  $e_s^i$  and  $e_s^j$  only, and so all elements of the matrix  $T_{ij}$  can be updated asynchronously and in parallel. Hopfield gave a simple explanation as to why Hebb's rule (1.2) tends to result in the stored words being fixed points for the dynamics of (1.1):

$$\sum_{j=1}^N T_{ij} e^j = \sum_{s=1}^m e_s^i \left[ \sum_{j \neq i} e_s^j e_s^j \right] \approx e_s^i (N-1),$$

where the last approximation appeals to the argument that the stored vectors are expected to be "pseudo-orthogonal," i.e., the square-bracketed term is approximately  $(N-1)\delta_{st}$ . Furthermore, since  $T_{ij}$  is symmetric it is readily verified that the "energy function"  $-\sum_i e^i \sum_j T_{ij} e^j$  is decreased by the substitution (1.1) whenever  $e^i$  changes

\* Received by the editors June 22, 1987; accepted for publication (in revised form) January 20, 1988.

† AT&T Bell Laboratories, Holmdel, New Jersey 07733.

(provided individual neurons are updated one at a time and are left unchanged when the argument of  $\text{sgn}(\cdot)$  is zero). Thus (1.1) must converge to a fixed point. This constitutes an error-correcting property in the sense that patterns close to a fixed point are attracted to that point through the iteration of (1.1).

These ideas are closely related to schemes developed and/or analyzed by Widrow and Hoff [3], Kohonen [2], [4], and Stone [5] (see [6] for extensive references). Most of these rules were stated for the more general *heteroassociative* storage problem in which “keys and contents” or “inputs and targets” are associated; however, we will rephrase these results here in the autoassociative theme. An appealing learning rule in place of (1.2) has been called the Widrow-Hoff, or delta, rule and is described by<sup>1</sup>

$$(1.3) \quad T_n = T_{n-1} + \eta \delta_n e'_n,$$

where  $T_n$  is the synaptic matrix after word  $e_n$  has been stored and  $\eta$  is a scalar. The term  $\delta_n$  is given by

$$(1.4) \quad \delta_n = e_n - T_{n-1} e_n$$

and so represents the deviation between a new word and its image under the synaptic matrix just before it is stored. The most important result about these methods [2], [5] is that as the number of stored words increases, the  $T$  matrix converges to the matrix that (when applied to the input vector) results in minimal least square error for the estimation of the target vector from the input vector. In the autoassociative case this matrix is just the identity matrix. Kohonen [4] describes algorithms that do even more—they recursively produce the  $T_n$  matrix which for every  $n$  is the optimal estimator, although they are considerably more complex than the simple Hebb or delta rules. In exchange for the simplicity of the Hebb rule, these rules (including the delta rule) do not impose any requirement of orthogonality or even linear independence in achieving their properties. To place Hopfield’s work in perspective with prior and contemporary work, it should be noted that it is more the retrieval process of (1.1) that is unusual in that it specifies iterated (or feedback) threshold dynamics, as opposed to a retrieval process which is typically a single application of the synaptic matrix to the input vector, often without any thresholding. For more complete surveys of neural and associative learning models, see [1], [4], [6], [7].

There have been numerous variations on the memories described by (1.1) and (1.2). Soulie [8] and Soulie and Weisbuch [9] consider variations of (1.1) according to how points lying on the discontinuity of the  $\text{sgn}(\cdot)$  are treated and the order in the updating of the neurons. Definitions are given for *parallel iteration*, where all the neurons are updated simultaneously, *sequential iteration*, where all the neurons are repeatedly updated in the order of a permutation of  $\{1, \dots, N\}$ , and *random iteration*, as in Hopfield [1].

Personnaz, Guyon, and Dreyfus [10] consider variations on (1.2) which also mitigate the pseudo-orthogonality requirement of the Hebb rule. They note that for a state  $e_s$  to be stable with respect to (1.1) it is necessary and sufficient that

$$(1.5) \quad T e_s = A e_s,$$

for some  $A$  a diagonal matrix with nonnegative elements. For the simplest case of  $A = I$  and  $e_s$ ,  $s = 1, \dots, m$  linearly independent, (1.3) can be solved by taking

$$T = \Sigma(\Sigma'\Sigma)^{-1}\Sigma'$$

<sup>1</sup> We use the notation that  $T_{ij}$  represents the  $(i, j)$ th element of matrix  $T$ , whereas  $T_n$  represents the  $n$ th matrix in a sequence. As above,  $e_n$  refers to the  $n$ th column vector in a sequence and  $e'_i$  the  $i$ th component of vector  $e$ .

where  $\Sigma$  is the matrix with columns  $e_1, \dots, e_m$ . They referred to this prescription for storage as the Generalized Hebb rule.  $T$  is one form of a pseudo-inverse, and so this method is closely related to the other methods described above. One further distinction between [1] and the other work cited above is the use of zero diagonal elements of  $T$ ; we will return to the role of diagonal elements in § 2.

The memories described above have limited capacities, and as new words continue to be stored, reach a point where no words can be reliably retrieved or corrected. This collapse in capacity can occur in two distinct ways. One way, exhibited by Hopfield's method ((1.1), (1.2)), is that "pseudo-orthogonality" between all pairs of stored words becomes less likely. In addition, the unbounded growth of the  $T$  matrix typically implies decreasing relatively smaller contributions of any stored word. This results in the inability to retrieve any, including recently stored, words at steady state. A second method of degeneracy which is exhibited, for example, by the delta method of storage ((1.3), (1.4)), is the convergence of  $T$  to the identity matrix after the input of a long sequence of random vectors. As pointed out above, this is by design in the sense that with  $T = I$  the memory optimally recalls any vector offered to it. But the identity matrix fails to exhibit any capability to correct errors or produce a word from its partial presentation, and arguably has degenerated to a state of zero storage capacity. These two modes of collapse are discussed more precisely in § 2 and demonstrated experimentally in § 3.

If a neural network memory is to be used on an ongoing basis, it either needs to be reset to the *tabula rasa* ( $T = 0$ ) state periodically, or old words need to be removed or "forgotten" in some way. Besides physiological evidence that short-term memories, called *palimpsests*, exist in nature, perhaps the most common application of associative memories in computers is in hierarchical memory systems. These cache memories are often found between CPU and pinout on a chip, between CPU and backplane bus on single board computers, in memory management units and in disk buffer caches. In these caches it is desirable that words that have not been used for some time be replaced by current words according to the long recognized principle of locality of reference [11]. Additionally, the capability of neural memories to provide rapid, highly parallel but sometimes imperfect retrieval, and the presence of backup copies of data in lower levels of memory hierarchies suggest a possible role for neural architectures in hierarchical memory systems.

In the original paper by Hopfield [1] it was proposed that a short-term memory could be obtained by limiting the range of values  $T_{ij}$  could assume. This idea was pursued further by Nadal et al. [12] who called it "learning within bounds." Both [1] and [12] required that changes in  $T_{ij}$  satisfy a constraint such as  $|T_{ij}| \leq A$ . The experiments in [12] suggested that for optimal choice of parameters, about  $.016N$  words could be stored in this memory before the bit error rate (probability that an arbitrary bit of a retrieved word is in error) exceeded 1.5 percent. The paper [12] also proposed another scheme called "marginalist learning," but that scheme resulted in exponential growth of synaptic values causing the authors to regard it as somewhat unrealistic for use on an ongoing basis. Subsequently Mezard, Nadal, and Toulouse [13] revised the marginalist scheme by requiring that the average squared synaptic efficacy remain constant and obtained an update rule

$$(1.6) \quad T_{n+1} = (1 + \varepsilon^2/N)^{-1/2}[(\varepsilon/N)e_{n+1}e'_{n+1} + T_n]$$

where  $T_n$  is the synaptic matrix after learning  $e_1, \dots, e_n$ , and  $\varepsilon$  is a tuning parameter. Using this scheme and  $\varepsilon = 4$ , they observed a capacity of approximately  $.06N$  at a bit error rate of 1.5 percent.



In this paper we describe and analyze a short-term neural network memory that is based on a simple modification to Hopfield's model. The storage rule is simply described and implemented: the new word is impressed on the memory with sufficient intensity that it is memorized. This provides an automatic forgetting of older words in the memory. We present a detailed analysis and comparison of this scheme with Hopfield's memory, the delta method, and the revised marginalist method. Several important insights emerge regarding the role of the diagonal elements and a number of techniques are developed that allow approximation of various error rates.

**2. A new short-term memory.** In this section we present a mathematical analysis of the new update model. We break the analysis into two parts depending on whether the diagonal elements in the  $T$  matrix are present.

**2.1. New model without diagonal terms.** To make the analysis precise, we assume we have a sequence of independent words,  $e_1, e_2, e_3, \dots$ , each of which is an  $N$ -dimensional vector of *uniform signary* random variables. By a uniform signary random variable we mean a random variable that attains values 1 and  $-1$  with equal probability of one half. We represent the synaptic matrix after storing  $n$  words by  $T_n$ . Define the  $i$ th bit *impression* of the word  $e_k$  on  $T_n$ ,  $I_i(n, k)$ , by the relation

$$\begin{bmatrix} I_1(n, k) & & 0 \\ & \ddots & \\ 0 & & I_N(n, k) \end{bmatrix} = \langle T_n e_k e_k' \rangle,$$

where  $e'$  denotes the transpose of  $e$  and  $\langle \cdot \rangle$  is a linear operator on the space of  $N \times N$  matrices defined by

$$\langle X \rangle_{ij} = \begin{cases} 0, & i \neq j, \\ X_{ii} & \text{otherwise.} \end{cases}$$

Thus, the word  $e_k$  is correctly stored in  $T_n$  if and only if the impression  $I_i(n, k) > 0$ , for  $1 \leq i \leq N$ .

The new method of update requires that  $\langle T_n \rangle = 0$ , and  $I_i(n, n) = r$ , where  $r > 0$  is a storage threshold, for all  $i$  and  $n$ . That is, the latest word is always stored to precisely impression  $r$  on each coordinate. We will see that  $r > 0$  is purely a scale factor and does not affect the results in any qualitative way. Let  $S_n = e_n e_n'$ , and  $M = N - 1$ , and assume  $M > 0$ . The update rules are

$$(2.1a) \quad \begin{aligned} T_1 &= \frac{r}{M} (S_1 - I_N), \\ T_n &= T_{n-1} + D_n (S_n - I_N), \quad n \geq 2, \end{aligned}$$

where  $D_n$  is the diagonal matrix defined by

$$(2.1b) \quad D_n = \frac{r}{M} I_N - \frac{1}{M} \langle T_{n-1} S_n \rangle$$

and  $I_N$  is the  $N$ -by- $N$  identity matrix.

Note that the complexity of updating the synaptic matrix,  $T_n$ , is of order  $N^2$ , which is the same order as that of the Hebb or delta rule. Moreover, the update rule given by (2.1) has a very simple interpretation and implementation. An individual neuron  $e^i$  can be considered merely to update its incoming weights  $T_{ij}$ ,  $j \neq i$ , at a rate

proportional to  $e^i e^j$  until the threshold condition is met. Thus, like Hebb's rule, the update can be accomplished simply and in parallel, with all neurons revising their corresponding row of the  $T$  matrix simultaneously and independently.

PROPOSITION 1. For  $n \geq 1$ ,  $T_n e_n = r e_n$ .

Proof. By direct substitution and the observation that  $S_n e_n = N e_n$  and  $\langle T_{n-1} S_n \rangle e_n = T_{n-1} e_n$ .  $\square$

From (2.1) it is easy to see that the threshold,  $r$ , is simply a scaling factor and can be taken equal to  $M$ . Unlike the Hopfield case,  $T_n$ , for  $n \geq 2$ , is in general nonsymmetric. Let  $\alpha = 1 - 1/M$ ,  $\gamma = 1 - 2/M + 2/M^2$ . In the Appendix, we prove the following theorem.

THEOREM 1.

- (i)  $\mathbf{E} T_k = 0$ ,  $k \geq 1$ ,
- (ii)  $\mathbf{E} T_k T'_k = M^2(1 - \alpha^k) I_N$ ,  $k \geq 1$ ,
- (iii)  $\mathbf{E} T_{n+k} S_{n+1} = M \alpha^{k-1} I_N$ ,  $n \geq 0$ ,  $k \geq 1$ ,
- (iv)  $\mathbf{E} e'_{n+1} T'_{n+k} T_{n+k} e_{n+1} = \mathbf{E} \text{tr} T_{n+k} S_{n+1} T'_{n+k} = M^2 N(1 - \alpha^{n+1}(\alpha^{k-1} - \gamma^{k-1}))$ ,  $n \geq 0$ ,  $k \geq 1$ .

Theorem 1 implies that for all  $i$ ,  $\mathbf{I}_i(n+k, n+1)$  has a mean of  $m(n+k, n+1) = M \alpha^{k-1}$ , and a standard deviation of

$$(2.2) \quad \sigma(n+k, n+1) = M \sqrt{1 - \alpha^{2k-2} - \alpha^{n+1}(\alpha^{k-1} - \gamma^{k-1})}.$$

Define the last element stored as having lag 0, and so the  $k$ th last element stored has lag  $k-1$ . The residual impression  $\mathbf{I}_i(n+k, n+1)$  is of interest because it is nonnegative if and only if the  $i$ th bit of a word at lag  $k-1$  is stable under (1.1) after the storage of  $n+k$  words. In the transient case starting from the initial condition  $T=0$  at  $n=0$ , (called the tabula rasa case), the standard deviation for the residual impression of the  $(k-1)$ th lag work after  $k$  words are stored is  $\sigma(k, 1) \approx M \sqrt{1 - \alpha^k}$ . At steady state, the standard deviation for the residual impression of the  $(k-1)$ th lag is  $\lim_{n \rightarrow \infty} \sigma(n+k, n+1) \approx M \sqrt{1 - \alpha^{2k-2}}$ . So even if the lag is fixed, the new model shows some degradation as it approaches steady state. However, unlike the Hopfield case, the capacity does not degenerate to zero at any time.

To better understand the capacity of the new model in storing words, notice that if  $N$  is large,  $\mathbf{I}_i(n+k, n+1)$  can be approximated by a Gaussian random variable. If the word  $e_{n+1}$  is presented for retrieval after storing the  $(n+k)$ th word, one of three things may happen: (1) the system may stabilize at  $e_{n+1}$ ; (2) the system may stabilize at a state different from  $e_{n+1}$ ; (3) the system may cycle through a chain of states. Only the first case corresponds to error-free recall. The condition that  $e_{n+1}$  is a stable point is equivalent to  $\mathbf{I}_i(n+k, n+1)$  positive for all  $i$ . Let  $\phi_b$  be the  $(100b)$ th percentile point of the standard normal distribution. Let  $\text{ber}_1(n+k, n+1)$  denote the probability that  $\mathbf{I}_i(n+k, n+1)$  is negative. (By symmetry, it is easy to see that the probability is independent of  $i$ .) The term  $\text{ber}_1(n+k, n+1)$  can be thought of as the bit error rate of the word  $e_{n+1}$  if all neurons update exactly once synchronously. Under the Gaussian assumption,  $1 - \text{ber}_1(n+k, n+1) = b$  when

$$(2.3) \quad \frac{m(n+k, n+1)}{\sigma(n+k, n+1)} = \phi_b.$$

For the tabula rasa case with  $b = 0.95$ , this implies

$$\sqrt{\frac{\alpha^{2k-2}}{1 - \alpha^k}} \approx 1.64,$$

or  $\alpha^{k-1} \approx 0.78$ . At steady state, the formula becomes

$$\sqrt{\frac{\alpha^{2k-2}}{1-\alpha^{2k-2}}} \approx 1.64,$$

or  $\alpha^{k-1} \approx 0.85$ .

These expressions can be solved to yield the maximum number of words that can be stored with  $\text{ber}_1(n+k, n+1) = 0.05$ . For the tabula rasa case the result is roughly  $0.25N$ , and roughly  $0.15N$  at steady state.

Assuming the  $I_i$ 's are independent, the probability that the  $(k-1)$ th lag word stabilizes with an error,  $\text{BER}(n+k, n+1)$ , is simply

$$\text{BER}(n+k, n+1) = 1 - (1 - \text{ber}_1(n+k, n+1))^N.$$

It is clear that  $\text{BER}(n+k, n+1) \rightarrow 1$  as  $N \rightarrow \infty$ . Another quantity of interest is the probability that an arbitrary bit is in error when the retrieval process has settled to a fixed point, or that the retrieval process is oscillating, denoted  $\text{ber}_\infty(n+k, n+1)$ . The quantity  $\text{ber}_\infty(n+k, n+1)$  is hard to compute theoretically, but will be simulated and approximated in § 3.

**2.2. New model with diagonal contribution.** The new model discussed previously does not allow self-excitation ( $T_{ii}$ ) terms. Here, we want to examine their effects. Consider a second model with synaptic matrix updated by the following rules:

$$(2.4) \quad T_1^d = \frac{r}{N} S_1, \quad T_n^d = T_{n-1}^d + \frac{1}{N} (rI_N - T_{n-1}^d) S_n.$$

Then it is easy to show that  $T_n^d e_n = r e_n$ . Again, it is clear that  $r$  is simply a scaling factor, and so we choose  $r = N$ . Let  $Q_n = I_N - (S_n/N)$ . Then the update rule can be rewritten as

$$(2.5) \quad T_n^d = (T_{n-1}^d - NI_N) Q_n + NI_N.$$

Let  $\alpha_d = 1 - 1/N$ , and  $\gamma_d = 1 - 2/N + 2/N^2$ . Then the following results can be proved using techniques similar to the proof of Theorem 1.

**THEOREM 2.**

- (i)  $\mathbf{E} T_k^d = N(1 - \alpha_d^k) I_N, k \geq 1,$
- (ii)  $\mathbf{E} T_k^d T_k^{d'} = N^2(1 - \alpha_d^k) I_N, k \geq 1,$
- (iii)  $\mathbf{E} T_{n+k}^d S_{n+1} = NI_N, n \geq 0, k \geq 1,$
- (iv)  $\mathbf{E} \text{tr} T_{n+k}^d S_{n+1} T_{n+k}^{d'} = N^3(1 + \alpha_d^{n+1}(\alpha_d^{k-1} - \gamma_d^{k-1})), n \geq 0, k \geq 1.$

Theorem 2 implies that at steady state, (as  $n$  tends to infinity),  $\mathbf{E}(T_n^d - NI_N)$  ( $T_n^{d'} - NI_N$ ) =  $N^2 \alpha_d^n I_N \rightarrow 0$ . Since for a matrix  $A$ ,

$$\text{tr} AA' = \sum_{ij=1}^N A_{ij}^2,$$

this implies that  $T_n^d$  converges to  $NI_N$  with probability 1. Hence, the new model with diagonal entries collapses in the long run under the assumption of a long stream of uniform signary inputs. On the other hand, for transient use, a scheme of this type may be practical (see [2] for illustrations). It is interesting to observe that choosing  $r = 1$  and  $\eta = 1/N$  causes the new method with diagonal to coincide with the delta method of (1.3) and (1.4). Thus the result shown here regarding the convergence of  $T_n^d$  to a multiple of the identity matrix is the same as the phenomenon discussed in § 1.

**2.3. Hopfield's model.** In order to facilitate the discussion in comparing the models, we recapitulate here some results concerning Hopfield's model. The synaptic

matrix of Hopfield's model can be represented as

$$T_n^H = \sum_{i=1}^n (S_i - I_N).$$

THEOREM 3.

- (i)  $\mathbf{E} T_k^H = 0, k \geq 1,$
- (ii)  $\mathbf{E} T_k^H T_k^{H'} = kMI_N, k \geq 1,$
- (iii)  $\mathbf{E} T_{n+k}^H S_{n+1} = MI_N, n \geq 0, k \geq 1,$
- (iv)  $\mathbf{E} \text{tr} T_{n+k}^H S_{n+1} T_{n+k}^{H'} = M(M+n+k-1)N, n \geq 0, k \geq 1.$

Theorem 3 implies that the variance of  $I_i(n+k, n+1)$  is  $M(n+k-1)$ , which goes to infinity as  $n \rightarrow \infty$ . Hence, the capacity of Hopfield's model degenerates to zero for large  $n$ .

**3. Experimental results and discussion.** We now describe the experimental results, which were obtained by simulation. Three memory-storage rules were simulated: Hopfield's model (1.2), the new methods of §§ 2.1 and 2.2 (2.1) and (2.4), and the revised marginalist model ((1.6) with zero diagonal terms). In each case the model was required to store a sequence of words that were independently generated, uniform signary random strings. The state evolution rule in each case was (1.1), and the individual neurons were updated randomly and asynchronously until a fixed point or cycle was observed.

The following performance measures were instrumented as a means of relating the experimental results to the analytical results of § 2:

$\text{ber}_1(n+k, n+1)$ : The probability that the impression of an arbitrary bit is negative before any bits of the word are updated, when retrieval of the  $(n+1)$ th word stored is attempted after storage of the  $(n+k)$ th word, (i.e., lag  $k-1$ ).

$\text{ber}_\infty(n+k, n+1)$ : The probability that an arbitrary bit is recalled with error after the memory has settled to a fixed point, or that the retrieval process is oscillating (i.e., (1.1) cycles), when retrieval of the  $(n+1)$ th word stored is attempted after storage of the  $(n+k)$ th word.

$\text{BER}(n+k, n+1)$ : The block error rate, or probability that any bit of word is in error, when retrieval of the  $(n+1)$ th word stored is attempted after storage of the  $(n+k)$ th word.

Note that when the memory is presented with an exact version of a stored vector for retrieval and any bit of the word is returned in error after an application of (1.1), then the memory will either retrieve the word incorrectly or it will cycle. We count both of these events as a block error.

Figure 1 shows a transient study of the case of  $N = 100$  neurons, with 7, 10, and 15 words stored, starting from tabula rasa. We show the block error rates at lags ranging from the last word stored back to the first. The results are shown for Hopfield's and the new method (without diagonal). As expected it is seen that Hopfield's method maintains an error rate independent of the lag, but increasing sharply as more words are stored. The new method shows errors that, as expected for a short-term memory, increase as a function of lag. The new method significantly outperforms Hopfield's for recent lags and is superior except for the most distant three or four words. The results were obtained using 1000-5000 ensemble averages.

Figure 2 is another transient study where the lag is fixed at 2 (i.e., the third last word stored is attempted for retrieval),  $N = 30$  neurons, and 80 words are stored. In this case both  $\text{BER}$  and  $\text{ber}_\infty$  results are shown. These results verify the statement in § 2 that as more words are stored, Hopfield's method quickly collapses ( $\text{BER}$

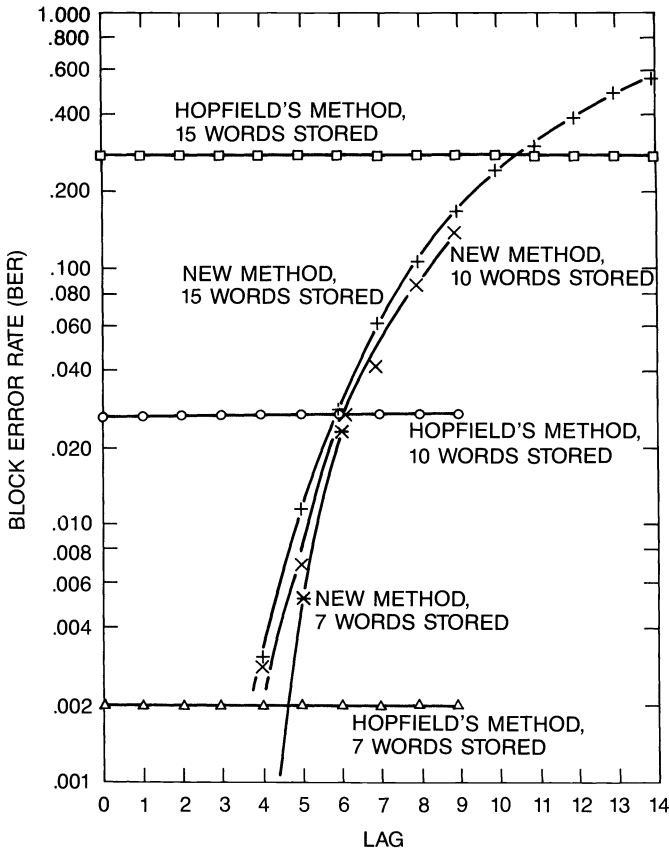


FIG. 1. Transient block error rates as a function of lag, starting from tabula rasa, for Hopfield and new methods. ( $N = 100$  neurons.)

approaches 1 when 25 words have been stored), whereas the new method settles to a bit error rate of about 5 percent and a block error rate of about 30 percent. For these results 1000 ensemble averages were used for each point.

As discussed in § 1, a second type of collapse of a short-term autoassociative memory is exhibited by the new method with diagonal, or the delta method. Figure 3 demonstrates this behavior. In this case, since the synaptic matrix is converging towards a multiple of the identity matrix (Theorem 2) it is necessary to test the ability of the memory to correct errors, or to perform autoassociative recall from the presentation of part of a word. In Fig. 3 we show results from the attempted retrieval of a word with  $b = 1, 2,$  or  $3$  bit errors introduced (at random). Shown is the probability that the iteration (1.1) (random and asynchronous updates) fails to converge to the exact initially stored word. This is denoted in Fig. 3 as  $BER_{\infty}$ , to emphasize the fact that (1.1) is iterated. Again, cycles are recorded as block errors and 1000 ensemble averages are used. It is seen that the new method with diagonal, which is equivalent to the delta method, is initially quite effective but quickly reaches a point where it is unable to correct any bit errors and consequently has  $BER_{\infty} = 1$ . In contrast, the new method without diagonal is able to correct errors at steady state.

Figures 4(a) and 4(b) concentrate on short-term memories at steady state ( $n \rightarrow \infty$ ) and compare the performance of the new method with the revised marginalist method ([13] with  $\varepsilon = 4.108$ ). These results do not utilize ensemble averages but instead use

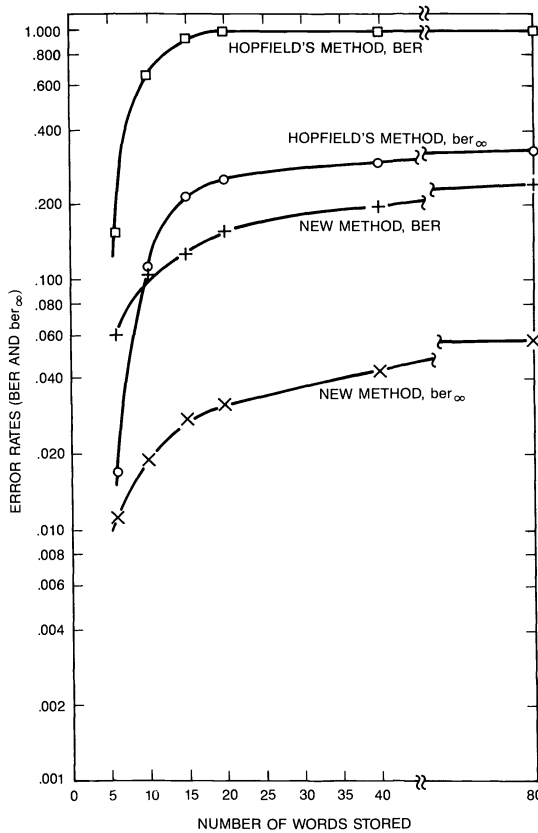


FIG. 2. Transient error rates at lag two (i.e., third last element stored), as a function of number of words stored, for the Hopfield and new methods. ( $N = 30$  neurons.)

running averages collected over long runs (allowing 200,000 to 1,000,000 observations) but after first discounting a warming-up period. Figure 4a considers the case of  $N = 30$  neurons and shows BER and  $ber_{\infty}$  for the new and revised marginalist method. Both show good performance as short-term memories. The new method exhibits smaller settled bit errors except at lags 1 and 2 and smaller block errors except at lag 1. Figure 4b repeats the experiment for  $N = 100$  and shows results for BER and  $ber_{\infty}$ , although in this case the new method was observed to have smaller settled bit error rates except at lags 2 through 6, and uniformly smaller block error rates. (See also Note Added in Proof.)

In the remainder of this section we discuss experimental results that relate back to the analysis carried out in § 2. We compare the  $ber_1$  results observed in the simulations with the results predicted using the assumption of a Gaussian distribution as proposed in § 2. The Tables 1, 2, and 3 for cases  $N = 10, 30,$  and  $100,$  respectively, report the results in their columns 1 and 2. It is seen that the approximation works well when ( $ber_1$ ) bit error rates are at least 10 percent for the case of  $N = 10,$  down to at least 2 percent for the case of  $N = 100.$  Thus this approximation might be stated to be quite accurate for the case of a large number of neurons, and particularly when the threshold bit error rate which is associated with “nominal capacity” is not too small. However, for small numbers of neurons and/or very low error rate regimes the error might be significant. It could be argued that the case of a small number of neurons is not as challenging to the investigator since these systems are easily simulated to high accuracy.

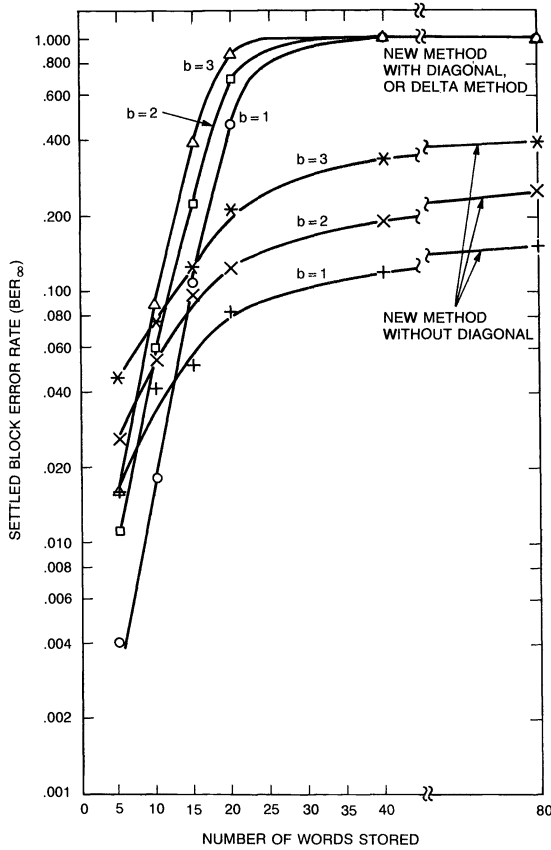


FIG. 3. Transient error rates at lag one (i.e., second last element stored), as a function of number of words stored, when the word presented for retrieval has  $b$  bit errors. Shown are the new method without and with diagonal, the latter being a delta storage rule. ( $N = 30$  neurons.)

The second analytical approximation we investigate in the tables refers to the computation of block error rates, or BER. As described in § 2, if we are willing to make an assumption of independence between bit errors in a word, the result  $BER = 1 - (1 - ber_1)^N$  is obtained. The quality of this approximation is shown in columns 3 and 4 of the tables, with and without the effect of the Gaussian approximation. That is, column 4 uses the independence assumption on the observed  $ber_1$  values, whereas column 3 uses it on the theoretically predicted  $ber_1$  values. Column 5 shows the observed values. It is seen that the independence assumption does introduce some errors at the lower error rates. The agreement between the theoretical (column 3) and observed (column 5) is seen to be fairly good, partly due to a fortuitous cancellation of errors. It should also be noted that the methodology of § 2 allows the theoretical BER results to be calculated without recourse to the independence assumption. However, this procedure involves the numerical evaluation of a multidimensional integral (first orthant probability of a Gaussian distribution), which we decided to avoid.

Our final topic in this section discusses the prospects for the computation of the  $ber_\infty$ , or settled bit error rate, statistics. Let  $ber_\infty(j)$  denote  $\lim_{n \rightarrow \infty} ber_\infty(n+j+1, n+1)$ , and  $BER(j)$  denote  $\lim_{n \rightarrow \infty} BER(n+j+1, n+1)$ . Then,

$$ber_\infty(j) = E[\text{settled bit errors at lag } j | \text{block error at lag } j] BER(j) / N.$$

We observed that the above conditional error rate was quite large, implying that when

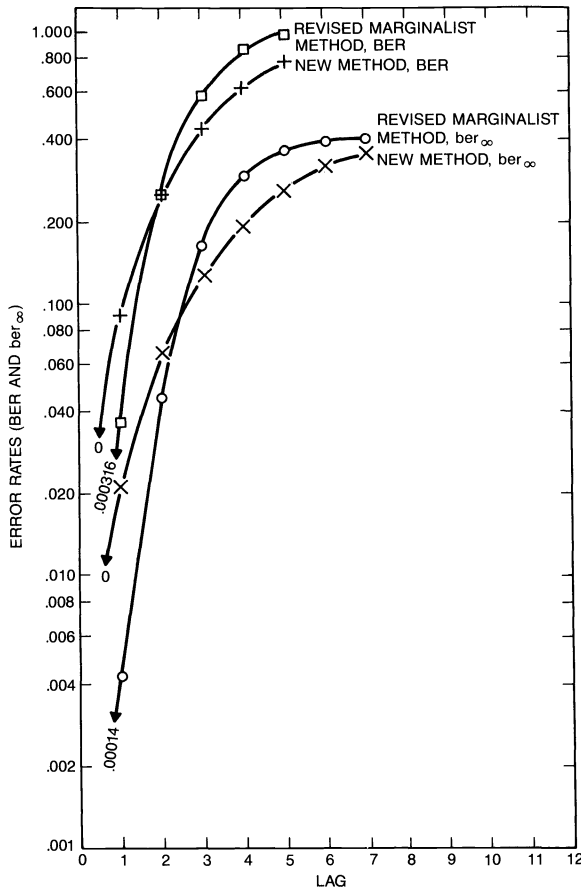


FIG. 4(a). Steady state error rates as a function of lag, for revised marginalist and new methods. ( $N = 30$  neurons.)

a block error occurred the model settled to a point far from the stored value we were trying to retrieve, i.e., the stored word was completely or almost completely lost from the memory, and convergence to another quite different stored word or spurious state occurred. The ranges of observed values are recorded in Table 4; the ranges noted are those observed as the lag varies. A simple explanation for and estimate of these observed values can be obtained as follows. If we were to ignore the erroneous and spurious words stored, then the expected number of words stored would be  $p$  pairs, where  $p = \sum_{i=0}^{\infty} (1 - \text{BER}(i))$ —the pairs arise from the observation that a word and its complement are always both fixed points of (1.1). Then when a block error occurs, we assume that the word stored is completely lost and that the memory settles to the nearest in Hamming distance of  $p$  randomly chosen words. Thus the expected number of bit errors  $Q$  given that a block error occurs is given by

$$Q = \sum_{k=0}^{\lfloor N/2 \rfloor - 1} \left( 1 - 2 \sum_{j=0}^k \binom{N}{j} \left( \frac{1}{2} \right)^N \right)^p.$$

Computational experience suggests that this expression is relatively insensitive to  $p$  in the neighborhood of its estimate given above. The values of  $Q/N$  are given in Table 4 and show a rough approximation to the observed rates. Thus combining this with



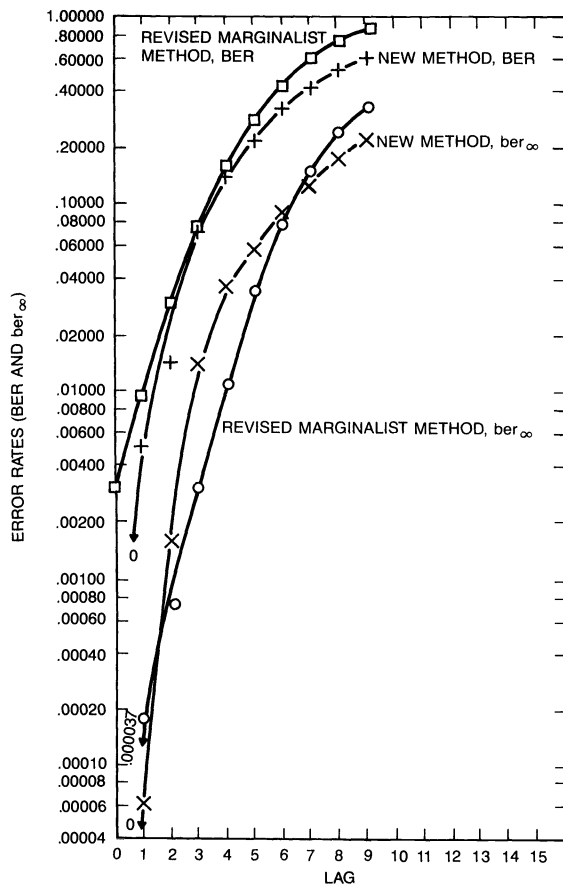


FIG. 4(b). Steady state error rates as a function of lag, for revised marginalist and new methods. ( $N = 100$  neurons.)

TABLE 1  
New method with  $N = 10$ .

	1	2	3	4	5
lag	One step ber predicted using Gaussian distribution	One step ber observed	BER predicted using independence and column 1	BER predicted using independence and column 2	BER observed
0	0	0	0	0	0
1	.026	.0487	.232	.393	.3162
2	.098	.1075	.643	.679	.6407
3	.161	.1615	.827	.828	.8427
4	.212	.2082	.908	.903	.9325

TABLE 2  
New method with  $N = 30$ .

	1	2	3	4	5
lag	One step ber predicted using Gaussian distribution	One step ber observed	BER predicted using independence and column 1	BER predicted using independence and column 2	BER observed
0	0	0	0	0	0
1	.00011	.0053	.0033	.147	.0862
2	.00495	.0161	.138	.385	.251
3	.0194	.0313	.444	.615	.442
4	.0392	.0491	.699	.779	.624
5	.0615	.0670	.853	.875	.759
6	.0838	.0866	.928	.934	.861
7	.105	.1056	.964	.965	.927

TABLE 3  
New method with  $N = 100$ .

	1	2	3	4	5
lag	One step ber predicted using Gaussian distribution	One step ber observed	BER predicted using independence and column 1	BER predicted using independence and column 2	BER observed
0	0	0	0	0	0
5	.0012	.0043	.113	.350	.215
7	.0052	.0093	.406	.607	.418
10	.017	.022	.820	.892	.701
15	.047	.050	.992	.994	.935
20	.079	.080	.9997	.998	.976

TABLE 4

$N$	$\text{ber}_{\infty}(k) \text{block error}$	$Q/N$
10	.27 ~ .34	.31
30	.23 ~ .38	.37
100	.2 ~ .48	.40

the previous methodology for finding the block error rate BER we obtain a rough method of calculation of the settled bit error rate  $\text{ber}_{\infty}$ .

**4. Conclusion.** The proposed neural network memory exhibits the behavior required of a short-term memory and its capacity compares favorably with previously proposed approaches. Its derivation is based on the simple and intuitive principle that the last word stored should be remembered correctly. The model's qualitative behavior is well explained by an analysis based on the mean and variance of the bit impression. Further work is needed to investigate the potential role of memories such as these in hierarchical memory systems and connectionist architectures.

**Appendix.** To prove Theorem 1, we need some simple algebraic formulas summarized in the following lemma.

LEMMA A: Let  $z' = (z_1, \dots, z_N)$ , where  $z_i$ 's are independent uniform signary random variables,  $S = zz'$  and  $R = I_N - (1/M)S$ . Let  $X, Y$  be  $N$ -by- $N$  matrices with coefficients independent of the  $z_i$ 's. Let  $E_z$  denote the expectation taken with respect to variables  $z_1, \dots, z_N$  only. Then

- (i)  $E_z R X R = (1 - 1/M)^2 X + (1/M^2)(X' + (\text{tr } X)I_N - 2\langle X \rangle)$ .
- (ii)  $E_z \langle X R \rangle Y R = E_z \langle R X' \rangle Y R = (1 - 2/M + 1/M^2)\langle X \rangle Y + (1/M^2)(\langle Y \rangle X + \langle X Y' \rangle - 2\langle X \rangle \langle Y \rangle)$ .
- (iii)  $E_z \langle X R \rangle^2 = (1 - 2/M)\langle X \rangle^2 + (1/M^2)\langle X X' \rangle$ .

*Proof.* To prove (i), note that

$$E_z R X R = \left(1 - \frac{2}{M}\right) X + \frac{1}{M^2} E_z S X S,$$

$$E_z S X S = \left(E_z z_i z_j \sum_{k,l=1}^N X_{kl} z_k z_l\right).$$

Part (i) follows from

$$E_z z_i z_j \sum_{k,l=1}^N X_{kl} z_k z_l = \begin{cases} X_{ij} + X_{ji}, & i \neq j, \\ \sum_{k=1}^N X_{kk}, & i = j. \end{cases}$$

The other results can be proved similarly.  $\square$

From now on we assume the symbols and notation defined in § 2. For  $k \geq 1$ , define  $\tilde{T}_k = T_k - M I_N$  and  $R_k = I_N - (1/M)S_k$ . Then, (2.1) can be rewritten as

$$T_k = \tilde{T}_{k-1} R_k - \langle \tilde{T}_{k-1} R_k \rangle.$$

Let  $E_k$  represent expectation taken with respect to variables  $e_k^1, \dots, e_k^N$  only.

*Proof of Theorem 1(i).* Clearly  $E T_1 = 0$ . The result follows by induction, since if  $E T_{k-1} = 0$ , then

$$E T_k = E \tilde{T}_{k-1} R_k - E \langle \tilde{T}_{k-1} R_k \rangle = \left(1 - \frac{1}{M}\right) (E \tilde{T}_{k-1} - E \langle \tilde{T}_{k-1} \rangle) = \left(1 - \frac{1}{M}\right) E T_{k-1} = 0. \quad \square$$

*Proof of Theorem 1(ii).* It is easy to see that  $E T_1 T_1' = M I_N = M^2(1 - \alpha) I_N$ . For  $k > 1$ ,

$$(A1) \quad \begin{aligned} E \tilde{T}_k \tilde{T}_k' &= E T_k T_k' + M^2 I_N \\ &= E \tilde{T}_{k-1} R_k^2 \tilde{T}_{k-1}' - E \langle \tilde{T}_{k-1} R_k \rangle R_k \tilde{T}_{k-1}' - E \tilde{T}_{k-1} R_k \langle R_k \tilde{T}_{k-1}' \rangle + E \langle \tilde{T}_{k-1} R_k \rangle^2 + M^2 I_N. \end{aligned}$$

Since  $E R_k^2 = (1 - 1/M + 1/M^2) I_N$ , the first term of the right-hand side of (A1) simplifies to  $(1 - 1/M + 1/M^2) \tilde{T}_k \tilde{T}_k'$ . Now let  $Y$  in Lemma A(ii) be the identity matrix. It follows that

$$\begin{aligned} E \langle \tilde{T}_{k-1} R_k \rangle R_k \tilde{T}_{k-1}' &= E (E_k \langle \tilde{T}_{k-1} R_k \rangle R_k) \tilde{T}_{k-1}' \\ &= \left(1 - \frac{2}{M}\right) E \langle \tilde{T}_{k-1} \rangle \tilde{T}_{k-1}' + \frac{1}{M^2} E \tilde{T}_{k-1} \tilde{T}_{k-1}' \\ &= \left(1 - \frac{2}{M}\right) M^2 I_N + \frac{1}{M^2} E \tilde{T}_{k-1} \tilde{T}_{k-1}'. \end{aligned}$$

Since  $\tilde{T}_{k-1}R_k\langle R_k\tilde{T}'_{k-1}\rangle$  is the transpose of  $\langle\tilde{T}_{k-1}R_k\rangle R_k\tilde{T}'_{k-1}$ , the second and third term of the right-hand side of (A1) are equal. Applying Lemma A(iii) to the fourth term, we have the following equation:

$$\mathbf{E}\tilde{T}_k\tilde{T}'_k = \left(1 - \frac{1}{M} - \frac{1}{M^2}\right)\mathbf{E}\tilde{T}_{k-1}\tilde{T}'_{k-1} + \frac{1}{M^2}\mathbf{E}\langle\tilde{T}_{k-1}\tilde{T}'_{k-1}\rangle + 2MI_N.$$

Using this relation, it is easy to show by induction that  $\mathbf{E}\tilde{T}_k\tilde{T}'_k = \mathbf{E}\langle\tilde{T}_k\tilde{T}'_k\rangle$  for all  $k$ . Hence, letting  $\alpha = 1 - 1/M$ , we have

$$\begin{aligned} \mathbf{E}\tilde{T}_k\tilde{T}'_k &= \left(1 - \frac{1}{M}\right)\mathbf{E}\tilde{T}_{k-1}\tilde{T}'_{k-1} + 2MI_N \\ &= \alpha^{k-1}\mathbf{E}\tilde{T}_1\tilde{T}'_1 + 2M(1 + \dots + \alpha^{k-2})I_N \\ &= M^2(2 - \alpha^k)I_N. \end{aligned} \quad \square$$

*Proof of Theorem 1(iii).* For  $k > 1$ ,

$$\begin{aligned} \mathbf{E}\tilde{T}_{n+k}S_{n+1} &= \mathbf{E}\tilde{T}_{n+k-1}R_{n+k}S_{n+1} - \mathbf{E}\langle\tilde{T}_{n+k-1}R_{n+k}\rangle S_{n+1} - MI_N \\ &= \left(1 - \frac{1}{M}\right)\mathbf{E}\tilde{T}_{n+k-1}S_{n+1} + \left(M\left(1 - \frac{1}{M}\right) - M\right)I_N \\ &= \alpha^{k-1}\mathbf{E}\tilde{T}_{n+1}S_{n+1} - (1 + \dots + \alpha^{k-2})I_N \\ &= M(\alpha^{k-1} - 1)I_N, \end{aligned}$$

since  $\mathbf{E}T_{n+1}S_{n+1} = MI_N$ . It follows for  $k \geq 1$  that

$$\mathbf{E}T_{n+k}S_{n+1} = M\alpha^{k-1}I_N. \quad \square$$

*Proof of Theorem 1(iv).* The result clearly holds for  $k = 1$ . For  $k > 1$ ,

$$\begin{aligned} \mathbf{E} \operatorname{tr} T_{n+k}S_{n+1}T'_{n+k} &= \mathbf{E} \operatorname{tr} T'_{n+k}T_{n+k}S_{n+1} \\ &= \mathbf{E} \operatorname{tr} R_{n+k}\tilde{T}'_{n+k-1}\tilde{T}_{n+k-1}R_{n+k}S_{n+1} \\ \text{(A2)} \quad &\quad - \mathbf{E} \operatorname{tr} \langle R_{n+k}\tilde{T}'_{n+k-1}\rangle\tilde{T}_{n+k-1}R_{n+k}S_{n+1} \\ &\quad - \mathbf{E} \operatorname{tr} R_{n+k}\tilde{T}'_{n+k-1}\langle\tilde{T}_{n+k-1}R_{n+k}\rangle S_{n+1} + \mathbf{E} \operatorname{tr} \langle\tilde{T}_{n+k-1}R_{n+k}\rangle^2 S_{n+1}. \end{aligned}$$

Applying Lemma A(i) to the first term, we have

$$\begin{aligned} \text{First term} &= \mathbf{E} \operatorname{tr} \tilde{T}_{n+k-1}(\mathbf{E}_k R_{n+k}S_{n+1}R_{n+k})\tilde{T}'_{n+k-1} \\ &= \left(1 - \frac{2}{M} + \frac{2}{M^2}\right)\mathbf{E} \operatorname{tr} \tilde{T}_{n+k-1}S_{n+1}\tilde{T}'_{n+k-1} + \left(\frac{1}{M} - \frac{1}{M^2}\right)\mathbf{E} \operatorname{tr} \tilde{T}_{n+k-1}\tilde{T}'_{n+k-1}. \end{aligned}$$

Applying Lemma A(ii) and Theorem 1(iii), we have

$$\begin{aligned} &\mathbf{E} \operatorname{tr} \langle R_{n+k}\tilde{T}'_{n+k-1}\rangle\tilde{T}_{n+k-1}R_{n+k}S_{n+1} \\ &= \left(1 - \frac{2}{M} + \frac{2}{M^2}\right)\mathbf{E} \operatorname{tr} \langle\tilde{T}_{n+k-1}\rangle\tilde{T}_{n+k-1}S_{n+1} \\ &\quad + \frac{1}{M^2}(\mathbf{E} \operatorname{tr} \langle\tilde{T}_{n+k-1}\tilde{T}'_{n+k-1}\rangle S_{n+1} - 2\mathbf{E} \operatorname{tr} \langle\tilde{T}_{n+k-1}\rangle^2 S_{n+1}) \\ &= (M-2)MN - (M^2 - 2M + 2)N\alpha^{k-2} + \frac{1}{M^2}\mathbf{E} \operatorname{tr} \tilde{T}_{n+k-1}\tilde{T}'_{n+k-1}. \end{aligned}$$

Since the third term on the right-hand side of (A2) is equal to  $\mathbf{E} \operatorname{tr} S_{n+1} R_{n+k} \tilde{T}'_{n+k-1} \langle \tilde{T}_{n+k-1} R_{n+k} \rangle$ , it is equal to the second term. Finally, by applying Lemma A(iii), we have

$$\text{Fourth term} = (M-2)MN + \frac{1}{M^2} \mathbf{E} \operatorname{tr} \tilde{T}_{n+k-1} \tilde{T}'_{n+k-1}.$$

So, letting  $\gamma = 1 - 2/M + 2/M^2$ , and using Theorem 1(ii),

$$\begin{aligned} \mathbf{E} \operatorname{tr} T'_{n+k} T_{n+k} S_{n+1} &= \gamma \mathbf{E} \operatorname{tr} \tilde{T}'_{n+k-1} \tilde{T}_{n+k-1} S_{n+1} + \left( \frac{1}{M} - \frac{2}{M^2} \right) M^2 (2 - \alpha^{n+k-1}) N \\ &\quad + (2\gamma M^2 \alpha^{k-2} - (M-2)M) N \\ &= \gamma \mathbf{E} \operatorname{tr} T'_{n+k-1} T_{n+k-1} S_{n+1} + (2 + (M-2)(2 - \alpha^{n+k-1})) N \\ &= \gamma^{k-1} \mathbf{E} \operatorname{tr} T'_{n+1} T_{n+1} S_{n+1} + 2(M-1)N(1 + \dots + \gamma^{k-2}) \\ &\quad - (M-2)N(\alpha^{n+k-1} + \alpha^{n+k-2}\gamma + \dots + \alpha^{n+1}\gamma^{k-2}) \\ &= M^2 N(1 - \alpha^{n+1}(\alpha^{k-1} - \gamma^{k-1})). \quad \square \end{aligned}$$

**Acknowledgments.** The authors thank S. Solla for her careful reading of this paper. We are also greatly indebted to a conscientious referee who made numerous suggestions that resulted in a much better paper.

**Note Added in Proof.** A generalization of the above new method which exhibits further improved performance is described in "A Decentralized Tunable Short Term Neural Network Memory and Application to Tracking", R. J. T. Morris, L. D. Rubin, and W. S. Wong, Second International Conference on Neural Networks, San Diego, CA, July 24-27, 1988.

#### REFERENCES

- [1] J. J. HOPFIELD, *Neural networks and physical systems with emergent collective computational abilities*, Proc. Nat. Acad. Sci. U.S.A., 79 (1982), pp. 2554-2558.
- [2] T. KOHONEN, *Associative Memory: A System-Theoretical Approach*, Springer-Verlag, Berlin, 1978.
- [3] G. WIDROW AND M. E. HOFF, *Adaptive switching circuits*, Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, 1960, Part 4, pp. 96-104.
- [4] T. KOHONEN, *Self Organization and Associative Memory*, Springer-Verlag, Berlin, 1984.
- [5] G. O. STONE, *An analysis of the delta rule and the learning of statistical associations*, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: Foundations, D. E. Rumelhart, J. L. McClelland, et al., MIT Press, Cambridge, MA, 1986.
- [6] D. E. RUMELHART, J. L. MCCLELLAND et al., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: Foundations, MIT Press, Cambridge, MA, 1986.
- [7] J. S. DENKER, *Neural network models of learning and adaptation*, Phys. D, 22D (1986), pp. 216-232.
- [8] F. FOGELMAN SOULIE, *Lyapunov functions and their use in automata networks*, in *Disordered Systems and Biological Organization*, E. Bienenstock et al., eds., NATO ASI Series in Systems and Computer Science, F20, Springer-Verlag, Berlin, 1986, pp. 85-100.
- [9] F. FOGELMAN SOULIE AND G. WEISBUCH, *Random iterated threshold networks and associative memory*, SIAM J. Comput., 16 (1987), pp. 203-220.
- [10] L. PERSONNAZ, I. GUYON, AND G. DREYFUS, *Neural network design for efficient information retrieval*, in *Disordered Systems and Biological Organization*, E. Bienenstock et al., eds., NATO ASI Series in Systems and Computer Science, Springer-Verlag, Berlin, 1986, pp. 227-231.
- [11] E. G. COFFMAN, JR. AND P. J. DENNING, *Operating System Theory*, Prentice-Hall, Englewood Cliffs, NJ, 1973.
- [12] J. P. NADAL, G. TOULOUSE, J. P. CHANGEUX, AND S. DEHAENE, *Networks of formal neurons and memory palimpsests*, Europhys. Lett., 10 (1986), pp. 535-542.
- [13] M. MEZARD, J. P. NADAL, AND G. TOULOUSE, *Solvable models of working memories*, J. Physique, 47 (1986), pp. 1457-1462.

## TRANSITIVE ORIENTATIONS OF GRAPHS\*

BÉLA BOLLOBÁS<sup>†‡</sup> AND GRAHAM BRIGHTWELL<sup>†</sup>

**Abstract.** Suppose that we have a set  $X$  of  $n$  objects in some unknown total order  $<$ . For  $G$  a graph on  $X$ , we ask, for each edge  $xy$ , "Is  $x < y$ ?" On processing the information thus gained we may be able to deduce more comparisons. The information we have is then in the form of a partial order  $P(G; <)$ . Let  $t(G)$  denote the maximum, over all linear orders  $<$  on  $X$ , of the number of pairs *not* related in  $P(G; <)$ ; and let  $t(n, p)$  denote the minimum of  $t(G)$  over all graphs  $G$  with  $n$  vertices and  $\lfloor pn^2/2 \rfloor$  edges. We find upper and lower bounds for  $t(n, p)$  throughout the range of  $p = p(n)$ .

**Key words.** graph, sorting algorithm, comparison sorting, orientations

**AMS(MOS) subject classifications.** 68R10, 68Q20, 05C80

**1. Introduction.** We are concerned with 2-round sorting problems of the following type. Suppose that we have a set  $V$  of size  $n$  in an unknown order  $<$ . We wish to ask a set of  $\lfloor pn^2/2 \rfloor$  questions all at once, such that no matter what answers we get we can deduce all but at most  $t(n, p)$  of the  $\binom{n}{2}$  relations. We can then, in the second round, complete our sort by asking at most  $t(n, p)$  questions. How large need  $t(n, p)$  be? Here a *question* or *probe* is a pair  $(a, b)$  of objects, and the answer reveals whether  $a < b$  or  $b < a$ .

As the probes have to be made simultaneously, the set of probes can be thought of as a graph on  $V$ . We can thus reformulate our problem in terms of graphs. Given a graph  $G = (V, E)$  of order  $n = |G|$  and size  $e(G)$ , say, consider an acyclic orientation of the edges. Let  $\vec{G} = (V, \vec{E})$  be the directed graph obtained in this way and let  $C(\vec{G})$  be the *acyclic closure* of  $\vec{G}$ :  $\vec{xy}$  is an arc of  $C(\vec{G})$  if  $\vec{G}$  contains a directed path from  $x$  to  $y$ . As every acyclic orientation of  $G$  is induced by a total order on  $V$ , we may assume that  $\vec{E}$  is induced by a total order  $<$  on  $V$ . The arcs of  $C(\vec{G})$  define a partial order  $P(G; <)$ ; we call  $P(G; <)$  the *transitive orientation* of  $G$  associated with  $<$ . So  $x < y$  in  $P(G; <)$  if there are elements  $x_1, x_2, \dots, x_r$  of  $V(G)$  such that  $x = x_1 < x_2 < \dots < x_r = y$  and  $x_i x_{i+1}$  is an edge of  $G$  for  $1 \leq i < r$ .

Thus we are interested in graphs  $G$  such that every transitive orientation of  $G$  contains many relations. Let  $r(G; <)$  denote the number of relations in  $P(G; <)$ , and let  $t(G) = \max_{<} \{ \binom{n}{2} - r(G; <) \}$ ; i.e.,  $t(G)$  is the maximum over all orders  $<$  of the number of pairs not related in  $P(G; <)$ . The question we are interested in is the following. For a graph  $G$  with  $n$  vertices and  $\lfloor pn^2/2 \rfloor$  edges, how small can  $t(G)$  be? Accordingly, let  $t(n, p) = \min \{ t(G) : |G| = n, e(G) = \lfloor pn^2/2 \rfloor \}$ . Our aim is to give bounds on  $t(n, p)$  throughout the range of  $p$ . (The notation  $t(n, p)$  is intended to suggest that the problem is closely related to random graphs, and a random graph of order  $n$ , with probability  $p$  of an edge, has about  $pn^2/2$  edges.)

One particular version of this problem is the following. How many questions do we have to ask in order to ensure that, when we know the answers to these questions, we will know all but  $o(n^2)$  of the relations? In other words, how large must  $p(n)$  be before  $t(n, p)$  becomes  $o(n^2)$ ? This question was proposed by Rabin (see [7]), and considered by the present authors [6] as well as by Ajtai et al. [1], and by Alon, Azar, and Vishkin [4].

\* Received by the editors February 9, 1987; accepted for publication December 4, 1987.

<sup>†</sup> Department of Pure Mathematics and Mathematical Statistics, University of Cambridge, Cambridge, United Kingdom CB2 1SB.

<sup>‡</sup> The work of this author was partially supported by MCS grant 8104854.

Related problems have been studied by Bollobás and Rosenfeld [8], Häggkvist and Hell [12]–[14], Ajtai, Komlós, and Szemerédi [2], [3], Bollobás and Thomason [9], and others (see Bollobás and Hell [7] and Bollobás [5]).

A slightly extended version of this paper, together with [6], occurs as § 8 of the second author’s thesis [10].

Before embarking on the results, we recall the concept of a random graph. Let  $G_{n,p} \equiv G_p$  denote a random graph on  $n$  vertices, with each pair of vertices joined by an edge with probability  $p = p(n)$ , each pair considered independently. We say that *almost every*  $G_p$  has property  $Q$ , or that  $Q$  holds *almost surely*, if the probability that  $G_{n,p}$  has  $Q$  tends to 1 as  $n \rightarrow \infty$ . For the general theory of random graphs, the reader is referred to Bollobás [5].

We note here one more piece of terminology: if  $x_1 x_2 \cdots x_r$  is a chain in  $P(G; <)$ , we say that the chain *spans*  $s$  vertices if there are precisely  $s$  vertices between  $x_1$  and  $x_r$  in  $<$ .

**2. Summary of results.** The function  $t(n, p)$  exhibits very different behaviour in three different ranges of  $p = p(n)$ . Of course, the boundaries of these ranges are chosen somewhat arbitrarily. We deal first with the middle range, with  $\frac{1}{3} \cong p \cong c \log n \log \log n / n \log \log \log n$ , it seems that almost every  $G_{n,p}$  has  $t(G_{n,p})$  close to  $t(n, p)$ . In this range, we shall prove the following general bound.

$$(1) \quad c_1 n p^{-1} \log n \leq t(n, p) \leq c_2 n p^{-1} \log(np) \log(p^{-1}).$$

(Here and throughout  $c, c_1, c_2$ , etc., are unspecified absolute constants.) It seems likely that, at any rate for most of this range, the correct order of magnitude of  $t(n, p)$  is  $n p^{-1} \log(np) \log(p^{-1}) / \log \log(np)$ . We offer several pieces of evidence in support of this conjecture: see also the remark after Theorem 8. First, we prove that, if  $p > (\log n)^3 / n$  and  $p(n) \rightarrow 0$  then, for almost every  $G_p$ ,

$$(2) \quad t(G_{n,p}) \cong \frac{c n p^{-1} \log(np) \log(p^{-1})}{\log \log(np)}.$$

In [6, Thms. 1 and 5] we proved that, for the special case  $p = \alpha \log n \log \log n / n \log \log \log n$ , with  $\alpha > 100$  a constant, that, for almost every  $G_p$ ,

$$(3) \quad \frac{n^2}{4\alpha^2} \leq t(G_{n,p}) \leq \frac{4n^2}{\alpha^{1/2}},$$

which fits in with our conjecture, and for the other end of the range we prove in this paper that our estimate is correct: if  $\frac{1}{3} \cong p \cong c_2^{-1} \log \log n / \log n$ , then

$$(4) \quad c_1 n p^{-1} \log n \leq t(n, p) \leq c_2 n p^{-1} \log n.$$

Alon, Azar, and Vishkin [4, Thm. 3.3] assert, essentially, that for  $p = c_1 n^{-1/2} \log n / (\log \log n)^{1/2}$ , we have

$$(5) \quad t(n, p) \leq \frac{c_2 n^{3/2} \log n}{(\log \log n)^{1/2}},$$

which serves to provide evidence in favour of our conjecture.

Let us give two special cases of (1). If  $p = n^{-r}$ , where  $r$  is a constant strictly between 0 and 1, we have:

$$(6) \quad c_1 n^{1+r} \log n \leq t(n, p) \leq c_2 n^{1+r} (\log n)^2,$$

and if  $p = r \log n / n$ , where  $\log \log n \leq r \leq (\log n)^s$ , then our “gap” is just  $O(\log \log n)$ :

$$(7) \quad \frac{c_1 n^2}{r} \leq t(n, p) \leq (s+1) c_2 \frac{n^2}{r} \log \log n.$$

In [10], we proved that if  $p \geq \alpha(n) \log n \log \log n / n \log \log \log n$ , where  $100 \leq \alpha(n) \leq (\log \log n)^{1/4}$ , then

$$(8) \quad t(n, p) \leq \frac{4n^2}{\alpha^{1/2}},$$

whereas ([6, Thm. 6], also Alon, Azar, and Vishkin [4, Prop. 3.5(i)], and Ajtai et al. [1, Thm. 2]), if  $p \leq c \log n / n$ , then

$$(9) \quad t(n, p) \geq \binom{n}{2} \max \left\{ \frac{4}{3c+5}, \left(1 - \frac{c}{2}\right) \left|1 - \frac{c}{2}\right| (1 - \varepsilon) \right\}.$$

Thus for smaller  $p$  the nature of the question changes, and we ask instead for the order of magnitude of  $r(n, p) = \binom{n}{2} - t(n, p)$ , the maximum number of relations a graph with  $n$  vertices and  $\lfloor pn^2/2 \rfloor$  edges can guarantee us. Here the graphs giving our lower bound are of the following form. We use about half of the edges to form a random graph on a set  $R$  of the vertices, with  $|R| = r$ , with  $r$  chosen just small enough to guarantee that, in every transitive orientation of  $G[R]$ , at least  $(8/9)\binom{r}{2}$  of the pairs of vertices of  $R$  are related. Other vertices of  $G$  have degree 1, sending one edge to  $R$ . Thus, in every transitive orientation, most of these vertices are related to a positive proportion of the vertices of  $R$ . By considering graphs of the above form, we prove the following results. For  $\frac{1}{4} \log n / n \geq p \geq 4/n$ , we have

$$(10) \quad \frac{c_1 n^3 p}{\log n} \geq \binom{n}{2} - t(n, p) \geq \frac{c_2 n^3 p \log \log \log n}{\log n \log \log n},$$

and for  $m = \lfloor pn^2/2 \rfloor \leq 2n$ , we have

$$(11) \quad \frac{c_1 m^2}{\log m} \geq \binom{n}{2} - t(n, p) \geq \frac{c_2 m^2 \log \log \log m}{\log m \log \log m}.$$

The final case we have to consider is the range  $p \geq \frac{1}{3}$ . This is covered by the following result. We prove that there are absolute constants  $d_1$  and  $d_2$  such that, whenever  $m \leq \frac{2}{3}\binom{n}{2}$ , we have

$$(12) \quad d_1 m^{1/2} \log m \leq t\left(n, 1 - m / \binom{n}{2}\right) \leq d_2 m^{1/2} \log m.$$

Also in this case random graphs  $G_p$  almost surely do not have  $t(G_{n,p})$  attaining the bound  $t(n, p)$ , up to a constant. Rather, we prove the upper bound in (12) by considering graphs where the  $m$  non-edges form a random graph on a set of  $cm^{1/2}$  vertices, and every other vertex has degree  $n - 1$ .

There are thus a variety of results in this paper, and it might be hoped that some of the ‘‘gaps’’ between our lower and upper bounds could be closed.

**3. The middle range.** Our first step in proving the results is to investigate a technique which gives a good lower bound for  $t(G)$  for many graphs  $G$ . If we have a set  $X$  of  $n$  objects,  $s$  of which are red, and we totally order  $X$  by  $<$ , then a *run* of red objects is simply an interval in  $(X, <)$  all elements of which are red. If a vertex  $x$  of our graph has many non-neighbours (the red vertices) in a set  $X$ , then, on taking a random ordering  $<$  of  $X$ , we shall find, with high probability, a longish run of non-neighbours; we can then find an extension of  $<$  to  $V(G)$  in which  $x$  is not related to any of these non-neighbours. What is more, we can do this simultaneously for many vertices  $x$  of our graph, thus yielding many pairs unrelated in  $P(G; <)$ . We first need an elementary



lemma concerning the existence of a long run. This is, of course, a well-known combinatorial problem, and variations of Lemma 1 can be found in several textbooks (for instance, David and Barton [11]). Some details are omitted in the proof.

LEMMA 1. *Suppose  $0 \leq l \leq s < n$  and  $2l + 1 < n$ . Let  $X$  be a set of  $n$  objects,  $s$  of which are red. If we take a random total ordering  $<$  of  $X$ , all orderings being equiprobable, then the probability that there is a run of  $l$  red objects in  $(X, <)$  is at least*

$$1 - \left(\frac{n-l}{s-l}\right)^l (n-s)^{-1} - \frac{2l+1}{n-2l-1}.$$

Furthermore, there is an  $n_0$  such that if  $n \geq n_0$  and  $l \leq \lfloor \log(\frac{1}{5}(n-s))/\log(n/s) \rfloor$ , then this probability is at least  $\frac{1}{2}$ .

*Proof.* Without loss of generality  $X$  is the set  $[n] = \{1, 2, \dots, n\}$ , the ordering  $<$  is the usual total ordering of  $[n]$ , and we are selecting the  $s$  red elements at random from  $[n]$ . Let  $M$  be the number of elements  $m \in [n]$  such that  $m$  is not red but  $m+1, m+2, \dots, m+l$  are. The random variable  $M$  thus depends on the set of red elements of  $[n]$ .

We see that

$$\mathbb{E}M = (n-l) \binom{n-l-1}{s-l} / \binom{n}{s},$$

and

$$\mathbb{E}M^2 \leq \mathbb{E}M + (n-l)^2 \binom{n-2l-2}{s-2l} / \binom{n}{s},$$

where  $\mathbb{E}$  denotes expectation. Hence by Chebyshev's inequality

$$\begin{aligned} P(M=0) &\leq \frac{\mathbb{E}M^2 - (\mathbb{E}M)^2}{(\mathbb{E}M)^2} \\ &\leq \frac{1}{\mathbb{E}M} + \left[ \binom{n}{s} \binom{n-2l-2}{s-2l} / \binom{n-l-1}{s-l}^2 - 1 \right]. \end{aligned}$$

Expanding the binomial coefficients and rearranging, we find that

$$\begin{aligned} P(M=0) &\leq \frac{1}{\mathbb{E}M} + \left[ \frac{(n-1)(s-l)}{(n-l-1)s} \cdot \frac{(n-2)(s-l-1)}{(n-l-2)(s-1)} \right. \\ &\quad \left. \dots \frac{(n-l)(s-2l+1)}{(n-2l)(s-l+1)} \cdot \frac{n}{n-2l-1} - 1 \right] \\ &\leq \frac{1}{\mathbb{E}M} + \frac{2l+1}{n-2l-1}. \end{aligned}$$

But

$$\mathbb{E}M \geq (n-s) \binom{s-l}{n-l}.$$

So

$$P(M=0) \leq \left(\frac{n-l}{s-l}\right)^l (n-s)^{-1} + \frac{2l+1}{n-2l-1},$$

as desired.

It is straightforward but tedious to check the second assertion, namely that for  $n$  sufficiently large and  $l = \lfloor \log(\frac{1}{5}(n-s))/\log(n/s) \rfloor$ , the expression above is at most  $\frac{1}{2}$ . For details see [10].  $\square$

Now we show how Lemma 1 can be used to find transitive orientations  $<$  of our graph  $G$  in which many pairs are unrelated.

**THEOREM 2.** *Let  $G$  be any graph. Suppose there are subsets  $U$  and  $W$  of  $V(G)$  of sizes  $|U| = u$  and  $|W| = w$ , respectively, such that every vertex of  $U$  sends at least  $\gamma$  non-edges to  $W$  (i.e., sends at most  $w - \gamma$  edges to  $W$ ) and  $w \geq n_0$ . Then  $t(G) \geq \frac{1}{2}ul(w, \gamma)$ , where*

$$l = l(w, \gamma) = \left\lfloor \frac{\log(\frac{1}{3}(w - \gamma))}{\log(w/\gamma)} \right\rfloor.$$

*Proof.* Consider a random total ordering of  $W$  (all orderings equiprobable), and let  $x$  be a vertex in  $U$ . By Lemma 1, with the “red” vertices as the elements of  $W$  which are non-neighbours of  $x$ , the probability that there is a run of  $l$  non-neighbours is at least  $\frac{1}{2}$ . Hence the expected number of vertices of  $U$  with a run of at least  $l$  non-neighbours is at least  $u/2$ . We now fix an ordering  $<$  of  $W$  such that the set  $X$  of vertices in  $U$  which do have a run of  $l$  non-neighbours in  $(W, <)$  has order at least  $u/2$ . Without loss of generality  $(W, <) = [w]$ . For each  $x \in X$ , fix some  $k(x)$  such that  $x$  is adjacent to  $k(x)$  but to none of  $k(x) + 1, \dots, k(x) + l$ .

We now extend  $<$  to a total order of  $V(G)$  as follows:

- (i) The order  $<$  remains unaltered on  $W$ .
- (ii) We choose any total order on  $V(G) \setminus (W \cup X)$ , and set  $(V(G) \setminus (W \cup X)) < W \cup X$ .
- (iii) For each  $x \in X$ , we set  $k(x) < x < k(x) + 1$ .
- (iv) We order each of the sets  $\{x \in X : k(x) = k\}$  in any way.

This is clearly an extension of  $<$  to a total order of  $V(G)$ . Also, each  $x \in X$  is incomparable with all of  $k(x) + 1, \dots, k(x) + l$  in  $P(G; <)$ . Indeed, if  $y \in X$  is above  $x$  in  $<$ , then  $k(y) \geq k(x)$ , and so  $y$  is not less than any vertex of  $W$  below  $k(y) + l$ , which is at least  $k(x) + l$ .

Therefore the  $|X|l$  relations  $x < k(x) + j$ , for  $x \in X$  and  $1 \leq j \leq l$ , are in  $<$  but not in  $P(G; <)$ , and therefore  $t(G) \geq |X|l \geq \frac{1}{2}ul$ , as desired.  $\square$

We shall make use of Theorem 2 both for large and medium values of  $p$ . We shall first consider the middle range, with  $p \leq \frac{1}{3}$  and  $pn/\log n \geq \frac{1}{2}$ . In this range, as mentioned earlier, it turns out that the upper bounds given by considering random  $G_p$ 's are at least close to best possible. Let us first prove the lower bound for  $t(n, p)$  given by (1). This in fact requires two quite different techniques, useful for different ranges of  $p$ . The first bound we prove, using Theorem 2, is good for fairly large  $p$ . Here and throughout, we omit integrality signs, which do not affect the argument. All our inequalities are claimed to hold only for  $n$  sufficiently large, even when this is not explicitly stated.

**THEOREM 3.** *If  $np \rightarrow \infty$ , and  $p \leq \frac{1}{3}$ , then  $t(n, p) \geq (1/30)np^{-1} \log(np)$ , for  $n$  sufficiently large, say  $n \geq n_1$ .*

*Proof.* Let  $G$  be any graph with  $n$  vertices and  $pn^2/2$  edges.

We consider first random partitions of  $V(G)$  into two components  $V_1$  and  $V_2$  of size  $m = n/2$ . The expected number of  $V_1 - V_2$  edges is  $pn^2/4$ . We fix one such partition with at most  $pn^2/4$  edges from  $V_1$  to  $V_2$ . At most  $n/4$  vertices of  $V_1$  send more than  $pn$  edges to  $V_2$ , and so there is a subset  $U$  of  $V_1$  with  $|U| \geq n/4$  such that no vertex of  $U$  sends more than  $pn$  edges to  $V_2$ . Hence the conditions of Theorem 2 are satisfied with  $W = V_2$  and  $\gamma = (\frac{1}{2} - p)n$ . Therefore

$$t(G) \geq \frac{1}{2} \frac{n}{4} \left\lfloor \frac{\log(\frac{1}{3}pn)}{\log((1-2p)^{-1})} \right\rfloor.$$

Since  $p \leq \frac{1}{3}$ , we have  $\log(1 - 2p) \geq -2p - 4p^2 \geq -(10/3)p$ , and hence

$$t(G) \geq \frac{n}{8} \left\lfloor \frac{3 \log(pn/5)}{10p} \right\rfloor$$

$$\geq \frac{1}{30} np^{-1} \log(pn),$$

for sufficiently large  $n$ .  $\square$

Let us note that a special case of Theorem 3 gives a result of Alon, Azar, and Vishkin [4], which is an improvement of a result of Bollobás and Thomason [9] and Häggkvist and Hell [13]. It proves that, to sort  $n$  objects in two rounds, at least  $(1/12)n^{3/2}(\log n)^{1/2}$  parallel processors are required. Bollobás and Thomason proved that  $n^{3/2} \log n$  processors suffice; this will also follow, up to a constant factor, from the upper bound on  $t(n, p)$  given by (1). In [4], Alon, Azar, and Vishkin announced that, in fact,  $n^{3/2} \log n / (\log \log n)^{1/2}$  processors suffice.

**COROLLARY 4.** *Let  $G$  be any graph with  $n$  vertices and  $(1/12)n^{3/2}(\log n)^{1/2}$  edges. If  $n$  is sufficiently large, then there is a transitive orientation of  $G$  with at most*

$$\binom{n}{2} - (1/12)n^{3/2}(\log n)^{1/2}$$

*relations.*

*Proof.* Set  $p = \frac{1}{6}n^{-1/2}(\log n)^{1/2}$  in Theorem 3.  $\square$

The drawback of the method used in the proof of Theorem 3 is that it considers only “short-range” implications, in the sense that if  $x < y$  in  $P(G; <)$  and  $y < z$ , we do not look to see whether the relation  $x < z$  is missing from  $P(G; <)$ . If  $p$  is small, this is obviously a serious restriction, and in fact we can improve on Theorem 3 by just taking a random ordering of  $V(G)$  and counting chains spanning fairly few vertices. This proof is essentially the same as one sketched in [6].

**THEOREM 5.** *Suppose that  $p(n) \rightarrow 0$  and  $(pn/\log n) \geq \frac{1}{2}$ . Then  $t(n, p) \geq (1/64)np^{-1} \log(p^{-1})$ .*

*Proof.* Let  $G$  be any graph with  $n$  vertices and  $pn^2/2$  edges. There are at most  $n/2$  vertices of degree greater than  $2pn$ . We place these vertices at the top of the ordering, and ignore them. The graph  $G'$  spanned by the remaining vertices has order  $m \geq n/2$  and maximal degree at most  $2pn$ .

We next take a random ordering  $<$  of  $V(G)$ , and calculate the expected number of chains spanning at most  $l = \frac{1}{3}p^{-1} \log(p^{-1}) \leq n/4$  vertices.

The number of chains of length  $k$  in  $G'$  is at most  $n(2pn)^k$ , and the probability that such a chain spans at most  $l$  vertices is at most

$$n \binom{l}{k} \frac{(m - k - 1)!}{m!}.$$

Hence the expected number of chains spanning at most  $l$  vertices is at most

$$\sum_{k=0}^l n^2(2pn)^k \binom{l}{k} \frac{(m - k - 1)!}{m!} \leq n \sum_{k=0}^l (2p)^k \binom{l}{k} \frac{2n^{k+1}}{m^{k+1}}$$

$$\leq 4n \sum_{k=0}^l (4p)^k \binom{l}{k}$$

$$= 4n(1 + 4p)^l.$$

Thus there is an ordering of  $G'$  such that at least  $(m-l)l-4n(1+4p)^l$  relations are omitted. But  $(1+4p)^l \leq e^{4pl} = p^{-1/2} \leq l/32$  and  $(m-l)l \geq nl/4$ , so

$$t(G) \geq t(G') \geq (m-l)l-4n(1+4p)^l \geq \frac{nl}{8} = \frac{1}{64} np^{-1} \log(p^{-1}),$$

as required.  $\square$

Combining Theorems 3 and 5, we obtain the desired lower bound for  $t(n, p)$ .

**THEOREM 6.** *Suppose  $pn/\log n \geq \frac{1}{2}$  and  $p \leq \frac{1}{3}$ . Then  $t(n, p) \geq (1/100)np^{-1} \log n$ .*

*Proof.* The maximum of  $(1/64) \log(p^{-1})$  and  $(1/30) \log(pn)$  is always at least  $(1/100) \log n$ .  $\square$

The lower bounds for  $t(n, p)$  in (1), (4), (6), and (7) follow from Theorem 6. We now turn our attention to an upper bound. As mentioned earlier, we shall prove this using the techniques of random graphs. We first give properties of a graph  $G$  which imply that  $t(G)$  is small, and then prove that almost every  $G_p$  has these properties, for appropriate values of the parameters.

**THEOREM 7.** *Suppose  $l \leq n$ , and let  $G$  be a graph on  $n$  vertices. Let  $(Q_1)$  and  $(Q_2)$  be the following properties of graphs, depending on  $l$ .*

$(Q_1)$  *For every pair  $(U, V)$  of subsets of  $V(G)$  with  $|U|=|V|=l$ , there is a  $U-V$  edge.*

$(Q_2)$  *Every triple  $(A, B, C)$  of disjoint subsets of  $V(G)$  with  $|A|=|B|=|C|=6l$  has the following property. For at most  $l^2$  pairs  $(x, y)$  with  $x \in A, y \in C$  there is no  $z \in B$  with  $xzy$  a path in  $G$ .*

(i) *If  $G$  has  $(Q_1)$ , then  $t(G) \leq 8nl \log l$ .*

(ii) *If, in addition,  $G$  has  $(Q_2)$ , then  $t(G) \leq 11nl$ .*

*Proof.* Let  $G$  be any graph on  $n$  vertices satisfying  $(Q_1)$ , and let  $<$  be any ordering of  $V(G)$ . Without loss of generality  $(V(G), <)= [n]$  with its natural order. We split  $V(G)$  into  $s = n/6l$  sets,  $V_1, \dots, V_s$ , where  $|V_i|=6l$  for each  $i$ , and  $V_1 < V_2 < \dots < V_s$ .

Suppose first that  $G$  has property  $(Q_1)$ , and it may or may not have  $(Q_2)$ . We shall find many implications in  $P(G; <)$  using just  $(Q_1)$ . Our method is to construct subsets  $W_i$  of the  $V_i$ , each of order at least  $5l$ , such that every subset  $Y$  of  $W_i$  of order at most  $l$  has at least  $2|Y|$  neighbours in  $W_{i-1}$ , for each  $i$ . Thus a vertex  $x$  in  $W_i$  will be above many vertices in  $W_{i-k}$ , for large  $k$ . Our final step will be to show that the vertices we discard in the construction of the  $W_i$  do not contribute excessive numbers of relations not in  $P(G; <)$ .

We define the  $W_i$  inductively, beginning with  $W_1 = V_1$ . Given  $W_{i-1}$  as required, set

$$\mathcal{A} = \{A \subseteq V_i: |\Gamma(A) \cap W_{i-1}| < 2|A|\}.$$

Clearly  $\mathcal{A}$  is closed under disjoint unions. We claim that  $\mathcal{A}$  contains no set of size between  $l$  and  $2l$ . Indeed, suppose that  $A \in \mathcal{A}$  and  $l \leq |A| \leq 2l$ . Then  $|W_{i-1} \setminus \Gamma(A)| > 5l - 4l = l$ , and so, since  $G$  has  $(Q_1)$ , there is an  $A - (W_{i-1} \setminus \Gamma(A))$  edge, which is not possible.

Let  $X_i$  be an element of  $\mathcal{A}$  which is maximal subject to  $|X_i| < l$ , and set  $W_i = V_i \setminus X_i$ . If  $Y \subset W_i$  has order at most  $l$ , and  $Y \in \mathcal{A}$ , then  $X_i \cup Y \in \mathcal{A}$ , but  $|X_i \cup Y| < 2l$  and so  $|X_i \cup Y| < l$ , contradicting the maximality of  $X_i$ . Therefore  $|\Gamma(Y) \cap W_{i-1}| \geq 2|Y|$  for every small subset  $Y$  of  $W_i$ , and so  $W_i$  is as desired.

Set  $W = \bigcup_{i=1}^s W_i$  and  $X = \bigcup_{i=1}^s X_i$ , so that  $V(G)$  is the disjoint union of  $W$  and  $X$ . We claim that, for every vertex  $x$  in  $W$ , there are at most  $7l \log l$  vertices  $y \in V(G)$  with  $y < x$  but  $y \not< x$  in  $P(G; <)$ . Indeed, suppose that  $x \in W_k$ , where we may assume that  $k \geq \log l$ . By induction on  $j$ , we see that for  $1 \leq j \leq \lceil \log_2 l \rceil$ , there are  $2^j$  vertices  $v$

in  $W_{k-j}$  with  $v < x$  in  $P(G; <)$ , and therefore in particular there are at least  $l$  vertices  $v$  in  $W_{k-\lceil \log_2 l \rceil}$  with  $v < x$  in  $P(G; <)$ . All but at most  $l$  vertices of  $G$  send an edge to one of these  $l$  vertices, since  $G$  has  $(Q_1)$ , and therefore there are at most  $6l(1 + \lceil \log_2 l \rceil) + l \leq 7l \log l$  vertices  $y \in V(G)$  with  $y < x$  but  $y \not\prec x$  in  $P(G; <)$ , as claimed.

Now we suppose that  $G$  also has property  $(Q_2)$ . In this case, we shall show that rather more is true: we can select a large subset  $W$  of  $V(G)$  such that, for every vertex  $x$  of  $W$ , there are at most  $8l$  vertices  $y$  of  $V(G)$  with  $y < x$  but  $y \not\prec x$  in  $P(G; <)$ .

We construct subsets  $W_i$  of  $V_i$  of size at least  $5l$  with the property that, for all  $x \in W_i$ , there are at least  $5l$  vertices  $y$  in  $V_{i-2}$  with  $y < x$  in  $P(G; <)$ . We set  $W_1 = V_1$  and  $W_2 = V_2$ . For  $k \geq 2$ , let  $A = V_{k+1}$ ,  $B = V_k$ , and  $C = V_{k-1}$ , and use  $(Q_2)$ . If  $xzy$  is a path in  $G$ , for  $x \in A$ ,  $z \in B$ , and  $y \in C$ , then  $y < x$  in  $P(G; <)$ , and so there are at most  $l^2$  pairs  $(x, y)$  with  $x \in A$ ,  $y \in C$ , and  $y \not\prec x$  in  $P(G; <)$ . Hence the set

$$X_{k+1} = \{x \in V_{k+1} : y \not\prec x \text{ in } P(G; <) \text{ for at least } l \text{ vertices } y \in V_{k-1}\}$$

has order at most  $l$ . The set  $W_{k+1} = V_{k+1} \setminus X_{k+1}$  then has the desired properties.

Every vertex  $x$  of  $W = \bigcup_{i=1}^s W_i$ , say  $x \in W_k$  with  $k \geq 3$ , has at least  $5l$  vertices  $y$  in  $V_{k-2}$  which are less than  $x$  in  $P(G; <)$ , and as before at most  $l$  vertices of  $G$  do not send an edge to one of these  $5l$  vertices. Thus at most  $8l$  vertices  $y < x$  are such that  $y \not\prec x$  in  $P(G; <)$ , as claimed.

In both cases, we have found sets  $W_i \subseteq V_i$  such that  $|W_i| \geq 5l$ , and every vertex  $x$  of  $W = \bigcup_{i=1}^s W_i$  has at most  $m$  vertices  $y$  below  $x$  with  $y \not\prec x$  in  $P(G; <)$ , where  $m \leq 7l \log l$ , and if  $(Q_2)$  holds, then  $m \leq 8l$ .

Now let us turn our attention to the vertices  $x$  in  $X = V(G) \setminus W$ . For such  $x$ , define

$$r_x = \max \{r : \{x-1, x-2, \dots, x-r\} \cap \Gamma(x) \cap W = \emptyset\},$$

so that  $x$  is joined to  $x - r_x - 1$  by an edge of  $G$ , and  $x - r_x - 1 \in W$ , but this is not true for any smaller  $r$ . There are at most  $m$  vertices below  $x - r_x - 1$  which are not less than  $x - r_x - 1$  in  $P(G; <)$ , and hence there are at most  $r_x + m$  vertices  $y$  such that  $y < x$  but  $y \not\prec x$  in  $P(G; <)$ .

Thus  $t(G)$ , the total number of pairs not related in  $P(G; <)$ , is at most  $mn + \sum_{x \in X} r_x$ . It remains to be shown that  $\sum_{x \in X} r_x$  cannot be too large.

For  $x \in X$ , let  $I_x = W \cap \{x-1, \dots, x-r_x\}$ . We note that, for  $r_x > 4l$ , we have  $|I_x| > r_x/2$ , and so

$$\sum_{x \in X} r_x \leq \frac{2}{3}nl + \sum_{x \in X} |I_x|.$$

Each  $I_x$  is an interval in  $(W, <|_W)$ , and we know that, for any subset  $Y$  of  $X$  of size  $l$ , the intersection of the  $I_x$  for  $x \in Y$  is not too large:  $|\bigcap_{x \in Y} I_x| < l$ , since otherwise  $(Y, \bigcap_{x \in Y} I_x)$  is a pair of subsets of  $V(G)$  which violates  $(Q_1)$ .

We relabel  $W$  so that  $(W, <|_W) = [l, 2l, \dots, sl] \in W$ , where  $s = |W|/l$ . Now consider the set

$$J = \{(x, j) : x \in X, \{jl, (j+1)l\} \subseteq I_x\}.$$

Essentially, we count  $|J|$  in two ways. First,  $|J| \leq (s-1)(l-1) < |W|$  since, for each  $j$ , there are at most  $l-1$  vertices  $x$  with  $(x, j) \in J$ .

But also  $|I_x| \leq l(2 + |\{j : (x, j) \in J\}|)$ , and so

$$\sum_{x \in X} |I_x| \leq l(2|X| + |J|) \leq 2ln.$$

So  $t(G) \leq mn + 3ln$ , for  $n$  sufficiently large. Therefore if  $(Q_1)$  holds, then  $t(G) \leq 8nl \log l$ , and if  $(Q_2)$  also holds, then  $t(G) \leq 11nl$ , which is as required.  $\square$

To use Theorem 7 to find bounds on  $t(n, p)$ , we need to show that there are graphs  $G$  with  $n$  vertices and  $pn^2/2$  edges with properties  $(Q_1)$  and  $(Q_2)$ , for appropriate  $l$ . As usual, we in fact show that *almost every*  $G_{n,p}$  will do.

THEOREM 8. (i) *If  $\alpha \geq 2$ ,  $p > \alpha^{-1} \log \log n / \log n$ , and  $n$  is sufficiently large, then  $t(n, p) \leq \alpha np^{-1} \log n$ .*

(ii) *If  $pn \rightarrow \infty$ , and  $n$  is sufficiently large, then  $t(n, p) \leq 50np^{-1} \log(np) \log(p^{-1})$ .*

*Proof.* We first show that, for  $l = \alpha p^{-1} \log(np)$  and  $\alpha$  any constant at least 2, almost every  $G_{n,p}$  has  $(Q_1)$ , provided  $np \rightarrow \infty$ .

Indeed, the expected number of pairs  $(U, V)$  of subsets with  $|U| = |V| = l$  and no  $U - V$  edge is at most

$$\begin{aligned} \binom{n}{l}^2 (1-p)^{l^2} &\leq \left(\frac{en}{l}\right)^{2l} e^{-pl^2} \\ &= \exp \left\{ l \left[ 2 \log \left( \frac{en}{l} \right) - pl \right] \right\} \\ &\leq \exp \left\{ l \left[ 2 \log \left( \frac{enp}{\alpha \log(np)} \right) - \alpha \log(np) \right] \right\} \\ &\rightarrow 0 \quad \text{as } n \rightarrow \infty. \end{aligned}$$

Next we show that, if  $p > \alpha^{-1} \log \log n / \log n$ , and  $l = \alpha p^{-1} \log n$ , then almost every  $G_{n,p}$  has property  $(Q_2)$ .

Indeed, the probability that  $(A, B, C)$  violates  $(Q_2)$  is at most

$$\sum_{j=l^2}^{36l^2} \binom{36l^2}{j} q^{36l^2-j} (1-q)^j,$$

where  $q = 1 - (1-p^2)^{6l}$  is the probability that there is a vertex  $z$  of  $B$  with  $xzy$  a path in  $G$ , for fixed vertices  $x$  and  $y$ . Hence the expected number of triples violating  $(Q_2)$  is rather crudely, at most

$$\begin{aligned} n^{18l} 36l^2 (36l^2)^{l^2} (1-p^2)^{6l^3} &\leq \exp \{ 18\alpha p^{-1} (\log n)^2 + 5\alpha^2 p^{-2} (\log n)^2 \log \log n \\ &\quad - 6\alpha^3 p^{-1} (\log n)^3 \} \\ &\leq \exp \{ \frac{1}{2} \alpha^3 p^{-1} (\log n)^3 + 5\alpha^3 p^{-1} (\log n)^3 - 6\alpha^3 p^{-1} (\log n)^3 \} \\ &\rightarrow 0 \quad \text{as } n \rightarrow \infty. \end{aligned}$$

Thus almost every  $G_{n,p}$  does have  $(Q_2)$ .

Applying Theorem 7(ii) with  $l = \alpha p^{-1} \log n$ , we obtain that, if  $p > \alpha^{-1} \log \log n / \log n$ , then  $t(G_{n,p}) \leq \alpha np^{-1} \log n$  almost surely, and so certainly  $t(n, p) \leq \alpha np^{-1} \log n$ , thus proving part (i) of the theorem.

We now apply Theorem 7(i) with  $l = 3p^{-1} \log(np)$ , and we find that

$$t(G_{n,p}) \leq 25np^{-1} \log(np) \log(p^{-1} \log(np)).$$

But by part (i) we may assume that  $\log(np) < p^{-1}$  and so, for  $n$  sufficiently large,

$$t(n, p) \leq 50np^{-1} \log(np) \log(p^{-1}),$$

as desired.  $\square$

Thus we have shown that inequalities (1), (4), (6), and (7) hold.

Theorem 8(ii) is probably not the best possible result. In the proof, all we use is that most small sets “spread” by a factor of 2 from one  $V_i$  to the next. In fact, it is almost certain that most sets spread by a factor of, say,  $(\log(np))^{1/2}$ . This would suggest that  $t(n, p) \leq cnp^{-1} \log(np) \log(p^{-1})/\log \log(np)$ , for some constant  $c$  and sufficiently large  $n$ . Our next result, establishing inequality (2), shows that, for random graphs, this would be best possible: almost every  $G_{n,p}$  has  $t(G_{n,p})$  at least as large as this bound.

**THEOREM 9.** *Suppose that  $p > 3 \log n \log \log n/n \log \log \log n$ , and that  $p(n) \rightarrow 0$ . Then almost every  $G_{n,p}$  has  $t(G_{n,p}) \geq np^{-1} \log(np) \log(p^{-1})/1000 \log \log(np)$ .*

*Proof.* We shall use the following elementary facts concerning random graphs, all of which can be found in Bollobás [5]. Almost every  $G_{n,p}$  has maximal degree at most  $2pn$  and chromatic number  $\chi(G_{n,p}) \leq 2pn/\log(np)$ , and almost no  $G_{n,p}$  has an independent set of size larger than  $2p^{-1} \log(np)$ .

Combining the last two facts, we see that almost every  $G_{n,p}$  has  $l \geq pn/4 \log(np)$  colour classes of size at least  $\frac{1}{4}p^{-1} \log(np)$ . The union  $H$  of these classes has order  $m$  at least  $n/16$ .

We consider orderings  $<$  of  $H$  in which colour classes appear one after the other, so that the only chains we need to consider are ones with vertices in separate colour classes. Such chains give rise to relations in  $P(G; <)$  if and only if the colour classes are in the “right” order in  $<$ .

We take a random permutation of the  $l$  colour classes of  $H$  and consider the expected number of chains of length  $k$  which span at most  $r = \log(p^{-1})/3 \log \log(np)$  classes, for  $k \leq r$ .

This is at most

$$m(2pn)^k l \binom{r}{k} \frac{(l-k-1)!}{l!} \leq 2m \left(\frac{2pn}{l}\right)^k \binom{r}{k}.$$

Thus the expected number of chains spanning at most  $r$  classes is at most

$$2m \left(1 + \frac{2pn}{l}\right)^r \leq 2m(9 \log(np))^r \leq 2mp^{-1/2}.$$

Therefore there are at least  $\frac{1}{2}mr\frac{1}{4}p^{-1} \log(np) - 2mp^{-1/2}$  relations omitted in some ordering of this form. Thus

$$t(G_{n,p}) \geq \frac{1}{1000} \frac{np^{-1} \log(np) \log(p^{-1})}{\log \log(np)},$$

as desired.  $\square$

**4. Sparse graphs.** In this section we consider graphs with fewer than  $\frac{1}{4}n \log n$  edges. As mentioned in the introduction, we are interested now in the approximate size of  $r(n, p) = \binom{n}{2} - t(n, p)$ , which is the maximum number of relations a graph of order  $n$  and size  $m \equiv \lfloor pn^2/2 \rfloor$  can guarantee us. We prove the following result, establishing inequalities (10) and (11).

**THEOREM 10.** *Let  $m = \lfloor pn^2/2 \rfloor$ .*

(i) *If  $\frac{1}{4}n \log n \geq m \geq 2n$ , then*

$$\frac{5nm}{\log n} \geq r(n, p) \geq \frac{nm \log \log \log n}{5 \times 10^5 \log n \log \log n}.$$

(ii) If  $m \leq 2n$ , then

$$\frac{10m^2}{\log m} \geq r(n, p) \geq \frac{m^2 \log \log \log m}{10^6 \log m \log \log m}.$$

*Proof.* First we suppose that  $\frac{1}{4}n \log n \geq m \geq 2n$ , and define  $r = m \log \log \log n / 6000 \log n \log \log n$ . Let  $X$  be a set of  $n$  vertices, and  $R \subseteq X$  be a subset of order  $r$ , which without loss of generality we take to be  $[r]$ . We construct a graph  $G$  on  $X$  as follows. We know from Theorem 1 of [6] (inequality (8) of this paper) with  $\alpha = 72^2$  that there is a graph  $H$  on  $R$  with at most  $2600r \log r \log \log r / \log \log \log r \leq m/2$  edges such that, for every transitive orientation  $<$  of  $H$ , there are at least  $\frac{8}{9}\binom{r}{2}$  relations in  $P(H; <)$ . We define  $G[R]$  to be equal to such an  $H$ . Next we partition the  $n - r$  vertices of  $X \setminus R$  into  $r$  sets  $S_1, \dots, S_r$  of equal size, and join  $x$  to every vertex of  $S_x$ , for  $x \in R$ . This uses at most  $n \leq m/2$  edges, and so our resulting graph  $G$  has at most  $m$  edges.

Now let  $<$  be any transitive orientation of  $G$ . We claim that there is a set  $W$  consisting of at least  $r/9$  vertices of  $R$  such that, for every vertex  $x$  of  $W$ , there are at least  $r/9$  vertices of  $R$  above  $x$  in  $P(G; <)$  and at least  $r/9$  vertices of  $R$  below  $x$  in  $P(G; <)$ . This will imply that, for every  $x \in W$  and every  $y \in S_x$ , whatever the orientation of the edge  $xy$  in  $<$ , the vertex  $y$  will be related to at least  $r/9$  elements of  $R$  in  $P(G; <)$ , and hence  $P(G; <)$  contains at least  $(n - r)/9 \cdot r/9 \geq nm \log \log \log n / 5 \times 10^5 \log n \log \log n$  relations, which gives the lower bound in (i).

Suppose then that our claim is false. For every vertex  $x \notin W$  with  $r/9 < x < 8r/9$ , there are at least  $\min\{x - r/9, 8r/9 - x\}$  vertices  $y \in R$  with  $x$  not related to  $y$  in  $P(H; <)$ . An elementary calculation now shows that there are at least  $r^2/9$  ordered pairs  $(x, y)$  of vertices of  $R$  not related in  $P(H; <)$ , which is a contradiction. Thus the lower bound in (i) is proven. (If we use some facts established in the proof of [6, Thm. 1], we can obtain rather better constants here.)

The lower bound in (ii) is obtained by putting a graph of the form described above on a subset of the vertices of order  $m/2$ , giving every other vertex degree 0, and applying part (i).

The upper bounds are fairly straightforward. We suppose first that  $\frac{1}{4}n \log n \geq m \geq n/2$ . At most  $4m/\log n$  vertices have degree at least  $\frac{1}{2} \log n$ . Consider the graph  $H$  spanned by the vertices of smaller degree, and consider a random ordering  $<$  of  $V(H)$ . The expected number of chains in  $(H; <)$  is at most

$$\sum_{l=1}^n n \left( \frac{1}{2} \log n \right)^l \frac{1}{(l+1)!} \leq ne^{(1/2) \log n} = n^{3/2}.$$

Thus there is an ordering of  $H$  with fewer than  $n^{3/2}$  chains. We take such an ordering of  $H$  and place all vertices of higher degree at the top, so that our graph has at most  $4m/\log n \cdot n + n^{3/2} < 5mn/\log n$  relations in this ordering, which gives the upper bound in (i). For (ii), we simply note that at most  $2m$  vertices have non-zero degree, and so we can delete all vertices of degree 0 and apply the above result to the remaining graph.  $\square$

**5. Dense graphs.** Finally, we turn our attention to very dense graphs, i.e., graphs with more than  $\frac{1}{3}\binom{n}{2}$  edges.

For very large values of  $p$ , a random  $G_p$  is not likely to be a good choice, i.e.,  $t(G_p)$  is almost surely much larger than the minimum of  $t(G)$  over all graphs of order  $n$  and size  $pn^2/2$ . To see this, let us consider the extreme case when  $p$  is so large that



the non-edges of  $G_p$  are almost surely independent (i.e., there is no triple  $x, y, z$  of vertices with neither  $xy$  nor  $yz$  an edge). In this case, by taking an ordering  $<$  of  $V(G)$  such that, whenever  $xy$  is a non-edge of  $G$ ,  $x$  and  $y$  are consecutive in  $<$ , we can arrange that  $t(G_p) = \binom{n}{2} - e(G_p)$ , the lowest possible value, almost surely.

We can do considerably better as follows. Let  $m$  be the (given) number of non-edges. Let  $V$  be a set of size  $n$  and  $X$  be a subset of  $V$  of size  $(3m)^{1/2} \leq n$ . We form a graph with vertex set  $V$  and  $m$  non-edges as follows. If  $x$  and  $y$  are in  $V$ , not both in  $X$ , we join  $x$  and  $y$  by an edge. We know by Theorem 3 that there is a graph  $H$  with vertex set  $X$ ,  $m$  non-edges, and  $t(H) \leq 6(3m)^{1/2} \log(3m^{1/2}) \leq 6m^{1/2} \log m$ , provided  $m$  is sufficiently large. We set  $G[X] = H$ . Our graph  $G$  thus has  $m$  non-edges, and clearly  $t(G) = t(H) \leq 6m^{1/2} \log m$ .

In fact, graphs of this type, with all vertices except those in a set  $W$  of order  $m^{1/2}$  having codegree 0, and those vertices in  $W$  having codegree about  $\frac{2}{3}m^{1/2}$ , are apparently not extremal. For instance, if we take a graph of this type and, for every vertex in  $W$ , delete  $\log m$  edges from that vertex to  $V(G) \setminus W$ , selecting different vertices each time, then the “best” ordering of  $V(G)$  still has the vertices of  $W$  appearing consecutively, so  $t(G)$  is not significantly increased. However, as shown by the next result, the bound  $6m^{1/2} \log m$  is, up to a constant factor, best possible.

**THEOREM 11.** *There are absolute constants  $\lambda, \eta > 0$  such that, whenever  $\binom{n}{2} \geq m$ , we have*

$$\lambda m^{1/2} \log m > t\left(n, 1 - m / \binom{n}{2}\right) > \eta m^{1/2} \log m.$$

*Proof.* We have already seen that the upper bound holds for  $\lambda > 6$ . We now consider the lower bound.

Let  $m_0$  be sufficiently large that every  $m \geq m_0$  satisfies the following inequalities.

- (i)  $m \geq 2n_1^2$ ;
- (ii)  $\log(m/3)/\log m > \sqrt{3}/2$ ;
- (iii)  $m^{1/2} \log m \leq m/12$ ;
- (iv)  $m \geq n_0^2$ ;
- (v)  $m^{1/4}/(\log m)^{3/2} > \frac{1}{5}$ ;
- (vi)  $\log(m^{1/2}/15) > \frac{1}{4} \log m$ .

Here  $n_0$  and  $n_1$  are the constants in Lemma 1 and Theorem 3, respectively. We shall make use of inequalities (i) to (vi) in the course of the proof, without always making explicit reference to the fact.

Our aim is to prove the result by induction on  $n$  and  $m$ . To get this induction started, we first need a few very simple facts.

We may assume that  $G$  has no vertex  $x$  of codegree at least  $m^{1/2} \log m$ , since otherwise we can place  $x$  at the top of our ordering  $<$ , and its non-neighbours immediately below it, thus forming a transitive orientation of  $G$  with at least  $m^{1/2} \log m$  pairs unrelated. Similarly, if there are  $m^{1/2} \log m$  independent non-edges  $x_i y_i$  in  $G$ , we make each pair  $(x_i, y_i)$  adjacent in  $<$ , and again we are done. Hence we may assume that there are at most  $2m(\log m)^2$  vertices of non-zero codegree in  $G$ . Furthermore, if  $G'$  is the graph obtained from  $G$  by deleting all vertices of codegree 0, then clearly  $t(G) = t(G')$ . Therefore we may assume that  $G$  has no vertices of codegree 0, and hence that  $n \leq 2m(\log m)^2$ .

We take  $\eta$  sufficiently small that  $t(n, 1 - m/\binom{n}{2}) > \eta m^{1/2} \log m$  for every  $m \leq m_0$  and  $n \leq 2m_0(\log m_0)^2$ . We also insist that  $\eta < 1/160$ .

Next we note that if  $m \geq m_0 \geq 2n_1^2$  and  $n \leq (3m)^{1/2}$ , then we are done by Theorem 3, since  $\eta < 1/30$ .

We have thus chosen a value of  $\eta$  such that our result is true for every pair  $(n, m)$  with  $m \leq m_0$  or  $n \leq (3m)^{1/2}$ . We now suppose that  $G$  has  $n > (3m)^{1/2}$  vertices,  $m > m_0$  non-edges, and that the result is true for all strictly smaller values of  $(n, m)$ .

We now eliminate the case when the non-edges of  $G$  tend to fall into two separate "components." Suppose there is a partition  $V_1 \cup V_2$  of  $V(G)$  such that  $\bar{e}(V_1), \bar{e}(V_2) \geq m/3$ . By induction there are orderings of  $G[V_1]$  and  $G[V_2]$  each omitting at least  $\eta(m^{1/2}/\sqrt{3}) \log(m/3)$  relations. So, by concatenating these orders, we see that there is an orientation of  $G$  omitting

$$\frac{2}{\sqrt{3}} \frac{\log(m/3)}{\log m} \eta m^{1/2} \log m \geq \eta m^{1/2} \log m$$

relations, since  $m > m_0$ . Therefore we may assume that there is no such partition.

We now begin the main part of the proof. We wish to apply Theorem 2, and so we need a fairly small set of vertices with many incident non-edges. Let  $W$  be a subset of  $V(G)$  of minimum order satisfying  $\bar{e}(W) \geq m/3$ , and also take  $W$  such that  $\bar{e}(W)$  is maximum subject to  $W$  being of this order  $w$ . Note that  $\bar{e}(W) \leq m/3 + m^{1/2} \log m \leq m/2$ , as we are assuming that there is no vertex of codegree greater than  $m^{1/2} \log m$ . Let  $d_w(x)$  denote the  $W$ -codegree of  $x$ , the number of non-neighbours of  $x$  which are in  $W$ . We see that  $d_w(x) \geq d_w(y)$  for every  $x \in W$  and  $y \in V(G) \setminus W$ , since otherwise  $\bar{e}(W \cup \{y\} \setminus \{x\}) > \bar{e}(W)$ . Let  $\gamma_0$  be the minimum of  $d_w(x)$  over  $x \in W$ .

Also, by our assumption on partitions of the vertex set,  $\bar{e}(V(G) \setminus W) < m/3$ , and hence there are at least  $m/3 - m^{1/2} \log m \geq m/4$  edges between  $W$  and  $V(G) \setminus W$ .

Let us give some fairly simple bounds for  $w$  and  $\gamma_0$ . First we check that  $\gamma_0$  cannot be too close to  $w$ . Indeed, if  $\gamma_0 \geq \frac{2}{3}w$ , then  $m/2 \geq \bar{e}(W) \geq \frac{1}{2}w\gamma_0 \geq w^2/3$ , so  $w \leq m^{1/2}$  and, as before, we are done by Theorem 3 applied to  $G[W]$ . Hence we may assume that  $\gamma_0 < \frac{2}{3}w$ . If, on the other hand,  $w$  is large, say  $w \geq m^{1/2} \log m$ , then consider  $U = \{y \in V(G) \setminus W : \exists x \in W \text{ with } xy \notin E(G)\}$ , the set of "coneighbours" of  $W$ . We have

$$\frac{m}{w} \geq \frac{1}{w} \sum d_w(x) \geq \gamma_0 \geq \frac{1}{|U|} \sum d_w(y) \geq \frac{m}{4|U|},$$

and so  $|U| > w/4 \geq \frac{1}{4}m^{1/2} \log m$ . For every  $y \in U$ , take a vertex  $x \in W$  with  $xy \notin E(G)$ , and say that  $y$  belongs to  $x$ . Form an ordering of  $V(G)$  in which each  $x \in W$  is immediately preceded by those  $y$  belonging to  $x$ . In such an ordering  $<$ , none of the relations  $xy$  with  $y \in U$  belonging to  $x \in W$ , are in  $P(G; <)$ , and so  $t(G) \geq \frac{1}{4}m^{1/2} \log m$ , and we are done. Thus we may and shall assume that

$$m^{1/2} \leq w \leq m^{1/2} \log m \quad \text{and} \quad \gamma_0 \leq \min \left\{ \frac{2}{3}w, \frac{m}{w} \right\}.$$

Now we shall apply Theorem 2 to  $(U, W)$  for all values of  $\gamma$  up to  $\gamma_0$ . If  $r(\gamma)$  vertices of  $U$  send at least  $\gamma$  non-edges to  $W$ , where  $15 \leq \gamma \leq \gamma_0$ , then Theorem 2 tells us that

$$\begin{aligned} t(G) &\geq \frac{1}{2} r(\gamma) \left[ \frac{\log(\frac{1}{5}(w-\gamma))}{\log(w/\gamma)} \right] \\ &\geq \frac{1}{2} r(\gamma) \left[ \frac{\log(w/15)}{\log(w/\gamma)} \right], \end{aligned}$$

since  $\gamma \leq \gamma_0 \leq \frac{2}{3}w$ . Hence if  $r(\gamma) \geq 2\eta m^{1/2} \log m [\log(w/15)/\log(w/\gamma)]^{-1}$  for any value of  $\gamma$  ( $15 \leq \gamma \leq \gamma_0$ ), we are done.

If  $\gamma \geq 4w^{1/2}$ , then  $\lceil \log(w/15)/\log(w/\gamma) \rceil \geq 2$ , so  $\lceil \log(w/15)/\log(w/\gamma) \rceil^{-1} \leq 2 \log(w/\gamma)/\log(w/15)$ .

But for  $\gamma < 4w^{1/2}$ , we certainly have the bound  $t(G) \geq r(\gamma)$ , and indeed we are done unless  $r(\gamma) \leq \eta m^{1/2} \log m$  for all  $\gamma > 0$ .

Thus we may and shall assume that the total number of  $U - W$  edges is at most

$$\begin{aligned} & 4w^{1/2} \eta m^{1/2} \log m + \sum_{\gamma=1}^{\gamma_0} 4\eta m^{1/2} \log m \frac{\log(w/\gamma)}{\log(w/15)} \\ &= 4w^{1/2} \eta m^{1/2} \log m + 4\eta m^{1/2} \frac{\log m}{\log(w/15)} [\gamma_0 \log w - \log(\gamma_0!)] \\ &\leq 4w^{1/2} \eta m^{1/2} \log m + 4\gamma_0 \eta m^{1/2} \frac{\log m}{\log(w/15)} \log(ew/\gamma_0). \end{aligned}$$

It now remains only to be shown that this quantity is always less than  $m/4$ , as that will imply that there are fewer than  $m/4$  edges from  $U$  to  $V$ , a contradiction. Thus we shall have proved the result for every pair  $(n, m)$ , and we shall be done.

The first term in the above expression is at most  $4\eta m^{3/4} (\log m)^{3/2} < m/8$ , and so it remains to be shown that  $m/8 \geq 4\gamma_0 \eta m^{1/2} (\log m / \log(w/15)) \log(ew/\gamma_0)$ , or in other words that

$$(13) \quad \frac{m^{1/2}}{\log m} \geq 32\eta \frac{\gamma_0 \log(ew/\gamma_0)}{\log(w/15)},$$

for all feasible values of  $w$  and  $\gamma_0$ .

For each fixed  $w$ , the right-hand side of (13) is increasing for  $\gamma_0 < w$ . We have that  $\gamma_0 \leq m/w \leq w$ , and so, setting  $\gamma_0 = m/w$  in (13), it is sufficient to show that

$$\frac{w \log(w/15)}{\log(ew^2/m)} \geq 32\eta m^{1/2} \log m.$$

Certainly  $\log(w/15) \geq \log(m^{1/2}/15) > \frac{1}{4} \log m$ , and  $w/\log(ew^2/m)$  is minimised at  $w = (em)^{1/2}$ . Therefore

$$\begin{aligned} \frac{w \log(w/15)}{\log(ew^2/m)} &\geq e^{1/2} m^{1/2} \frac{1}{4} \log m \frac{1}{2} \\ &\geq 32\eta m^{1/2} \log m, \end{aligned}$$

as desired, since  $\eta \leq 1/160$ . Therefore there are fewer than  $m/4$   $U - W$  edges, which is a contradiction. Thus the result is true for every pair  $(n, m)$  and we are done.  $\square$

REFERENCES

[1] M. AJTAI, J. KOMLÓS, W. STEIGER, AND E. SZEMERÉDI, *Almost sorting in one round*, preprint.  
 [2] M. AJTAI, J. KOMLÓS, AND E. SZEMERÉDI, *An  $O(n \log n)$  sorting network*, ACM Symposium on Theory of Computing, 15 (1983), pp. 1-9.  
 [3] ———, *Sorting in  $c \log n$  parallel steps*, Combinatorica, 3 (1983), pp. 1-19.  
 [4] N. ALON, Y. AZAR, AND U. VISHKIN, *Tight complexity bounds for parallel comparison sorting*, in Proc. 27th Annual Symposium on the Foundations of Computer Science, Toronto, IEEE, 1986, pp. 502-510.  
 [5] B. BOLLOBÁS, *Random Graphs*, Academic Press, London, 1985.  
 [6] B. BOLLOBÁS AND G. R. BRIGHTWELL, *Graphs whose every orientation contains almost every relation*, Israel J. Math., 59 (1987), pp. 112-128.

- [7] B. BOLLOBÁS AND P. HELL, *Sorting and graphs*, in *Graphs and Order*, I. Rival, ed., NATO ASI Series; Reidel, Dordrecht, Boston, Lancaster, 1984, pp. 169–184.
- [8] B. BOLLOBÁS AND M. ROSENFELD, *Sorting in one round*, *Israel J. Math.*, 38 (1981), pp. 154–160.
- [9] B. BOLLOBÁS AND A. THOMASON, *Parallel sorting*, *Discr. Appl. Math.*, 6 (1983), pp. 1–11.
- [10] G. R. BRIGHTWELL, *Linear extensions of partially ordered sets*, Ph.D. thesis, University of Cambridge, 1987.
- [11] F. N. DAVID AND D. E. BARTON, *Combinatorial Chance*, Griffin, London, 1962.
- [12] R. HÄGGKVIST AND P. HELL, *Graphs and parallel comparison algorithms*, *Congr. Numer.*, 29 (1980), pp. 497–509.
- [13] ———, *Parallel sorting with constant time for comparisons*, *SIAM J. Comput.*, 10 (1981), pp. 465–472.
- [14] ———, *Sorting and merging in rounds*, *SIAM J. Algebraic Discrete Methods*, 3 (1982), pp. 465–473.

## READIES AND FAILURES IN THE ALGEBRA OF COMMUNICATING PROCESSES\*

J. A. BERGSTRA†, J. W. KLOP‡, AND E.-R. OLDEROG§

**Abstract.** Readiness and failure semantics are studied in the setting of Algebra of Communicating Processes (ACP). A model of process graphs modulo readiness equivalence, respectively, failure equivalence, is constructed, and an equational axiom system is presented which is complete for this graph model. An explicit representation of the graph model is given, the failure model, whose elements are failure sets. Furthermore, a characterisation of failure equivalence is obtained as the maximal congruence which is consistent with trace semantics. By suitably restricting the communication format in ACP, this result is shown to carry over to subsets of Hoare's Communicating Sequential Processes (CSP) and Milner's Calculus of Communicating Systems (CCS). Also, the characterisation implies a full abstraction result for the failure model. In the above we restrict ourselves to finite processes without  $\tau$ -steps. At the end of the paper a comment is made on the situation for infinite processes with  $\tau$ -steps: notably we obtain that failure semantics is incompatible with Koomen's fair abstraction rule, a proof principle based on the notion of bisimulation. This is remarkable because a weaker version of Koomen's fair abstraction rule is consistent with (finite) failure semantics.

**Key words.** process algebra, concurrency, readiness semantics, failure semantics, bisimulation semantics

**AMS(MOS) subject classifications.** 68Q05, 68Q10, 68Q55, 68Q45

**Introduction.** This paper is concerned with the *failure semantics* for communicating processes as introduced by Brookes, Hoare, and Roscoe [BHR84] (see also Rounds and Brookes [RB81].) This notion of failure semantics is based on the assumption that all possible knowledge about a process takes the form of a set of pairs  $[\sigma, X]$ , where  $\sigma$  is a linear history of events (actions) in which the process has engaged in cooperation with its environment, and where  $X$  is a set of events which are impossible after  $\sigma$ . Thus failure semantics can be seen as a linear history semantics enriched by "local branching information."

Two further semantic models of processes will play an auxiliary role in our paper: Milner's model based on the notion of *observational equivalence* [Mi80] or *bisimulation* (see Park [Pa83]) and the *readiness semantics* described in [OH83]. Processes which are equivalent in the sense of bisimulation semantics are also failure equivalent, but failure semantics identifies more processes. Intermediate between bisimulation and failure semantics is the readiness semantics; here positive information  $(\sigma, Y)$  is given about a process:  $Y$  is a set of possible actions after the history  $\sigma$ .

Related to the study of failure semantics which was done by Brookes, Hoare, and Roscoe [BHR84] and Brookes [Br83] in the context of Communicating Sequential Processes (CSP) (see [Ho78], [Ho80]) is the work of De Nicola and Hennessy [DH84], where some equivalences, based on the notion of *test*, are introduced, one of which

---

\* Received by the editors February 18, 1986; accepted for publication (in revised form) December 1, 1987. This research was partially supported by the European Communities under ESPRIT contract 432, An Integrated Formal Approach to Industrial Software Development (Meteor).

† Computer Science Department, University of Amsterdam, Kruislaan 409, 1098 SJ Amsterdam and the Department of Philosophy, State University of Utrecht, Heidelberglaan 2, 3584 CS Utrecht, the Netherlands.

‡ Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, the Netherlands.

§ Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, 2300 Kiel 1, Federal Republic of Germany.

coincides on a class of simple expressions with failure equivalence. The work of De Nicola and Hennessy [DH84] takes place in the context of Milner's Calculus of Communicating Systems (CCS). Connections between CCS and CSP, in regard to failure semantics, were given by Brookes [Br83].

Most of the work just mentioned was carried out in a context where both recursion and hiding (abstraction from silent  $\tau$ -steps) were present. This combination has complicated matters significantly. The aim of our paper is therefore to investigate the "pure" failure semantics *without recursion* and *hiding* (except for an interesting digression in its final section where the intricate interplay of these phenomena is highlighted). Our context will be ACP, the axiomatic system for the *Algebra of Communicating Processes* as introduced and studied in the series of papers [BK83], [BK84a], [BK84b], [BK84c], [BK85], [BBK85], [BK86a], [BK86b]. (For an introductory survey see [BK86b].) As we shall see, one advantage of this choice is that the different communication concepts of CSP and CCS can be treated in a uniform way (cf. also Milner [Mi83] and Winskel [Wi83]). In fact, to achieve this uniformity we will work here with a mild extension of ACP, where *renaming operators* are present. This system is called  $ACP_r$  and is displayed in Table 1. Note that  $ACP_r$  is purely equational and, for a finite alphabet of actions, it is a finite axiom system.

It turns out that in our restricted setting readiness and failure semantics have a neat *axiomatisation*, by means of two equations R1,2, which on top of  $ACP_r$  yield readiness semantics, and a "saturation" axiom S which, when added to  $ACP_r + R1,2$ , yields failure semantics.  $ACP_r$  alone corresponds to bisimulation semantics. These results are established in the first part of the paper. In §§ 1-3 we construct models for these axiom systems, starting from a domain of finite process graphs on which equivalences  $\Leftrightarrow, =_{\mathcal{R}}, =_{\mathcal{F}}$  (bisimulation equivalence, readiness equivalence, failure equivalence, respectively) are divided out. Next, in § 4, the axiom systems for these quotient structures are presented and shown to be complete. The extra axioms R1,2 and S are not new; in a form disguised by many  $\tau$ 's they appear already in [Br83], and they are derivable from the axioms given in [DH84] (see our comparison in Remark 7.3.3). The definitions of  $\Leftrightarrow, =_{\mathcal{R}}, =_{\mathcal{F}}$  are also standard. What seems new in our treatment is the strategy of the completeness proofs by means of a decomposition of  $\Leftrightarrow, =_{\mathcal{R}}, =_{\mathcal{F}}$  on process graphs in a small number of very simple *process graph transformations* (§ 3).

So we obtain a "graph model" for  $ACP_r$  satisfying failure semantics. In § 5, an *explicit representation* of this graph model, called the *failure model* is constructed directly from the failure sets. This links our work with that of [BHR84]. The graph model and the failure model are shown to be isomorphic. In § 6 we restrict the general communication format of  $ACP_r$  to one-to-one communication. We show that subsets of CSP and CCS can be interpreted within this framework. This serves as a preparation for § 7, where we prove that for  $ACP_r$  with one-to-one communication failure equivalence is the *maximal trace respecting congruence*. Here traces are understood as *complete* histories recording all communications up to a final process state. This simple characterisation of failure equivalence seems new. In the proof we use the readiness semantics as a "stepping stone" towards failure equivalence. The characterisation is shown to carry over to the subsets of CSP and CCS introduced in § 6. For CCS we relate our result to the notion of testing introduced in [DH84]. Further on, the characterisation implies that for  $ACP_r$  with one-to-one communication the failure model is *fully abstract* with respect to trace equivalence.

The paper concludes in § 8 with a digression in which processes under failure semantics are considered in the context of recursion and hiding. The main point made here is that the proof principle *Koomen's Fair Abstraction Rule* (KFAR), which is

TABLE 1

ACP<sub>r</sub>

*Algebra of Communicating Processes with renaming. Here  $a, b$  range over the set  $A_\delta (= A \cup \{\delta\})$  of atomic processes or actions;  $\delta \notin A$  is a constant denoting deadlock;  $x, y, z$  range over the set of all processes which includes  $A_\delta$  and is closed under the binary operations  $+$ ,  $\cdot$ ,  $\parallel$ ,  $\llbracket \_ \rrbracket$ ,  $|$  and the unary operations  $\partial_H, a_H$ , where  $H \subseteq A$ . See § 1.2 for further explanation.*

$x + y = y + x$	A1
$x + (y + z) = (x + y) + z$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5
$x + \delta = x$	A6
$\delta x = \delta$	A7
$a b = b a$	C1
$(a b) c = a (b c)$	C2
$\delta a = \delta$	C3
$x\parallel y = x\llbracket \_ \rrbracket y + y\llbracket \_ \rrbracket x + x y$	CM1
$a\llbracket \_ \rrbracket x = ax$	CM2
$ax\llbracket \_ \rrbracket y = a(x\parallel y)$	CM3
$(x + y)\llbracket \_ \rrbracket z = x\llbracket \_ \rrbracket z + y\llbracket \_ \rrbracket z$	CM4
$ax b = (a b)x$	CM5
$a bx = (a b)x$	CM6
$ax by = (a b)(x\parallel y)$	CM7
$(x + y) z = x z + y z$	CM8
$x (y + z) = x y + x z$	CM9
$\partial_H(a) = a$ if $a \notin H$	D1
$\partial_H(a) = \delta$ if $a \in H$	D2
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	D3
$\partial_H(xy) = \partial_H(x) \cdot \partial_H(y)$	D4
$a_H(b) = b$ if $b \notin H$	RN1
$a_H(b) = a$ if $b \in H$	RN2
$a_H(x + y) = a_H(x) + a_H(y)$	RN3
$a_H(xy) = a_H(x) \cdot a_H(y)$	RN4

important in system verification and which can be justified in bisimulation semantics, is not valid in any extension of (finite) failure semantics. As far as we know this observation, which is supported by deriving a formal inconsistency, is new. Remarkably, a weaker version of KFAR turns out to be both useful for verification and consistent with finite failure semantics (see [BKO86]).

**1. The domain  $\mathbb{H}_\delta$  of finite acyclic process graphs.** In order to build a “graph model” for the axiomatisation ACP<sub>r</sub> (see Introduction, Table 1) which also satisfies failure semantics, we start by introducing a domain of process graphs ( $\mathbb{H}_\delta$ ) enriched with a number of operations  $+$ ,  $\cdot$ ,  $\parallel$ ,  $\llbracket \_ \rrbracket$ ,  $|$ ,  $\partial_H, a_H (a \in A)$  corresponding to the operators in ACP<sub>r</sub>. It should be emphasized that this structure  $\mathbb{H}_\delta(+, \cdot, \parallel, \llbracket \_ \rrbracket, |, \partial_H, a_H, a, \delta) (a \in A)$  is not yet a model of ACP<sub>r</sub>; it becomes so after dividing out by a suitable equivalence on  $\mathbb{H}_\delta$  (which, of course, should be a congruence with respect to the operations). For example, dividing out by bisimulation equivalence (as defined in § 2.3 below) yields a model of ACP<sub>r</sub>, and in fact one that is isomorphic to the initial model of ACP<sub>r</sub>. This

matter does not, however, concern us in this paper. What we are interested in is the quotient structure obtained by dividing out by readiness equivalence or failure equivalence, respectively (defined below in 2.2), that is, what we will call (in analogy with “term model”) the graph model for  $ACP_r$ , satisfying readiness semantics or failure semantics, respectively.

**1.1. Finite acyclic process graphs in  $\delta$ -normal form.** A process graph over a set is a rooted, directed multigraph whose edges are labeled by elements of this set. Let  $\mathbb{H}$  be the collection of *finite acyclic* process graphs over the alphabet  $A_\delta = A \cup \{\delta\}$  (here  $\delta \notin A$ ) consisting of actions  $a, b, \dots \in A$  and the constant  $\delta$  denoting deadlock. In the sequel we will work with  $\mathbb{H}_\delta \subseteq \mathbb{H}$ , the subset of  $\delta$ -normal process graphs. A process graph  $g \in \mathbb{H}$  is  $\delta$ -normal if whenever an edge  $s \xrightarrow{\delta} t$  occurs in  $g$ , then the node  $s$  has outdegree 1 and the node  $t$  has outdegree 0. In anthropomorphic terminology, let us say that an edge  $s \rightarrow t$  is an *ancestor* of  $s' \rightarrow t'$  if it is possible to move along edges from  $t$  to  $s'$ ; likewise the latter edge will be called a *descendant* of the former. Edges having the same initial node are *brothers*. So, a process graph  $g$  is  $\delta$ -normal if all its  $\delta$ -edges have no brothers and no descendants.

Note that for  $g \in \mathbb{H}$  the ancestor relation is a partial order on the set of edges of  $g$ .

We will now associate to a process graph  $g \in \mathbb{H}$  a unique  $g'$  in  $\delta$ -normal form, by the following procedure:

- (1) *nondeterministic  $\delta$ -removal* is the elimination of a  $\delta$ -edge having at least one brother.
- (2)  *$\delta$ -shift* of a  $\delta$ -edge  $s \xrightarrow{\delta} t$  in  $g$  consists of deleting this edge, creating a fresh node  $t'$ , and adding the edge  $s \xrightarrow{\delta} t'$ .

Now it is not hard to see that the procedure of repeatedly applying (in arbitrary order) (1), (2) in  $g$  will lead to a unique graph  $g'$  which is  $\delta$ -normal; this  $g'$  is the  *$\delta$ -normal form* of  $g$ . It is understood that pieces of the graph which have become inaccessible from the root, are discarded.

*Example 1.1.1.* See Fig. 1, where  $g'$  is the  $\delta$ -normal form of  $g$ .

**1.2. Operations on process graphs.** On  $\mathbb{H}_\delta$  we define the operations  $+, \cdot, \parallel, \llbracket, \lceil, \partial_H$ , as in [BK85], [BK86a], and moreover rename operators  $a_H$ . The constants  $a, \delta (a \in A)$  are represented by graphs consisting of a single arrow labeled by  $a, \delta$ , respectively. For the sake of completeness we repeat the definitions briefly:

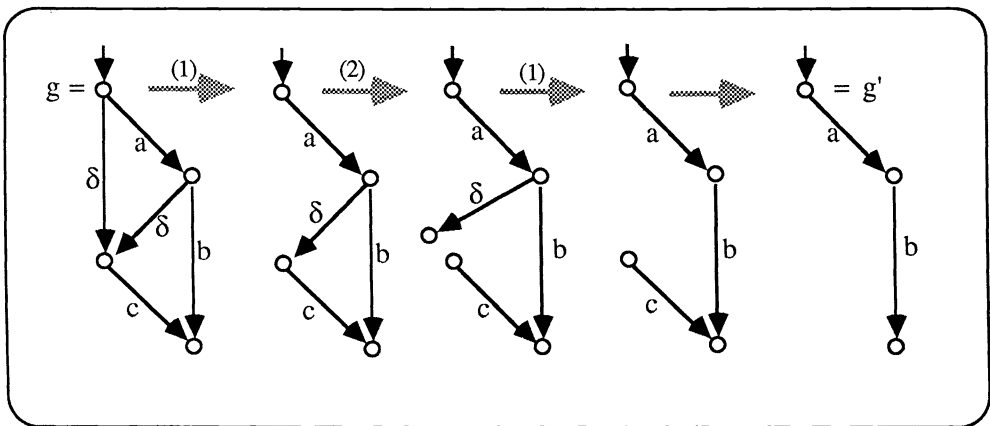


FIG. 1



(i) The *sum*  $g + h$  is the graph obtained by identifying the roots of  $g, h$  and taking the  $\delta$ -normal form (this is necessary if  $g$  or  $h$  is the graph consisting of a single step labeled with  $\delta$ ).

(ii) The *product*  $g \cdot h$  is obtained by appending  $h$  at all terminal nodes which are not terminal nodes of a  $\delta$ -step.

(iii) The *merge*  $g \parallel h$  consists of the  $\delta$ -normal form of the process graph obtained as the Cartesian product of  $g, h$  augmented with diagonal edges for successful communications.

(iv) The *left-merge*  $g \underline{\parallel} h$  is the subgraph of  $g \parallel h$ , where an initial step must be one from  $g$ .

(v) The *communication merge*  $g | h$  is the subgraph of  $g \parallel h$ , where an initial step must be a communication result of an initial step in  $g$  and an initial step in  $h$ .

(vi) The *encapsulation*  $\partial_H(g)$  is the result of renaming all (labels of) steps in  $H \subseteq A$  by  $\delta$ , and taking the  $\delta$ -normal form.

(vii) The *renaming*  $a_H(g)$  is the result of renaming all (labels of) steps in  $H \subseteq A$  by  $a$ . We have renamings  $a_H$  for each  $a \in A$ .

*Example 1.2.1.* Let  $g$  be the process graph in Fig. 2(a) and  $h$  the process graph in Fig. 2(b). Let the communication function  $|\cdot| : A_\delta \times A_\delta \rightarrow A_\delta$  be such that  $a | c = e$  and  $b | d = f$ , all other communications equal  $\delta$ . Then  $g + h$  is the graph in Fig. 2(c);  $g \cdot h$  is the graph in Fig. 2(d);  $g \parallel h$  is the  $\delta$ -normal form of the graph in Fig. 2(e), which is the graph in Fig. 2(f);  $g \underline{\parallel} h$  is the graph in Fig. 2(g);  $g | h$  is the graph in Fig. 2(h);  $\partial_{\{a,d\}}(g)$  is the graph in Fig. 2(i);  $\partial_{\{a,d\}}(h)$  is the graph in Fig. 2(j); and  $a_{(b)}(g)$  is the graph in Fig. 2(k).

**2. Equivalences on process graphs.** Though in this paper our main interest is in the *ready equivalence* and the *failure equivalence*, we also will consider trace equivalence and bisimulation equivalence. In this section these notions are introduced and compared. At the end of the section the concept of a *convexly saturated* process graph is introduced, which illuminates the relationship between ready and failure equivalence and which will play an important role in establishing the completeness of the axiom systems for ready and failure equivalence, respectively, presented in § 4.

**2.1. Trace equivalence.** Consider a process graph  $g \in \mathbb{H}_\delta$ . Every path in  $g$  from the root of  $g$  to some node in  $g$  determines a word  $s \in A_\delta^*$  formed by concatenating the labels in the consecutive steps in the path. Any such word  $\sigma$  will be called a *history* of (the path in)  $g$ . We are particularly interested in *complete histories*, i.e., words determined by paths ending in a terminal node. Throughout this paper complete histories will be called *traces*. By *trace* ( $g$ ) we denote the set of all traces of  $g$ . *Trace equivalence*  $\sim_{\text{tr}}$  of process graphs  $g, h \in \mathbb{H}_\delta$  is defined as follows:

$$g \sim_{\text{tr}} h \quad \text{iff} \quad \text{trace}(g) = \text{trace}(h).$$

Note that there are two types of traces: *successful traces*  $\sigma \in A^*$  ending in a successful termination node (see § 2.2) and *deadlocking traces*  $\sigma \cdot \delta \in A^* \cdot \{\delta\}$  ending in  $\delta$ .

**2.2. Ready equivalence and failure equivalence.** We will distinguish four types of nodes of  $g \in \mathbb{H}_\delta$ .

- (i) End nodes of  $\delta$ -steps in  $g$  are *improper*.
- (ii) Begin nodes of  $\delta$ -steps are called *deadlock nodes*.
- (iii) Termination nodes of  $g$  other than those in (i) are *successful termination nodes*.
- (iv) Nonterminal nodes which are not deadlock nodes.

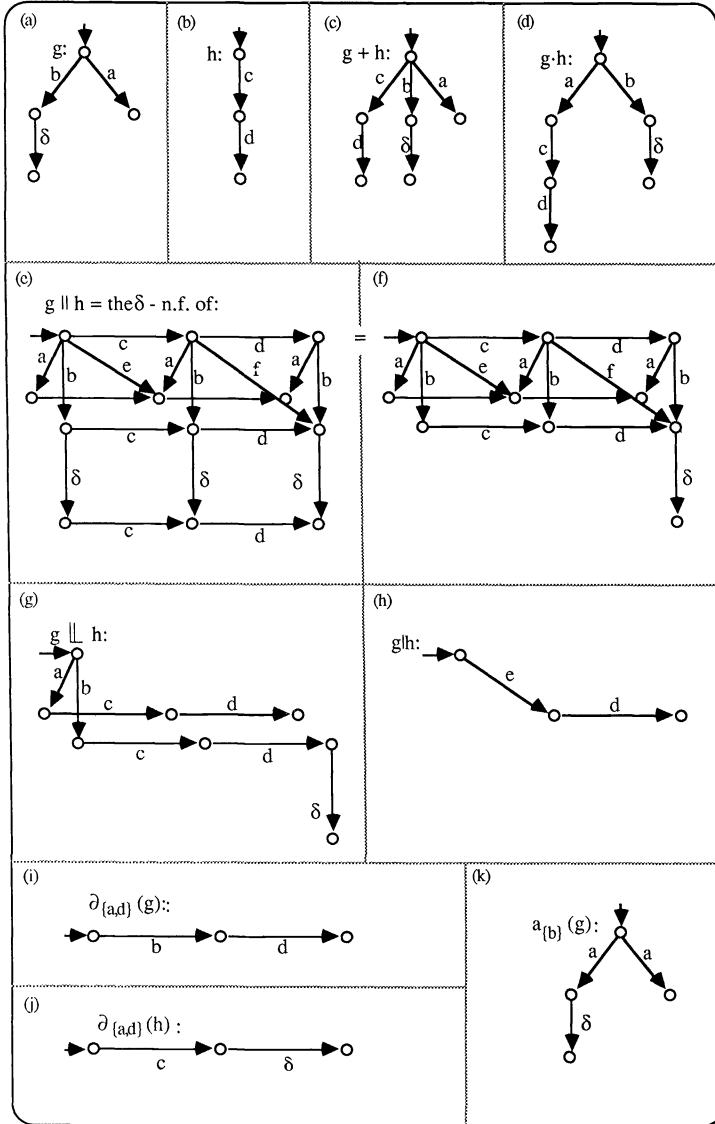


FIG. 2

The *successor set* of node  $s$  as in (ii) is, by definition,  $\emptyset$ . The successor set of a node  $s$  as in (iv) is the set of labels  $\in A$  of edges with begin node  $s$ . A node as in (i) or (iii) has no successor set.

Now  $(\sigma, X)$  where  $\sigma \in A^*$ ,  $X \subseteq A$  is a *ready pair* of  $g$  if there is a path from root  $s_0$  to some proper node  $s$  which is not a successful termination node, with history  $\sigma$  and  $X$  as the successor set of  $s$ . The *ready set* of  $g$  is the set of all ready pairs of  $g$  together with all successful traces. Notation:  $\mathcal{R}[g]$ .

The *failure set* of  $g$ , notation:  $\mathcal{F}[g]$ , is defined as follows. If  $(\sigma, X) \in \mathcal{R}[g]$ , then  $[s, Y]$  is a *failure pair* of  $g$  if  $Y \subseteq X^c$ , and  $Y$  is called a *refusal set*. Here and in the sequel we use the following notation:  $X^c = A - X$ . Now  $\mathcal{F}[g]$  is the set of all failure

pairs of  $g$ , together (again) with the successful traces of  $g$ . Thus we have

$$\mathcal{R}[g] = \{\sigma \mid \sigma \text{ is a successful trace of } g\} \cup \{(\sigma, X) \mid (\sigma, X) \text{ is a ready pair of } g\},$$

$$\mathcal{F}[g] = \{\sigma \mid \sigma \text{ is a successful trace of } g\} \cup \{[\sigma, Y] \mid Y \subseteq X^c \text{ for some } (\sigma, X) \in \mathcal{R}[g]\}.$$

Note that  $\delta$  does not appear anywhere in  $\mathcal{R}[g]$  and  $\mathcal{F}[g]$ .

*Example 2.2.1.* Consider  $g$  as in Fig. 3; at each node its type (i)–(iv) is indicated. Moreover Table 2 contains the contribution of each node to the failure and ready set of  $g$ .

*Example 2.2.2.* (i) Let  $\delta$  be the graph consisting of one  $\delta$ -step. Then  $\mathcal{R}[\delta] = \{(\varepsilon, \emptyset)\}$  and  $\mathcal{F}[\delta] = \{[\varepsilon, Y] \mid Y \subseteq A\}$ .

(ii) Let  $a \in A$ . Then  $\mathcal{R}[a] = \{(\varepsilon, \{a\}), a\}$  and  $\mathcal{F}[a] = \{a\} \cup \{[\varepsilon, Y] \mid Y \subseteq A - \{a\}\}$ .

(iii) Let  $a\delta$  be the graph consisting of a consecutive  $a$ - and  $\delta$ -step. Then  $\mathcal{R}[a\delta] = \{(\varepsilon, \{a\}), (a, \emptyset)\}$  and  $\mathcal{F}[a\delta] = \{[\varepsilon, Y] \mid Y \subseteq A - \{a\}\} \cup \{[a, Z] \mid Z \subseteq A\}$ .

**DEFINITION 2.2.3.** Let  $g, h \in \mathbb{H}_\delta$ . Then  $g \equiv_{\mathcal{R}} h$  if  $\mathcal{R}[g] = \mathcal{R}[h]$  and  $g \equiv_{\mathcal{F}} h$  if  $\mathcal{F}[g] = \mathcal{F}[h]$ . In other words,  $g, h$  are *ready equivalent*, and *failure equivalent*, respectively.

**2.3. Bisimulation equivalence.** For the sake of completeness we include the definition of the well-known notion of a bisimulation.

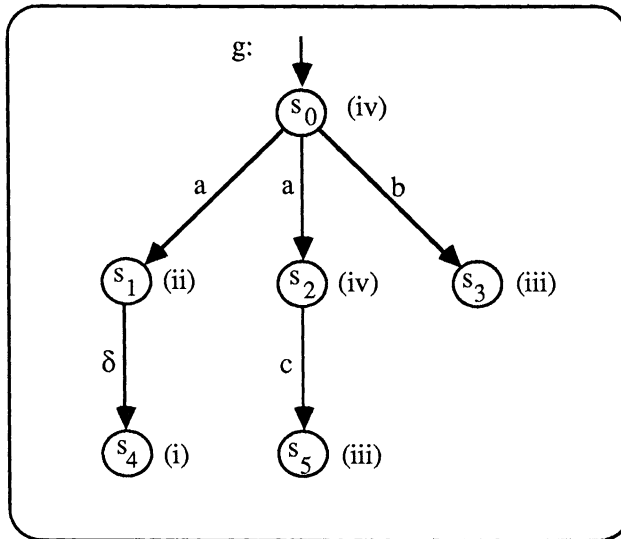


FIG. 3

TABLE 2

	$\mathcal{R}[g]$	$\mathcal{F}[g]$
$s_0$	$(\varepsilon, \{a, b\})$	$[\varepsilon, Y], Y \subseteq A - \{a, b\}$
$s_1$	$(a, \emptyset)$	$[a, Y], Y \subseteq A$
$s_2$	$(a, \{c\})$	$[a, Y], Y \subseteq A - \{c\}$
$s_3$	$b$	$b$
$s_4$		
$s_5$	$ac$	$ac$

DEFINITION 2.3.1. Let  $g, h \in \mathbb{H}_\delta$ . Let  $\text{ROOT}(g)$ ,  $\text{ROOT}(h)$  denote the root of  $g, h$ , respectively, and let  $\text{NODES}(g)$ ,  $\text{NODES}(h)$  denote the set of nodes of  $g, h$ , respectively.

Then  $R \subseteq \text{NODES}(g) \times \text{NODES}(h)$  is a *bisimulation* from  $g$  to  $h$  if:

- (i)  $(\text{ROOT}(g), \text{ROOT}(h)) \in R$ ;
- (ii) If  $(s, t) \in R$  and  $s \xrightarrow{u} s'$  (where  $u \in A_\delta$ ) is an edge in  $g$ , then  $(s', t') \in R$  for some  $t'$  such that  $t \xrightarrow{u} t'$ ;
- (iii) If  $(s, t) \in R$  and  $t \xrightarrow{u} t'$  (where  $u \in A_\delta$ ) is an edge in  $h$ , then  $(s', t') \in R$  for some  $s'$  such that  $s \xrightarrow{u} s'$ .

Notation:  $g \Leftrightarrow h$  ( $g, h$  are *bisimulation equivalent*, or *bisimilar*) if there is a bisimulation from  $g$  to  $h$  (or vice versa).

As we will want to model the axiom  $\delta \cdot x = \delta$  later, we profit here from the fact that only  $\delta$ -normal process graphs are considered. Otherwise the definition of bisimulation would be more involved.

**2.4. Comparing the equivalences.** It is not hard to compare the four equivalences  $\sim_{\text{tr}}$ ,  $\equiv_{\mathcal{R}}$ ,  $\equiv_{\mathcal{F}}$ , and  $\Leftrightarrow$ : for  $g, h \in \mathbb{H}_\delta$  we have

$$g \Leftrightarrow h \Rightarrow g \equiv_{\mathcal{R}} h \Rightarrow g \equiv_{\mathcal{F}} h \Rightarrow g \sim_{\text{tr}} h$$

and in general none of these implications can be reversed as some of the following examples (e.g., Example 2.4.2) show. Lemma 2.5.5 states a sufficient condition for reversing the second implication.

In the sequel we will prove (Proposition 4.2.3) that  $g \equiv_{\mathcal{R}} h$  and  $g \equiv_{\mathcal{F}} h$  are *congruences* with respect to the operations defined above in § 1.2. Also  $\Leftrightarrow$  is a congruence; see Theorem 2.5 of [BK85] for the more complicated situation where  $\tau$ -steps are present. Trace equivalence however is *not* a congruence with respect to these operations, as the following example shows.

Example 2.4.1. Let  $\mathcal{C}[\xi]$  be the context  $\partial_{\{b,c\}}(\xi \parallel c)$ , and let  $a, b, b^o, c, c^o$  be atoms with communications  $b \mid b = b^o, c \mid c = c^o$  and all other communications resulting in  $\delta$ . Consider the trace equivalent processes  $a(b + c)$  and  $ab + ac$ . Then  $\mathcal{C}[a(b + c)] = ac^o \uparrow_{\text{tr}} ad + ac^o = \mathcal{C}[ab + ac]$ .

Example 2.4.2. See Fig. 4.

**2.5. Convexly saturated process graphs.** Following [Br83] and [DH84] we introduce the following notion of convexity.

DEFINITION 2.5.1.  $\mathcal{X} \subseteq \mathcal{P}(A)$  is *convex* if

- (i)  $X, Y \in \mathcal{X} \Rightarrow X \cup Y \in \mathcal{X}$ ;
- (ii)  $X, Y \in \mathcal{X}, X \subseteq Z \subseteq Y \Rightarrow Z \in \mathcal{X}$ .

(Here  $\mathcal{P}(A)$  is the power set of  $A$ . In particular,  $\emptyset \subseteq \mathcal{P}(A)$  is convex.)

DEFINITION 2.5.2. (i) Let  $g \in \mathbb{H}_\delta$  and  $\sigma \in A^*$ . Then  $g \mid \sigma = \{X \mid (\sigma, X) \in \mathcal{R}[\![g]\!]\}$ .

(ii)  $g$  is *convexly saturated* (or just “convex” or “saturated”) if  $g \mid \sigma$  is convex, for all  $\sigma \in A^*$ .

Example 2.5.3. In Fig. 5,  $g_1, g_2$  are not convexly saturated, but their “convex saturations”  $g'_1, g'_2$  are.

PROPOSITION 2.5.4. Let  $\mathcal{X} \subseteq \mathcal{P}(A)$  be convex, and let  $Y \subseteq A$  be a finite set such that  $Y \notin \mathcal{X}, Y \subseteq \cup \mathcal{X}$ . Then for no  $X \in \mathcal{X}$  we have  $Y^c \subseteq X^c$ .

Proof. Consider a finite  $Y$  such that  $Y \notin \mathcal{X}, Y \subseteq \cup \mathcal{X}$ . Suppose that there is an  $X \in \mathcal{X}$  such that  $Y^c \subseteq X^c$ , or equivalently  $X \subseteq Y$ . Clearly,  $Y$  is covered by finitely many members from  $\mathcal{X}$ , hence (since  $\mathcal{X}$  is convex) by some  $Z \in \mathcal{X}$ . From  $X \subseteq Y \subseteq Z$  it follows that  $Y \in \mathcal{X}$ , a contradiction.  $\square$

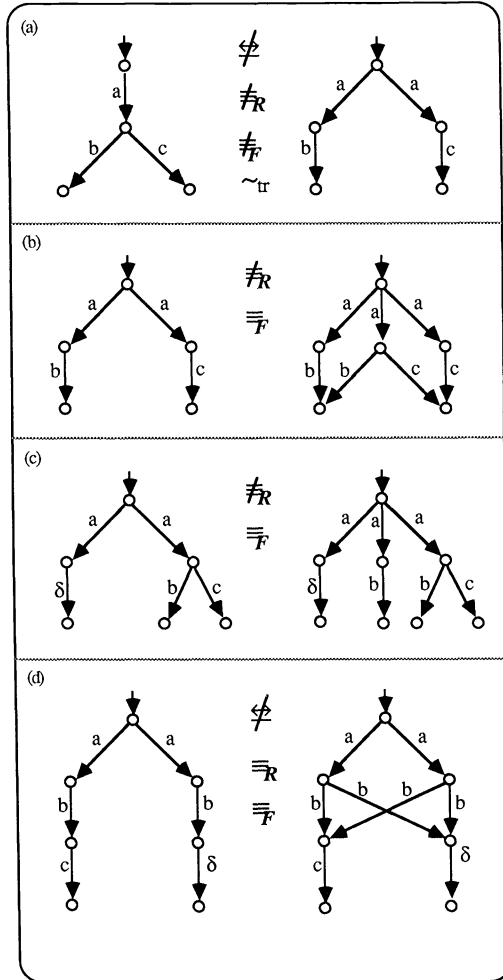


FIG. 4

LEMMA 2.5.5. *Let  $g, h \in \mathbb{H}_\delta$  be convexly saturated. Then*

$$g \equiv_{\mathcal{R}} h \Leftrightarrow g \equiv_{\mathcal{F}} h.$$

*Proof.* Only to prove ( $\Leftarrow$ ). So, we suppose  $g \not\equiv_{\mathcal{R}} h$  and we want to prove  $g \not\equiv_{\mathcal{F}} h$ . Furthermore, we may suppose that  $g, h$  have the same trace set; otherwise  $g \not\equiv_{\mathcal{F}} h$  is immediate. Now there is a ready pair  $(\sigma, X)$  in (say)  $\mathcal{R}[[g]]$  but not in  $\mathcal{R}[[h]]$ . By  $(\sigma, X) \in \mathcal{R}[[g]]$  we have the failure pair  $[\sigma, X^c] \in \mathcal{F}[[g]]$ . Now consider  $h \upharpoonright \sigma$ , which is by assumption convex. Since  $g \sim_{\text{tr}} h$ , we have  $X \subseteq \cup (h \upharpoonright \sigma)$ . Furthermore,  $(\sigma, X) \notin \mathcal{R}[[h]]$  entails  $X \not\subseteq h \upharpoonright \sigma = \{X_i \mid i \in I\}$ . So, by Proposition 2.5.4: for no  $i \in I$  we have  $X^c \subseteq X_i^c$ . But then  $[\sigma, X^c] \notin \mathcal{F}[[h]]$  and we have  $g \not\equiv_{\mathcal{F}} h$ .  $\square$

**3. Transformations on process graphs.** We now introduce four *elementary transformations* on process graphs  $\in \mathbb{H}_\delta$  with the following property: the first two of them generate, when applied on  $g \in \mathbb{H}_\delta$ , all process graphs  $g'$  *bisimilar* to  $g$ ; further, the first three generate the *ready equivalence* class of  $g$ ; and finally, the four together generate the *failure equivalence* class of  $g$ .

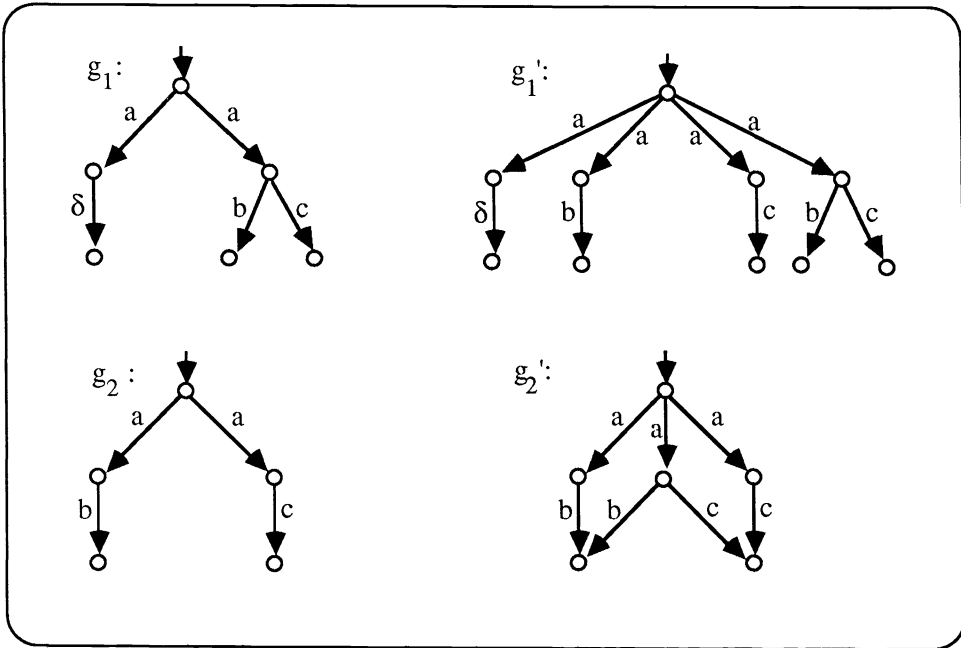


FIG. 5

**3.1. The transformations double edge, sharing, cross, and fork.**

[i] **Double edge.** This process graph transformation step removes in a *double edge* as in Fig. 6 (where  $a \in A$ ), one of the edges. Notation:  $g \Rightarrow_{[i]} h$ .

[ii] **Sharing.** Suppose  $g \in \mathbb{H}_\delta$  contains two nodes  $s, t$  determining isomorphic subgraphs  $(g)_s, (g)_t$ . Then the nodes  $s, t$  may be identified. Notation:  $g \Rightarrow_{[ii]} h$ .

[iii] **Cross.** If  $s$  is a node of  $g \in \mathbb{H}_\delta$ ,  $\sigma$  is a *history* of  $s$  if there is a path from the root of  $g$  to  $s$  yielding the word  $\sigma$ . Furthermore,  $\text{history}(s)$  is the set of all histories of  $s$ . Now let  $g \in \mathbb{H}_\delta$  contain a part as in Fig. 7(a), where  $\text{history}(s_1) = \text{history}(s_2)$ . Then edges, as in Fig. 7(b), may be inserted. Notation:  $g \Rightarrow_{[iii]} h$ .

Note that the condition on histories is fulfilled when  $g$  is a process tree. Furthermore, note that the condition on histories is necessary: it is easy to give an example where this requirement is violated and such that after insertion of the two new steps we have new ready pairs or new completed traces.

[iv] **Fork.** Let  $g \in \mathbb{H}_\delta$  contain a part as in Fig. 8(a), where all successor steps  $b_1, \dots, b_n$  of the left  $a$ -step are displayed. Then a part as indicated in Fig. 8(b) may be inserted. Notation:  $g \Rightarrow_{[iv]} h$ .

Here it is not required that all steps  $b_1, \dots, b_n, c_1, \dots, c_m$  have different end nodes. If  $n = 1, b_1$  may be  $\delta$ ; likewise  $c_1$  may be  $\delta$ . In such a case, after inserting the fork we have to  $\delta$ -normalise the resulting graph again. We emphasize that a fork connects all of the successor steps of the left  $a$ -step with some of those of the right  $a$ -step.

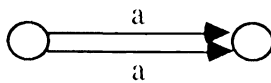


FIG. 6

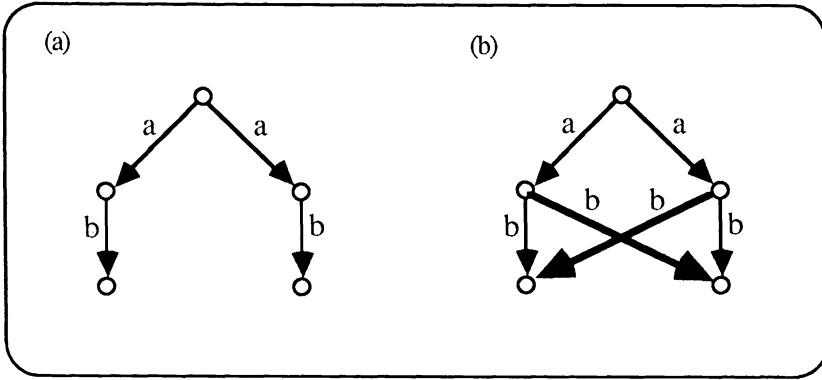


FIG. 7

Notation 3.1.1. (i)  $\Rightarrow$  is  $\Rightarrow_{[i]} \cup \dots \cup \Rightarrow_{[iv]}$ ;

(ii)  $\Rightarrow^*$  is the transitive reflexive closure of  $\Rightarrow$ ;

(iii)  $\Leftrightarrow^*$  is the equivalence relation generated by  $\Rightarrow$ .

Example 3.1.2. (i) See Fig. 9. Note how  $\Rightarrow_{[iii]}$  enables us to switch subgraphs  $x, y$  at the end of paths with the same history ( $abc$  in Fig. 9(b)).

(ii) (See Fig. 10.) Figure 10(a) contains an example of a fork transformation. Figure 10(b) contains an example of a fork transformation involving a  $\delta$ -step. Figure 10(c) shows that complete branches can be pruned by successive transformations.

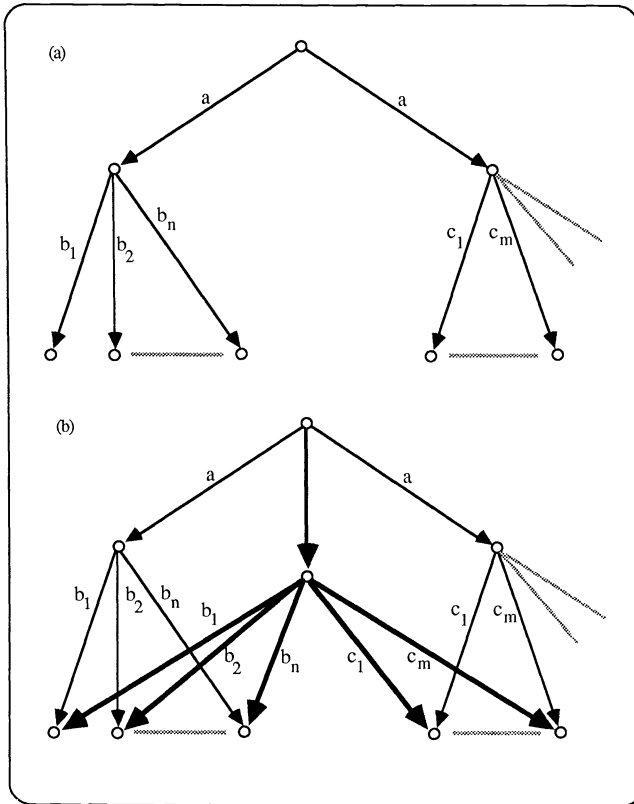


FIG. 8

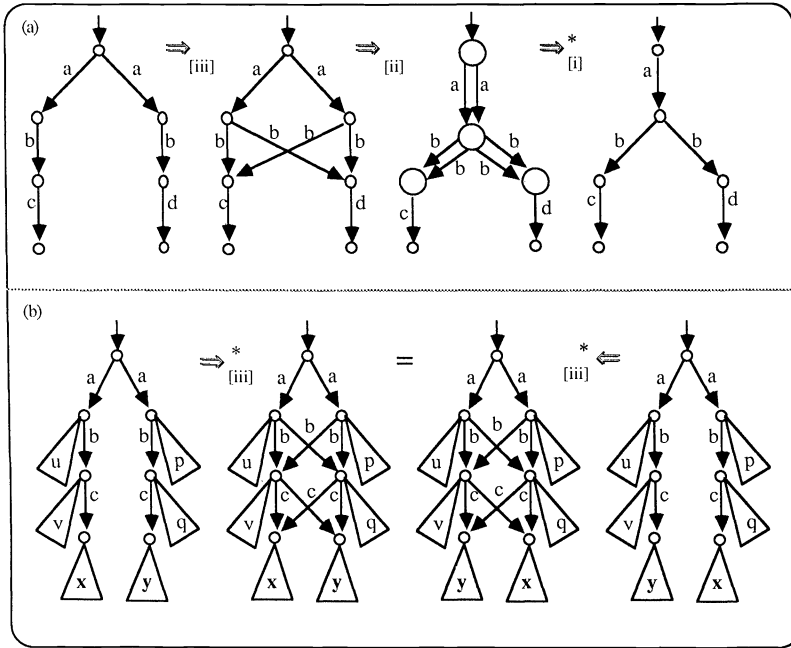


FIG. 9

DEFINITION 3.1.3. (i) A transformation step  $g \Rightarrow_{[iii]} h$  is called *restricted* if  $g$  is a process tree (i.e., a process graph without sharing of subgraphs).

(ii) Let  $\Leftrightarrow$  be the symmetric closure of  $\Rightarrow$ . A transformation  $g \Leftrightarrow \dots \Leftrightarrow h$  is restricted if every  $[iii]$ -step in the transformation is restricted.

**3.2. Connecting process graph equivalences with process graph transformations.**

PROPOSITION 3.2.1. Let  $g, h \in H_\delta$ . Then we have the following:

- (i)  $g \Rightarrow_{[i-iii]} h$  implies  $g \equiv_{\mathcal{R}} h$ ;
- (ii)  $g \Rightarrow_{[i-iv]} h$  implies  $g \equiv_{\mathcal{F}} h$ .

*Proof.* Item (i) follows at once from the definitions.

(ii) We must only prove that the new node  $s$  introduced in a fork does not generate new failure pairs (see Fig. 8(b)).

Case 1. Let  $(\sigma a, \{b_1, \dots, b_n\})$  be the ready pair contributed by node  $t_1$ , where  $n \geq 1$  and the  $b_i$  are not  $\delta$ . The ready pair of the new node  $s$  is  $(\sigma a, \{b_1, \dots, b_n, c_1, \dots, c_m\})$ . Hence the failure pairs contributed by  $s$  are among those of  $t_1$ .

Case 2.  $n = 1$  and  $b = \delta$ . Then  $(\sigma a, \emptyset)$  is the ready pair of  $t_1$  so the failure pairs of  $t_1$  are  $[\sigma a, X]$ ,  $X \subseteq A$  and again these cover the failure pairs of  $s$ .

Case 3. The cases where  $m = 1, c_1 = \delta$  are trivial.

So in all cases the *new* failure pairs (of  $s$ ) were already present as failure pairs of  $t_1$ . The part of  $\mathcal{F}[g]$  that consists of successful traces is invariant.  $\square$

We will now prove the reverse implications in Proposition 3.2.1. To this end the *ready normal form*  $\mathcal{R}(g)$  and the *failure normal form*  $\mathcal{F}(g)$  will be defined. First we define a map  $\gamma$  from the collection of ready sets  $\{\mathcal{R}[g] \mid g \in H_\delta\}$  to  $H_\delta$ .

DEFINITION 3.2.2. (i) Let  $g \in H_\delta$  have ready set  $\mathcal{R}[g]$ . Then  $\gamma(\mathcal{R}[g])$  is the process graph with  $\mathcal{R}[g] \cup \{o\}$  as set of nodes, with  $(\varepsilon, X) \in \mathcal{R}[g]$  as root, and with edges



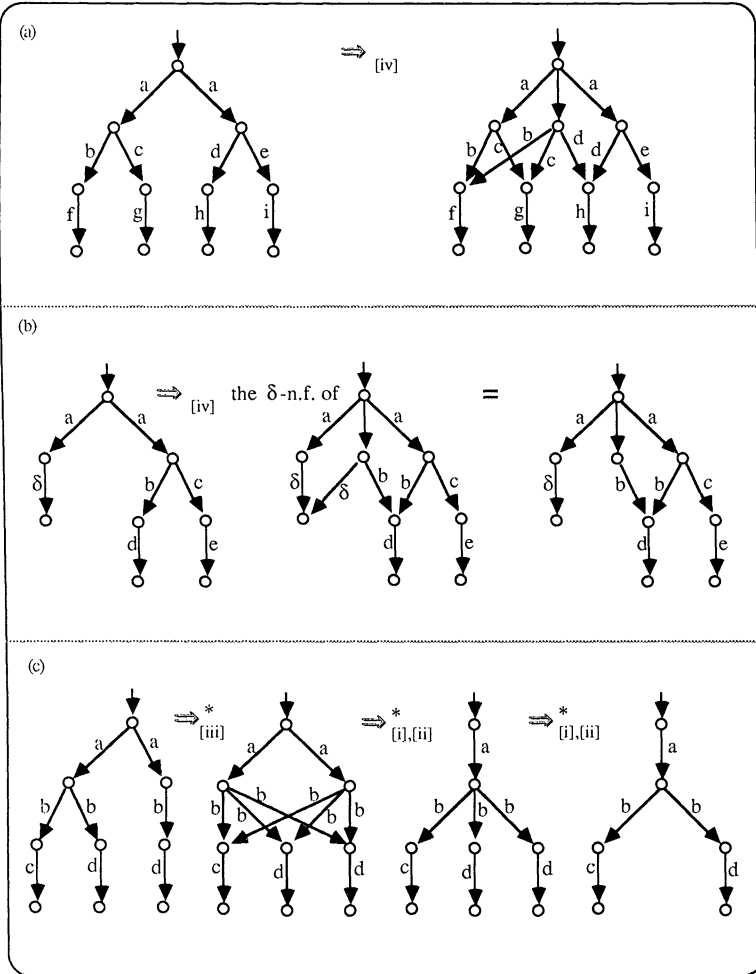


FIG. 10

given by

$$\begin{aligned}
 (\sigma, \{a\} \cup X) &\xrightarrow{a} (\sigma a, Y), \\
 (\sigma, \{a\} \cup X) &\xrightarrow{a} \sigma a, \\
 (\sigma, \emptyset) &\xrightarrow{\delta} o
 \end{aligned}$$

(whenever the left-hand side, right-hand side  $\in \mathcal{R}[[g]] \cup \{o\}$ ).

(ii)  $\mathcal{R}(g) = \gamma(\mathcal{R}[[g]])$  is the *ready normal form* of  $g$ .

(iii) The *convex closure*  $\text{cl}(\mathcal{R}[[g]])$  of  $\mathcal{R}[[g]]$  is obtained as the smallest set containing  $\mathcal{R}[[g]]$  and satisfying

$$(\sigma, X), (\sigma, Y \cup Z) \in \text{cl}(\mathcal{R}[[g]]) \Rightarrow (\sigma, X \cup Y) \in \text{cl}(\mathcal{R}[[g]]).$$

(iv)  $\mathcal{F}(g) = \gamma(\text{cl}(\mathcal{R}[[g]]))$  is the *failure normal form* of  $g$ .

*Example 3.2.3.* Let  $g$  be as in Fig. 11(a). Then  $\mathcal{R}(g)$ ,  $\mathcal{F}(g)$  are as in Fig. 11(b), 11(c), respectively.

**PROPOSITION 3.2.4.**

- (i)  $g \Leftrightarrow_{[i-iii]}^* \mathcal{R}(g)$  via a restricted transformation;
- (ii)  $g \Leftrightarrow^* \mathcal{F}(g)$  via a restricted transformation;

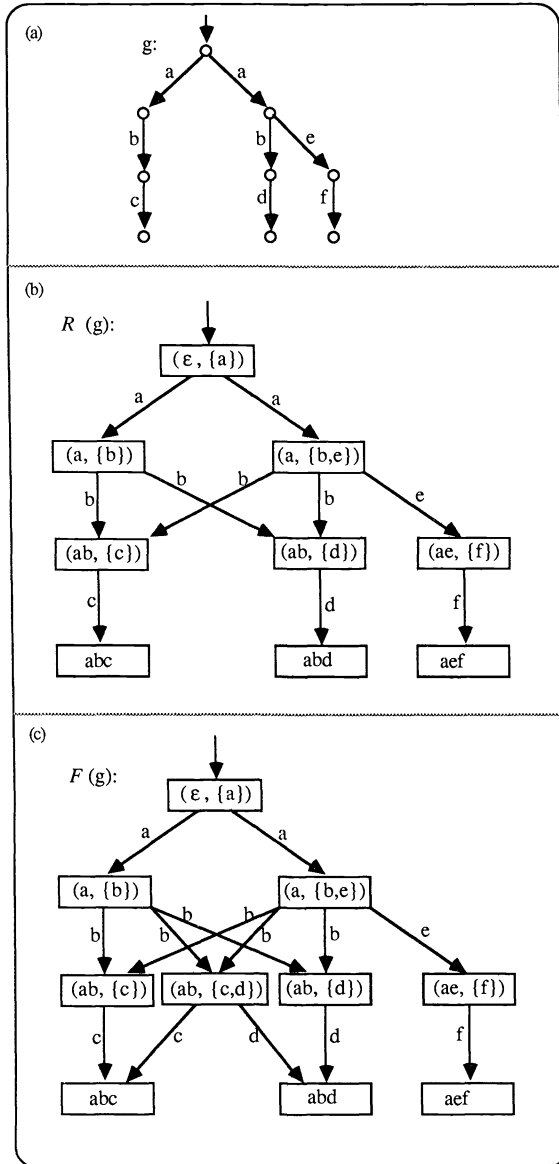


FIG. 11

- (iii)  $g \equiv_{\mathcal{R}} \mathcal{R}(g)$ ;
- (iv)  $g \equiv_{\mathcal{F}} \mathcal{F}(g)$ ;
- (v)  $\mathcal{R}(\mathcal{F}(g)) = \mathcal{F}(g)$ ;
- (vi)  $g \equiv_{\mathcal{R}} h \Rightarrow \mathcal{R}(g) = \mathcal{R}(h)$ ;
- (vii)  $g \equiv_{\mathcal{F}} h \Rightarrow \mathcal{F}(g) = \mathcal{F}(h)$ .

*Proof.* (i) (The following proof was kindly provided to us by R. J. van Glabbeek (personal communication).) Let  $g \in \mathbb{H}_\delta$  be given. We will transform  $g$  via a restricted transformation to a process graph  $g^*$  such that  $g^* \Leftrightarrow \mathcal{R}(g)$ . Since  $\Leftrightarrow$  coincides with  $\Leftrightarrow_{\{i,iii\}}^*$  (see Corollary 3.2.5(i)) this suffices.

If each node in  $g$  has a unique history,  $g$  is called *history unambiguous*. So in particular, process trees are history unambiguous. For a history-unambiguous process graph  $g$ , the *level* of node  $s$  in  $g$  is the length in symbols of the history of  $s$ . The root of  $g$ , therefore, has level 0. We will use the following notation: if  $s$  is a node of  $g$ ,  $\text{ready}(s)$  is the ready contribution of  $s$  to the ready set of  $g$ ; so  $\text{ready}(s) = (\sigma, X)$  or  $\sigma$  for some  $\sigma, X$ . If  $s, t$  are nodes of  $g$ , we write  $s \Leftrightarrow t$  to indicate that the subgraphs with  $s, t$  as roots, respectively, are bisimilar.

The transformation of  $g$  to  $g^*$  such that  $g^* \Leftrightarrow \mathcal{R}(g)$  will be done in stages, level by level, starting from the top level (level 0). The induction hypothesis for the transformation is that after the  $n$ th stage  $g$  ( $=g_0$ ) is transformed to  $g_n$  satisfying the following property  $\mathbb{H}_n$ : *Suppose  $p, q$  are nodes of  $g_n$  such that  $p$  has level  $n$ ,  $\text{ready}(p) = (\sigma, X \cup \{a\})$  and  $\text{ready}(q) = (\sigma a, Y)$  or  $\sigma a$ . Then  $g_n$  contains a node  $r$  such that  $q \Leftrightarrow r$  and  $p \rightarrow_a r$ .*

For  $g_0$  we have indeed  $\mathbb{H}_0$ ;  $p$  is then the root and for  $r$  we just take  $q$ . Now suppose  $g_n$  is constructed such that  $\mathbb{H}_n$  holds. We will construct  $g_{n+1}$  such that  $\mathbb{H}_{n+1}$  holds. So consider a node  $p$  of level  $n+1$  in  $g_n$  admitting an  $a$ -step (see Fig. 12) with  $\text{ready}(p) = (\sigma b, X \cup \{a\})$ , and a node  $q$  with  $\text{ready}(q) = (\sigma b a, Y)$ . Hence there is a node  $p'$  on level  $n$  such that  $p' \rightarrow_b p$ ; say  $\text{ready}(p') = (\sigma, X')$ . Also there must be a node  $q'$  such that  $q' \rightarrow_a q$  and, say,  $\text{ready}(q') = (\sigma b, Z)$ . By  $\mathbb{H}_n$ , therefore, there is a node  $r'$  such that  $q' \Leftrightarrow r'$  and  $p' \rightarrow_b r'$ . By the definition of  $\Leftrightarrow$ , there is a node  $r$  such that  $r' \rightarrow_a r$  and  $q \Leftrightarrow r$ . Now we insert a cross (i.e., two  $a$ -steps) as in the figure. The result is unshared by backward application of  $\Rightarrow_{(iii)}$  to a process tree. This unsharing does not increase the number of nodes of level  $n+1$ , and also does not increase the number of nodes of level  $n+2$  modulo  $\Leftrightarrow$ . The procedure is repeated for all  $p$  of level  $n+1$  and equivalence classes  $q/\Leftrightarrow$ . As there are only finitely many such pairs  $p, q/\Leftrightarrow$  the procedure stops eventually; the resulting tree is  $g_{n+1}$ . Clearly  $g_{n+1}$  satisfies  $\mathbb{H}_{n+1}$ . The construction of the sequence  $g_0, g_1, \dots, g_n$  stops when  $n$  is equal to the depth of  $g$ . The result is called  $g^*$ , and we claim that  $g^* \Leftrightarrow \mathcal{R}(g)$  via the bisimulation that relates nodes  $s, t$  in  $g^*, \mathcal{R}(g)$ , respectively, such that  $\text{ready}(s) = \text{ready}(t)$ .

*Proof of the Claim.* Suppose  $s, t$  are nodes in  $g^*, \mathcal{R}(g)$ , respectively, such that  $\text{ready}(s) = \text{ready}(t) = (\sigma, X \cup \{a\})$ . Let  $s \rightarrow_a s'$ . Then take the unique node  $t'$  in  $\mathcal{R}(g)$  with  $\text{ready}(t') = \text{ready}(s')$ . This must be  $(\sigma a, Y)$  or  $\sigma a$ . By definition of the edges in  $\mathcal{R}(g)$  we have  $t \rightarrow_a t'$ ; and indeed  $s' \Leftrightarrow t'$  because  $\text{ready}(s') = \text{ready}(t')$ . The other side of the bisimulation requirements: Let  $s, t$  be as before, and let  $t \rightarrow_a t'$  with  $\text{ready}(t') = (\sigma a, Y)$  or  $\sigma a$ . Let  $s^*$  be a node in  $g^*$  such that  $\text{ready}(s^*) = \text{ready}(t')$ . By construction

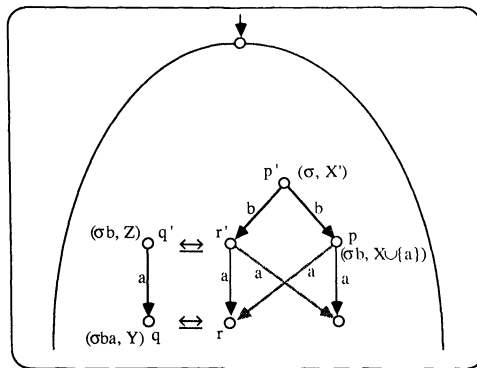


FIG. 12

of  $g^*$  there is a node  $s'$  in  $g^*$  with  $s^* \Leftrightarrow s'$  and  $s \rightarrow_a s'$ . Clearly,  $\text{ready}(s') = \text{ready}(s^*) = \text{ready}(t')$ , hence  $s' \Leftrightarrow t'$ .

(ii) Let  $g$  be given. According to (i) there is a restricted transformation of  $g$  to  $\mathcal{R}(g)$ . We will transform (via a restricted transformation)  $\mathcal{R}(g)$  further into  $\mathcal{F}(g)$ , as follows. Take nodes  $(\sigma, X)$ ,  $(\sigma, Y \cup Z)$  from  $\mathcal{R}(g)$  such that  $(\sigma, X \cup Y)$  is not yet a node of  $\mathcal{R}(g)$ . (If such nodes do not exist then  $\mathcal{R}(g)$  is already equal to  $\mathcal{F}(g)$ .) Now it is easy to see that there are paths in  $\mathcal{R}(g)$  from the root to the nodes  $(\sigma, X)$ ,  $(\sigma, Y \cup Z)$  such that these paths coincide in all but their last step, i.e., the paths split up as late as possible. At the split-up node we now insert a fork into  $\mathcal{R}(g)$ , with central node  $(\sigma, X \cup Y)$ , which is a new node. Call the result:  $\mathcal{R}(g)^+$ . Next,  $\mathcal{R}(g)^+$  is transformed (according to (i)) to  $\mathcal{R}(\mathcal{R}(g)^+)$ . Iteration of this procedure, via  $\mathcal{R}(\mathcal{R}(g)^+)^+$ ,  $\mathcal{R}(\mathcal{R}(\mathcal{R}(g)^+)^+)$ , etc., obviously will stop in  $\mathcal{F}(g)$ .

Parts (iii) and (iv) are left to the reader.

(v) By Definition 3.2.2,  $\mathcal{R}(\mathcal{F}(g)) = \mathcal{F}(g)$  means

$$\gamma(\mathcal{R}[\gamma(\text{cl}(\mathcal{R}[\mathcal{R}(g)]))]) = \gamma(\text{cl}(\mathcal{R}[\mathcal{R}(g)])),$$

which is equivalent to

$$\mathcal{R}[\gamma(\text{cl}(\mathcal{R}[\mathcal{R}(g)]))] = \text{cl}(\mathcal{R}[\mathcal{R}(g)]).$$

So we must check that the set of ready pairs of the graph determined by the set of ready pairs  $\text{cl}(\mathcal{R}[\mathcal{R}(g)])$  is just  $\text{cl}(\mathcal{R}[\mathcal{R}(g)])$ ; this seems obvious.

(vi)  $g \equiv_{\mathcal{R}} h$  by definition means  $\mathcal{R}[g] = \mathcal{R}[h]$ . Hence  $\mathcal{R}(g) = \gamma(\mathcal{R}[g]) = \gamma(\mathcal{R}[h]) = \mathcal{R}(h)$ .

(vii) Suppose  $g \equiv_{\mathcal{F}} h$ . Then by (iv)  $g \equiv_{\mathcal{F}} \mathcal{F}(g)$ ,  $h \equiv_{\mathcal{F}} \mathcal{F}(h)$ , so  $\mathcal{F}(g) \equiv_{\mathcal{F}} \mathcal{F}(h)$ . Since both  $\mathcal{F}(g)$ ,  $\mathcal{F}(h)$  are convexly closed, we have  $\mathcal{F}(g) \equiv_{\mathcal{R}} \mathcal{F}(h)$  (by Lemma 2.5.5). So

(vi)  $\mathcal{R}(\mathcal{F}(g)) = \mathcal{R}(\mathcal{F}(h))$ . Hence by (v):  $\mathcal{F}(g) = \mathcal{F}(h)$ .  $\square$

**COROLLARY 3.2.5.** *Let  $g, h \in \mathbb{H}_\delta$ . Then we have the following:*

- (i)  $g \Leftrightarrow h$  if and only if  $g \Leftrightarrow_{\{\text{ii}, \text{iii}\}}^* h$ ;
- (ii)  $g \equiv_{\mathcal{R}} h$  if and only if  $g \Leftrightarrow_{\{\text{i}-\text{iii}\}}^* h$ ;
- (iii)  $g \equiv_{\mathcal{F}} h$  if and only if  $g \Leftrightarrow^* h$ .

*Proof.* Item (i) is (essentially) proved in the Appendix of [BK83] and also in Corollary 2.13 of [BK85]: the proofs there also take  $\tau$ -steps into account; after leaving out all mention of  $\tau$ -steps, the result follows.

(ii) The implication from right to left follows from Proposition 3.2.1(i). The other direction follows from Proposition 3.2.4(i), (vi).

(iii) The proof is similar to (ii).  $\square$

**4. Axiomatising the equivalences on process graphs.** We will now use our analysis of  $\equiv_{\mathcal{R}}, \equiv_{\mathcal{F}}$  on the graph domain  $\mathbb{H}_\delta$  to formulate complete axiom systems for these notions. First this will be done for the signature of  $+, \cdot$  alone, later on (in § 4.2) also  $\parallel, \ll, \mid, \partial_H$  will be taken into account.

**4.1. The case without communication.** We start with the observation (whose proof is simple and omitted) that  $\equiv_{\mathcal{R}}, \equiv_{\mathcal{F}}$  are congruences on  $\mathbb{H}_\delta(+, \cdot)$ , and hence can be factored out to yield  $\mathbb{H}_\delta(+, \cdot)/\equiv_{\mathcal{R}}$  and  $\mathbb{H}_\delta(+, \cdot)/\equiv_{\mathcal{F}}$ , respectively. These are the structures which we will now axiomatise.

We will prove that the axiom system  $\text{BPA}_\delta + \text{R1}, 2 + \text{S}$  in Table 3 is a complete axiomatisation for  $\mathbb{H}_\delta(+, \cdot)/\equiv_{\mathcal{F}}$ ; after leaving out axiom *S* we have a complete axiomatisation for  $\mathbb{H}_\delta(+, \cdot)/\equiv_{\mathcal{R}}$ . Here  $a, b$  vary over  $A \cup \{\delta\}$ ;  $x, y, z, u, v$  are variables

for processes. Note that R2 is not derivable from R1 because in  $\text{BPA}_\delta + \text{R1}, 2 + \text{S}$  there is no process  $x$  satisfying  $bx = b$  when  $b \neq \delta$ . On the other hand,  $x$  should be present in axiom S as the equation

$$a + a(y + z) = a + a(y + z) + ay$$

would yield the failure-inconsistent equation

$$a + ab = a + ab + a\delta.$$

*Remark 4.1.1.* (i) The axioms R1, 2 and S (R for readiness, S for saturation), which are specific for failure equivalence, appear already in [Br83] in a slightly different form. [Br83] considers also  $\tau$ -steps and presents as laws valid for failure equivalence in Proposition 1.3.6:

$$(1) \quad \tau(\mu x + u) + \tau(\mu y + v) = \tau(\mu x + \mu y + u) + \tau(\mu x + \mu y + v),$$

$$(2) \quad \mu x + \mu y = \mu(\tau x + \tau y)$$

(here  $\mu \in A_\delta \cup \{\tau\}$ ;  $x, y, u, v$  are arbitrary processes), and in Proposition A.3 in [Br83]:

$$(3) \quad \tau x + \tau y = \tau x + \tau y + \tau(x + y),$$

$$(4) \quad \tau x + \tau(x + y + z) = \tau x + \tau(x + y) + \tau(x + y + z).$$

Clearly (1), (2) imply R1 in Table 3; and using the  $\tau$ -law  $x\tau = x$ , also valid in failure semantics, we also derive R2. Further, (3), (4) together with (2) yield the pair

$$ax + ay = ax + ay + a(x + y),$$

$$ax + a(x + y + z) = ax + a(x + y) + a(x + y + z)$$

(where  $a \in A_\delta$ ), which is equivalent to axiom S in Table 3.

TABLE 3  
 $\text{BPA}_\delta + \text{R1}, 2 + \text{S}$

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5
$x + \delta = x$	A6
$\delta x = \delta$	A7
$a(bx + u) + a(by + v) = a(bx + by + u) + a(bx + by + v)$	R1
$a(b + u) + a(by + v) = a(b + by + u) + a(b + by + v)$	R2
$ax + a(y + z) = ax + a(y + z) + a(x + y)$	S

(ii) The axioms R1, 2 and S are also immediate consequences of the proof system of De Nicola and Hennessy [DH84] for strong testing equivalence  $\approx_2$ , to be discussed and related with failure equivalence later in Remark 7.3.3. This can be seen as follows:

(1) *Axiom S* in Table 3:  $ax + a(y + z) = ax + a(y + z) + a(x + y)$  implies

$$ax + ay = ax + ay + a(x + y)$$

by taking  $z = y$ ; this is (D5) in [DH84]. Further, (S) implies

$$ax + a(x + y + z) = ax + a(x + y + z) + a(x + y)$$

by replacing  $y$  in (S) by  $x+y$ . This is (D6) in [DH84]. Vice versa, (S) follows from (D5), (D6):

$$ax + a(y+z) \tag{D5}$$

$$= ax + a(y+z) + a(x+y+z) \tag{D6}$$

$$= ax + a(y+z) + a(x+y+z) + a(x+y) \tag{D5}$$

$$= ax + a(y+z) + a(x+y).$$

(2) *Axiom R1*:  $a(bx+u) + a(by+v) = a(bx+by+v) + a(bx+by+u)$  is derived from the axiom system in [DH84] as follows:

$$bx + \tau(by+v) = \tau(bx+by+v) \tag{N3},$$

$$by + \tau(bx+u) = \tau(bx+by+u) \tag{N3},$$

$$bx + by + \tau(by+v) + \tau(bx+u) = \tau(bx+by+v) + \tau(bx+by+u)$$

$$bx + \tau(bx+u) = \tau(bx+u) \tag{D9},$$

$$by + \tau(by+v) = \tau(by+v) \tag{D9},$$

$$\tau(by+v) + \tau(bx+u) = \tau(bx+by+v) + \tau(bx+by+u),$$

$$a[\tau(by+v) + \tau(bx+u)] = a[\tau(bx+by+v) + \tau(bx+by+u)],$$

$$a(by+v) + a(bx+u) = a(bx+by+v) + a(bx+by+u) \tag{N1}.$$

Here (N1), (N3), and (D9) are axioms in [DH84].

(3) *Axiom R2*:  $a(b+u) + a(by+v) = a(b+by+u) + a(b+by+v)$  is not needed in [DH84] because a process  $b$ , which first performs action  $b$  and then successfully terminates, is not considered there. Note that the process  $b\text{NIL}$  of [DH84] corresponds to  $b \cdot \delta$  and is thus different from  $b$ .

**4.1.2. Connecting terms with process graphs.** Let  $\text{Ter}(\text{BPA}_\delta)$  be the set of closed terms in the signature of  $\text{BPA}_\delta$  (=the signature of  $\text{BPA}_\delta + \text{R1, 2+S}$ ). We define the following translations:

$$\text{graph: } \text{Ter}(\text{BPA}_\delta) \rightarrow \mathbb{H}_\delta,$$

$$\text{ter: } \mathbb{H}_\delta \rightarrow \text{Ter}(\text{BPA}_\delta).$$

Here  $\text{graph}(T)$  is the process graph obtained by first normalizing  $T$  with respect to A4, A6, A7 in Table 3 and second by interpreting  $a, +, \cdot$  as the corresponding ‘‘one-edge graphs’’ and operators  $+, \cdot$  on  $\mathbb{H}_\delta$ .

Further, to define  $\text{ter}(g)$  we first define  $\text{tree}(g)$  as the tree obtained from  $g$  by ‘‘unsharing.’’ Now we define  $\text{ter}(g)$  as the term corresponding in the obvious way to  $\text{tree}(g)$ .

*Example 4.1.2.1.* (i)  $\text{graph}(a(b+c+d)d+de+ed) = \text{graph}(a(bd+cd)+ed)$  is the graph in Fig. 13(a).

(ii) If  $g$  is as in Fig. 13(b), then  $\text{tree}(g)$  is as in Fig. 13(c).

(iii) If  $g$  is as in (ii), then  $\text{ter}(g) = ace + b(de + ab)$ .

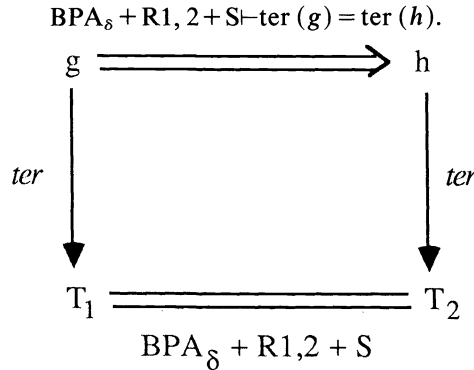
*Remark 4.1.3.* Note that  $\text{ter}, \text{graph}$  are ‘‘almost’’ inverse to each other:

$$\text{BPA}_\delta \vdash (\text{ter} \circ \text{graph})(T) = T,$$

$$(\text{graph} \circ \text{ter})(g) \Leftrightarrow g$$

where  $\Leftrightarrow$  (bisimilarity) coincides with  $\Leftrightarrow_{\text{III,III}}^*$ .

TRANSFER LEMMA 4.1.4 (see diagram). Let  $g, h \in \mathbb{H}_\delta$  be such that  $g \Rightarrow h$ . In case  $\Rightarrow$  is  $\Rightarrow_{(iii)}$  we require moreover that  $g$  be a process tree. Then



*Proof.* A transformation  $g \Rightarrow_{(ii)} h$  (removing a double edge) “translates” into some applications of A3:  $x + x = x$ .

A transformation  $g \Rightarrow_{(iii)} h$  is invisible on the level of terms, i.e.,  $\text{ter}(g)$  and  $\text{ter}(h)$  are identical terms. Next consider a transformation  $g \Rightarrow_{(iii)} h$ , which consists of adding two edges in  $g$  as in Fig. 14. (Note that in this case  $g$  is assumed to be a tree.) This translates to an application of R1 if the subtrees  $x, y$  are nonempty, and to R2 if one of these subtrees is empty. In case both subtrees  $x, y$  are empty we have an application of axiom A3.

Finally, a transformation  $g \Rightarrow_{(iv)} h$  (see also Fig. 8) translates into some applications of axiom S in Table 3.  $\square$

THEOREM 4.1.5. (i)  $\text{BPA}_\delta + \text{R1}, 2 \vdash T_1 = T_2 \Leftrightarrow \text{graph}(T_1) \equiv_{\mathcal{G}} \text{graph}(T_2)$ .

(ii)  $\text{BPA}_\delta + \text{R1}, 2 + \text{S} \vdash T_1 = T_2 \Leftrightarrow \text{graph}(T_1) \equiv_{\mathcal{F}} \text{graph}(T_2)$ .

*Proof.* We prove (ii); the proof of (i) is similar.

Checking the soundness ( $\Rightarrow$ ) is routine and will not be done here. As to the completeness ( $\Leftarrow$ ): suppose  $\text{graph}(T_1) \equiv_{\mathcal{F}} \text{graph}(T_2)$ . Then by Proposition 3.2.4(ii), (vii):  $\text{graph}(T_1) \Leftrightarrow^* \text{graph}(T_2)$  via a restricted transformation. Now by the Transfer Lemma 4.1.4 we have

$$\text{BPA}_\delta + \text{R1}, 2 + \text{S} \vdash (\text{ter} \circ \text{graph})(T_1) = (\text{ter} \circ \text{graph})(T_2)$$

and by Remark 4.1.3:

$$\text{BPA}_\delta + \text{R1}, 2 + \text{S} \vdash T_1 = T_2. \quad \square$$

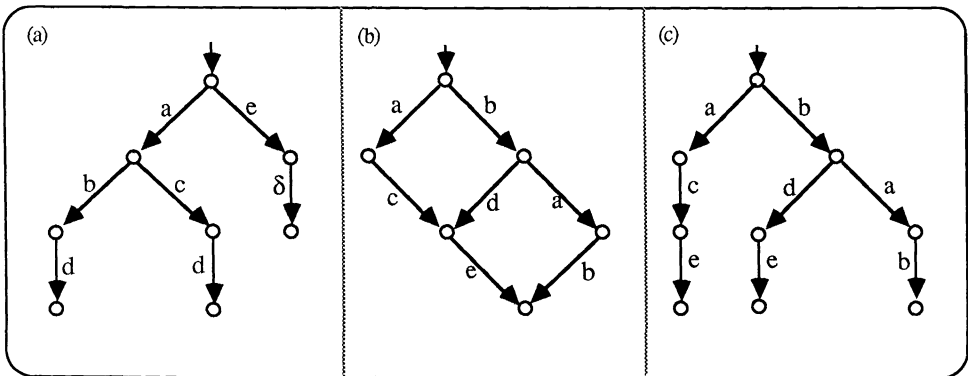


FIG. 13

*Notation 4.1.6.* (i) If  $(\Sigma, E)$  is a specification (sometimes only written as  $E$  if the signature  $\Sigma$  is clear), then  $I(\Sigma, E)$  is its initial algebra.

(ii)  $\cong$  denotes isomorphism between algebras.

COROLLARY 4.1.7. (i)  $\mathbb{H}_\delta(+, \cdot, a, \delta) / \equiv_{\mathcal{F}} \cong I(\text{BPA}_\delta + \text{R1}, 2)$ .

(ii)  $\mathbb{H}_\delta(+, \cdot, a, \delta) / \equiv_{\mathcal{F}} \cong I(\text{BPA}_\delta + \text{R1}, 2 + \text{S})$ .  $\square$

**4.2. The case with communication: the graph model of  $\text{ACP}_r$ .** Finally we will prove the results above in the presence of communication. The operators  $\parallel, \llbracket, \cdot, |, \partial_H, a_H (a \in A)$  on  $\mathbb{H}_\delta$  were already introduced in § 1.2. They are the semantical counterparts of the same operators in the axiom system  $\text{ACP}_r$ , as in the upper part of Table 4, which presents the axiom system  $\text{ACP}_r + \text{R1}, 2 + \text{S}$ , and which extends our earlier axiom system  $\text{BPA}_\delta + \text{R1}, 2 + \text{S}$  in Table 3.

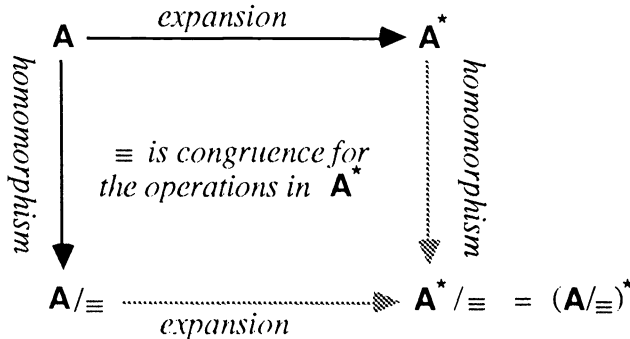
As before, in Table 4 a, b, c vary over  $A \cup \{\delta\}$ , and  $x, y, z, u, v$  vary over processes.

We want to prove that the initial algebra of  $\text{ACP}_r + \text{R1}, 2 + \text{S}$  is isomorphic to the model of finite acyclic graphs modulo failure equivalence  $\equiv_{\mathcal{F}}$ , called the *graph model* for  $\text{ACP}_r + \text{R1}, 2 + \text{S}$ . To this end we have first to prove that  $\equiv_{\mathcal{F}}$  is a *congruence* with respect to also the new operators. Once we have this, and knowing from [BK85], [BK86a] (after leaving out all reference to  $\tau$ -steps) that there is the isomorphism

$$I(\text{ACP}_r) \cong \mathbb{H}_\delta(+, \cdot, \parallel, \llbracket, \cdot, |, \partial_H, a_H, a, \delta) / \Leftrightarrow$$

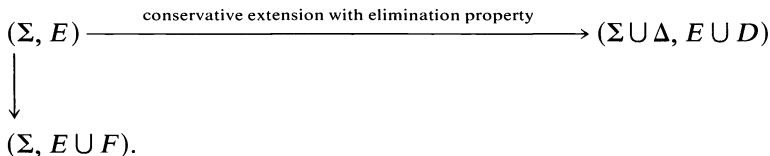
where  $\Leftrightarrow$  is bisimulation (which coincides with  $\Leftrightarrow_{\{\text{III}, \text{III}\}}^*$ ; Corollary 3.2.5(i)), the derived isomorphism is a consequence from some general facts which we will state now.

**4.2.1. General intermezzo.** Let  $\mathbf{A}$  be an algebra that on the one hand can be expanded to  $\mathbf{A}^*$  (i.e., enriched with new functions; the domain is invariant) and on the other hand can be factored out via  $\equiv$ , a congruence on  $\mathbf{A}$ , to  $\mathbf{A}/\equiv$ . Suppose moreover that  $\equiv$  is also a congruence on  $\mathbf{A}^*$ . (See the following diagram.)



Then this expansion and factorisation are compatible (or commuting):  $\mathbf{A}^*/\equiv$  equals  $(\mathbf{A}/\equiv)^*$ . Now let  $\mathbf{A}, \mathbf{A}^*, \mathbf{A}/\equiv$  be isomorphic respectively to the initial algebras of the equational specifications  $(\Sigma, E), (\Sigma \cup \Delta, E \cup D), (\Sigma, E \cup F)$ . Then it follows that  $(\Sigma \cup \Delta, E \cup D)$  is

- (1) a conservative extension of the “base” specification  $(\Sigma, E)$  (i.e., no new identities between closed terms in the base signature  $\Sigma$  are provable from  $(\Sigma \cup \Delta, E \cup D)$ ), and
- (2) moreover, the extra operators in  $\Delta$  can be *eliminated*:





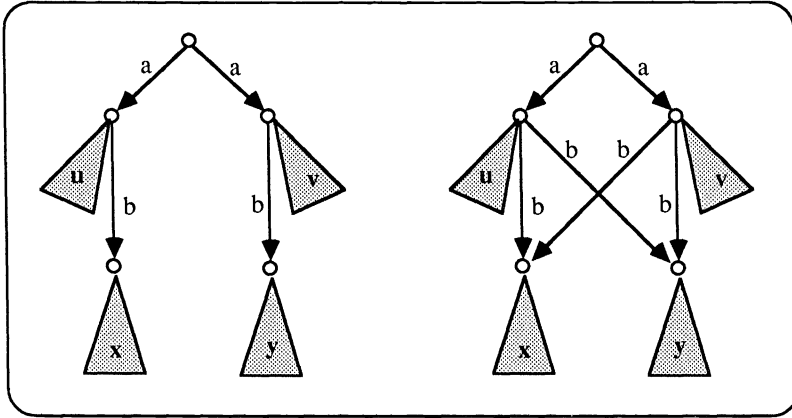


FIG. 14

TABLE 4  
ACP<sub>r</sub>+R1, 2+S

$x + y = y + x$	A1
$x + (y + z) = (x + y) + z$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5
$x + \delta = x$	A6
$\delta x = \delta$	A7
$a b = b a$	C1
$(a b) c = a (b c)$	C2
$\delta a = \delta$	C3
$x  y = x  y + y  x + x y$	CM1
$a  x = ax$	CM2
$ax  y = a(x  y)$	CM3
$(x + y)  z = x  z + y  z$	CM4
$ax b = (a b)x$	CM5
$a bx = (a b)x$	CM6
$ax by = (a b)(x  y)$	CM7
$(x + y) z = x z + y z$	CM8
$x (y + z) = x y + x z$	CM9
$\partial_H(a) = a \quad \text{if } a \notin H$	D1
$\partial_H(a) = \delta \quad \text{if } a \in H$	D2
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	D3
$\partial_H(xy) = \partial_H(x) \cdot \partial_H(y)$	D4
$a_H(b) = b \quad \text{if } b \notin H$	RN1
$a_H(b) = a \quad \text{if } b \in H$	RN2
$a_H(x + y) = a_H(x) + a_H(y)$	RN3
$a_H(xy) = a_H(x) \cdot a_H(y)$	RN4
$a(bx + u) + a(by + v) = a(bx + by + u) + a(bx + by + v)$	R1
$a(b + u) + a(by + v) = a(b + by + u) + a(b + by + v)$	R2
$ax + a(y + z) = ax + a(y + z) + a(x + y)$	S

Furthermore (and this is what we are interested in) we may conclude from the given isomorphisms that

$$\mathbf{A}^*/\equiv = (\mathbf{A}/\equiv)^* \cong I(\Sigma \cup \Delta, E \cup D \cup F)$$

where the last algebra is the initial algebra of the *union* of  $(\Sigma, E \cup F)$  and  $(\Sigma \cup \Delta, E \cup D)$ .

(In the statement of the next theorem, as well as in its proof and Table 5, we have suppressed mention of the constants  $a, \delta$  in, e.g.,  $\mathbb{H}_\delta(+, \cdot)$ , which actually should read  $\mathbb{H}_\delta(+, \cdot, a, \delta)(a \in A)$ .)

**THEOREM 4.2.2.** *Let the initial algebras  $I(\text{BPA}_\delta)$  etc. as in Table 5(ii) of the axiom systems  $\text{BPA}_\delta$  etc. as in Table 5(i) be given. Furthermore, consider the graph models  $\mathbb{H}_\delta(+, \cdot) / \cong$  etc. as in Table 5(iii).*

*Then corresponding initial models and graph models are isomorphic. In particular:*

$$I(\text{ACP}_r + \text{R1}, 2 + \text{S}) \cong \mathbb{H}_\delta(+, \cdot, \parallel, \llbracket, \lrcorner, \partial_H, a_H) / \equiv_{\mathcal{F}}$$

*Proof.* Consider, for example,

$$\begin{array}{ccc} \text{BPA}_\delta & \longrightarrow & \text{ACP}_r \\ \downarrow & & \\ \text{BPA}_\delta + \text{R1}, 2 + \text{S} & & \end{array}$$

and the corresponding initial algebras

$$\begin{array}{ccc} I(\text{BPA}_\delta) & \longrightarrow & I(\text{ACP}_r) \\ \downarrow & & \\ I(\text{BPA}_\delta + \text{R1}, 2 + \text{S}) & & \end{array}$$

and furthermore (by position in the diagram in Table 5) the corresponding graph models

$$\begin{array}{ccc} \mathbb{H}_\delta(+, \cdot) / \cong & \xrightarrow{\text{exp}} & \mathbb{H}_\delta(+, \cdot, \parallel, \llbracket, \lrcorner, \partial_H, a_H) / \cong \\ \downarrow \text{hom} & & \\ \mathbb{H}_\delta(+, \cdot) / \equiv_{\mathcal{F}} & & \end{array}$$

By Corollary 4.1.7(ii) we have  $I(\text{BPA}_\delta + \text{R1}, 2 + \text{S}) \cong \mathbb{H}_\delta(+, \cdot) / \equiv_{\mathcal{F}}$ , and by results in [BK85], [BK86a], [BK86b] we have  $I(\text{BPA}_\delta) \cong \mathbb{H}_\delta(+, \cdot) / \cong$  and  $I(\text{ACP}_r) \cong \mathbb{H}_\delta(+, \cdot, \parallel, \llbracket, \lrcorner, \partial_H, a_H) / \cong$ .

Therefore, by 4.2.1, it suffices to prove that  $\equiv_{\mathcal{F}}$  is a congruence with respect to the “new” operators on  $\mathbb{H}_\delta$  in order to conclude that

$$I(\text{ACP}_r + \text{R1}, 2 + \text{S}) \cong \mathbb{H}_\delta(+, \cdot, \parallel, \llbracket, \lrcorner, \partial_H, a_H) / \equiv_{\mathcal{F}}$$

This is proved in the next proposition.  $\square$

**PROPOSITION 4.2.3.** (i) *Failure equivalence is a congruence with respect to the operators  $\parallel, \llbracket, \lrcorner, \partial_H, a_H$  on  $\mathbb{H}_\delta$ .*

(ii) *The same holds for ready equivalence.*

*Proof.* (i) We consider some typical cases.

*The case of  $\partial_H$ .* To prove  $g \equiv_{\mathcal{F}} h \Rightarrow \partial_H(g) \equiv_{\mathcal{F}} \partial_H(h)$ . By Corollary 3.2.5 it suffices to check that  $g \Rightarrow h$  implies  $\partial_H(g) \equiv_{\mathcal{F}} \partial_H(h)$ . The cases that  $\Rightarrow$  is  $\Rightarrow_{[i]}$  or  $\Rightarrow_{[ii]}$  present no problem. As to  $\Rightarrow_{[iii]}$ : it is easy to verify that

$$g \Rightarrow_{[iii]} h \text{ implies } \partial_H(g) = \partial_H(h) \text{ or } \partial_H(g) \Rightarrow_{[iii]} \partial_H(h).$$

As to  $\Rightarrow_{[iv]}$ , as in the previous case, the effect of  $\partial_H$  (renaming some atoms in  $g, h$  into  $\delta$  and  $\delta$ -normalising the resulting graphs again) is such that either the “same” fork can be inserted or  $\partial_H(g) = \partial_H(h)$ .

TABLE 5

(i)	
$BPA_\delta$	$\longrightarrow$ $ACP_r$
$\downarrow$	$\downarrow$
$BPA_\delta + R1, 2$	$\longrightarrow$ $ACP_r + R1, 2$
$\downarrow$	$\downarrow$
$BPA_\delta + R1, 2 + S$	$\longrightarrow$ $ACP_r + R1, 2 + S$
(ii)	
$I(BPA_\delta)$	$\xrightarrow{\text{exp}}$ $I(ACP_r)$
$\downarrow \text{hom}$	$\downarrow \text{hom}$
$I(BPA_\delta + R1, 2)$	$\xrightarrow{\text{exp}}$ $I(ACP_r + R1, 2)$
$\downarrow \text{hom}$	$\downarrow \text{hom}$
$I(BPA_\delta + R1, 2 + S)$	$\xrightarrow{\text{exp}}$ $I(ACP_r + R1, 2 + S)$
(iii)	
$\mathbb{H}_\delta(+, \cdot) / \simeq$	$\xrightarrow{\text{exp}}$ $\mathbb{H}_\delta(+, \cdot, \parallel, \perp,  , \partial_H, a_H) / \simeq$
$\downarrow \text{hom}$	$\downarrow \text{hom}$
$\mathbb{H}_\delta(+, \cdot) / \equiv_{\mathcal{R}}$	$\xrightarrow{\text{exp}}$ $\mathbb{H}_\delta(+, \cdot, \parallel, \perp,  , \partial_H, a_H) / \equiv_{\mathcal{R}}$
$\downarrow \text{hom}$	$\downarrow \text{hom}$
$\mathbb{H}_\delta(+, \cdot) / \equiv_{\mathcal{F}}$	$\xrightarrow{\text{exp}}$ $\mathbb{H}_\delta(+, \cdot, \parallel, \perp,  , \partial_H, a_H) / \equiv_{\mathcal{F}}$

(Note here that it is crucial that process graphs  $g, h$  as in Fig. 15 are not failure equivalent, since  $\partial_{\{b\}}$  would yield a trace  $a\delta$  in  $h$  but not in  $g$ .)

The case of  $\parallel$ . It suffices to prove

$$g \Rightarrow g' \text{ implies } g \parallel h \equiv_{\mathcal{F}} g' \parallel h.$$

As above, only the cases [iii], [iv] (cross and fork, respectively) are of interest. In fact we will prove the following:

- (1)  $g \Rightarrow_{\text{[iii]}} g'$  implies  $g \parallel h \Rightarrow_{\text{[iii]}} g' \parallel h$ .
- (2)  $g \Rightarrow_{\text{[iv]}} g'$  implies  $g \parallel h \equiv_{\mathcal{F}} g' \parallel h$ .

*Proof of (1).* Due to the construction of a merge as a Cartesian product with diagonal edges for communications (Fig. 16), it is “geometrically” clear (see Fig. 17) that inserting a cross in  $g$  amounts to inserting several crosses (also possibly diagonal ones, depending on the communication function) in the merge  $g \parallel h$ . So  $g \parallel h \Rightarrow_{\text{[iii]}} g' \parallel h$ . (It is not hard to see that the condition on histories, which is stated in the definition

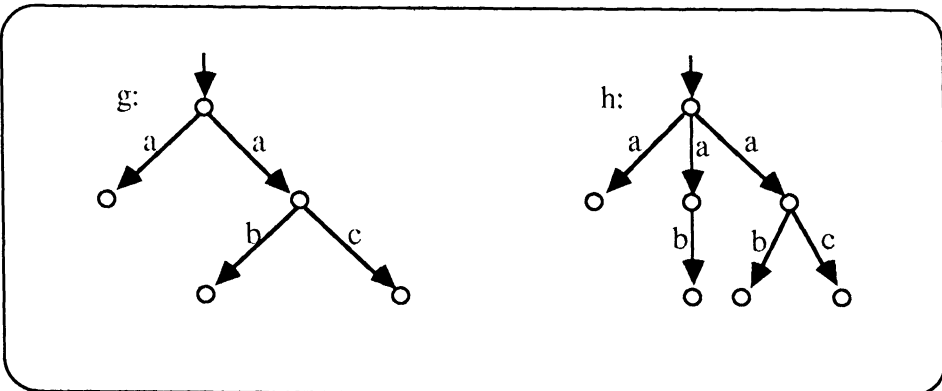


FIG. 15

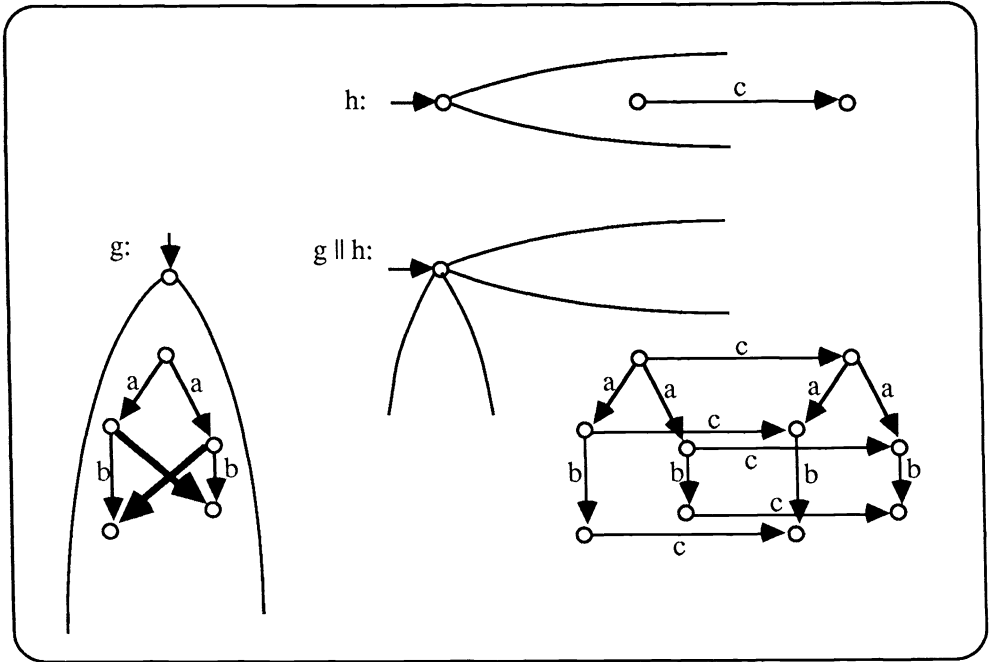


FIG. 16

of  $\Rightarrow_{(iii)}$ , stays satisfied in such a way that insertion of these crosses in  $g \parallel h$  is indeed legitimate.)

*Proof of (2).* Under the assumption  $g \Rightarrow_{(iv)} g'$  we now prove  $g \parallel h \equiv_{\mathcal{F}} g' \parallel h$  directly from the definition  $\equiv_{\mathcal{F}}$ . So consider the addition in  $g$  of a fork that connects all successors of  $s_1$  (see Fig. 18) to some of those of  $s_3$ . That is, the failure pairs contributed by the new node  $s_2$  are contained in those of  $s_1$ . Then we must check that the new

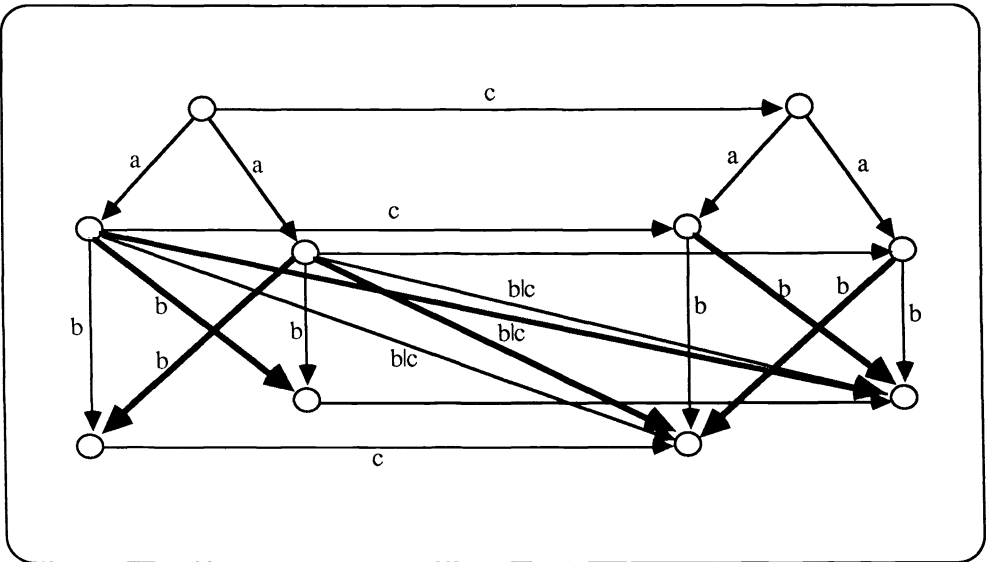


FIG. 17

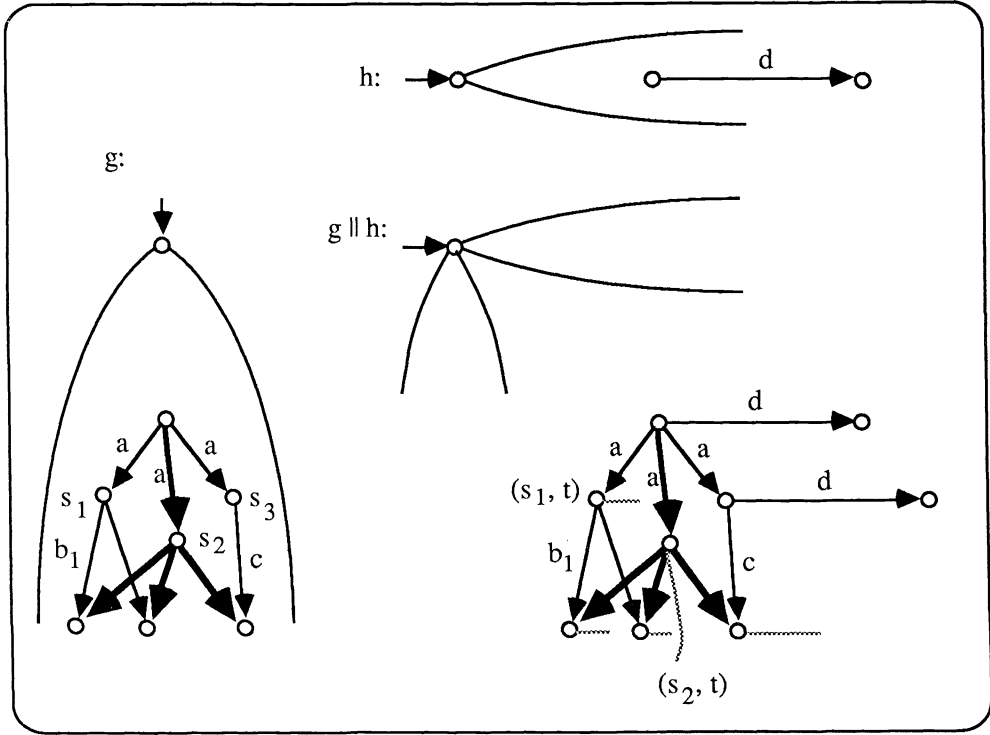


FIG. 18

nodes  $(s_2, t)$  in  $g' || h$  caused by this addition, contribute no new failure pairs. It is not hard to check that indeed the failure pairs of  $(s_2, t)$  are contained in those of  $(s_2, t)$  by some consideration of the outgoing edges of  $(s_1, t)$  and  $(s_2, t)$ . The precise verification is omitted here.

The proof of part (ii) of the proposition is as for (i)—but simpler. It is omitted here.  $\square$

**5. The failure model of  $ACP_r$ .** In the previous sections the notion of failure equivalence was introduced for the process graph domain  $\mathbb{H}_\delta$ , and it was shown to be a congruence with respect to the operators of  $ACP_r$  in  $\mathbb{H}_\delta$ . The quotient  $\mathbb{H}_\delta / \equiv_{\mathcal{F}}$  was shown to be a model of  $ACP_r$ , called the graph model of  $ACP_r$ . Furthermore, a complete axiomatisation  $ACP_r + R1, 2 + S$  was given for  $\equiv_{\mathcal{F}}$  in the sense of

$$I(ACP_r + R1, 2 + S) \cong \mathbb{H}_\delta / \equiv_{\mathcal{F}}.$$

Here  $\mathbb{H}_\delta / \equiv_{\mathcal{F}}$  is short for  $\mathbb{H}_\delta(+, \cdot, \parallel, \lfloor \_ \rfloor, \lfloor \_ \rfloor, \partial_H, a_H, a, \delta) / \equiv_{\mathcal{F}}$ . In this section we will provide an *explicit representation* of the quotient structure  $\mathbb{H}_\delta(+, \cdot, \parallel, \lfloor \_ \rfloor, \lfloor \_ \rfloor, \partial_H, a_H, a, \delta) / \equiv_{\mathcal{F}}$ , called the *failure model* of  $ACP_r$ . The model will shed more light into the structure of failures, and—in connection with § 6.2—it will link our definitions with the original work on failures in [BHR84].

**5.1. The domain  $\mathbb{F}$  of failure sets.** First we introduce the domain of failure sets, denoted by  $\mathbb{F}$ . It consists of all finite subsets

$$F \subseteq A^+ \cup (A^* \times \mathcal{P}(A))$$

(where  $A^*$  is the set of finite words over  $A$ ,  $A^+$  is the set of nonempty finite words

over  $A$ , and  $\mathcal{P}(A)$  is the power set of  $A$ ) which satisfy the following closure properties:

- (i)  $[\varepsilon, \emptyset] \in F$ ;
- (ii)  $[\sigma_1 \sigma_2, \emptyset] \in F \Rightarrow [\sigma_1, \emptyset] \in F$ ;
- (iii)  $X \subseteq Y$  and  $[\sigma, Y] \in F \Rightarrow [\sigma, X] \in F$ ;
- (iv)  $[\sigma, X] \in F$  and  $[\sigma, X \cup \{a\}] \notin F \Rightarrow \sigma a \in F$  or  $[\sigma a, \emptyset] \in F$ ;
- (v)  $\sigma a \in F \Rightarrow [\sigma, \emptyset] \in F$ ;
- (vi)  $[\varepsilon, X] \in F \ \& \ a \in X \Rightarrow [a, \emptyset] \notin F$ .

The conditions (i)-(iv) on failure sets are exactly as in [BHR84]. Condition (v) deals with traces  $\sigma \in A^+$  which allow a direct definition of sequential composition without using (and later hiding again) an extra action  $\surd$  coding the event of successful termination as in [BHR84]. In § 6.2 on CSP we will restrict ourselves to CSP without successful termination. Then this difference is irrelevant. Condition (vi) is needed because we do not consider  $\tau$ -steps and hence no initial nondeterminism.

**5.2. Operations on failure sets.** Now we define the constants  $\delta, a (a \in A)$  and the operations  $+, \cdot, \parallel, \llbracket, \partial_H, a_H$  of ACP, directly on  $F$ . For  $F, G \in \mathbb{F}$  we put the following:

- (i)  $\delta = \{[\varepsilon, X] \mid X \subseteq A\}$ .
- (ii)  $a = \{[a, X] \mid X \subseteq A - \{a\}\} \cup \{a\}$ .  
Initially “ $a$ ” can refuse anything except “ $a$ .” After “ $a$ ” has occurred, the process successfully terminates.

- (iii)  $F + G = \{[\varepsilon, X] \mid [\varepsilon, X] \in F \cap G\} \cup \{\sigma \mid \sigma \in F \cup G\} \cup \{[\sigma, X] \mid \sigma \neq \varepsilon \wedge [\sigma, X] \in F \cup G\} \cup \{\sigma \mid \sigma \in F \cup G\} \cup \{[\sigma, X] \mid \sigma \neq \varepsilon \wedge [\sigma, X] \in F \cup G\}$ .

In its first step  $F + G$  can refuse only those actions which can be refused by both  $F$  and  $G$ . In all subsequent steps  $F + G$  behaves as  $F \cup G$ .

- (iv)  $F \cdot G = \{[\sigma, X] \mid [\sigma, X] \in F\} \cup \{\sigma_1 \sigma_2 \mid \sigma_1 \in F \wedge \sigma_2 \in G\} \cup \{[\sigma_1 \sigma_2, X] \mid \sigma_1 \in F \wedge [\sigma_2, X] \in G\}$ .

$F \cdot G$  first behaves like  $F$  and after successful termination of  $F$  in a trace  $\sigma_1$  continues to behave as  $G$ .

- (v) (1)  $F \parallel G = \{\sigma \mid \exists \sigma_1 \in F, \sigma_2 \in G: \sigma \in \sigma_1 \parallel \sigma_2\} \cup \{[\sigma, X] \mid \exists [\sigma_1, X_1] \in F, [\sigma_2, X_2] \in G: \sigma \in \sigma_1 \parallel \sigma_2 \wedge X \subseteq (X_1 \cap X_2) - \{(a \mid b) \mid a \notin X_1 \wedge b \notin X_2\}\}$
- (2)  $\wedge X \subseteq (X_1 \cap X_2) - \{(a \mid b) \mid a \notin X_1 \wedge b \notin X_2\}$
- (3)  $\cup \{[\sigma, X] \mid \exists \sigma_1 \in F, [\sigma_2, X_2] \in G: \sigma \in \sigma_1 \parallel \sigma_2 \wedge X = X_2\}$
- (4)  $\cup \{[\sigma, X] \mid \exists [\sigma_1, X_1] \in F, \sigma_2 \in G: \sigma \in \sigma_1 \parallel \sigma_2 \wedge X = X_1\}$

where  $\sigma_1 \parallel \sigma_2$  is the set of traces in  $A^*$  defined inductively by

$$\varepsilon \parallel \sigma = \sigma \parallel \varepsilon = \{\sigma\},$$

$$a \sigma_1 \parallel b \sigma_2 = a \cdot (\sigma_1 \parallel b \sigma_2) \cup b \cdot (a \sigma_1 \parallel \sigma_2) \cup [a \mid b] \cdot (\sigma_1 \parallel \sigma_2)$$

with  $[a \mid b] = \{(a \mid b)\}$  if  $a \mid b \neq \delta$  and  $\emptyset$  if  $a \mid b = \delta$ .

Thus  $\sigma_1 \parallel \sigma_2$  is the set of successful traces obtained by merging and communicating between  $\sigma_1$  and  $\sigma_2$ . For all traces  $\sigma_1 \in F$  and  $\sigma_2 \in G$  this set is included in  $F \parallel G$  (clause (1)). Besides traces  $F \parallel G$  contains certain failure pairs  $[\sigma, X]$ . If either  $F$  or  $G$  have already terminated,  $X$  is just the refusal set of the other, not-yet-terminated component  $G$  or  $F$  (clauses (3) and

(4)). If neither  $F$  nor  $G$  have terminated,  $X$  contains only actions that both  $F$  and  $G$  can refuse. This suggests  $X \subseteq X_1 \cap X_2$ , where  $X_1$  and  $X_2$  are the refusal sets of  $F$  and  $G$ . However,  $F \parallel G$  cannot refuse the possible communications between  $F$  and  $G$ . These communications can only be of the form  $(a|b)$  with  $a \notin X_1$  and  $b \notin X_2$ . This explains the condition

$$X \subseteq X_1 \cap X_2 - \{(a|b) \mid a \notin X_1 \wedge b \notin X_2\}$$

for the refusal set  $X$  of  $F \parallel G$  (clause (2)). Note that in case of  $(a|b) = \delta$  nothing is deduced from  $X_1 \cap X_2$ .

Clearly,  $F \parallel G$  and  $F|G$  are just variations of  $F \parallel G$  differing only in their first actions.

$$(vi) \quad F \parallel G = \{\sigma \mid \exists \sigma_1 \in F, \sigma_2 \in G: \sigma \in \sigma_1 \parallel \sigma_2\}$$

$$\cup \{[\varepsilon, X] \mid [\varepsilon, X] \in F\}$$

$$\cup \{[\sigma, X] \mid \sigma \neq \varepsilon \wedge \exists [\sigma_1, X_1] \in F, [\sigma_2, X_2] \in G: \sigma \in \sigma_1 \parallel \sigma_2 \\ \wedge X \subseteq (X_1 \cap X_2) - \{(a|b) \mid a \notin X_1 \wedge b \notin X_2\}\}$$

$$\cup \{[\sigma, X] \mid \sigma \neq \varepsilon \wedge \exists \sigma_1 \in F, [\sigma_2, X_2] \in G: \sigma \in \sigma_1 \parallel \sigma_2 \wedge X = X_2\}$$

$$\cup \{[\sigma, X] \mid \sigma \neq \varepsilon \wedge \exists [\sigma_1, X_1] \in F, \sigma_2 \in G: \sigma \in \sigma_1 \parallel \sigma_2 \wedge X = X_1\}$$

where  $\sigma_1 \parallel \sigma_2$  is the set of traces in  $A^*$  defined inductively by

$$\varepsilon \parallel \sigma = \emptyset,$$

$$a\sigma_1 \parallel \sigma_2 = a \cdot (\sigma_1 \parallel \sigma_2).$$

Until the completion of its first communication  $F \parallel G$  behaves as  $F$ . This explains why  $F \parallel G$  inherits all initial failure pairs  $[\varepsilon, X]$  of  $F$ . Afterwards  $F \parallel G$  behaves as  $F \parallel G$ .

$$(vii) \quad F|G = \{\sigma \mid \exists \sigma_1 \in F, \sigma_2 \in G: \sigma \in \sigma_1 | \sigma_2\}$$

$$\cup \{[\varepsilon, X] \mid \exists [\varepsilon, X_1] \in F, [\varepsilon, X_2] \in G: X \subseteq A \\ - \{(a|b) \mid a \notin X_1 \wedge b \notin X_2\}\}$$

$$\cup \{[\sigma, X] \mid \sigma \neq \varepsilon \wedge \exists [\sigma_1, X_1] \in F, [\sigma_2, X_2] \in G: \sigma \in \sigma_1 | \sigma_2 \\ \wedge X \subseteq (X_1 \cap X_2) - \{(a|b) \mid a \notin X_1 \wedge b \notin X_2\}\}$$

$$\cup \{[\sigma, X] \mid \sigma \neq \varepsilon \wedge \exists \sigma_1 \in F, [\sigma_2, X_2] \in G: \sigma \in \sigma_1 | \sigma_2 \wedge X = X_2\}$$

$$\cup \{[\sigma, X] \mid \sigma \neq \varepsilon \wedge \exists [\sigma_1, X_1] \in F, \sigma_2 \in G: \sigma \in \sigma_1 | \sigma_2 \wedge X = X_1\}$$

where  $\sigma_1 | \sigma_2$  is the set of traces in  $A^*$  defined inductively by

$$\varepsilon | \sigma_2 = \sigma_1 | \varepsilon = \emptyset,$$

$$a\sigma_1 | b\sigma_2 = [a|b] \cdot (\sigma_1 \parallel \sigma_2).$$

In its first step  $F|G$  requires a communication between  $F$  and  $G$ . Here initially  $F|G$  can refuse every set  $X$  of actions not containing possible communications between  $F$  and  $G$ . This explains the condition

$$X \subseteq A - \{(a|b) \mid a \notin X_1 \wedge b \notin X_2\}$$

for the failure pairs  $[\varepsilon, X]$ . After its first step  $F|G$  behaves like  $F \parallel G$ .

- (viii)  $\partial_H(F) = \{\sigma \mid \sigma \in F \text{ does not contain any } a \in H\}$   
 $\cup \{[\sigma, X \cup Y] \mid [\sigma, X] \in F, s \text{ does not contain any } a \in H, \text{ and } Y \subseteq H\}.$

In  $\partial_H(F)$  only those traces that do not contain any  $a \in H$  are successful, and the actions in  $H$  can be refused at any moment.

- (ix)  $a_H(F) = \{a_H(\sigma) \mid \sigma \in F\}$   
 $\cup \{[a_H(\sigma), X] \mid a \in X \wedge [\sigma, X \cup H] \in F\}$   
 $\cup \{[a_H(\sigma), X] \mid a \notin X \wedge [\sigma, X - H] \in F\}$

where the renaming operator  $a_H$  is applied pointwise to the elements in  $\sigma$ . A set  $X$  can be refused by  $a_H(F)$  if  $a_H^{-1}(X) = \{b \mid \exists c \in X: a_H(b) = c\}$  can be refused by  $F$ .

Except for the different representation of successful termination, the definitions of  $\delta, a, +, \cdot, a_H$  are as for STOP,  $a \rightarrow$  SKIP,  $\square, ;$  and direct image in [BHR84]. The definition of  $\parallel$  differs from the parallel composition operators in [BHR84]. In § 6.2 we will show how to interpret in  $ACP_r$  synchronous parallel composition of [BHR84]. The operators  $\parallel, |, \partial_H$  are not present in [BHR84].

**5.3. The failure model.** The *failure model* of  $ACP_r$  is now given by the structure  $\mathbb{F}(+, \cdot, \parallel, \llcorner, |, \partial_H, a_H, a, \delta)(a \in A)$ .

**THEOREM 5.3.1.** *The failure model of  $ACP_r$  is isomorphic to the graph model of  $ACP_r$ :*

$$\mathbb{H}_\delta(+, \cdot, \parallel, \llcorner, |, \partial_H, a_H, a, \delta) / \equiv_{\mathcal{F}} \cong \mathbb{F}(+, \cdot, \parallel, \llcorner, |, \partial_H, a_H, a, \delta).$$

*Proof.* Consider the mapping  $\mathcal{F}: \mathbb{H}_\delta \rightarrow \mathbb{F}$  introduced in § 2.2. It is clear that  $\mathcal{F}$  is well defined, i.e., that  $\mathcal{F}[g] \in \mathbb{F}$  holds for every  $g \in \mathbb{H}_\delta$ . Also, by Definition 2.2.3,  $g \equiv_{\mathcal{F}} h$  if and only if  $\mathcal{F}[g] = \mathcal{F}[h]$  for all  $g, h \in \mathbb{H}_\delta$ . Thus  $\mathcal{F}$  is also well defined and injective as a mapping:

$$\mathcal{F}: \mathbb{H}_\delta / \equiv_{\mathcal{F}} \rightarrow \mathbb{F}$$

(which, by abuse of language, we denote also with  $\mathcal{F}$ ). Now  $\mathcal{F}$  is surjective and behaves homomorphically over the operations  $+, \cdot, \parallel, \llcorner, |, \partial_H$ , and  $a_H$ . The proofs of these facts are tedious but follow in a straightforward way from the definitions of these operators on graphs (in § 1.2) and the definitions of the corresponding operators on  $\mathbb{F}$  (in § 5.1). We will not spell out these proofs. Thus  $\mathcal{F}$  is the required isomorphism from  $\mathbb{H}_\delta(\dots)$  to  $\mathbb{F}(\dots)$ .  $\square$

**6.  $ACP_r$  with one-to-one communication.** As a preparation for the subsequent section we now introduce some additional structure on the alphabet  $A_\delta$  and the communication function  $|: A_\delta \times A_\delta \rightarrow A_\delta$  of  $ACP_r$ .

**6.1. One-to-one communication.** First we assume that  $A$  (with typical elements  $a, b \in A$ ) is partitioned into  $A = C \cup I$ , where  $C$  (with typical elements  $c, d \in C$ ) is the set of *communicating* actions and  $I$  (disjoint from  $C$  and with typical elements  $i, j \in I$ ) is the set of *internal* actions. The set  $I$  will serve as an auxiliary tool for the communication function  $|$ .

Second, we denote by  $\alpha(x)$ , the *alphabet* of  $x$ , the set of non- $\delta$  actions occurring in the closed  $ACP_r$ -term  $x$ . For example,  $\alpha(a\delta + cd) = \{a, c, d\}$ . In subsequent results we will usually be interested in terms  $x$  with  $\alpha(x) \subseteq C$ , i.e., not involving internal,



auxiliary actions. Formally, the alphabet of a closed  $ACP_r$ -term  $x$  is defined by first eliminating the operators  $\parallel$ ,  $\ll$ ,  $|$ ,  $\partial_H$ , and  $a_H$  from  $x$ , using the axioms of  $ACP_r$ . (This is possible by virtue of an elimination theorem to this effect proved in [BK84a] for  $ACP$ ; the extra operators  $a_H$  in  $ACP_r$  present no problem.) The resulting closed term  $x'$  contains only the “basic constructors”  $+$  and  $\cdot$ , and we may further suppose that  $x'$  contains no subterm of the form  $(p+q)r$  (by some applications of axiom A4 of  $ACP_r$ , see Table 1); that is,  $x'$  uses only prefix multiplication. Now we define  $\alpha(x)$  to be  $\alpha(x')$ , where  $\alpha(x')$  is defined by the following clauses, using induction on the structure of  $x'$ :

$$\begin{aligned}\alpha(\delta) &= \emptyset, \\ \alpha(a) &= \{a\} \quad (a \in A), \\ \alpha(\delta x) &= \emptyset, \\ \alpha(ax) &= \{a\} \cup \alpha(x) \quad (a \in A), \\ \alpha(x+y) &= \alpha(x) \cup \alpha(y).\end{aligned}$$

(That  $\alpha(x)$  is indeed well defined in this way, follows from the confluence property of the rewriting procedure used in obtaining  $x'$  from  $x$ . This fact is for  $ACP$  also proved in [BK84a] and is easily carried over to  $ACP_r$ .)

LEMMA 6.1.1. *For closed terms  $x, y$  over  $ACP_r$ , with  $\alpha(x), \alpha(y) \subseteq C$  we have*

$$\partial_C(x \parallel y) = \partial_C(x | y).$$

*Proof.* It suffices to show that  $\partial_C(x \ll y) = \delta$ . Recall that  $x$  can be normalized in  $ACP_r$  to

$$x = \sum_i c_i x_i + \sum_j d_j$$

with  $c_i, d_j \in C$ , and with the empty sum  $\Sigma$  denoting  $\delta$ . Thus

$$x \ll y = \sum_i c_i (x_i \parallel y) = \sum_j d_j y$$

which implies  $\partial_C(x \ll y) = \delta$ .  $\square$

DEFINITION 6.1.2. Assuming the above partition of the alphabet  $A$  we say  $ACP_r$  has *one-to-one communication* if for the communication merge  $|$  there exists a bijection  $\varphi: C \rightarrow C$  such that  $c | \varphi(c) \in I$  for every  $c \in C$ , and  $a | b = \delta$  otherwise.

Note that  $c | \varphi(c) \in I$  implies  $c | \varphi(c) \neq \delta$ . Next, we show that the definitions of parallel composition used in CSP and CCS are typical examples of one-to-one communication.

**6.2. Hoare's parallel composition  $\parallel_{\mathscr{H}}$  in CSP.** In [BHR84] Hoare et al. propose an operation  $x \parallel_{\mathscr{H}} y$  modelling the full synchronization of processes  $x$  and  $y$ . We shall consider  $\parallel_{\mathscr{H}}$  here within a small subset of the language CSP [BHR84] which we call “CSP.” The signature of “CSP” is given by

- the constant STOP,
- unary prefix operators  $c \rightarrow$ , for  $c \in C$ ,
- the binary infix operators  $\square$  and  $\parallel_{\mathscr{H}}$ .

Here  $C$  is a given set of communication actions, contained in the overall alphabet  $A$ .

The semantics of “CSP” is determined by the failures model of [BHR84]. It is based on the failures domain  $F_{\text{BHR}}$  consisting of all subsets

$$F \subseteq A^* \times \mathcal{P}(A)$$

satisfying the closure properties (i)–(iv) discussed in § 5.1. The additional closure property (v) on traces is not needed here since the failure sets  $F \in \mathbb{F}_{\text{BHR}}$  contain only failure pairs  $[\sigma, X]$ .

The failure model assigns to each closed ‘‘CSP’’ term  $x$  a failure set  $\mathcal{F}_{\text{BHR}}\llbracket x \rrbracket$  in the domain  $\mathbb{F}_{\text{BHR}}$ . According to [BHR84] the definition is as follows:

- (i)  $\mathcal{F}_{\text{BHR}}\llbracket \text{STOP} \rrbracket = \{[\varepsilon, X] \mid X \subseteq A\}$ ;
- (ii)  $\mathcal{F}_{\text{BHR}}\llbracket c \rightarrow x \rrbracket = \{[\varepsilon, X] \mid X \subseteq A - \{c\}\} \cup \{[c \cdot \sigma, X] \mid [\sigma, X] \in \mathcal{F}_{\text{BHR}}\llbracket x \rrbracket\}$ ;
- (iii)  $\mathcal{F}_{\text{BHR}}\llbracket x \square y \rrbracket = \{[\varepsilon, X] \mid [\varepsilon, X] \in \mathcal{F}_{\text{BHR}}\llbracket x \rrbracket \cap \mathcal{F}_{\text{BHR}}\llbracket y \rrbracket\}$   
 $\cup \{[\sigma, X] \mid \sigma \neq \varepsilon \wedge [\sigma, X] \in \mathcal{F}_{\text{BHR}}\llbracket x \rrbracket \cup \mathcal{F}_{\text{BHR}}\llbracket y \rrbracket\}$ ;
- (iv)  $\mathcal{F}_{\text{BHR}}\llbracket x \parallel y \rrbracket = \{[\sigma, X \cup Y] \mid [\sigma, X] \in \mathcal{F}_{\text{BHR}}\llbracket x \rrbracket \wedge [\sigma, Y] \in \mathcal{F}_{\text{BHR}}\llbracket y \rrbracket\}$ .

The failure model induces the following failure equivalence  $\equiv_{\mathcal{F}, \text{BHR}}$  on closed ‘‘CSP’’ terms  $x$  and  $y$ :

$$x \equiv_{\mathcal{F}, \text{BHR}} y \quad \text{iff} \quad \mathcal{F}_{\text{BHR}}\llbracket x \rrbracket = \mathcal{F}_{\text{BHR}}\llbracket y \rrbracket.$$

We now link these definitions of [BHR84] to our present setting by interpreting ‘‘CSP’’ in  $\text{ACP}_r$  with one-to-one communication. Let  $C = \{c_1, \dots, c_n\}$ . Then we take  $A = C \cup I$  with

$$I = \{\hat{c}_1, \dots, \hat{c}_n\}$$

where the  $\hat{c}_i (i = 1, \dots, n)$  are new copies of the actions  $c_i$  in  $C$ . Furthermore, one-to-one communication is introduced by putting  $\varphi(c) = c$  and  $c \mid c = \hat{c}$  for every  $c \in C$ . The interpretation of ‘‘CSP’’ in  $\text{ACP}_r$  is given by a mapping  $\mathcal{I}$  from closed ‘‘CSP’’ terms into closed  $\text{ACP}_r$  terms defined as follows:

- (i)  $\mathcal{I}(\text{STOP}) = \delta$ ;
- (ii)  $\mathcal{I}(c \rightarrow x) = c \cdot \mathcal{I}(x)$ ;
- (iii)  $\mathcal{I}(x \square y) = \mathcal{I}(x) + \mathcal{I}(y)$ ;
- (iv)  $\mathcal{I}(x \parallel y) = C_I(\partial_C(\mathcal{I}(x) \parallel \mathcal{I}(y)))$

where  $C_I$  abbreviates the composite operator  $(c_i)_{\{\hat{c}_i\}} \circ \dots \circ (c_n)_{\{\hat{c}_n\}}$ , built from the renaming operators  $(c_i)_{\{\hat{c}_i\}} (i = 1, \dots, n)$  that rename  $c_i$  into  $\hat{c}_i$ .

This interpretation is justified by the following result.

**PROPOSITION 6.2.1.** *For closed ‘‘CSP’’ terms  $x$*

$$\mathcal{F}_{\text{BHR}}\llbracket x \rrbracket = \mathcal{F}\llbracket \mathcal{I}(x) \rrbracket \subseteq C^* \times \mathcal{P}(A)$$

*holds where  $\mathcal{F}$  is the  $\text{ACP}_r$  failures model of § 5. In particular  $\mathcal{F}\llbracket \mathcal{I}(x) \rrbracket$  does not contain any traces  $\sigma$  signaling successful termination, only failure pairs  $[\sigma, X]$ .*

*Proof.* By induction on the structure of  $x$ . The cases (i)–(iii) are immediate. Case (iv), parallel composition, is more tedious. It is easy to see that both

$$\mathcal{F}_{\text{BHR}}\llbracket x \parallel y \rrbracket, \mathcal{F}\llbracket C_I(\partial_C(\mathcal{I}(x) \parallel \mathcal{I}(y))) \rrbracket \subseteq C^* \times \mathcal{P}(A).$$

Hence the closure properties of the failure domains  $\mathbb{F}_{\text{BHR}}$  and  $\mathbb{F}$ , respectively, imply

$$\begin{aligned} [\sigma, X] \in \mathcal{F}_{\text{BHR}}\llbracket x \parallel y \rrbracket & \quad \text{iff} \quad [\sigma, X \cup Y] \in \mathcal{F}_{\text{BHR}}\llbracket x \parallel y \rrbracket, \\ [\sigma, X] \in \mathcal{F}\llbracket C_I(\partial_C(\mathcal{I}(x) \parallel \mathcal{I}(y))) \rrbracket & \quad \text{iff} \quad [\sigma, X \cup Y] \in \mathcal{F}\llbracket C_I(\partial_C(\mathcal{I}(x) \parallel \mathcal{I}(y))) \rrbracket \end{aligned}$$

for arbitrary  $Y \subseteq A - C$ . Thus it suffices to show

$$[\sigma, X] \in \mathcal{F}_{\text{BHR}}\llbracket x \parallel y \rrbracket \quad \text{iff} \quad [\sigma, X] \in \mathcal{F}\llbracket C_I(\partial_C(\mathcal{I}(x) \parallel \mathcal{I}(y))) \rrbracket$$

for  $\sigma \in C^*$  and  $X \subseteq C$ .

Let  $\hat{\sigma}$  and  $\hat{X}$  result from  $\sigma$  and  $X$  by replacing pointwise each action  $c$  by  $\hat{c}$ . In particular, we have  $\hat{C} = A - C$ . Then for  $\sigma \in C^*$  and  $X \subseteq C$

$$[\sigma, X] \in \mathcal{F}_{\text{BHR}}[x \parallel_{\neq} y]$$

if and only if (induction hypothesis, definition of  $\parallel_{\neq}$ )

$$\exists X_1 \subseteq C, X_2 \subseteq C:$$

$$[\sigma, X_1] \in \mathcal{F}[\mathcal{I}(x)] \wedge [\sigma, X_2] \in \mathcal{F}[\mathcal{I}(y)] \wedge X \subseteq X_1 \cup X_2$$

if and only if (definition  $\hat{X}$ )

$$\exists X_1 \subseteq C, X_2 \subseteq C:$$

$$[\sigma, X_1] \in \mathcal{F}[\mathcal{I}(x)] \wedge [\sigma, X_2] \in \mathcal{F}[\mathcal{I}(y)] \wedge \hat{X} \subseteq \{\hat{c} \mid c \in X_1 \cup X_2\}$$

if and only if (closure properties of the failure domain F)

$$\exists X_1, X_2: \hat{C} \subseteq X_1 \subseteq A \wedge \hat{C} \subseteq X_2 \subseteq A$$

$$\wedge [\sigma, X_1] \in \mathcal{F}[\mathcal{I}(x)] \wedge [\sigma, X_2] \in \mathcal{F}[\mathcal{I}(y)]$$

$$\wedge \hat{X} \subseteq X_1 \cap X_2 - \{\hat{c} \mid c \notin X_1 \cup X_2\}$$

if and only if (one-to-one communication, definition  $\parallel$ )

$$[\hat{\sigma}, \hat{X}] \in \mathcal{F}[\mathcal{I}(x) \parallel \mathcal{I}(y)]$$

if and only if (definition  $C_I, \partial_C$ )

$$[\sigma, X] \in \mathcal{F}[C_I(\partial_C(\mathcal{I}(x) \parallel \mathcal{I}(y)))]$$

This finishes our proof.  $\square$

Consequently, for ‘‘CSP’’ the original failure equivalence  $\equiv_{\mathcal{F}_{\text{BHR}}}$  of [BHR84] coincides with our definition of failure equivalence  $\equiv_{\mathcal{F}}$  in § 2. More precisely we have the following corollary:

**COROLLARY 6.2.2.** *For closed ‘‘CSP’’ terms  $x$  and  $y$*

$$x \equiv_{\mathcal{F}_{\text{BHR}}} y \quad \text{iff} \quad \mathcal{I}(x) \equiv_{\mathcal{F}} \mathcal{I}(y).$$

For closed ‘‘CSP’’ terms  $x$  and  $y$  the notions of trace and trace equivalence are defined via the interpretation in ACP<sub>r</sub>:

$$\text{trace}(x) = \text{trace}(\mathcal{I}(x)),$$

$$x \sim_{\text{tr}} y \quad \text{iff} \quad \mathcal{I}(x) \sim_{\text{tr}} \mathcal{I}(y).$$

(Actually, trace is in § 2.1 only defined on graphs; using the operation graph from § 4.1.2 we now define for a term  $x$ , trace( $x$ ) as trace(graph( $x$ )).) Using Proposition 6.2.1 the trace set of a term  $x$  can also be computed directly from its failure set  $\mathcal{F}_{\text{BHR}}[x]$ :

$$\text{trace}(x) = \{\sigma \cdot \delta \mid [\sigma, A] \in \mathcal{F}_{\text{BHR}}[x]\}.$$

Recall that in our paper we only consider *complete* traces, either leading to a deadlock  $\delta$  or to successful termination (not possible for ‘‘CSP’’). In [BHR84] the word ‘‘trace’’ is used as well, but it refers to any sequence  $\sigma$  with

$$[\sigma, \emptyset] \in \mathcal{F}_{\text{BHR}}[x].$$

Such sequences were called *histories* in § 2.

**6.3. Milner's parallel composition  $\parallel_{\mathcal{M}}$  in CCS.** Since the parallel composition  $\parallel$  in  $ACP_r$  can be seen as a generalization of Milner's operation  $\parallel_{\mathcal{M}}$  in CCS [Mi80], it is easy to regain the original definition. As for CSP, we do this within a small subset of CCS which we call "CCS." Milner stipulates that the set  $C$  of communicating actions is equipped with a bijection  $\bar{\cdot} : C \rightarrow C$  satisfying  $\bar{\bar{c}} = c$ . Here  $\bar{c}$  is called the *matching* action of  $c$ . In addition to communicating actions Milner uses a symbol  $\tau$  denoting the so-called *silent* action. We will write  $\hat{\tau}$  because we work here without Milner's  $\tau$ -laws that make  $\tau$  silent or invisible (see the discussion below and § 8). Hence the alphabet for "CCS" will be  $A = C \cup \{\hat{\tau}\}$ .

The signature of "CCS" consists of the following:

- the constant NIL;
- unary prefix operators  $a \cdot$ , for  $a \in A$ ;
- unary postfix operators  $\setminus H$ , for  $H \subseteq C$ ;
- the binary infix operators  $+$  and  $\parallel_{\mathcal{M}}$ .

Informally,  $x \parallel_{\mathcal{M}} y$  denotes the nondeterministic interleaving of  $x$  and  $y$ , plus the communication of  $x$  and  $y$  via matching actions which then yield  $\hat{\tau}$  as a result. Following [Mi80], this can be expressed by the infinite axiom scheme:

$$(*) \quad (\sum_i a_i x_i) \parallel_{\mathcal{M}} (\sum_j b_j y_j) = \sum_i a_i (x_i \parallel_{\mathcal{M}} y) + \sum_j b_j (x \parallel_{\mathcal{M}} y_j) + \sum_{a_i = \bar{b}_j} \hat{\tau} \cdot (x_i \parallel_{\mathcal{M}} y_j)$$

where  $x = \sum_i a_i x_i$  and  $y = \sum_j b_j y_j$ .

We shall define the semantics of  $\parallel_{\mathcal{M}}$  via an interpretation  $\mathcal{F}$  of "CCS" in  $ACP_r$ , with one-to-one communication. To this end, take  $I = \{\hat{\tau}\}$  and define

$$\varphi(c) = \bar{c} \text{ and } c | \bar{c} = \hat{\tau}.$$

Then  $\mathcal{F}$  is rather trivial:

- (i)  $\mathcal{F}(\text{NIL}) = \delta$ ;
- (ii)  $\mathcal{F}(a \cdot x) = a \cdot \mathcal{F}(x)$ ;
- (iii)  $\mathcal{F}(x \setminus H) = \partial_H(\mathcal{F}(x))$ ;
- (iv)  $\mathcal{F}(x + y) = \mathcal{F}(x) + \mathcal{F}(y)$ ;
- (v)  $\mathcal{F}(x \parallel_{\mathcal{M}} y) = \mathcal{F}(x) \parallel \mathcal{F}(y)$ .

Note that the auxiliary operations  $\parallel$  and  $|$  in  $ACP_r$  serve to replace the infinite axiom scheme (\*) by finitely many  $ACP_r$  axioms.

In [Mi80] Milner studies CCS terms under the (weak) bisimulation equivalence [Pa83]; however, here we shall study "CCS" under the failure equivalence. For closed "CCS" terms  $x$  and  $y$  we define the notions of failure equivalence, trace equivalence and alphabet via the interpretation  $\mathcal{F}$  in  $ACP_r$ :

$$\begin{aligned} x \equiv_{\mathcal{F}} y & \text{ iff } \mathcal{F}(x) \equiv_{\mathcal{F}} \mathcal{F}(y), \\ x \sim_{\text{tr}} y & \text{ iff } \mathcal{F}(x) \sim_{\text{tr}} \mathcal{F}(y), \\ \alpha(x) & = \alpha(\mathcal{F}(x)). \end{aligned}$$

In general, these definitions are not quite appropriate for CCS because  $\tau$  should be silent or invisible; more formally  $\tau$  should be subject to Milner's  $\tau$ -laws. In the above interpretation of "CCS"  $\hat{\tau}$  remains visible, i.e., recorded in the traces and failure pairs. The reason for this clash is that CCS indivisibly couples parallel composition  $\parallel_{\mathcal{M}}$  and  $\tau$ , whereas we decided to separate failure equivalence  $\equiv_{\mathcal{F}}$  from  $\tau$ .

However, we can regain the spirit of CCS if we restrict the failure equivalence to  $\hat{\tau}$ -free "CCS" terms  $x$  and  $y$ , i.e., with

$$\hat{\tau} \notin \alpha(x), \alpha(y).$$

Unfortunately,  $\hat{\tau}$ -free “CCS” terms are not closed under parallel composition  $\parallel_{\mathcal{M}}$ . Therefore we shall consider also a modified trace set

$$\text{trace}_{\hat{\tau}}(x)$$

for “CCS” terms  $x$  which results from  $\text{trace}(x)$  by deleting in every trace  $\sigma \cdot \delta \in \text{trace}(x)$  all occurrences of  $\hat{\tau}$  in  $\sigma$ . Then  $\text{trace}_{\hat{\tau}}(x)$  represents the set of complete traces in the sense of CCS. For example,

$$\begin{aligned} \text{trace}(c\text{NIL} \parallel_{\mathcal{M}} \bar{c}\text{NIL}) &= \{c\bar{c}\delta, \bar{c}c\delta, \hat{\tau}\delta\}, \\ \text{trace}_{\hat{\tau}}(c\text{NIL} \parallel_{\mathcal{M}} \bar{c}\text{NIL}) &= \{c\bar{c}\delta, \bar{c}c\delta, \delta\}. \end{aligned}$$

**7. The maximal trace respecting congruence.** In § 4 (Proposition 4.2.3) it was shown that failure equivalence  $\equiv_{\mathcal{F}}$  is a congruence with respect to the operators of  $\text{ACP}_r$ . In this section we will prove that for  $\text{ACP}_r$  with one-to-one communication failure equivalence is in fact the maximal trace respecting congruence. This implies a full abstraction result for the failure model of § 5. But first let us introduce the relevant concepts.

**7.1. Preliminaries.** Let  $\Sigma$  be a signature with  $\text{Ter}(\Sigma)$  denoting the set of closed terms over  $\Sigma$ . By  $\text{Ter}(\Sigma)[\xi]$  we denote the set of terms over  $\Sigma$  with  $\xi$  as free variable. These terms are called *contexts* and are typically written as  $\mathcal{C}[\xi]$ .

Let  $\mathcal{T} \subseteq \text{Ter}(\Sigma)$ . A *congruence for  $\mathcal{T}$*  is an equivalence relation  $\equiv$  on  $\mathcal{T}$ , such that

$$x \equiv y \text{ implies } \mathcal{C}[x] \equiv \mathcal{C}[y]$$

for all terms  $x, y \in \mathcal{T}$  and contexts  $\mathcal{C}[\xi] \in \text{Ter}(\Sigma)[\xi]$  with  $\mathcal{C}[x], \mathcal{C}[y] \in \mathcal{T}$ . A congruence  $\equiv$  for  $\mathcal{T}$  is *trace respecting* if

$$x \equiv y \text{ implies } \text{trace}(x) = \text{trace}(y)$$

for all  $x, y \in \mathcal{T}$ . A trace respecting congruence  $\equiv$  for  $\mathcal{T}$  is called *maximal* if for all  $x, y \in \mathcal{T}$ ,  $x \not\equiv y$  implies that there exists some context  $\mathcal{C}[\xi] \in \text{Ter}(\Sigma)[\xi]$  with  $\mathcal{C}[x], \mathcal{C}[y] \in \mathcal{T}$  and  $\text{trace}(\mathcal{C}[x]) \neq \text{trace}(\mathcal{C}[y])$ .

**PROPOSITION 7.1.1.** *For each  $\mathcal{T} \subseteq \text{Ter}(\Sigma)$  the maximal trace respecting congruence for  $\mathcal{T}$  exists and is unique.*

*Proof. Uniqueness.* Suppose  $\equiv_1$  and  $\equiv_2$  are different maximal trace respecting congruences on  $\mathcal{T}$ . Then for some  $x, y \in \mathcal{T}$  we have

$$x \equiv_1 y, \text{ but } x \not\equiv_2 y.$$

Since  $\equiv_1$  is a trace respecting congruence on  $\mathcal{T}$ ,  $\text{trace}(\mathcal{C}[x]) = \text{trace}(\mathcal{C}[y])$  holds for every context  $\mathcal{C}[\xi] \in \text{Ter}(\Sigma)[\xi]$  with  $\mathcal{C}[x], \mathcal{C}[y] \in \mathcal{T}$ . But this contradicts the maximality of  $\not\equiv_2$ .

*Existence.* Define  $\equiv$ , a binary relation on  $\mathcal{T}$ , as follows:  $x \equiv y$  if and only if for every context  $\mathcal{C}[\xi] \in \text{Ter}(\Sigma)[\xi]$  with  $\mathcal{C}[x], \mathcal{C}[y] \in \mathcal{T}$ ,  $\text{trace}(\mathcal{C}[x]) = \text{trace}(\mathcal{C}[y])$  holds.

It is easy to see that  $\equiv$  is a trace respecting congruence for  $\mathcal{T}$ ; maximality follows from its definition.  $\square$

**7.2. A characterisation of failure equivalence.** Let us now turn to  $\text{ACP}_r$ . We write  $\text{Ter}(\text{ACP}_r)$  instead of  $\text{Ter}(\Sigma)$ . From § 4 we know that failure equivalence  $\equiv_{\mathcal{F}}$  is a trace respecting congruence for  $\text{Ter}(\text{ACP}_r)$ . (For the sake of convenience, we have identified here the semantical notion  $\equiv_{\mathcal{F}}$  with the equivalence induced by  $\equiv_{\mathcal{F}}$  on  $\text{Ter}(\text{ACP}_r)$  via the correspondence between process graphs and terms, explained in § 4.1.) Thus for  $\text{ACP}_r$ , in general, we have

$$\equiv_{\mathcal{F}} \subseteq \equiv_{\max}$$

with  $\equiv_{\max}$  denoting the maximal trace respecting congruence for  $\text{Ter}(\text{ACP}_r)$ . If we specialize  $\text{ACP}_r$  to the case of one-to-one communication, we can actually prove that

$$\equiv_{\mathcal{F}} = \equiv_{\max},$$

and thus arrive at a very pleasing characterization of failure equivalence.

**THEOREM 7.2.1.** *Consider  $\text{ACP}_r$  with one-to-one communication. Then failure equivalence  $\equiv_{\mathcal{F}}$  is the maximal trace respecting congruence for the set  $\mathcal{T}_C$  of all closed terms  $x$  over  $\text{ACP}_r$  with alphabet  $\alpha(x) \subseteq C$ .*

*Proof.* Suppose  $x \not\equiv_{\mathcal{F}} y$ ; i.e.,  $\mathcal{F}[x] \neq \mathcal{F}[y]$  holds for  $x, y \in \mathcal{T}_C$ . If  $\text{trace}(x) \neq \text{trace}(y)$ , the trivial context  $\mathcal{C}[\xi] = \xi$  will do. Now suppose that  $\text{trace}(x) = \text{trace}(y)$  holds. Because of  $x \not\equiv_{\mathcal{F}} y$  we can assume without loss of generality that there exists a failure pair  $[\sigma, X]$  with

$$[\sigma, X] \in \mathcal{F}[x], [\sigma, X] \notin \mathcal{F}[y].$$

By the definition of  $\mathcal{F}$ ,  $[\sigma, X] \in \mathcal{F}[x]$  implies that there exists some ready pair  $(\sigma, Z) \in \mathcal{R}[x]$  with  $X \subseteq Z$ . Note that  $Z \neq \emptyset$ . Suppose we had  $(\sigma, \emptyset) \in \mathcal{R}[x]$ . Then  $\sigma\delta \in \text{trace}(x) = \text{trace}(y)$  and  $(\sigma, \emptyset) \in \mathcal{R}[y]$ . Thus  $[\sigma, C] \in \mathcal{F}[y]$  and therefore also  $[\sigma, X] \in \mathcal{F}[x]$ , a contradiction.

Trace equivalence of  $x$  and  $y$  implies that there exists a ready pair  $(\sigma, Y) \in \mathcal{R}[y]$  with  $Y \neq \emptyset$ . Again by the definition of  $\mathcal{F}$ ,  $[\sigma, X] \notin \mathcal{F}[y]$  implies that for every such ready pair  $(\sigma, Y) \in \mathcal{R}[y]$  there exists some  $d \in X \cap Y$ . Now consider a context of the form

$$\mathcal{C}[\xi] = (c_{1\{i_1\}} \circ \dots \circ c_{n\{i_n\}} \circ \partial_C)(x \parallel \varphi(\sigma) \cdot \Sigma \varphi(d) \cdot \delta)$$

where the sum  $\Sigma$  is taken over all  $d \in X \cap Y$  such that  $(\sigma, Y) \in \mathcal{R}[y]$ . Furthermore  $I = \{i_1, \dots, i_n\}$ ,  $c_1, \dots, c_n \in C$ ,  $\varphi$  is the bijection describing the one-to-one communication in  $\text{ACP}_r$  and  $\varphi(\sigma)$  is the result of applying  $\varphi$  pointwise to  $\sigma$ . Note that  $\mathcal{C}[\xi]$  is uniquely determined by  $x$  and  $y$  except for the choice of the  $c_1, \dots, c_n$  in the renaming operators. Note that indeed  $\mathcal{C}[x], \mathcal{C}[y] \in \mathcal{T}_C$  due to the presence of operators  $\partial_C$  and  $c_{j\{i_j\}}$  in  $\mathcal{C}[\xi]$ . We now claim that

$$(c_{1\{i_1\}} \circ \dots \circ c_{n\{i_n\}})(\sigma \mid \varphi(\sigma)) \cdot \delta \in \text{trace}(\mathcal{C}[x]), \notin \text{trace}(\mathcal{C}[y])$$

where  $\sigma \mid \varphi(\sigma)$  is understood by applying  $\mid$  pointwise to  $\sigma$  and  $\varphi(\sigma)$ .

To prove this claim we first state a general observation about ready sets  $\mathcal{R}[z]$  of closed terms  $z$  over  $\text{ACP}_r$ . Let  $\sigma = a_1 \dots a_m$  and  $Z = \{b_1, \dots, b_n\}$ . Then  $(\sigma, Z) \in \mathcal{R}[z]$  if and only if there exist  $x_1, \dots, x_m, y_1, \dots, y_n \in \text{Ter}(\text{ACP}_r)$  with

$$\text{ACP}_r \vdash x = a_1(a_2 \dots (a_m(b_1y_1 + \dots + b_ny_n) + x_m) \dots + x_2) + x_1.$$

This observation is obvious from §§ 3 and 4.

Next we recall from Lemma 6.1.1 that due to the encapsulation  $\partial_C$  we can replace the general parallel composition  $\parallel$  in  $\mathcal{C}[\xi]$  by the communication operator  $\mid$  which enforces synchronization.

Combining these two facts, it is easy to calculate that  $(\sigma, Z) \in \mathcal{R}[x]$  with  $X \subseteq Z$  yields

$$(c_{1\{i_1\}} \circ \dots \circ c_{n\{i_n\}})(\sigma \mid \varphi(\sigma)) \cdot \delta \in \text{trace}(\mathcal{C}[x]).$$

Now suppose that this trace is also present in  $\text{trace}(\mathcal{C}[y])$ . Since  $\text{ACP}_r$  allows only one-to-one communication, there exists a history  $\sigma \in C^*$  such that every ready pair  $(\sigma, Y) \in \mathcal{R}[y]$  satisfies  $X \cap Y = \emptyset$ . This is a contradiction. This finishes our proof.  $\square$

**7.3. Application to CSP and CCS.** The characterization of failure equivalence for  $ACP_r$  yields corresponding results for the subsets “CSP” and “CCS” or [BHR84] and [Mi80].

**COROLLARY 7.3.1.** *For closed “CSP” terms the failure equivalence  $\equiv_{\mathcal{F},\text{BHR}}$  of [BHR84] is the maximal trace respecting congruence.*

*Proof.* Via the interpretation  $\mathcal{I}$  the failure equivalence  $\equiv_{\mathcal{F},\text{BHR}}$  is a trace respecting congruence for “CSP.” To show maximality, suppose  $x \not\equiv_{\mathcal{F},\text{BHR}} y$  for closed terms  $x$  and  $y$ . Then  $\mathcal{I}(x) \not\equiv_{\mathcal{F}} \mathcal{I}(y)$  by Corollary 6.2.2. Since  $\alpha(\mathcal{I}(x)), \alpha(\mathcal{I}(y)) \subseteq C$ , Theorem 7.2.1 applies and yields a context  $\mathcal{C}[\xi]$  in  $ACP_r$  with

$$\mathcal{C}[\mathcal{I}(x)] \not\sim_{\text{tr}} \mathcal{C}[\mathcal{I}(y)].$$

Looking at the proof of Theorem 7.2.1 we see that  $\mathcal{C}[\xi]$  can be expressed in “CSP”; i.e., there exists a context  $\mathcal{C}'[\xi]$  in “CSP” with

$$I(\mathcal{C}')[\xi] = \mathcal{C}[\xi]$$

where we stipulate  $\mathcal{I}(\xi) = \xi$ . Thus

$$\mathcal{I}(\mathcal{C}')[\mathcal{I}(x)] \not\sim_{\text{tr}} \mathcal{I}(\mathcal{C}')[\mathcal{I}(y)].$$

Since  $\mathcal{I}$  is defined by structural induction, we have  $\mathcal{I}(\mathcal{C}')[\mathcal{I}(x)] = \mathcal{I}(\mathcal{C}'[x])$  and likewise for  $y$ . Thus

$$\mathcal{C}'[x] \not\sim_{\text{tr}} \mathcal{C}'[y]$$

by the definition of trace equivalence for “CSP.”  $\square$

Due to the differences of  $\tau$  and  $\hat{\tau}$  in CCS and  $ACP_r$  (see § 6.3), we can characterize failure equivalence only for  $\hat{\tau}$ -free “CCS” terms.

**COROLLARY 7.3.2.** *On the subset of closed,  $\hat{\tau}$ -free “CCS” terms failure equivalence  $\equiv_{\mathcal{F}}$  coincides with the maximal trace respecting congruence defined for full “CCS.” This result holds for both notions of trace introduced for “CCS” terms, viz.,  $\text{trace}(\cdot)$  and  $\text{trace}_{\hat{\tau}}(\cdot)$ .*

*Proof.* Via the interpretation  $\mathcal{I}$  failure equivalence  $\equiv_{\mathcal{F}}$  is a trace respecting congruence for “CCS.” This holds for the original definition of  $\text{trace}(\cdot)$ , however, since

$$\text{trace}(x) = \text{trace}(y) \quad \text{implies} \quad \text{trace}_{\hat{\tau}}(x) = \text{trace}_{\hat{\tau}}(y),$$

it holds for  $\text{trace}_{\hat{\tau}}(\cdot)$  as well.

Now consider two closed,  $\hat{\tau}$ -free “CCS” terms  $x, y$  such that  $x \not\equiv_{\mathcal{F}} y$ , i.e.,  $\mathcal{I}(x) \not\equiv_{\mathcal{F}} \mathcal{I}(y)$ . Since  $\hat{\tau}$ -freeness means  $\alpha(\mathcal{I}(x)), \alpha(\mathcal{I}(y)) \subseteq C$ , the proof technique for Theorem 7.2.1 applies and yields an  $ACP_r$  context of the form

$$\mathcal{C}[\xi] = \partial_C(\xi \parallel \mathcal{I}(z))$$

where  $z$  is a closed,  $\hat{\tau}$ -free “CCS” term such that for some  $n \geq 0$

$$\hat{\tau}^n \cdot \delta \in \text{trace}(\mathcal{C}[\mathcal{I}(x)]), \notin \text{trace}(\mathcal{C}[\mathcal{I}(y)]).$$

Note that in the definition of  $\mathcal{C}[\xi]$  we deviate slightly from Theorem 7.2.1 and omit the renaming operator, which would yield here  $c_{\{\hat{\tau}\}}$  for some  $c \in C$ . The reason is that  $\tau$  (respectively,  $\hat{\tau}$ ) cannot be renamed in Milner’s [Mi80] (and hence “CCS”).

The above  $\mathcal{C}[\xi]$  can be translated back into the “CCS” context

$$\mathcal{C}'[\xi] = (\xi \parallel_{\mathcal{M}} z) \setminus C,$$

which yields

$$\hat{\tau}^n \cdot \delta \in \text{trace}(\mathcal{C}'[x]), \notin \text{trace}(\mathcal{C}'[y]),$$

and thus

$$\delta \in \text{trace}_{\hat{\tau}}(\mathcal{C}'[x]), \notin \text{trace}_{\hat{\tau}}(\mathcal{C}'[y]).$$

This proves the maximality of failure equivalence for  $\hat{\tau}$ -free ‘‘CCS’’ terms with respect to both notions of trace.  $\square$

Thus the (proof of) Theorem 7.2.1 gives a uniform argument for the communication mechanisms of both ‘‘CSP’’ and CCS.’’

*Remark 7.3.3.* (Comparison with the work of De Nicola and Hennessy [DH84].) We have proved that (under a restricted communication format) processes are failure equivalent if and only if they cannot be separated by any context where ‘‘separated’’ refers to the criterion of having different traces. This characterisation is easy to understand, as it involves only the notions of *trace* and *context*. It is interesting to compare our result with a result in [DH84]. Since the settings are quite different (here finite processes in  $ACP_r$ , there CCS with recursion,  $\tau$ -steps and an additional constant  $\Omega$  denoting the undefined state), we state the comparison for the greatest common denominator of  $ACP_r$  and CCS, viz., the language ‘‘CCS’’ of § 6.3.

De Nicola and Hennessy [DH84] set up a notion of *testing* and consider two processes  $p$  and  $q$  as equivalent if and only if they pass exactly the same tests. This idea of testing is very appealing, but the formal definitions are somewhat more technical. Both processes and tests are just terms over the signature of ‘‘CCS.’’ However, in the alphabet  $A$  we assume a distinguished action  $\omega$ , which may appear in tests only. The action  $\omega$  is interpreted as reporting success; it is needed in the definition of a process passing a test. Due to the restriction to ‘‘CCS,’’ we can phrase De Nicola and Hennessy’s definition as follows.

For ‘‘CCS’’ terms  $p, q, r$  and actions  $a \in A$  we write

$$p \xrightarrow{a} q \quad \text{if } \exists r: \text{‘‘CCS’’} \vdash p = a \cdot q + r,$$

$$p \xrightarrow{a} \quad \text{if } \exists q: p \xrightarrow{a} q.$$

Intuitively,  $p \xrightarrow{a} q$  states that  $p$  can perform an action  $a$  and then behave like  $q$ . A *computation* is a sequence of ‘‘CCS’’ terms of the form

$$p_1 \xrightarrow{\hat{\tau}} p_2 \xrightarrow{\hat{\tau}} \cdots \xrightarrow{\hat{\tau}} p_n;$$

it is called *maximal* if there is no ‘‘CCS’’ term  $q$  with  $p_n \xrightarrow{\hat{\tau}} q$ . Since ‘‘CCS’’ does not include recursion, any computation is finite here.

There are two forms of a process  $p$  passing a test  $t$ :

- (i)  $p$  may pass  $t$  if there exists a computation

$$p \parallel_{\mathcal{M}} t = p_1 \parallel_{\mathcal{M}} t_1 \xrightarrow{\hat{\tau}} \cdots \xrightarrow{\hat{\tau}} p_n \parallel_{\mathcal{M}} t_n$$

with  $t_n \xrightarrow{\omega}$ , or equivalently if there exists some  $n \geq 0$  with

$$\hat{\tau}^n \cdot \omega \in \text{trace}(p \parallel_{\mathcal{M}} t),$$

- (ii)  $p$  must pass  $t$  if whenever

$$p \parallel_{\mathcal{M}} t = p_1 \parallel_{\mathcal{M}} t_1 \xrightarrow{\hat{\tau}} \cdots \xrightarrow{\hat{\tau}} p_n \parallel_{\mathcal{M}} t_n$$

is a *maximal computation* then there exists some  $m$  with  $1 \leq m \leq n$  and  $t_m \xrightarrow{\omega}$ .

Thus a term  $t_n$  that can perform an  $\omega$ -action serves as a criterion for success. For examples of (i) and (ii) we refer to [DH84].

Then De Nicola and Hennessy [DH84] introduce three so-called *testing equivalences* on processes  $p, q$ :

- (i)  $p \approx_1 q$  if for every test  $t$ :  $p$  may pass  $t$  if and only if  $q$  may pass  $t$ .



- (ii)  $p \approx_2 q$  if for every test  $t$ :  $p$  must pass  $t$  if and only if  $q$  must pass  $t$ .
- (iii)  $p \approx_3 q$  if  $p \approx_1 q$  and  $p \approx_2 q$ .

It is now very interesting that for  $\hat{\tau}$ -free “CCS” the strong testing equivalence coincides with the failure equivalence  $\equiv_{\mathcal{F}}$ . This is an immediate consequence of Corollary 6.2.6 of [DH84] stated for the class of so-called strongly convergent CCS terms, which in particular includes all  $\hat{\tau}$ -free “CCS” terms. Thus at least for  $\hat{\tau}$ -free “CCS” terms we have a pleasing convergence of ideas:

*strong testing equivalence = failure equivalence = maximal trace respecting congruence.*

Conceptually, we find the notion of a maximal trace respecting congruence simpler than the definition of passing a test.

**7.4. Full abstraction.** The notion of *full abstraction* is due to Milner [Mi77] (see also [HP79], [Pl77]). It is a relationship between models (of an axiomatic system) and equivalence relations (on the terms of that system) whose definition is motivated by the following question:

*Under what circumstances can we replace a term  $x$  by a term  $y$  without noticing this change by a given equivalence  $\equiv$ ?*

Using the notion of a context introduced above, this question amounts to:

*Under what conditions on  $x$  and  $y$  do we have  $\mathcal{C}[x] \equiv \mathcal{C}[y]$  for every context  $\mathcal{C}[\xi]$ ?*

Full abstraction can be seen as looking for a sufficient and necessary condition that answers this question. Formally, we state the following definition.

**DEFINITION 7.4.1.** A model  $\mathcal{M}$  for  $\mathcal{T} \subseteq \text{Ter}(\Sigma)$  is called *fully abstract with respect to an equivalence relation  $\equiv$  on  $\mathcal{T}$*  if for all terms  $x, y \in \mathcal{T}$ :

$$\mathcal{M}[x] = \mathcal{M}[y] \text{ iff } \mathcal{C}[x] \equiv \mathcal{C}[y] \text{ holds for every context } \mathcal{C}[\xi] \in \text{Ter}(\Sigma)[\xi] \text{ with } \mathcal{C}[x], \mathcal{C}[y] \in \mathcal{T}.$$

Thus a fully abstract model  $\mathcal{M}$  optimally fits the equivalence  $\equiv$  in the sense that it just makes the identifications on terms that are forced by  $\equiv$ . Usually, it is quite difficult to discover fully abstract models (see [HP79], [Mi77], [Pl77]), but for the failure model  $\mathcal{F} = \mathcal{F}(+, \cdot, \parallel, \llbracket, \lrcorner, \partial_H, a_H, a, \delta)(a \in A)$  of § 5 and the trace equivalence  $\sim_{\text{tr}}$  of § 2 we can now state such a result.

**THEOREM 7.4.2.** *Consider  $\text{ACP}_r$  with one-to-one communication. Then for the set  $\mathcal{T}_C$  of all closed terms  $x$  over  $\text{ACP}_r$  with alphabet  $\alpha(x) \subseteq C$  the failure model  $\mathcal{F}$  is fully abstract with respect to the trace equivalence  $\sim_{\text{tr}}$ .*

*Proof.* By Definition 7.4.1, it suffices to show that for all  $x, y \in \mathcal{T}_C$ :

$$\mathcal{F}[x] = \mathcal{F}[y] \quad \text{iff } x \equiv_{\max} y$$

where  $\equiv_{\max}$  is the maximal trace respecting congruence. But this is immediate from Theorem 7.2.1.  $\square$

**COROLLARY 7.4.3.** *For the set of closed “CSP” terms the failure model  $\mathcal{F}_{\text{BHR}}$  of [BHR84] is fully abstract with respect to the trace equivalence  $\sim_{\text{tr}}$ .*

For “CCS” we cannot state the analogous result due to the  $\hat{\tau}$  mismatch discussed above.

## 8. Processes with recursion and abstraction: bisimulation versus failure equivalence.

**8.1. Preliminaries.** In the preceding sections we have been exclusively concerned with the failure semantics for finite processes without abstraction, i.e., not involving  $\tau$ -steps. In this section we will set aside that restriction and comment also on infinite

(recursive) processes with abstraction, in regard to bisimulation and failure equivalence. The crucial point is the way in which infinite sequences of  $\tau$ -steps in a process are treated.

In the failure semantics proposed in [BHR84], all processes having an infinite  $\tau$ -sequence from the root are set equal (to the process CHAOS). The notion of bisimulation is more discriminating. The advantage is that models obtained by bisimulation equivalence satisfy a useful abstraction principle: *Koomen's Fair Abstraction Rule* (KFAR), as introduced in [BK84b]. Roughly, this rule gives a way of simplifying processes by elimination of (some) infinite  $\tau$ -sequences. This elimination can be understood as *fairness* of (visible) actions over silent  $\tau$ -steps. A more precise description is given below. (Of course, setting all processes having an infinite  $\tau$ -sequence from the root equal to CHAOS also eliminates infinite  $\tau$ -sequences, but then all information is lost.)

Since KFAR is a very useful tool for system verification (e.g., in [BK84b] it was used to verify an alternating bit protocol), it is natural to ask whether  $\text{KFA}_{\kappa}$  is also compatible with the somewhat simpler failure semantics. More precisely, we can ask whether there exists a process model which for finite processes agrees with the failure semantics and for infinite processes satisfies KFAR. Interestingly, it turns out that such a model does not exist. To prove this result, we will formulate a set of assumptions embodying failure semantics and KFAR, and derive an inconsistency. Formally, the inconsistency arises from the following extension of the axiom system considered above:

$$\begin{aligned} & \text{ACP}_r + \text{R1}, 2 + \text{S} \\ & \quad + \text{Milner's } \tau\text{-laws} + \text{axioms for abstraction operators} \\ & \quad + \text{KFAR} \\ & \quad + \text{RSP (recursive specification principle)}. \end{aligned}$$

Here RSP is the assumption that guarded systems of recursion equations have a solution, which is moreover unique.

Now by virtue of our axiomatic approach we can pinpoint the origin of the inconsistency derived below with some accuracy. It turns out that the failure of KFAR in failure semantics holds already in ready semantics, and moreover that communication does not play a role in the inconsistency. That is, the inconsistency already appears in the subsystem

$$\text{BPA} + \text{T1} + \text{TI1} - 5 + \text{R1} + \text{KFAR} + \text{RSP}$$

which we will explain now. BPA, for *basic process algebra*, consists of the axioms A1-5 of  $\text{ACP}_r$ , which specify the properties of  $+$  and  $\cdot$ . T1 is the simplest of Milner's  $\tau$ -laws [Mi80] (see Table 6). In addition, Table 6 contains axioms TI1-TI5; these specify the abstraction operators  $\tau_I$ , where  $I \subseteq A$  is a set of *internal* actions as simple renaming operators (cf. [BK84c] and [BK86a], [BK86b]).

R1 is the axiom for the readiness semantics (see Tables 3 and 4):

$$a(bx + u) + a(by + v) = a(bx + by + u) + a(bx + by + v).$$

The *recursive specification principle* RSP states that guarded systems  $E$  of recursive equations have unique solutions (see [BK84b] or [BBK85]):

$$\frac{E(x_1, \dots, x_n), E(y_1, \dots, y_n), \quad E \text{ guarded}}{x_i = y_i}.$$

Informally, "guarded" means that every recursive occurrence of  $x_i$  in  $E$  is preceded

TABLE 6  
BPA+T1+TI1-5

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5
$x\tau = x$	T1
$\tau_i(\tau) = \tau$	TI1
$\tau_i(a) = a \text{ if } a \notin I$	TI2
$\tau_i(a) = \tau \text{ if } a \in I$	TI3
$\tau_i(x + y) = \tau_i(x) + \tau_i(y)$	TI4
$\tau_i(xy) = \tau_i(x) \cdot \tau_i(y)$	TI5

by an action different from  $\tau$ . For example, the system

$$\begin{aligned} x_1 &= ax_2 + bx_2, \\ x_2 &= c(x_1 + x_2) + d \end{aligned}$$

is guarded and thus has a unique solution.

We will now explain KFAR. For each  $n \geq 1$ , we have a version  $\text{KFAR}_n$ .  $\text{KFAR}_1$  is as follows:

$$\frac{x = ix + y \quad (i \in I)}{\tau_I(x) = \tau \cdot \tau_I(y)}$$

The premise of  $\text{KFAR}_1$  says that  $x$  has an infinite  $i$ -trace (see Fig. 19). Now  $\text{KFAR}_1$  expresses the fact that  $x$  makes *fair* choices along its infinite  $i$ -trace, i.e., performing  $x$  entails at most finitely many choices against  $y$ . We may note here the necessity of the abstraction operator  $\tau_i$  in  $\text{KFAR}_1$ : From  $x = \tau x + y$  it does *not* follow that  $x = \tau \cdot \tau_i(y)$ , since the equation  $x = \tau x + y$  has infinitely many solutions (see [BK84c] or [BK86a]).

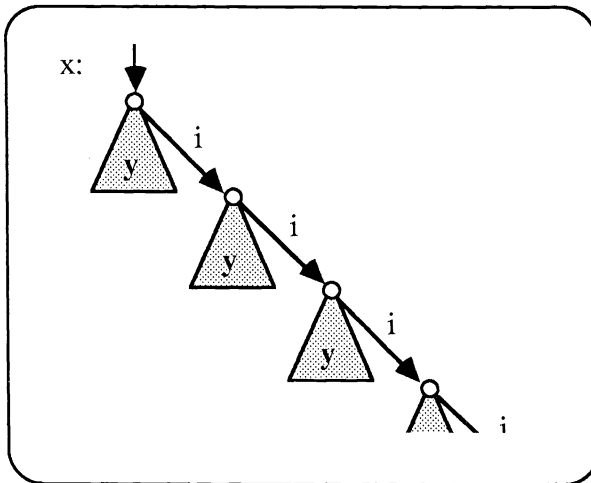


FIG. 19

The version of KFAR for  $n = 2$  is

$$\frac{x_1 = ix_2 + y_1, \quad x_2 = jx_1 + y_2 \quad (i, j \in I)}{\tau_I(x_1) = \tau \cdot \tau_I(y_1 + y_2)}.$$

In the general formulation of  $\text{KFAR}_n$  the premise displays an “ $I$ -cycle” of length  $n$ . For a precise formulation we refer to [BK84b] or [BBK85].

Note that except for KFAR all assumptions in  $\text{BPA}_\tau + \text{TI1-TI5} + \text{R1} + \text{RSP}$  are valid for failure semantics. To see that the  $\tau$ -laws TI1-TI3 (of which only the first one is needed for the derivation of the contradiction below) are valid for failure semantics, we refer to [Br83], which gives axioms describing failure semantics for finite processes involving  $\tau$ -steps; these axioms imply the  $\tau$ -laws.

**8.2. The inconsistency of failure semantics with KFAR.** We will now derive the announced contradiction. It is important to notice that this contradiction is entirely insensitive to how failure semantics works with processes that contain  $\tau$ -steps.

Consider the following systems of guarded recursion equations:

$$E_1 \begin{cases} x = ax_1 + ax_2, \\ x_1 = c + bx_2, \\ x_2 = d + bx_1, \end{cases}$$

and

$$E_2 \begin{cases} y = ay_1 + ay_2, \\ y_1 = c + by_2, \\ y_2 = d + by_1. \end{cases}$$

The systems  $E_1, E_2$  have solutions  $x, y$  which can be depicted as in Fig. 20.

CLAIM:  $x$  and  $y$  are failure equivalent.

Intuitively this may be clear since (as demonstrated in § 3.1) axiom R1 amounts to placing “crosses”; from the graphs for  $x, y$  we can thus obtain equivalent graphs as in Fig. 21. These two graphs are in fact identical.

*Proof of the Claim (Formally).* Consider the system  $E_3$  of guarded recursion equations:

$$E_3 \begin{cases} z = az_1 + az_2, \\ z_1 = c + bz_1 + bz_2, \\ z_2 = d + bz_1 + bz_2. \end{cases}$$

(This system corresponds with the graph in Fig. 21.) Now

$$\begin{aligned} x &= ax_1 + ax_2 = a(c + bx_2) + a(d + bx_1) \quad (\text{by R1}) \\ &= a(c + bx_2 + bx_1) + a(d + bx_1 + bx_2) = az'_1 + az'_2 \end{aligned}$$

where

$$z'_1 = c + bx_2 + bx_1 \quad \text{and} \quad z'_2 = d + bx_1 + bx_2.$$

Further,

$$\begin{aligned} z'_1 &= c + bx_2 + bx_1 = c + b(d + bx_1) + b(c + bx_2) \quad (\text{by R1}) \\ &= c + b(c + bx_2 + bx_1) + b(d + bx_1 + bx_2) \\ &= c + bz'_1 + bz'_2 \end{aligned}$$

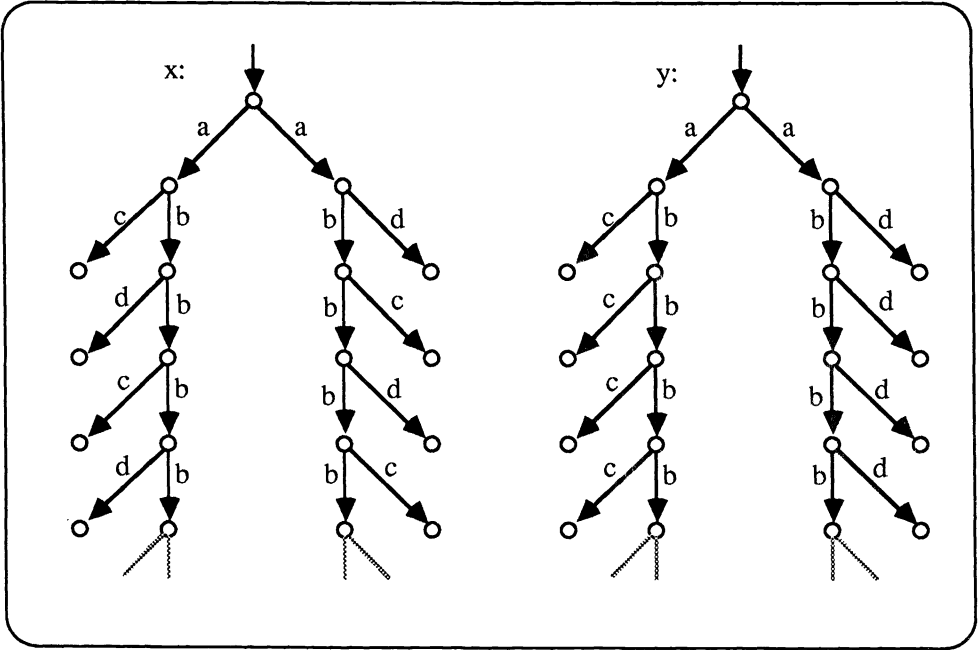


FIG. 20

and likewise

$$z'_2 = d + bz'_1 + bz'_2.$$

So  $(x, z'_1, z'_2)$  satisfies  $E_3$ . A similar computation shows that  $(y, z''_1, z''_2)$ , where

$$z''_1 = c + by_1 + by_2, \quad z''_2 = d + by_1 + by_2$$

satisfies  $E_3$ . Hence by RSP,

$$(x, z'_1, z'_2) = (y, z''_1, z''_2) = (z, z_1, z_2),$$

in particular  $x = y$ . This proves the claim.

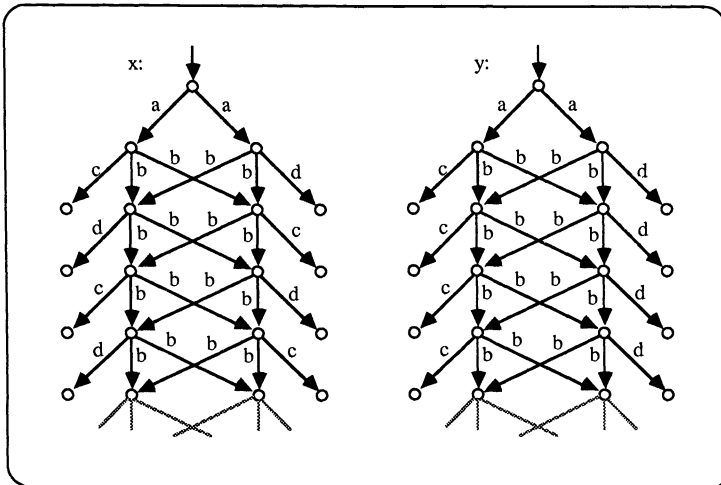


FIG. 21

In order to derive the inconsistency we will abstract from  $b$ , by means of  $\tau_{\{b\}}$ , in  $x$  and  $y$ . This yields corresponding process graphs as in Fig. 22. Next we apply KFAR on  $\tau_{\{b\}}(x)$  and  $\tau_{\{b\}}(y)$  and obtain  $a(c+d)$  and  $ac+ad$ , respectively. This can be seen graphically: KFAR shrinks the infinite  $\tau$ -traces to a point, obtaining the graphs as in Fig. 23.

Formally:

$$(*) \quad \tau_{\{b\}}(x) = \tau_{\{b\}}(ax_1 + ax_2) = a \cdot \tau_{\{b\}}(x_1) + a \cdot \tau_{\{b\}}(x_2).$$

Further,

$$x_1 = bx_2 + c, \quad x_2 = bx_1 + d$$

yields by KFAR<sub>2</sub>:

$$\tau_{\{b\}}(x_1) = \tau \cdot \tau_{\{b\}}(c+d) = \tau(c+d),$$

$$\tau_{\{b\}}(x_2) = \tau \cdot \tau_{\{b\}}(c+d) = \tau(c+d).$$

Hence from (\*)

$$\begin{aligned} \tau_{\{b\}}(x) &= a\tau(c+d) + a\tau(c+d) \quad (\text{by T1 in Table 6}), \\ &= a(c+d) + a(c+d) = a(c+d). \end{aligned}$$

Next consider  $y$ :

$$(**) \quad \tau_{\{b\}}(y) = a \cdot \tau_{\{b\}}(y_1) + a \cdot \tau_{\{b\}}(y_2).$$

Now  $y_1 = by_1 + c$  yields by KFAR<sub>1</sub>:  $\tau_{\{b\}}(y_1) = \tau c$ ; similarly  $\tau_{\{b\}}(y_2) = \tau d$ . Hence from (\*\*)

$$\tau_{\{b\}}(y) = a\tau c + a\tau d = ac + ad.$$

So, since  $x = y$ , we have proved  $a(c+d) = ac + ad$ . But  $a(c+d)$  and  $ac + ad$  are not failure equivalent.

**8.3. Further results.** The above inconsistency proves that the advantages of Koo-  
men's fair abstraction rule, KFAR, cannot be combined with the simplicity of failure  
semantics. We investigated this dichotomy further and were pleased to find a weaker

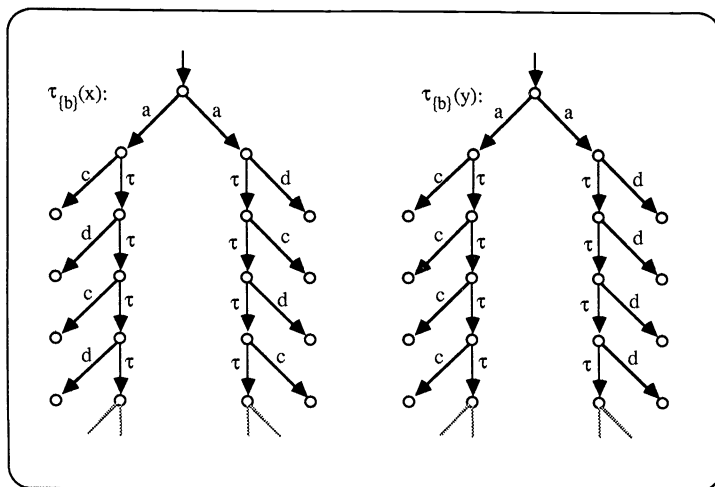


FIG. 22

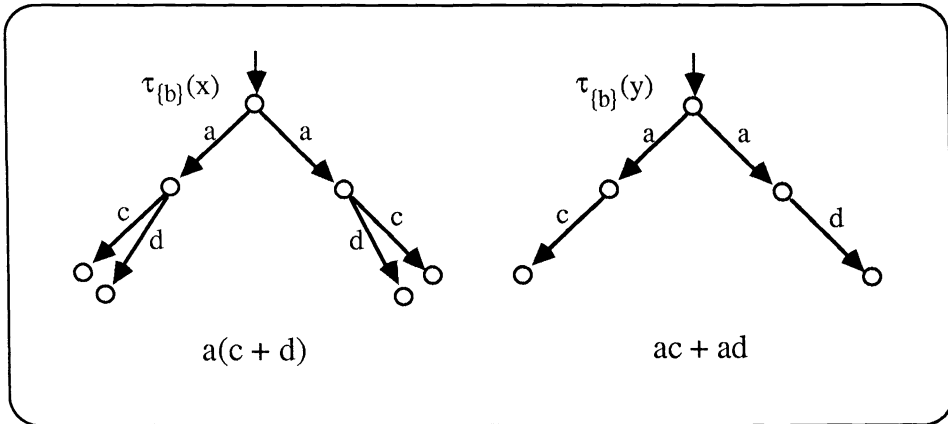


FIG. 23

fair abstraction rule, called  $\text{KFAR}^-$ , which is consistent with (finite) failure semantics, and which is still useful for many process verifications. More precisely, the new rule is consistent with a version of Brookes, Hoare, and Roscoe's failure semantics [BHR84] without catastrophic divergence, i.e., that does not identify processes having an infinite  $\tau$ -sequence from the root with the process CHAOS. The details and applications of the new rule  $\text{KFAR}^-$  can be found in [BKO86].

**Acknowledgment.** We thank R. van Glabbeek and one of the referees for pointing out some inconsistencies in a previous version of this paper and for many detailed suggestions and corrections.

## REFERENCES

- [BBK85] J. C. M. BAETEN, J. A. BERGSTRA, AND J. W. KLOP, *On the consistency of Koomen's Fair Abstraction Rule*, Report CS-R8511, Centre for Mathematics and Computer Science, Amsterdam, the Netherlands, 1985; Theoret. Comput. Sci., 51 (1987), pp. 129-176.
- [BK83] J. A. BERGSTRA AND J. W. KLOP, *An abstraction mechanism for process algebras*, Report IW 231/83, Centre for Mathematics and Computer Science, Amsterdam, the Netherlands, 1983.
- [BK85] ———, *Algebra of communicating processes with abstraction*, Theoret. Comput. Sci., 37 (1985), pp. 77-121.
- [BK84a] ———, *Process algebra for synchronous communication*, Inform. and Control, 60 (1984), pp. 109-137.
- [BK84b] ———, *Verification of an alternating bit protocol by means of process algebra*, Report CS-R8404, Centre for Mathematics and Computer Science, Amsterdam, the Netherlands, 1984; also in Math. Methods of Spec. and Synthesis of Software Systems 1985, W. Bibel and K. P. Jantke, eds., Math. Research 31, Akademie-Verlag, Berlin, 1986, pp. 9-23.
- [BK84c] ———, *A complete inference system for regular processes with silent moves*, Report CS-R8420, Centre for Mathematics and Computer Science, Amsterdam, the Netherlands, 1984; Proc. Logic Colloquium in Hull, J. Truss and F. Drake, eds., North-Holland, Amsterdam, 1986, pp. 21-81.
- [BK86a] ———, *Algebra of Communicating Processes*, in CWI Monographs I, Proc. CWI Symposium on Mathematics and Computer Science, J. W. de Bakker, M. Hazewinkel, and J. K. Lenstra, eds., North-Holland, Amsterdam, 1986, pp. 89-138.
- [BK86b] ———, *Process Algebra: Specification and Verification in Bisimulation Semantics*, in CWI Monograph 4, Proc. CWI Symposium Mathematics and Computer Science II, M. Hazewinkel, J. K. Lenstra, and L. G. L. T. Meertens, eds., North-Holland, Amsterdam, 1986, pp. 61-94.

- [BKO86] J. A. BERGSTRA, J. W. KLOP, AND E.-R. OLDEROG, *Failures without chaos: a new process semantics for fair abstraction*, in Proc. IFIP Working Conference on Formal Description of Programming Concepts, Gl. Avernoes 1986, M. Wirsing, ed., North-Holland, Amsterdam, 1987, pp. 77–101.
- [Br83] S. D. BROOKES, *On the relationship of CCS and CSP*, in Proc. 10th Internat. Colloquium on Automat. Lang. and Programming, Barcelona, J. Díaz, ed., Lecture Notes in Computer Science, 154, Springer-Verlag, New York, Berlin, 1983, pp. 83–96.
- [BHR84] S. BROOKES, C. HOARE, AND W. ROSCOE, *A theory of communicating sequential processes*, J. Assoc. Comput. Mach., 31 (1984), pp. 560–599.
- [DH84] R. DE NICOLA AND M. C. B. HENNESSY, *Testing equivalences for processes*, Theoret. Comput. Sci., 34 (1984), pp. 83–133.
- [He83] M. HENNESSY, *Synchronous and asynchronous experiments on processes*, Inform. and Control, 59 (1983), pp. 36–83.
- [HP79] M. HENNESSY AND G. D. PLOTKIN, *Full abstraction for a simple programming language*, in Proc. 8th Mathematical Foundations of Computer Science, J. Becvar, ed., Lecture Notes in Computer Science 74, Springer-Verlag, Berlin, New York, 1979, pp. 108–120.
- [Ho78] C. A. R. HOARE, *Communicating sequential processes*, Comm. ACM 21 (1978), pp. 666–677.
- [Ho80] ———, *A model for communicating sequential processes*, in On the Construction of Programs, R. M. McKeag and A. M. McNaughton, eds., Cambridge University Press, London, New York, 1980, pp. 229–243.
- [Mi77] R. MILNER, *Fully abstract models of typed  $\lambda$ -calculi*, Theoret. Comput. Sci., 4 (1977), pp. 1–22.
- [Mi80] R. MILNER, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science 92, Springer-Verlag, New York, Berlin, 1980.
- [Mi83] R. MILNER, *Calculi for synchrony and asynchrony*, Theoret. Comput. Sci., 25 (1983), pp. 267–310.
- [OH83] E. R. OLDEROG AND C. A. R. HOARE, *Specification-oriented semantics for communicating processes*, in Proc. 10th Internat. Colloquium on Automat. Lang. and Programming, Barcelona, J. Díaz, ed., Lecture Notes in Computer Science 154, Springer-Verlag, New York, Berlin, 1983, pp. 561–572; expanded version, Acta Informatica, 23 (1986), pp. 9–66.
- [Pa83] D. M. R. PARK, *Concurrency and automata on infinite sequences*, in Proc. 5th GI (Gesellschaft für Informatik) Conference, Lecture Notes in Computer Science 104, Springer-Verlag, New York, Berlin, 1981.
- [Pl77] G. D. PLOTKIN, *LCF considered as a programming language*, Theoret. Comput. Sci., 5 (1977), pp. 223–255.
- [RB81] W. C. ROUNDS, AND S. D. BROOKES, *Possible futures, acceptances, refusals, and communicating processes*, in Proc. 22nd IEEE Symposium on Foundations of Computer Science, Nashville, TN (IEEE Computer Society Press, 1981), pp. 140–149.
- [VG188] R. J. VAN GLABBEEK, *Personal communication*, 1988.
- [Wi83] G. WINSKEL, *Synchronisation trees*, in Proc. 10th ICALP, Barcelona, J. Díaz, ed., Lecture Notes in Computer Science 154, Springer-Verlag, New York, Berlin, 1983, pp. 695–711; expanded version in Theoret. Comput. Sci., 34 (1984), pp. 33–82.



## THE AVERAGE COMPLEXITY OF DETERMINISTIC AND RANDOMIZED PARALLEL COMPARISON-SORTING ALGORITHMS\*

N. ALON†‡ AND Y. AZAR†

**Abstract.** In practice, the average time of (deterministic or randomized) sorting algorithms seems to be more relevant than the worst-case time of deterministic algorithms. Still, the many known complexity bounds for parallel comparison sorting include no nontrivial lower bounds for the average time required to sort by comparisons  $n$  elements with  $p$  processors (via deterministic or randomized algorithms). We show that for  $p \geq n$  this time is  $\Theta(\log n / \log(1 + p/n))$  (it is easy to show that for  $p \leq n$  the time is  $\Theta(n \log n / p) = \Theta(\log n / (p/n))$ ). Therefore even the average-case behaviour of randomized algorithms is not more efficient than the worst-case behaviour of deterministic ones.

**Key words.** parallel sorting, comparison algorithms, randomized sorting

**AMS(MOS) subject classification.** 68E05

**1. Introduction.** Sorting is one of the central problems in computer science. For extensive lists of publications dealing with serial and parallel sorting algorithms see, e.g., [Ak85], [BHe85], [Kn73], and [Th83].

Most of the fastest serial and parallel sorting algorithms are based on binary comparisons. In these algorithms the number of comparisons is typically the primary measure of time complexity. Any lower bound on the number of comparisons required for a problem clearly implies a time lower bound for such algorithms.

It is well known that  $\Theta(n \log n)$  binary comparisons are both necessary and sufficient for sorting  $n$  elements in the serial comparison tree model. The situation is somewhat more complicated for parallel algorithms. The common parallel comparison model here is the one introduced by Valiant [Va75] (see also [BH085]), where only comparisons are counted.

In measuring time complexity within this model, we do not count steps in which communication among the processors, movement of data, and memory addressing are performed. We also avoid counting steps in which consequences are deduced from comparisons that were performed.

Note that any lower bound in this model implies the same bound for all algorithms, based on comparisons, in any parallel random access machine (PRAM), including PRAMs that allow simultaneous access to the same common memory location for read and write purposes.

In a serial decision-tree model, we wish to minimize the number of comparisons. The goal of an algorithm in a parallel comparison model is to minimize the number of comparison rounds as well as the total number of comparisons performed.

Let  $k$  stand for the number of comparison rounds (time) of an algorithm in the parallel comparison model. Let  $c(k, n)$  denote the *minimum total* number of comparisons required to sort any  $n$  elements in  $k$  rounds (over all possible algorithms).

Upper and lower bounds for  $c(k, n)$  appear in [AA87], [AAV86], [AKS83a], [AKS83b], [AV87], [BT83], [BHe85], [Bo86], [BH085], [HH80], [HH81], [Pi87]. The

---

\* Received by the editors June 15, 1987; accepted for publication (in revised form) January 14, 1988.

† Department of Mathematics, Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv, Israel.

‡ The research of this author was supported in part by an Allon Fellowship, a grant from the Bat Sheva de Rothschild Foundation, and by the Fund for Basic Research administered by the Israel Academy of Sciences.

best-known bounds for fixed  $k$  are (see [AA87], [AAV86], [BHe85])

$$\Omega(n^{1+1/k}(\log n)^{1/k}) \leq c(k, n) \leq O(n^{1+1/k} \log n)$$

and for general  $k \leq \log n$  (see [AV87], [AAV86])

$$(1.1) \quad \Omega(kn^{1+1/k}) \leq c(k, n) \leq kn^{1+b/k}$$

where  $b > 0$  is a constant. These bounds imply that the time required for sorting  $n$  elements, if  $p$  comparisons are performed in each time unit, is  $\Theta(\log n/(p/n))$  for  $p \leq n$  and  $\Theta(\log n/\log(1+p/n))$  for  $p \geq n$ .

These results determine up to a constant factor the (worst-case) time complexity of sorting in the parallel comparison tree model. However, the problem of estimating the *average* (over all orders) running time of the best sorting algorithm, as well as that of determining the time complexity of the best randomized sorting algorithm, is far from being settled. In fact, besides the relatively easy  $\Omega(n \log n)$  lower bound for randomized (or average) serial sorting (see, e.g., [AHU74]), there are no known lower bounds for the worst-case or average-time complexity of randomized sorting algorithms at all. Such bounds appear to be important, since in practical situations we are naturally interested in the average running time, and not necessarily in the worst-case behaviour. Similarly, fast randomized parallel sorting algorithms could be extremely helpful in practice.

Proving lower bounds for the average time of (deterministic or randomized) comparison algorithms appears to be much more complicated than obtaining lower bounds for the worst-case time of deterministic ones. In fact, there are several known results that show that for various comparison algorithms the average time, as well as the worst-case time of randomized algorithms, differs asymptotically from the worst-case time of their deterministic counterparts. One such result is due to Reischuk [Re81], who gave a randomized comparison parallel algorithm for selection, whose expected running time is bounded by a constant, using  $n$  processors. Together with the  $\Omega(\log \log n)$  lower bound of [Va75] for finding the maximum among  $n$  elements using  $n$  processors, we conclude that there exists a randomized algorithm for selection that performs better than any of its deterministic counterparts. The results of [BHe85] on approximate sorting in one round, those of [Rei85] on integer sorting and those of [AAV86] on sorting in a fixed number of rounds supply several other examples of parallel randomized comparison algorithms that perform better than the corresponding best-known deterministic ones. In view of these examples we might expect that randomized parallel sorting algorithms could work asymptotically faster than deterministic ones. Our main result in this paper is that this is not the case. In fact we prove the following theorem.

**THEOREM 1.1.** *The average time required for sorting  $n$  elements in the best randomized algorithm with  $p$  processors, (i.e., the best algorithm that performs  $p$  comparisons in each time unit), is  $\Theta(\log n/\log(1+p/n))$  for  $p \geq n$  (and is, easily,  $\Theta(\log n/(p/n))$  for  $p \leq n$ ).*

This matches, up to a constant factor, the worst-case running time for the deterministic case for all values of  $p$ .

To prove the lower bound we prove the following proposition.

**PROPOSITION 1.2.** *The average number of comparisons in the best deterministic algorithm that sorts  $n$  elements in  $k$  rounds is*

$$\Omega(kn^{1+1/k}) \quad \text{for all } k \leq \log n.$$

Note that for  $k = \Theta(\log n)$  this coincides with the known bound for the serial case.

A parallel algorithm is said to achieve average *optimal speedup* if its average running time is proportional to  $\text{Seq}(n)/p$ , where  $\text{Seq}(n)$  is the lower bound on the average serial running time,  $n$  is the size of the problem being considered, and  $p$  is the number of processors used.

An immediate consequence of Theorem 1.1 is that if the number  $p$  of processors is larger than  $n$  by an order of magnitude then it is impossible to design an average optimal speedup randomized comparison sorting algorithm. Note that, for  $p = O(n)$ , there is an optimal speedup (deterministic and therefore randomized or average deterministic) algorithm, given by the [AKS83a], [AKS83b] sorting network. These results enable us to identify asymptotically the parallelism average break point of sorting, which is the minimum average time that can be achieved by an average optimal speedup algorithm. Specifically,  $\Theta(\log n)$  is the average break point for sorting  $n$  elements, which is the same break point as that of deterministic algorithms (see [AAV86], [AV87]).

Note that for finding the maximum the average break point is better than the worst-case break point. Reference [Va75] proved that  $\Theta(\log \log n)$  is the break point for finding the maximum among  $n$  elements but [Re81] proved that  $\Theta(1)$  is the average break point of that problem.

The proof of Theorem 1.1 differs considerably and is far more complicated than that of the corresponding result for the worst-case time of deterministic algorithms (which, obviously, follows from it). The main difficulty lies in the proof of Proposition 1.2. As we are dealing with the average-case behaviour, we cannot use the traditional adversary way for choosing the unknown order; we need a new method. Since a direct proof seems elusive, we must prove a stronger result, which implies, e.g., that our bound holds for algorithms that allow, in addition to usual comparisons, questions of the form “is rank  $(x) \leq i$ ?” For the exact statement of the stronger result see § 2. We are unable to prove Proposition 1.2 without proving this stronger assertion.

The derivation of Theorem 1.1 from Proposition 1.2 is much easier than the proof of Proposition 1.2; however, it is not straightforward. It combines certain probabilistic arguments with a well-known observation of Yao [Ya77] and the upper bound in inequality (1.1). This is described in § 3. The final section, 4, contains concluding remarks, together with an application of our method to selection problems.

## 2. The average number of comparisons in deterministic algorithms.

**2.1. The parallel computation model.** Let  $N$  be a set of  $n$  elements taken from a totally ordered domain. The *parallel comparison model of computation* equivalent to the parallel computation tree model of [BHo85] allows algorithms that work as follows. The algorithm consists of timesteps called *rounds*. In each round binary comparisons are performed simultaneously. The input for each comparison is two elements of  $N$ . The output of each comparison is one of the following two:  $<$  or  $>$ . Each item may take part in several comparisons during the same round.

Our discussion uses the following correspondence between each round and a graph. The elements are the vertices. Each comparison to be performed is an undirected edge that connects its input elements. Each computation results in orienting this edge from the largest element to the smallest. Thus in each round we get an acyclic orientation of the corresponding graph, and the transitive closure of the union of the  $r$  oriented graphs obtained until round  $r$  represents the set of all pairs of elements whose relative order is known at the end of round  $r$ .

Suppose we performed  $r$  rounds where  $r > 0$  is some integer. The comparisons performed in round  $r + 1$  are chosen, of course, according to the results in all previous rounds.

Recall that  $c(k, n)$  denotes the minimum *total* worst-case number of comparisons required to sort  $n$  elements in  $k$  rounds (over all possible algorithms).

Let  $r(k, n)$  denote the *average* total number of comparisons, over all orders, required to sort  $n$  elements in  $k$  rounds, in the best algorithm (that necessarily stops after  $k$  rounds). Clearly  $r(k, n) \leq c(k, n)$ . Our objective is to prove Proposition 1.2, which supplies a lower bound for  $r(k, n)$ , where  $k$  is possibly a function of  $n$ .

**2.2. Legal situations.** We prove a stronger result that implies Proposition 1.2. We consider the following situation, called a *legal situation*: We have  $s + 1$  disjoint sets of elements, denoted  $Z$  and  $Y_1, \dots, Y_s$ , with a set  $E$  of edges (comparisons) between them.

The set  $Z$ ,  $|Z| = m$ ,  $0 \leq m \leq n$ , is a set of  $m$  elements such that the rank of each  $z \in Z$  in the total  $n$ -order is known.  $Y_1, \dots, Y_s$ ,  $|Y_i| = y_i > 0$ ,  $1 \leq i \leq s$  are  $s \geq 0$  sets of elements such that, for each  $i$ , the set of  $y_i$  ranks of the  $i$ th set in the total  $n$ -order is known, but all the  $y_i!$  orders of the elements of the sets are equally likely. The existing edges (comparisons) we have are only for this round and are only between pairs of elements of the same  $Y_i$ , for some  $i$ , or between an element of  $Y_i$  and an element of  $Z$ . The answers to these comparisons are known, but depend, of course, on the actual orders inside the sets  $Y_i$ . Let  $e_i$  be the number of edges in  $Y_i$ . Let  $\bar{e}_i$  be the number of edges between an element of  $Y_i$  and an element outside  $Y_i$ , whose relative order does not follow from the known information about the sets of ranks. Denote  $f_i = e_i + \frac{1}{2}\bar{e}_i$  and call it the “ $f$ -number” of the set  $Y_i$  in this situation.

The following facts are worth noting:

(1) The number of edges  $|E|$  satisfies

$$|E| \geq \sum_{i=1}^s \left( e_i + \frac{1}{2} \bar{e}_i \right) = \sum_{i=1}^s f_i.$$

(2) The number of elements  $n$  satisfies

$$n = m + \sum_{i=1}^s y_i.$$

(3) The (known) set of ranks of each  $Y_i$  is not necessarily a block of  $y_i$  consecutive ranks, and there are no edges between an element of  $Y_i$  and an element of  $Y_j$  for  $i \neq j$ . Before the next round of comparisons, all the information about the relative order of an element of  $Y_i$  and an element of  $Y_j$  is a consequence of either the known sets of ranks of  $Y_i$  and  $Y_j$  or the results of comparisons to elements of  $Z$  and transitivity.

(4) If for some set  $Y_i$ ,  $f_i = 0$  then all the  $y_i!$  orders in this set are equally likely for each possibility of the results of the comparisons of the present round (and independently of these results).

**2.3. The lower bound.** Let  $A$  denote the above legal situation. Denote by  $F(k, A)$  the average, over all the  $\prod_{i=1}^s (y_i!)$  possible orders, of the “ $f$ -number” of comparisons that are needed to sort the  $n$  elements in  $k$  more rounds, starting from situation  $A$ . Here the “ $f$ -number” means that a comparison between an element of  $Y_i$  and an element of  $Y_j$  ( $i$  can be equal to  $j$ ) is counted as one comparison and a comparison between an element of  $Y_i$  and an element of  $Z$  is counted as  $\frac{1}{2}$  a comparison.

Define

$$g_k(y, f) = \begin{cases} 1, & f = 0, \\ \left(\frac{c}{4}\right)^{1/k}, & 0 < \frac{f}{y} \leq \frac{1}{4}, \\ \left(\frac{cf}{y}\right)^{1/k}, & \frac{f}{y} \geq \frac{1}{4}, \end{cases}$$

where  $c$  is some positive constant to be chosen later. Define also

$$\varphi_k(y, f) = k \left[ \frac{y^{1+1/k}}{cg_k(y, f)} - y \right].$$

Note that  $\varphi_k$  is a monotone nonincreasing function of  $f$ . The key inequality is the following.

**THEOREM 2.1.** *In the above notation*

$$(2.1) \quad F(k, A) \geq \sum_{i=1}^s \varphi_k(y_i, f_i)$$

for every  $k \geq 1$ , and every legal situation  $A$ .

The (rather lengthy) proof of Theorem 2.1 is given below. It is based on moving from one legal situation to another by finding properties that hold for all orders (and not just for a special one chosen by an adversary). Some inequalities for convex functions and some statistical behaviour of random orders are used as well, together with a reduction of nonlegal situations that are created during the algorithm to legal ones.

Proposition 1.2 for  $k \leq \log n / (\log 2c)$  is the special case of Theorem 2.1 in which  $A$  is the legal situation with  $Z = \phi$ ,  $s = 1$ ,  $Y_1$  is the set of all elements, and  $f_1 = 0$ . For  $k = \Theta(\log n)$ , Proposition 1.2 follows immediately from the serial bound. Another special case of Theorem 2.1 corresponds to algorithms that allow queries of the form “is rank ( $x$ )  $\leq i$ ” besides usual comparisons. Indeed, suppose we have  $2n - 1$  elements. Let  $A$  be the legal situation in which  $|Z| = n - 1$ , the set of ranks of  $Z$  is  $\{2, 4, \dots, 2n - 2\}$ ,  $s = 1$ , the set of ranks of  $Y_1$  is  $\{1, 3, \dots, 2n - 1\}$  and  $f_1 = 0$ . Applying Theorem 2.1 to this situation we conclude that  $\Omega(kn^{1+1/k})$  comparisons are needed to sort  $Y_1$  in  $k$  ( $\leq \log n$ ) rounds, where here, clearly, a comparison to an element of  $Z$  corresponds to a query of the form “is rank( $x$ )  $\leq i$ ?”

*Proof of Theorem 2.1.* We prove the theorem for every fixed  $n \geq 2$  by induction on  $k$  and on the parameters of  $A$  with  $c = 256e$ . (We make no effort here to obtain the best possible  $c$ .) The base case is left to the end. Our induction hypothesis is that the assertion of Theorem 2.1 holds for any  $k'$  and any legal situation  $A'$  with sets  $Z'$ ,  $|Z'| = m'$  and  $Y'_1, \dots, Y'_{s'}$ ,  $|Y'_i| = y'_i > 0$ , where

$$m' + \sum_{i=1}^{s'} y'_i = n,$$

provided at least one of the following three cases holds:

- (a)  $k' < k$ .
- (b)  $k' = k$  and  $\sum_{i=1}^{s'} y'_i < \sum_{i=1}^s y_i$ .
- (c)  $k' = k$ ,  $\sum_{i=1}^{s'} y'_i \leq \sum_{i=1}^s y_i$  and  $s < s'$ . (Note that always  $s, s' \leq n$ .)

We have to prove the theorem for  $k$  and  $A$ . We consider two possible cases.

*Case 1.* There is a set  $Y_i$ , with  $f_i > 0$ .

*Case 2.* For all  $i$ ,  $1 \leq i \leq s$ ,  $f_i = 0$ , and  $k > 1$ . (The case  $k = 1$  and  $f_i = 0$  for all  $i$  will be the base case.)

In Case 1 we can assume, without loss of generality, that  $f_s > 0$ . Denote  $Y = Y_s$ ,  $|Y| = y = y_s$ ,  $f = f_s$ .

*Subcase 1a.*  $f \geq (1/c^{k+1})y^2$ . In this case  $\varphi_k(y, f) \leq k(y^{1+1/k}/c(cf/y)^{1/k} - y) \leq 0$ . For each order of  $Y$ , let the algorithm know this order for free; i.e., let  $A'$  be the legal situation obtained from  $A$  by replacing  $Z$  by  $Z \cup Y$ . Since  $\sum y'_i < \sum y_i$  and  $k' = k$  it follows from the induction hypothesis (case (b)) that

$$F(k', A') \geq \sum_{i=1}^{s'} \varphi_k(y'_i, f'_i) = \sum_{i=1}^{s-1} \varphi_k(y_i, f_i) \geq \sum_{i=1}^s \varphi_k(y_i, f_i).$$

Averaging over all orders of  $Y$  we conclude that  $F(k, A) \geq \sum_{i=1}^s \varphi_k(y_i, f_i)$ , as needed. (Note that each comparison that is counted as one comparison in  $F(k', A')$  is also counted as 1 in  $F(k, A)$  and each comparison that is counted as  $\frac{1}{2}$  in  $F(k', A')$  is counted either as  $\frac{1}{2}$  or as 1 in  $F(k, A)$ .)

*Subcase 1b.*  $0 < f \leq y/4$ . Since  $2f = 2e_s + \bar{e}_s$  there are at most  $2f$  elements in  $Y$ , which are compared to some other elements (in  $Y$  or in  $Z$ ). Let  $Y'_s \subseteq Y$  be a set of  $2f \leq y/2$  elements containing all those that are compared to members of  $Y \cup Z$ . Also define  $Y'_{s+1} = Y \setminus Y'_s$ . We let the algorithm know for free the set of ranks of  $Y'_s$  and  $Y'_{s+1}$ . This corresponds to the legal situation  $A'$  obtained from  $A$  by splitting  $Y_s$  into the two sets  $Y'_s$  and  $Y'_{s+1}$ . For  $A'$ ,  $s' = s + 1$ ,  $f'_s \leq f_s$ ,  $f'_{s+1} = 0$ . As  $s' > s$  we can apply the induction hypothesis (case (c)) to  $A'$  and conclude that

$$F(k, A') \geq \sum_{i=1}^{s+1} \varphi_k(y'_i, f'_i) \geq \sum_{i=1}^s \varphi_k(y_i, f_i) - \varphi_k(y, f) + \varphi_k(2f, f) + \varphi_k(y - 2f, 0).$$

In Appendix 1 below we show that for  $0 < f \leq y/4$ ,  $\varphi_k(2f, f) + \varphi_k(y - 2f, 0) \geq \varphi_k(y, f)$ . Thus, for each  $A'$  that arises as above  $F(k, A') \geq \sum_{i=1}^s \varphi_k(y_i, f_i)$ . Averaging over all possible assignments of the two sets of ranks to the elements in  $Y'_s$  and in  $Y'_{s+1}$ , we conclude that  $F(k, A)$ , which is at least this average, is at least  $\sum_{i=1}^s \varphi_k(y_i, f_i)$ , as needed.

*Subcase 1c.* (The main case.)  $y/4 < f < y^2/c^{k+1}$ . Define  $t = \lceil 4f/y \rceil (> 1)$ ,  $\beta = \lfloor y/t \rfloor$ . Note that  $2 \leq t \leq 8f/y$ , and  $y/t \geq y^2/8f > c^{k+1}/8$ , and hence  $\beta \geq 1$  and

$$(2.2) \quad \beta = \left\lfloor \frac{y}{t} \right\rfloor \geq \frac{y}{t} \left( 1 - \frac{8}{c^{k+1}} \right).$$

Randomly partition  $Y$  into  $t + 1$  blocks  $B_1, \dots, B_t, C$  where  $|B_i| = \beta$ , and  $|C| = y - t\beta$  (possibly  $|C| = 0$ ). For each such partition choose a random permutation  $\pi$  of  $[1, 2, \dots, t]$ . Assume that each element of  $C$  is greater than each element of  $Y \setminus C$ , and that each element of  $B_i$  is greater than each element of  $B_j$  if and only if  $\pi(i) > \pi(j)$ . These choices, together with the assumption that all the orders inside each block are equally likely, give each of the possible  $y!$  orders of  $y$  with equal probability. For each choice of  $B_i, C$ , and  $\pi$ , there are  $t$  pairwise disjoint sets  $Z_1, \dots, Z_t$  whose ranges of ranks lie inside those of  $B_1, \dots, B_t$ . Let  $g'_i$  be the number of comparisons between two elements of  $B_i$ , and let  $g''_i$  be the number of comparisons between an element of  $B_i$  and an element of  $Z_i$ . Also define  $g_i = g'_i + \frac{1}{2}g''_i$ . Note that  $g'_i, g''_i$ , and  $g_i$  are random variables (whose values depend on the choice of the partition and of  $\pi$ ), that  $2g_i$  is an integer, and that  $g_i$  is the  $f$ -number of  $B_i$  (since the result of each comparison between members of  $B_i$  and members of  $N \setminus (B_i \cup Z_i)$  follows from the assumptions on the sets of ranks of each  $B_i$ ). If  $2g_i < \beta$ , let  $G_i$  be a subset of  $B_i$ ,  $|G_i| = 2g_i$  that contains every member of  $B_i$ , which is compared to a member of  $B_i \cup Z_i$ . (Clearly there are at most  $2g_i$  such elements.) If  $2g_i \geq \beta$ , define  $G_i = B_i$ . Also define  $N_i = B_i \setminus G_i$ . We let our algorithm know for free the sets of ranks of each  $G_i$  and each  $N_i$ . By the definition of  $G_i$  there are no comparisons between  $G_i$  and  $N_i$ . Hence we now have, for each choice of  $B_i, C$ , and  $\pi$ , a new legal situation  $A'$ , which is obtained from  $A$

by replacing  $Z$  by  $Z \cup C$  and by replacing  $Y = Y_s$  by all the nonempty  $G_i$  and  $N_i$  (of which there are at least  $t \geq 2$ ). Note that for  $A'$ ,  $k' = k$ ,  $\sum y'_i \leq \sum y_i$ , and  $s' > s$ . Note also that the  $f$ -number of  $N_i$ ,  $f_{N_i} = 0$  for each nonempty  $N_i$ ,  $1 \leq i \leq t$ . Let  $h_i = f_{G_i}$  be the  $f$ -number of  $G_i$ . Note that  $h_i \leq g_i$ . By applying case (c) of the induction hypothesis to  $A'$  (and by the fact that  $\varphi_k(0, 0) = 0$ ) it suffices to prove that

$$\psi = \psi_1 - \psi_2 \geq 0 \quad \text{where}$$

$$(2.3) \quad \psi_1 = E\left(\sum_{i=1}^t \varphi_k(|N_i|, 0) + \sum_{i=1}^t \varphi_k(|G_i|, h_i)\right), \quad \psi_2 = \varphi_k(y, f),$$

and where the expected value is over all choices of the partition  $B_i$ ,  $C$ , and the permutation  $\pi$ . By the symmetry of the sets  $B_i$

$$E\left(\sum_{i=1}^t \varphi_k(|N_i|, 0) + \sum_{i=1}^t \varphi_k(|G_i|, h_i)\right) = tE(\varphi_k(|N_1|, 0) + \varphi_k(|G_1|, h_1))$$

$$\geq tE(\varphi_k(|N_1|, 0) + \varphi_k(|G_1|, g_1)).$$

Denote  $g = g_1$ . Let  $A_1$  be the event  $2g < \beta$  and let  $p = P(A_1)$  be its probability. Put  $\bar{p} = 1 - p$ . In this notation the right-hand side of the last inequality is simply

$$tpE_{A_1}(\varphi_k(\beta - 2g, 0) + \varphi_k(2g, g)) + t\bar{p}E_{\bar{A}_1}(\varphi_k(\beta, g)),$$

where  $E_{A_1}$ ,  $E_{\bar{A}_1}$  are the expectations given  $A_1$ ,  $\bar{A}_1$ , respectively. Hence

$$\psi_1 \geq tpE_{A_1}\left(k\left[\frac{(\beta - 2g)^{1+1/k}}{c} - (\beta - 2g)\right] + k\left[\frac{(2g)^{1+1/k}}{c(c/2)^{1/k}} - 2g\right]\right)$$

$$+ t\bar{p}E_{\bar{A}_1}\left(k\left[\frac{\beta^{1+1/k}}{c(cg/\beta)^{1/k}} - \beta\right]\right)$$

$$= \frac{tk}{c}\left[pE_{A_1}\left((\beta - 2g)^{1+1/k} + \frac{(2g)^{1+1/k}}{(c/2)^{1/k}}\right) + \bar{p}E_{\bar{A}_1}\frac{\beta^{1+1/k}}{(cg/\beta)^{1/k}}\right] - kt\beta(p + \bar{p}).$$

Put  $E_{A_1}(2g) = a_1$ ,  $E_{\bar{A}_1}(2g) = a_2$ ,  $E(2g) = a$ . Note that  $a_2 \geq \beta$ ,  $pa_1 + \bar{p}a_2 = a$ ,  $p + \bar{p} = 1$ , and  $\beta t \leq y$ . By the convexity of the functions  $x^{1+1/k}$  and  $1/x^{1/k}$  we can apply Jensen's inequality (see, e.g., [HLP59]) to get

$$\psi_1 \geq \frac{kt}{c}\left[p\left((\beta - a_1)^{1+1/k} + \frac{a_1^{1+1/k}}{(c/2)^{1/k}}\right) + \bar{p}\frac{\beta^{1+1/k}}{(\frac{1}{2}ca_2/\beta)^{1/k}}\right] - ky.$$

Put  $\alpha_1 = a_1/\beta$ ,  $\alpha_2 = a_2/\beta$ ,  $\alpha = a/\beta$ . Clearly

$$(2.4) \quad \alpha_2 \geq 1, \quad p\alpha_1 + \bar{p}\alpha_2 = \alpha,$$

and

$$\psi_1 \geq \frac{kt\beta^{1+1/k}}{c}\left[p\left((1 - \alpha_1)^{1+1/k} + \frac{\alpha_1^{1+1/k}}{(c/2)^{1/k}}\right) + \bar{p}\frac{1}{((c/2)\alpha_2)^{1/k}}\right] - ky.$$

Hence

$$\psi = \psi_1 - \psi_2 \geq \frac{kt\beta^{1+1/k}}{c}\left[p\left((1 - \alpha_1)^{1+1/k} + \frac{\alpha_1^{1+1/k}}{(c/2)^{1/k}}\right) + \frac{\bar{p}}{((c/2)\alpha_2)^{1/k}}\right]$$

$$- ky - k\left[\frac{y^{1+1/k}}{c(cf/y)^{1/k}} - y\right].$$

Since  $t < 8f/y$  we conclude, by (2.2), that

$$\psi \cong \frac{ky^{1+1/k}(1-8/c^{k+1})^{1+1/k}}{ct^{1/k}} \left[ p \left( (1-\alpha_1)^{1+1/k} + \frac{\alpha_1^{1+1/k}}{(c/2)^{1/k}} \right) + \frac{\bar{p}}{((c/2)\alpha_2)^{1/k}} - \frac{1}{(c/8)^{1/k}(1-8/c^{k+1})^{1+1/k}} \right].$$

As  $c = 256e$  we can check that  $(1-16/c^{k+1})^k \cong 1-16k/c^{k+1} \cong \frac{1}{2}$ , and hence

$$\left(1 - \frac{8}{c^{k+1}}\right)^{1+1/k} \cong \left(1 - \frac{8}{c^{k+1}}\right)^2 \cong 1 - \frac{16}{c^{k+1}} \cong \left(\frac{1}{2}\right)^{1/k}.$$

Therefore

$$\psi \cong \frac{ky^{1+1/k}(1-8/c^{k+1})^{1+1/k}}{ct^{1/k}} \left[ p \left( (1-\alpha_1)^{1+1/k} + \frac{\alpha_1^{1+1/k}}{(c/2)^{1/k}} \right) + \frac{\bar{p}}{((c/2)\alpha_2)^{1/k}} - \frac{1}{(c/16)^{1/k}} \right].$$

To establish (2.3) it suffices to show that

(2.5) 
$$h \cong 0 \quad \text{where}$$

$$h = p \left[ (1-\alpha_1)^{1+1/k} + \frac{\alpha_1^{1+1/k}}{(c/2)^{1/k}} \right] + \frac{\bar{p}}{((c/2)\alpha_2)^{1/k}} - \frac{1}{(c/16)^{1/k}}.$$

In Appendix 2 we prove that (2.5) holds, subject to the constraints (2.4) and

(2.6) 
$$\alpha \leq \frac{1}{2}.$$

In Appendix 3 we establish (2.6). Therefore  $\psi \cong 0$  and (2.3) holds, completing the proof of Subcase 1c and that of Case 1.

*Case 2.* For all  $i, 1 \leq i \leq s, f_i = 0$  and  $k > 1$ . Here the next round of comparisons ( $k$ th from the end) is performed. Let  $F$  be the set of these comparisons. Note that (if  $s > 1$ )  $F$  may contain comparisons between members of distinct sets  $Y_i$ , and hence the present situation is not necessarily a legal one. Let  $f_1, \dots, f_s$  be the new “ $f$ -numbers” of  $Y_1, \dots, Y_s$  (i.e.,  $f_i = e_i + \frac{1}{2}\bar{e}_i$ , where  $e_i, \bar{e}_i$  are as in the definition of a legal situation given in § 2.2). Clearly  $|F| \cong \sum_{i=1}^s f_i$ .

*Subcase 2a.*  $s = 1$ . In this case the present situation  $A'$  is a legal one. For  $A'$ , the number of remaining rounds is  $k' = k - 1$ , and there is a set  $Y = Y_1$  with  $f$ -number  $f = f_1$ , and  $|Y| = y$  elements and a set  $Z$ . By case (a) of the induction hypothesis  $F(k', A') \cong \varphi_{k-1}(y, f)$ . Therefore, to complete the proof for this subcase it suffices to check that

$$f + \varphi_{k-1}(y, f) \cong \varphi_k(y, 0).$$

If  $0 \leq f < y/4$  then

$$\begin{aligned} f + \varphi_{k-1}(y, f) &\cong \varphi_{k-1}(y, f) \cong (k-1) \left( \frac{y^{1+1/(k-1)}}{c(c/4)^{1/(k-1)}} - y \right) \\ &= ky \left[ \left( 1 - \frac{1}{k} \right) \frac{y^{1/(k-1)}}{c(c/4)^{1/(k-1)}} + \frac{1}{k} \cdot 1 \right] - ky. \end{aligned}$$

By the Arithmetic-Geometric Inequality  $\alpha a + \beta b \cong a^\alpha b^\beta$  for all  $\alpha, \beta, a, b \geq 0, \alpha + \beta = 1$ . Applying it with  $\alpha = 1 - 1/k, \beta = 1/k$  we get

$$\begin{aligned} f + \varphi_{k-1}(y, f) &\cong ky \left[ \frac{y^{(1/(k-1))(k-1/k)}}{c^{(k-1)/k} c^{1/k} (\frac{1}{4})^{1/k}} \cdot 1^{1/k} \right] - ky \\ &= \frac{ky^{1+1/k} \cdot 4^{1/k}}{c} - ky \cong \frac{ky^{1+1/k}}{c} - ky = \varphi_k(y, 0), \end{aligned}$$



as needed. Otherwise  $f > y/4$  and then, by a similar application of the Arithmetic-Geometric Inequality,

$$\begin{aligned}
 f + \varphi_{k-1}(y, f) &= f + (k-1) \left( \frac{y^{1+1/(k-1)}}{c(cf/y)^{1/(k-1)}} - y \right) \\
 &= ky \left[ \frac{1}{k} \left( \frac{f}{y} \right) + \left( 1 - \frac{1}{k} \right) \frac{y^{1/(k-1)}}{c^{1+1/(k-1)}(f/y)^{1/(k-1)}} \right] - (k-1)y \\
 &\geq ky \left( \frac{f}{y} \right)^{1/k} \frac{y^{(1/(k-1))((k-1)/k)}}{c^{(k/(k-1))((k-1)/k)}(f/y)^{1/k}} - (k-1)y \\
 &= \frac{ky^{1+1/k}}{c} - (k-1)y \geq k \left( \frac{y^{1+1/k}}{c} - y \right) = \varphi_k(y, 0).
 \end{aligned}$$

This completes the proof of Subcase 2a.

*Subcase 2b.*  $s > 1$ . Here the situation is not necessarily legal, as there may be comparisons between distinct sets  $Y_i$ . We will show that the average value of the  $f$ -number of comparisons of each set  $Y_i$  is at least  $\varphi_k(y_i, f_i)$ . As the total  $f$ -number is simply the sum of the  $f$ -numbers of the sets  $Y_i$ , this will give the desired result. Fix  $i$ ,  $1 \leq i \leq s$ . Let  $A_i''$  be the legal situation obtained from (the possibly nonlegal one)  $A'$  by defining  $Z'' = N \setminus Y_i$ , i.e., by giving the exact rank of each element outside  $Y_i$ . (Note that there are many distinct such  $A_i''$ , depending on the actual ranks of the elements in  $\cup_{j \neq i} Y_j$ .) As  $y_i < \sum_{j=1}^s y_j$ , we can apply case (b) of the induction hypothesis and conclude that  $F(k-1, A_i'') \geq \varphi_{k-1}(y_i, f_i)$ . It follows that any algorithm that sorts in  $k-1$  rounds, starting from the situation  $A'$  performs on the average at least  $\varphi_{k-1}(y_i, f_i)$  ( $f$ -number of) comparisons for  $Y_i$ , averaging only on those orders in which the ranks outside  $Y_i$  are as in the specific choice of  $A_i''$ . Averaging over all possible  $A_i''$  and summing over  $i$ ,  $1 \leq i \leq s$ , we conclude that

$$\begin{aligned}
 F(k, A) &\geq \sum_{i=1}^s f_i + \sum_{i=1}^s \varphi_{k-1}(y_i, f_i) \\
 &= \sum_{i=1}^s (f_i + \varphi_{k-1}(y_i, f_i)) \geq \sum_{i=1}^s \varphi_k(y_i, 0),
 \end{aligned}$$

where the last inequality follows from the computation of the previous subcase. This completes the proof of Case 2, and the proof of the induction step.

To complete the proof of Theorem 2.1, it remains to establish the base case of the induction. Clearly this case corresponds to a legal situation  $A$  with  $k = 1$ , with a set  $Z$  and sets  $Y_1, \dots, Y_s$ , where  $f_i = 0$  for all  $i$  (otherwise we are in either Case 1 or Case 2, and can apply the induction hypothesis). By the arguments given in the proof of Subcase 2b, we can reduce the case  $s > 1$  to the case  $s = 1$  (otherwise we bound the  $f$ -number for each  $Y_i$  separately, as in Subcase 2b). Hence we may assume that for  $A$ ,  $k = 1$ ,  $s = 1$ ,  $Y = Y_1$ ,  $|Y| = y$ ,  $f_1 = 0$ , and  $Z$  is a set of elements with known ranks. We must sort  $Y$  in one round. Note that since we must complete the sorting in one round we actually have to prove here a worst-case lower bound. If there are two successive ranks in the (known) set of ranks of  $Y$ , we must compare each pair of elements of  $Y$ . Indeed, suppose that a dispensed comparison is between such two successive elements of  $Y$  in the sorted order. The algorithm will clearly fail to determine their relative order. Hence, in this case,  $F(1, A) \geq \binom{y}{2} \geq \varphi_1(y, 0)$ , as needed. Therefore, we may assume that there is at least one element of  $Z$  between any two successive elements of  $Y$ . Clearly we can assume that there is precisely one element of  $Z$  between any two

successive elements of  $Y$  and that there are no elements of  $Z$  that are greater than all members of  $Y$  (or smaller than all members of  $Y$ ), as no additional information can be derived from comparisons to such elements. We can thus assume that  $|Z| = |Y| - 1 = y - 1$ , the (known) set of ranks of  $Z$  is  $\{2, 4, \dots, 2y - 2\}$ , and the (known) set of ranks of  $Y$  is  $\{1, 3, \dots, 2y - 1\}$ .

Let  $F$  be the set of comparisons performed by the algorithm. Suppose  $F = F' \cup F''$ , where  $F'$  are the comparisons between elements of  $Y$  and  $F''$  are those between elements of  $Y$  and elements of  $Z$ . We claim that if  $a, b$  are two distinct members of  $Y$  and the comparison (= edge)  $\{a, b\}$  satisfies  $\{a, b\} \notin F'$  then there are at least  $y - 1 = |Z|$  comparisons in  $F''$  that involve  $a$  or  $b$ . Indeed, if this is false, then there is an element  $z \in Z$  such that  $\{a, z\}, \{b, z\} \notin F''$ . If  $\text{rank}(z) = l$ , then the algorithm clearly will fail to distinguish between any order in which  $\text{rank}(a) = l - 1$  and  $\text{rank}(b) = l + 1$  and the order obtained from it by replacing the ranks of  $a$  and  $b$ . Thus the claim holds. Summing these comparisons of  $F''$  over all  $a, b$  with  $\{a, b\} \notin F'$ , we obtain at least  $(y - 1) \cdot (\binom{y}{2} - |F'|)$ . In this sum each comparison in  $F''$  is counted at most  $y - 1$  times, as for every  $a \in Y$  there are at most  $y - 1$   $b \in Y, b \neq a$  such that  $\{a, b\} \notin F'$ . Hence

$$(y - 1)|F''| \geq (y - 1) \left( \binom{y}{2} - |F'| \right), \quad \text{i.e., } |F'| + |F''| \geq \binom{y}{2}.$$

We conclude that the  $f$ -number of  $Y$  satisfies  $f = |F'| + \frac{1}{2}|F''| \geq \frac{1}{2}\binom{y}{2} \cong \varphi_k(y, 0)$ . This completes the proof of the base case of the induction and establishes Theorem 2.1.  $\square$

**3. Time lower bounds for randomized parallel sorting algorithms.** In this section we derive Theorem 1.1 from Proposition 1.2. The easy fact that for  $p \leq n$  the average time is  $\Theta(\log n / (p/n))$  follows from the existence of the sorting network [AKS83a], [AKS83b], together with the  $\Omega(n \log n)$  known bound for serial average randomized algorithms. The upper bound in inequality (1.1) implies that for every  $p \geq n$  time  $O(\log n / \log(1 + p/n))$  is sufficient, even for the worst case of deterministic algorithms. It remains to prove the lower bound  $\Omega(\log n / \log(1 + p/n))$  for  $p \geq n$ . As observed by Yao [Ya77], since any randomized algorithm is simply a probability distribution on deterministic ones, it suffices to establish the same lower bound for the average time of deterministic sorting algorithms with  $p$  processors. This does not follow immediately from Proposition 1.2, since in this proposition we considered only algorithms that necessarily stop after  $k$  rounds. Hence we need to do some more work. We first need the following simple probabilistic lemma. Let  $S_n$  denote the group of all permutations on  $n$  elements. For  $A \subseteq S_n$  and  $g_0 \in S_n$  define  $g_0A = \{g_0g \mid g \in A\}$ . Also define  $q(A) = |A|/|S_n| = |A|/n!$

**LEMMA 3.1.** *If  $A \subseteq S_n$  and  $q(A) \leq \frac{1}{2}$  then for every  $s \geq 1$  there are  $g_1, g_2, \dots, g_s \in S_n$  such that  $q(\bigcap_{i=1}^s g_iA) \leq 1/2^s$ .*

*Proof.* Choose, independently,  $s$  (not necessarily distinct) random elements  $g_1, g_2, \dots, g_s$  of  $S_n$ . If  $h \in S_n$ , the probability that  $h \in \bigcap_{i=1}^s g_iA$ , is precisely  $q(A)^s$ . Thus, the expected number of elements in  $\bigcap_{i=1}^s g_iA$  is  $n!q(A)^s \leq n!/2^s$ , and there are  $g_1, \dots, g_s \in S_n$  satisfying the conclusion of Lemma 3.1.  $\square$

**PROPOSITION 3.2.** *Suppose there is a deterministic algorithm  $M$  that sorts  $n$  elements with  $p$  processors in expected time  $T$ . Then, for every  $s \geq 1$ , there is a deterministic algorithm that sorts  $n$  elements in  $2T + \lceil T \rceil \leq 4T$  rounds (and necessarily stops after these  $4T$  rounds) with at most  $2Tps + 1/2^s \lceil T \rceil n^{1+b/T}$  average number of comparisons, where  $b > 0$  is the constant from (1.1).*

*Proof.* Let  $N$  be the set of elements we have to sort. Let  $A$  be the set of all permutations of  $N$  that  $M$  fails to sort in  $\leq 2T$  rounds. Clearly,  $q(A) \leq \frac{1}{2}$ . By Lemma 3.1 there exist  $g_1, \dots, g_s \in S_n$  such that  $q(\bigcap_{i=1}^s g_iA) \leq 1/2^s$ . Let  $M''$  be the algorithm

in which  $s$  copies of  $M$  run simultaneously for  $\lceil 2T \rceil$  rounds, where the  $i$ th copy runs on the elements of  $N$  permuted according to  $g_i^{-1}$ . Clearly  $M''$  finds the order of  $N$ , unless this order corresponds to a permutation in  $B = \bigcap_{i=1}^s g_i A$ . But this happens only on a  $1/2^s$ -fraction of the orders. Let  $M'$  be the algorithm that consists of  $M''$ , and if  $M''$  fails it uses the best deterministic algorithm for sorting  $N$  in  $\lceil T \rceil$  rounds. By (1.1) this part takes, in the worst case, at most  $\lceil T \rceil n^{1+b/T}$  additional comparisons. This completes the proof.  $\square$

*Proof of Theorem 1.1.* As observed in the beginning of this section, we only have to establish an  $\Omega(\log n / \log(1 + p/n))$  lower bound for the average time of deterministic algorithms for sorting  $n$  elements with  $p \geq n$  processors. By Proposition 1.2 there exists a (small) constant  $\frac{1}{2} > c > 0$  such that the average number of comparisons in any deterministic algorithm that sorts  $n \geq 2$  elements in  $k \leq \log n$  rounds is at least

$$(3.1) \quad ckn^{1+1/k}.$$

Let  $b \geq 1$  be the (large) constant from inequality (1.1). Let  $d$  be the (small) positive constant defined by

$$(3.2) \quad \frac{1}{16d} = 1 + \log(16b) + \log(1/2c).$$

(Notice that the right-hand side is positive, as  $c < \frac{1}{2}$ ,  $b \geq 1$ .)

Let  $M$  be a deterministic algorithm that sorts  $n$  elements with  $p \geq n$  processors in expected time  $T$ . To complete the proof we show that

$$(3.3) \quad T \geq d \log n / \log(1 + p/n).$$

If  $T \geq d \log n$  then (3.3) holds (for every  $p \geq n$ ). Hence, we may assume that  $T < d \log n (< b \cdot \log n)$ . Define  $s = \lceil b \log n / T \rceil$ . Clearly  $b \log n / T \leq s \leq 2b \log n / T$ . By Proposition 3.2, there is a deterministic algorithm that sorts  $n$  elements in at most  $4T$  rounds with at most

$$2Tps + \frac{1}{2^s} \lceil T \rceil n^{1+b/T} \leq 4Tp \frac{b \log n}{T} + 2Tn$$

average number of comparisons. Hence, by (3.1)

$$4Tp \frac{b \log n}{T} + 2Tn \geq c \cdot 4Tn^{1+1/(4T)},$$

i.e.,

$$4 \frac{p}{n} \frac{b \log n}{T} \geq 4cn^{1/(4T)} - 2 \geq 2cn^{1/(4T)},$$

where the last inequality holds since  $\log n / 4T > 1/4d > -\log(c)$ . By taking logarithms we obtain

$$\log \left( \frac{p}{n} \right) \geq \frac{\log n}{4T} - \log \left( \frac{\log n}{4T} \right) + \log(2c) - \log(16b).$$

As  $\log n / 4T > 1/4d > 4$  we have

$$\frac{\log n}{4T} - \log \left( \frac{\log n}{4T} \right) \geq \frac{\log n}{8T}.$$

Also, by (3.2),

$$\frac{\log n}{16T} > \frac{1}{16d} > \log(16b) - \log(2c).$$

These three inequalities imply

$$\log(p/n) > \frac{\log n}{16T},$$

and hence, for  $p \geq n$  the inequality

$$T > \frac{1}{16} \frac{\log n}{\log(p/n)} > d \log n / \log(1 + (p/n))$$

holds (provided our assumption  $T < d \log n$  holds). This establishes (3.3) and completes the proof.  $\square$

**4. Concluding remarks.** We have shown that the average running time of any comparison (deterministic or randomized) algorithm for sorting  $n$  elements with  $p$  processors is  $\Theta(\log n / \log(1 + p/n))$ , for all  $p \geq n$ , (and is  $\Theta(\log n / (p/n))$  for  $p \leq n$ .)

This is the first known nontrivial bound for randomized parallel comparison sorting algorithms. It shows that the average time of the best randomized algorithm is not smaller than the worst-case time of the best deterministic algorithm for all  $p$  and  $n$ , up to a constant factor.

We note that although Proposition 1.2 is mainly used as a tool for deriving this result, it actually gives additional information. This proposition shows that the average number of comparisons in any deterministic algorithm that sorts  $n$  elements in  $k \leq \log n$  rounds is  $\Omega(kn^{1+1/k})$ . As shown in [AAV86] this result is sharp for any fixed  $k$ . Note also that as shown in [AA87], the worst-case number of comparisons for such an algorithm is  $\Omega(n^{1+1/k}(\log n)^{1/k})$ , i.e., it is bigger for every fixed  $k$ . Thus we conclude that the average behaviour is somewhat different from the worst-case one for parallel comparison sorting algorithms. Our main result (Theorem 1.1) shows that this difference is, however, very small and shrinks to a constant factor if we fix the number of processors and estimate the running time. It is more than a constant factor if we fix the time and estimate the number of processors.

Our methods supply some results for randomized selection algorithms as well. In particular, we can show that the average number of comparisons needed in any randomized algorithm for finding the maximum of  $n$  elements in two rounds is  $\Theta(n^{4/3})$ , and for doing so in three rounds is  $\Theta(n)$ . We omit the details.

**Appendix 1.** We have to show that for  $0 < f \leq y/4$ ,  $\varphi_k(2f, f) + \varphi_k(y - 2f, 0) \geq \varphi_k(y, f)$ . That is,

$$k \left( \frac{(2f)^{1+1/k}}{c(c/2)^{1/k}} - 2f \right) + k \left( \frac{(y-2f)^{1+1/k}}{c} - (y-2f) \right) \geq k \left( \frac{y^{1+1/k}}{c(c/4)^{1/k}} - y \right),$$

or

$$(y-2f)^{1+1/k} + \frac{(2f)^{1+1/k}}{(c/2)^{1/k}} - \frac{y^{1+1/k}}{(c/4)^{1/k}} \geq 0.$$

Put  $\alpha = 2f/y$  and  $h(\alpha) = (1 - \alpha)^{1+1/k} + \alpha^{1+1/k}/(c/2)^{1/k} - 1/(c/4)^{1/k}$ . We have to show that  $h(\alpha) \geq 0$  for all  $0 < \alpha \leq \frac{1}{2}$ . This is done by checking that  $h$  is convex,  $h'(\frac{1}{2}) \leq 0$ , and  $h(\frac{1}{2}) \geq 0$ . Hence for all  $0 < \alpha \leq \frac{1}{2}$   $h'(\alpha) \leq 0$  and therefore  $h(\alpha) \geq h(\frac{1}{2}) \geq 0$ .

$h$  is convex, as it is a sum of convex functions.  $h'(\frac{1}{2}) = -(k+1)/k[(\frac{1}{2})^{1/k} - (1/2)^{1/k}/(c/2)^{1/k}] \leq 0$ , as  $c \geq 2$ .

$$\begin{aligned} h\left(\frac{1}{2}\right) &= \left(\frac{1}{2}\right)^{1+1/k} \left[ 1 + \left(\frac{2}{c}\right)^{1/k} - 2\left(\frac{8}{c}\right)^{1/k} \right] \\ &= \left(\frac{1}{2}\right)^{1+1/k} \left[ \left(1 - \left(\frac{8}{c}\right)^{1/k}\right)^2 + \left(\frac{64}{c^2}\right)^{1/k} \left(\left(\frac{c}{32}\right)^{1/k} - 1\right) \right] \geq 0, \end{aligned}$$

as  $c \geq 32$ .

**Appendix 2.** We have to show that

(2.5)  $h \geq 0$  where

$$h = p \left[ (1 - \alpha_1)^{1+1/k} + \frac{\alpha_1^{1+1/k}}{(c/2)^{1/k}} \right] + \frac{\bar{p}}{((c/2)\alpha_2)^{1/k}} - \frac{1}{(c/16)^{1/k}}$$

subject to

(2.4)  $\alpha_2 \geq 1, \quad p\alpha_1 + \bar{p}\alpha_2 = \alpha$

and

(2.6)  $\alpha \leq \frac{1}{2}$ .

(Recall that  $p, \bar{p}, \alpha, \alpha_1, \alpha_2 \geq 0, p + \bar{p} = 1, c = 256e$ .) Consider  $h$  as a function of  $\alpha_1$ , where  $\alpha, p, \bar{p}$  are constants and  $\alpha_2 = (\alpha - p\alpha_1)/\bar{p}$ . Since  $\alpha_2 \geq 1, \alpha \leq \frac{1}{2}$  we have  $0 \leq \alpha_1 \leq (\alpha - \bar{p})/p = 1 - (1 - \alpha)/p$ . Clearly  $h(\alpha_1)$  is a convex function of  $\alpha_1$ , i.e.,  $h''(\alpha_1) \geq 0$  for all admissible  $\alpha_1$ . Here we prove that  $h'(1 - (1 - \alpha)/p) \leq 0$  and hence, as  $h'(\alpha_1)$  is nonincreasing, we have  $h(\alpha_1) \geq h(1 - (1 - \alpha)/p)$ . We next show that  $h(1 - (1 - \alpha)/p) \geq 0$  and complete the proof. To check that  $h'(1 - (1 - \alpha)/p) \leq 0$ , observe that

$$\begin{aligned} h'(\alpha_1) &= p \left[ -\frac{k+1}{k} (1 - \alpha_1)^{1/k} + \frac{k+1}{k} \frac{\alpha_1^{1/k}}{(c/2)^{1/k}} \right] - \bar{p} \frac{c/2(-p/\bar{p})}{k((c/2)\alpha_2)^{1+1/k}} \\ &= \frac{p}{k} \left[ -(k+1)(1 - \alpha_1)^{1/k} + (k+1) \frac{\alpha_1^{1/k}}{(c/2)^{1/k}} + \frac{c}{2((c/2)\alpha_2)^{1+1/k}} \right]. \end{aligned}$$

For  $\alpha_1 = 1 - (1 - \alpha)/p$  we have  $\alpha_2 = 1$ ; hence (since  $0 \leq p \leq 1, \alpha \leq \frac{1}{2}$ )

$$\begin{aligned} h\left(1 - \frac{1 - \alpha}{p}\right) &= \frac{p}{k} \left[ -(k+1) \left(\frac{1 - \alpha}{p}\right)^{1/k} + (k+1) \left( \left(1 - \frac{1 - \alpha}{p}\right)^{1/k} / (c/2)^{1/k} \right) + \left(\frac{2}{c}\right)^{1/k} \right] \\ &\geq \frac{p}{k} \left[ -(k+1)(1 - \alpha)^{1/k} + (k+1)\alpha^{1/k} \left(\frac{2}{c}\right)^{1/k} + \left(\frac{2}{c}\right)^{1/k} \right] \\ &= -\frac{p}{k} \left[ (k+1) \left( (1 - \alpha)^{1/k} - \alpha^{1/k} \left(\frac{2}{c}\right)^{1/k} \right) - \left(\frac{2}{c}\right)^{1/k} \right] \\ &\geq -\frac{p}{k} \left[ (k+1) \left( \left(\frac{1}{2}\right)^{1/k} - \left(\frac{1}{2}\right)^{1/k} \left(\frac{2}{c}\right)^{1/k} \right) - \left(\frac{2}{c}\right)^{1/k} \right] \\ &= -\frac{p}{k} \left(\frac{1}{2}\right)^{1/k} \left[ (k+1) \left( 1 - \left(\frac{2}{c}\right)^{1/k} \right) - \left(\frac{4}{c}\right)^{1/k} \right]. \end{aligned}$$

The last quantity is clearly negative for  $k = 1$  (as  $c = 256e$ ). For  $k \geq 2$ ,  $(2/c)^{1/k} \leq (\frac{1}{4})^{1/k} \leq 1 - 1/k$ , and hence we have

$$h\left(1 - \frac{1-\alpha}{p}\right) \leq -\frac{p}{k} \left(\frac{1}{2}\right)^{1/k} \left[ (k+1) \left(1 - \left(1 - \frac{1}{k}\right)\right) - 1 \right] = -\frac{p}{k} \left(\frac{1}{2}\right)^{1/k} \leq 0,$$

as needed. It remains to check that  $h(1 - (1 - \alpha)/p) \geq 0$ . Indeed

$$h\left(1 - \frac{1-\alpha}{p}\right) = p \left[ \left(\frac{1-\alpha}{p}\right)^{1+1/k} + \left(1 - \frac{1-\alpha}{p}\right)^{1+1/k} / (c/2)^{1/k} \right] + \frac{\bar{p}}{(c/2)^{1/k}} - \left(\frac{16}{c}\right)^{1/k}.$$

Since  $(1-x)^\gamma \geq 1 - \gamma x$  for  $0 \leq x \leq 1$ ,  $\gamma \geq 1$  this implies

$$\begin{aligned} h\left(1 - \frac{1-\alpha}{p}\right) &\geq \frac{(1-\alpha)^{1+1/k}}{p^{1/k}} + p \frac{1 - ((1+1/k)(1-\alpha)/p)}{(c/2)^{1/k}} + \frac{1-p}{(c/2)^{1/k}} - \left(\frac{16}{c}\right)^{1/k} \\ &= \frac{(1-\alpha)^{1+1/k}}{p^{1/k}} + \frac{1 - (1+1/k)(1-\alpha)}{(c/2)^{1/k}} - \left(\frac{16}{c}\right)^{1/k}. \end{aligned}$$

As  $p \leq 1$  we conclude

$$\begin{aligned} h\left(1 - \frac{1-\alpha}{p}\right) &\geq \left(\frac{2}{c}\right)^{1/k} \left[ \left(\frac{c}{2}\right)^{1/k} (1-\alpha)^{1+1/k} + 1 - \left(1 + \frac{1}{k}\right)(1-\alpha) - 8^{1/k} \right] \\ &= \left(\frac{2}{c}\right)^{1/k} \left\{ (1-\alpha) \left[ \left(\frac{c(1-\alpha)}{2}\right)^{1/k} - \left(1 + \frac{1}{k}\right) \right] + 1 - 8^{1/k} \right\}. \end{aligned}$$

Since  $\alpha \leq \frac{1}{2}$  and  $c = 256e$  implies  $[(c/2)(1-\alpha)]^{1/k} \geq (c/4)^{1/k} = (64e)^{1/k} \geq 64^{1/k}$ .  $(1+1/k)$  we have

$$\begin{aligned} h\left(1 - \frac{1-\alpha}{p}\right) &\geq \left(\frac{2}{c}\right)^{1/k} \left\{ \frac{1}{2} \left[ \left(\frac{c}{4}\right)^{1/k} - \left(1 + \frac{1}{k}\right) \right] + 1 - 8^{1/k} \right\} \\ &= \frac{1}{2} \left(\frac{2}{c}\right)^{1/k} \left[ \left(\frac{c}{4}\right)^{1/k} + 1 - \frac{1}{k} - 2 \cdot 8^{1/k} \right] \\ &= \frac{1}{2} \cdot \left(\frac{2}{c}\right)^{1/k} \left[ (1 - 8^{1/k})^2 + \left(\frac{c}{4}\right)^{1/k} - 64^{1/k} - \frac{1}{k} \right] \\ &\geq \frac{1}{2} \cdot \left(\frac{2}{c}\right)^{1/k} \left[ (1 - 8^{1/k})^2 + 64^{1/k} \left(1 + \frac{1}{k}\right) - 64^{1/k} - \frac{1}{k} \right] \\ &= \frac{1}{2} \cdot \left(\frac{2}{c}\right)^{1/k} \left[ (1 - 8^{1/k})^2 + \frac{1}{k} (64^{1/k} - 1) \right] \geq 0. \end{aligned}$$

This completes the proof of (2.5).

**Appendix 3.** Here we show  $\alpha \leq \frac{1}{2}$ . Recall that  $\alpha = a/\beta$ , where  $a = E(2g) = E(2g' + g'')$ ,  $g'$  is the number of comparisons inside  $B_1$  and  $g''$  is the number of comparisons between  $B_1$  and  $Z_1$ . Let  $F'$  be the set of all comparisons (= edges) between elements of  $Y$  and let  $F''$  be the set of all comparisons between  $Y$  and  $Z$  whose results do not follow from the known information about the ranks. Put  $|F'| = e$ ,  $|F''| = \bar{e}$  and let  $f = e + \frac{1}{2}\bar{e}$  be the “ $f$ -number” of  $Y$ . As the members of  $B_1$  are  $\beta$  random elements of  $Y$  we have

$$E(g') = \binom{\beta}{2} / \binom{y}{2} |F'| \leq \frac{\beta^2}{y^2} e.$$

Consider a fixed edge (= comparison) in  $F''$ . Such a comparison compares some member  $m_y \in Y$  to some member  $m_z \in Z$ . The probability that the  $m_y$  will be in  $\cup_{i=1}^t B_i$  is, clearly,  $\beta t/y$ . As the permutation  $\pi$  is chosen randomly, the probability that  $m_y \in B_i$ , given that  $m_z \in Z_i$  (and that  $m_y \in \cup_{i=1}^t B_i$ ), is  $1/t$ . Therefore, the expected number of edges from  $F'$  that join members of  $B_i$  to members of  $Z_i$  for some  $1 \leq i \leq t$  is at most  $|F''| \cdot \beta t/y \cdot 1/t = \beta \bar{e}/y$ . By the symmetry of the sets  $B_i$  this gives  $E(g'') \leq \beta \bar{e}/y$ . Therefore, since  $\beta t \leq y$ ,

$$a = E(2g) = E(2g' + g'') \leq 2 \frac{\beta^2}{y^2} e + \frac{\beta \bar{e}}{yt} \leq 2 \frac{\beta}{yt} \left( e + \frac{1}{2} \bar{e} \right) = 2 \frac{b}{yt} f.$$

Recall that  $t \geq 4f/y$ , i.e.,  $f \leq ty/4$ . This implies  $\alpha = a/\beta \leq 2f/yt \leq \frac{1}{2}$ , as needed.

## REFERENCES

- [AA88] N. ALON AND Y. AZAR, *Sorting, approximate sorting, and searching in rounds*, SIAM J. Disc. Meth., 1 (1988), pp. 269–280.
- [AAV86] N. ALON, Y. AZAR, AND U. VISHKIN, *Tight complexity bounds for parallel comparison sorting*, in Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, Toronto, Ontario, Canada, 1986, pp. 502–510.
- [AHU74] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, London, 1974, pp. 92–93.
- [Ak85] S. AKL, *Parallel Sorting Algorithms*, Academic Press, New York, 1985.
- [AKS83a] M. AJTAI, J. KOMLÓS, AND E. SZÉMERÉDI, *An  $O(n \log n)$  sorting network*, in Proc. 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 1–9.
- [AKS83b] ———, *Sorting in  $c \log n$  parallel steps*, Combinatorica, 3 (1983), pp. 1–19.
- [AV87] Y. AZAR AND U. VISHKIN, *Tight comparison bounds on the complexity of parallel sorting*, SIAM J. Comput., 3 (1987), pp. 458–464.
- [BHe85] B. BOLLOBÁS AND P. HELL, *Sorting and graphs*, in Graphs and Orders, I. Rival, ed., D. Reidel, Boston, MA, 1985, pp. 169–184.
- [Bo86] B. BOLLOBÁS, *Random Graphs*, Academic Press, New York, 1986, Chap. 15.
- [BHo85] A. BORODIN AND J. E. HOPCROFT, *Routing, merging and sorting on parallel models of computation*, J. Comput. System Sci., 30 (1985), pp. 130–145.
- [BT83] B. BOLLOBÁS AND A. THOMASON, *Parallel sorting*, Discrete Applied Math., 6 (1983), pp. 1–11.
- [GK87] M. GERÉB-GRAUS AND D. KRIZANC, *The complexity of parallel comparison merging*, in Proc. 28th Annual IEEE Symposium on Foundations of Computer Science, Los Angeles, CA, 1987, pp. 195–201.
- [HH80] R. HÄGGKVIST AND P. HELL, *Graphs and parallel comparison algorithms*, Congr. Numer., 29 (1980), pp. 497–509.
- [HH81] ———, *Parallel sorting with constant time for comparisons*, SIAM J. Comput., 10 (1981), pp. 465–472.
- [HLP59] G. H. HARDY, J. E. LITTLEWOOD, AND G. POLYA, *Inequalities*, Cambridge University Press, Cambridge, 1959.
- [Kn73] D. E. KNUTH, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
- [Pi87] N. PIPPENGER, *Sorting and selecting in rounds*, SIAM J. Comput., 16 (1987), pp. 1032–1038.
- [Re81] R. REISCHUK, *A fast probabilistic sorting algorithm*, in Proc. 22nd Annual IEEE Symposium on Foundations of Computer Science, Nashville, TN, 1981, pp. 212–219.
- [Rei85] J. H. REIF, *An optimal parallel algorithm for integer sorting*, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, Portland, OR, 1985, pp. 496–503.
- [Th83] C. THOMPSON, *The VLSI complexity of sorting*, IEEE Trans. Comput., 32 (1983), pp. 1171–1184.
- [Va75] L. G. VALIANT, *Parallelism in comparison problems*, SIAM J. Comput., 4 (1975), pp. 348–355.
- [Ya77] A. C. C. YAO, *Probabilistic computations: Towards a unified measure of complexity*, in Proc. 18th Annual IEEE Symposium on Foundations of Computer Science, Providence, RI, 1977, pp. 222–227.

## P-PRINTABLE SETS\*

ERIC W. ALLENDER† AND ROY S. RUBINSTEIN‡

**Abstract.** P-printable sets arise naturally in the studies of generalized Kolmogorov complexity and data compression, as well as in other areas. We present new characterizations of the P-printable sets and present necessary and sufficient conditions for the existence of sparse sets in P that are not P-printable. As a corollary to one of our results, we show that the class of sets of small generalized Kolmogorov complexity is exactly the class of sets which are P-isomorphic to a tally language.

**Key words.** Kolmogorov complexity, sparse sets, P-isomorphisms, AuxPDAs, computational complexity

**AMS(MOS) subject classifications.** 68Q15, 68Q30, 68Q05, 03D15, 03D30

**1. Introduction.** Sparse sets have the useful property that for every  $n$  there is a table of size polynomial in  $n$  that lists the elements of the set of size less than or equal to  $n$ . Letting  $S$  be an arbitrary sparse set, it is conceivable that by storing a polynomial size table for sufficiently large  $n$ , we might have immediate access to all the information about  $S$  that we might ever need, despite the fact that the uniform complexity of  $S$  may be very high. Unfortunately, the complexity of producing such a table may be much greater than the complexity of recognizing the set  $S$ . If the complexity of producing a table for  $S$  is not much greater than the complexity of recognizing  $S$ , i.e., the complexity of producing a table for  $S$  is polynomial-time Turing reducible to  $S$ , then  $S$  is said to be *self P-printable*. If the complexity of producing a table for  $S$  is easy, i.e., a table for  $S$  can be produced in polynomial time without using  $S$  as an oracle, then  $S$  is *P-printable*. (Formal definitions for this and other concepts are given in § 2.) Obviously, P-printable sets belong to P. The notion of P-printability was introduced in [HY84] and was further explored in [HIS85].

The idea of generalized Kolmogorov complexity was introduced in [Har83] and [Sip83], and the connection between it and P-printability has been studied independently in [BB86], [HH86], and [Rub86b]. Generalized Kolmogorov complexity is a measure of how far a string can be compressed and how fast it can be restored. The relativized version of generalized Kolmogorov complexity allows the use of an oracle in the restoration. Sets containing only strings that can be greatly compressed and quickly restored (to be defined more precisely in the next section) are said to have *small generalized Kolmogorov complexity*. The papers [BB86] and [HH86] show that a set is self-P-printable if and only if it has small generalized Kolmogorov complexity relative to itself. This has the corollary (independently proved in [Rub86b]) that a set is P-printable if and only if it is in P and has small generalized Kolmogorov complexity.

P-printable sets are also shown here to have close connections with tally sets. Section 3 presents the result that a set is P-printable if and only if it is P-isomorphic to a tally language in P, if and only if it is in P and has small generalized Kolmogorov complexity. This has the important corollary that sets of small generalized Kolmogorov complexity are precisely those that are P-isomorphic to a tally language. This improves

---

\* Received by the editors November 18, 1986; accepted for publication (in revised form) February 3, 1988.

† Department of Computer Science, Rutgers University, New Brunswick, New Jersey 08903. The work of this author was supported in part by the National Science Foundation under grant MCS 81-03608 and by a 1986 Rutgers University Research Council Summer Fellowship.

‡ College of Computer Science, Northeastern University, Boston, Massachusetts 02115. The work of this author was supported in part by the National Science Foundation under grant DCR84-02033 and by the National Security Agency under grant MDA904-87-H-2020.



upon a result in [BB86] that the sets of small generalized Kolmogorov complexity are those that are “semi-isomorphic” to a tally language. Since most of the properties of sets studied in complexity theory are invariant under P-isomorphisms, the significance of this corollary is that the sets of small generalized Kolmogorov complexity are essentially identical to the tally languages from the point of view of complexity theory.

It is also shown that a set is P-printable if and only if it is sparse and has a ranking function computable in polynomial time. A ranking function is a function that maps an element of a set to its index in the lexicographic ordering of the set. They are presented in the context of data compression in [GS85] and are of independent interest [All85] and [Hem87].

Section 4 presents machine-based characterizations of the P-printable sets. It is shown there that a set is P-printable if and only if it is sparse and accepted by a deterministic one-way logspace-bounded AuxPDA, if and only if it is sparse and accepted by a nondeterministic one-way logspace-bounded AuxPDA. This presents a parallel to the result of Cook [Coo71] that a set is in P if and only if it is accepted by a (two-way) logspace-bounded AuxPDA. It is surprising that restricting AuxPDAs to have a one-way input head leads to a characterization of P-printable sets.

One-way AuxPDAs are not very powerful machines, as it was shown in [Bra77b] that some relatively “natural” languages in P are not accepted by one-way AuxPDAs of sublinear space complexity. With this result, any argument showing the existence of a sparse set in P (or even PSPACE) that is not accepted by any one-way logspace-bounded AuxPDA is strong enough to settle several outstanding problems in complexity theory, since, for example,  $P = PSPACE$  implies all sparse sets in PSPACE are P-printable [HY84].

Section 5 addresses some structural questions concerning P-printable sets. While it is clear that every P-printable set is sparse and in P, it is not known whether or not there is a sparse set in P that is not P-printable. Here we present the result that there is a sparse set in P that is not P-printable if and only if there is a sparse set in DLOG that is not P-printable, if and only if there is a sparse set in FewP – P. FewP was introduced in [All86b] as a class of sets between UP and NP (see § 2 for more details). This section concludes with the results that there are infinite sparse sets of time complexity arbitrarily close to polynomial that have finite intersection with every P-printable set, and that every infinite set in P has an infinite P-printable subset if and only if every infinite set in NP has an infinite P-printable subset.

**2. Preliminaries.** While it is assumed that the reader is familiar with the basic concepts and structures from complexity theory (such as P, NP, and DLOG), some of the more important and not universally known ones are presented here, along with notation.

We use the standard lexicographic ordering  $\leq$  on strings, and  $|w|$  denotes the length of the string  $w$ . All strings here are elements of  $\{0, 1\}^*$ , and all sets are subsets of  $\{0, 1\}^*$ . Although a language may be referred to as a subset of  $\{0, 1, \#\}^*$ , this is merely a notational convenience; such a language should be thought of as a subset of  $\{00, 11, 01\}^*$ . A tally language is a subset of  $\{0\}^*$ .

Strings are sometimes used to denote numbers (and vice versa) by letting the string  $w$  denote the number whose binary representation is  $1w$ . This preserves the ordering and allows us to write, for example,  $|w| = \lfloor \log w \rfloor$ . All logarithms are base two. We use EXPTIME to denote  $DTIME(2^{O(n)})$  and NEXPTIME to denote  $NTIME(2^{O(n)})$ .

**DEFINITION 1.** A set  $A$  is *sparse* if there exists a polynomial  $p$  such that the

number of strings in  $A$  of length less than or equal to  $n$  is less than or equal to  $p(n)$ .

The Kolmogorov complexity of finite strings was introduced independently by Kolmogorov [Kol65] and Chaitin [Cha66], [Cha75] as a way to measure the randomness of a finite string (or equivalently, the amount of information contained in a string). The Kolmogorov complexity of a finite binary string is the length of the shortest program that generates it. Intuitively it can be seen that if a string can be generated by a program shorter than itself (i.e., it can be *compressed*), it must contain some redundant information. A string is *random* if it cannot be compressed.

One limitation in using this as the definition of randomness is that it provides no limits or restrictions on the computation time to generate the original string from its smallest program. Time-bounded versions of Kolmogorov complexity have been considered by Ko [Ko86], Sipser [Sip83], and more recently by Hartmanis [Har83].

Hartmanis introduced a two-parameter version of Kolmogorov complexity (now called *generalized Kolmogorov complexity*) that includes information about not only how far a string can be compressed, but how fast it can be restored. This generalized Kolmogorov complexity is further explored in [All87], [BB86], [Huy86], [KOSW86], [Lon86], [Rub86b], and [Rub86a]. It is Hartmanis's definition of generalized Kolmogorov complexity that is presented here.

DEFINITION 2. For a Turing machine  $M_u$  and functions  $g$  and  $G$  mapping natural numbers to natural numbers, let

$$K_u[g(n), G(n)] = \{x \mid (\exists y)[|y| \leq g(|x|) \text{ and } M_u(y) = x \text{ in } G(|x|) \text{ or fewer steps}]\}.$$

We will refer to  $y$  as the *compressed string*,  $x$  as the *restored string*,  $g(n)$  as the *compression*, and  $G(n)$  as the *restoration time*. It was shown in [Har83] that there exists a Turing machine  $M_u$  (called a *universal Turing machine*) such that for any other Turing machine  $M_v$  there exists a constant  $c$  such that  $K_v[g(n), G(n)] \subseteq K_u[g(n) + c, cG(n) \log G(n) + c]$ . Dropping the subscript,  $K[g(n), G(n)]$  will actually denote  $K_u[g(n), G(n)]$ .

DEFINITION 3. A set is said to have *small generalized Kolmogorov complexity* if it is a subset of  $K[k \log n, n^k]$  for some  $k$ .

Clearly every set with small generalized Kolmogorov complexity is sparse. Note that the definition of small generalized Kolmogorov complexity is robust enough to handle the small difference between the universal machine mentioned above and any other machine.

We now give the definition of P-printability.

DEFINITION 4. A set  $S$  is *polynomial-time printable* (P-printable) if there exists a  $k$  such that all the elements of  $S$  up to size  $n$  can be printed by a deterministic machine in time  $n^k + k$ .

Clearly every P-printable set is necessarily sparse and in P.

Goldberg and Sipser [GS85] discuss compression of languages and ranking and present the following definitions.

DEFINITION 5. A function  $f: \Sigma^* \rightarrow \Sigma^*$  is a *compression* of language  $L$  if  $f$  is one-to-one on  $L$  and for all except finitely many  $x \in L$ ,  $|f(x)| < |x|$ .

DEFINITION 6. A language  $L$  is *compressible in time  $T$*  if there is a compression function  $f$  for  $L$  that can be computed in time  $T$ , and the "inverse" of  $f$ ,  $f^{-1}: f(L) \rightarrow L$ , such that for any  $x \in L$ ,  $f^{-1}(f(x)) = x$ , can be computed in time  $T$ .

This clearly relates to generalized Kolmogorov complexity in that compressible languages do not contain more than finitely many random strings, and the compression time (in this sense) relates to the second parameter of the generalized Kolmogorov complexity. Note that in this version, the compression time is a bound on both the

compression and restoration, whereas the time parameter of generalized Kolmogorov complexity refers only to the restoration time.

Reference [GS85] additionally presents the following definitions.

DEFINITION 7. A function  $f$  *optimally compresses* a language  $L$  if for any  $x \in L$  of length  $n$ ,  $|f(x)| \leq \lceil \log(\sum_{i=0}^n |L^i|) \rceil$ , where  $L^i$  is the set of strings in  $L$  of length  $i$ .

DEFINITION 8. For any set  $L \subseteq \Sigma^*$ , the *ranking function* for  $L$ ,  $r_L: \Sigma^* \rightarrow \mathbb{N}$ , is given by  $r_L(x) = |\{w \in L \mid w \leq x\}|$ .

The ranking is a special kind of optimal compression.

Allender [All86b] defined the complexity class FewP to be between UP and NP as follows.

DEFINITION 9. FewP is the class of languages that are accepted by nondeterministic polynomial-time Turing machines  $M$  for which there is a polynomial  $p$  such that for all inputs  $w$ , if  $M$  accepts  $w$ , there are fewer than  $p(|w|)$  accepting computations of  $M$  on  $w$ .

FewP is related to the class UP [Ber77], [Val76], [GS84] of languages in NP that are accepted by nondeterministic polynomial-time bounded Turing machines with unique accepting computations. Both UP and FewP are subclasses of NP defined by restricting the number of accepting computations. Densities of accepting computations were previously considered in [Mor82], but no class equivalent to FewP was introduced.

DEFINITION 10. A function  $f$  is a *P-isomorphism* if it is a bijection such that both  $f$  and  $f^{-1}$  are computable in polynomial time. Two sets  $A$  and  $B$  are *P-isomorphic* if there is some P-isomorphism  $f$  such that  $A = f(B)$ .

Auxiliary pushdown automata (AuxPDAs) are due to Cook [Coo71]. An AuxPDA is a Turing machine with a pushdown store in addition to a worktape. When we bound the space used by an AuxPDA, we bound only the space used on the worktape; the space used on the pushdown store is "free." Useful results about AuxPDAs are summarized in [HU79]. The fact that the languages accepted in time  $T(n)^{O(1)}$  are precisely the sets accepted by  $\log T(n)$  space-bounded deterministic and nondeterministic AuxPDAs [Coo71] will be used.

A one-way AuxPDA is an AuxPDA with a one-way input head. One-way AuxPDAs have been studied before in [Bra77b], [Bra77a], [Chy77], [WB79], and [Wec80]. In [BDG85] and [Huy85] one-way AuxPDAs were investigated in connection with restricted forms of nonuniform complexity. In most studies of one-way AuxPDAs, the machine starts its computation with  $\log n$  space marked off on its worktape; that is the model used here.

**3. Structural characterizations.** In this section the non-machine-based characterizations of the class of P-printable sets presented in the Introduction are proved.

THEOREM 1. *The following are equivalent:*

- (1)  $S$  is P-printable.
- (2)  $S$  is sparse and has a ranking function computable in polynomial time.
- (3)  $S$  is P-isomorphic to some tally set in P.
- (4)  $S \subseteq K[k \log n, n^k]$  and  $S \in P$ .

Note. The equivalence of (1) and (4) also appears in [BB86], [HH86], and [Rub86b].

*Proof.* [(1)  $\Rightarrow$  (2)]. The proof is immediate.

[(2)  $\Rightarrow$  (3)]. Let  $S$  have a ranking function  $r_1$  computable in polynomial time, and let there be fewer than  $p(n)$  strings of length  $n$  in  $S$ . Thus the function  $r_2$  given by  $r_2(w) = w - r_1(w)$  is a ranking function for the complement of  $S$ . (Recall that strings can represent numbers, as explained in § 2.) Also, the set  $T = \{0^{np(n)+i} \mid r_1(1^{n-1}) <$

$i \leq r_1(1^n)$  is a tally set in P; let  $r_3$  be a ranking function for the complement of  $T$ . As was noted in [GS85], all these ranking functions have inverses that are computable in time polynomial in the length of their output. It is now easy to show that the function that takes  $x$  of length  $n$  to  $0^{np(n)+r_1(x)}$  if  $x \in S$  and to  $r_3^{-1}(r_2(x))$  if  $x \notin S$  is a P-isomorphism mapping  $S$  onto  $T$ .

[(3)  $\Rightarrow$  (4)]. Let  $S$  be P-isomorphic to  $T \subseteq 0^*$  via some isomorphism  $f$  such that both  $f$  and  $f^{-1}$  are computable in time  $n^c$ . The compressed form of a string  $x \in S$  of length  $n$  is  $z$ , the binary representation of  $|f(x)|$ . Since  $f$  is computable in time at most  $n^c$ , it follows that  $|f(x)| \leq n^c$ , and thus  $|z| \leq c \log n$ . To get  $x$  back from  $z$ , simply compute  $f^{-1}(0^z)$ . Since  $|0^z| = |f(x)| \leq n^c$ , computing  $0^z$  from  $z$  takes time at most polynomial in  $n$ , call it  $n^l$ . Computing  $f^{-1}(0^z)$  takes time at most  $|0^z|^c \leq (n^c)^c = n^{c^2}$ . The computation time of  $x$  from  $z$  is thus  $\leq n^{c^2} + n^l \leq n^k$  for some  $k \geq c$ . Thus  $S \subseteq K[k \log n, n^k]$ . If  $T$  is in P, then  $S$  will also be in P since they are P-isomorphic.

[(4)  $\Rightarrow$  (1)]. Assume that  $S \in P$  and that for some  $k, S \subseteq K[k \log n, n^k]$ . On input  $n$ , for each of the  $n^{k+1} - 1$  strings of length  $\leq k \log n$ , run  $M_u$  for at most  $n^k$  steps and, if the computation has completed and the result is in  $S$ , print it. This process can clearly be done in time polynomial in  $n$ .  $\square$

It should be pointed out that results similar to those of Theorem 1 were presented in [BB86] as part of an investigation of sets of small generalized Kolmogorov complexity. In [BB86], Balcázar and Book define “semi-isomorphisms” and show that a set has small generalized Kolmogorov complexity if and only if it is semi-isomorphic to a tally set. Using Theorem 1 we can improve upon their result. First, however, we need the following easy result.

PROPOSITION 2. For all  $M_u$  and  $k, K_v[k \log n, n^k] \in P$ .

This proposition is readily seen to be true by the following procedure: to determine if a string  $x$  of length  $n$  is in  $K_v[k \log n, n^k]$ , run machine  $M_u$  on all strings of length less than or equal to  $k \log n$ , and accept if and only if  $M_u$  outputs  $x$  on one of these strings.

COROLLARY 3. There exists a  $k$  such that  $A \subseteq K[k \log n, n^k]$  if and only if  $A$  is P-isomorphic to a tally set.

Proof. The proof from right to left follows from the argument given in the proof of [(3)  $\Rightarrow$  (4)] of Theorem 1. For the forward direction, let  $A \subseteq K[k \log n, n^k]$ . By Theorem 1 and Proposition 2,  $K[k \log n, n^k]$  is P-isomorphic to some tally set  $T$  in P via some P-isomorphism  $f$ . It is now clear that  $A$  is P-isomorphic to  $f(A) \subseteq T \subseteq \{0^*\}$ .  $\square$

While no good upper bound is known for the generalized Kolmogorov complexity of sparse sets in P without assuming P-printability, it is not hard to show that every sparse set in P is a subset of  $K[O(\log n), 2^{O(n)}]$ . However, that is not very informative since every sparse set in EXPTIME is contained in  $K[O(\log n), 2^{O(n)}]$ .

We remark that, while it is not known whether or not there are sparse sets in P that are not subsets of  $K[k \log n, n^k]$  for any  $k$ , it is not hard to show that, for any time-constructible function  $T(n)$  that is greater than every polynomial, there is a sparse set in DTIME( $T(n)$ ) that is not a subset of  $K[k \log n, n^k]$  for any  $k$ . Also, there is a nonrecursive sparse set that is not a subset of  $K[S(n), T(n)]$  for any  $S(n) = o(n)$  and any recursive  $T(n)$ . As an example of such a set, consider a set consisting of exactly one Kolmogorov-random string of each length  $n$ . Sparse sets such as these are not P-isomorphic to any tally set.

**4. Machine-based characterizations.** We now present machine-based characterizations of the P-printable sets. Cook [Coo71] showed that a set is in P if and only if it is accepted by a (two-way) logspace-bounded AuxPDA. By restricting this machine

to be one way, we obtain a characterization of the P-printable sets.

**THEOREM 4.** *The following are equivalent:*

- (1)  $S$  is P-printable.
- (2)  $S$  is sparse and is accepted by a deterministic one-way logspace-bounded AuxPDA.
- (3)  $S$  is sparse and is accepted by a nondeterministic one-way logspace-bounded AuxPDA.

*Proof.* [(1)  $\Rightarrow$  (2)]. If  $S$  is P-printable, then there is a function  $f$  computable in polynomial time such that  $f(n)$  encodes the elements of  $S$  of length less than or equal to  $n$ . Thus the set  $A = \{n\#i \mid \text{the } i\text{th bit of } f(n) = 1\}$  is in EXPTIME and can thus be recognized by a deterministic AuxPDA that uses linear space. Equivalently, an AuxPDA with  $n\#i$  written on its worktape can determine if  $n\#i \in A$  without referencing its input tape (i.e., without moving its input head).

Using this trick, a one-way AuxPDA can easily check if the word on its input tape agrees with the  $i$ th word in the enumeration of  $S$ . If the  $i$ th word disagrees with the input in the  $j$ th position, it is not necessary to move the input head back to the start of the tape in order to compare the input with the  $(i+1)$ st word. Instead, find the next word in the enumeration which agrees with the  $i$ th word up to (and not including) the  $j$ th position. The following algorithm makes this precise. Let  $p$  be a polynomial such that for all  $n$  there are fewer than  $p(n)$  elements of  $S$  of length less than or equal to  $n$ . The following algorithm can be carried out by a deterministic logspace-bounded AuxPDA.

**begin**

At the start of the computation,  $w = w_1 \cdots w_n$  is on the input tape and  $\lfloor \log n \rfloor$  space is marked off on the worktape.

$r := 1^{\lfloor \log n \rfloor}$  //Note that the AuxPDA cannot know what  $n$  is until it has read the entire input. Clearly, however,  $n \leq r \leq 2n$ .

$i := 1$

$j := 0$  //The pair  $(i, j)$  will be maintained so that the word on the input tape will match the  $i$ th word in the enumeration of  $S$  up through position  $j$ .

**step one**

Increment  $j$  until a  $j$  is found (by repeated calls of the form  $r\#1 \in A?$ ,  $r\#2 \in A?$ ,  $\dots$ ) such that the  $j$ th input symbol differs from the  $j$ th symbol of the  $i$ th word of the enumeration of  $S$ , or until  $j = n$ .

If  $j = n$  and the first  $n$  symbols of the input match the first  $n$  symbols of the  $i$ th word of the enumeration and the  $i$ th word has length  $n$ , then halt and accept.

Otherwise go on to step two.

**step two**

By making repeated calls of the form  $r\#1 \in A?$ ,  $r\#2 \in A?$ ,  $\dots$ , find the least  $i'$  such that  $i < i' < p(r)$ , and such that the first  $j-1$  symbols of the  $i$ th and the  $i'$ th words of the enumeration of  $S$  agree.

If no such  $i'$  exists, halt and reject.

Otherwise set  $i := i'$  and go to step one.

**end**

The AuxPDA described above simply searches through the enumeration of  $S$  until some word is found that matches the input, and accepts if such a word is found. It is

easy to see that the algorithm is correct. Note that the only time the input head is moved is in step one, and that the input need only be read in one scan from left to right.

[(2)  $\Rightarrow$  (3)]. The proof is immediate.

[(3)  $\Rightarrow$  (1)]. Let  $S$  be sparse and accepted by a nondeterministic one-way logspace-bounded AuxPDA  $M$ . It is easy to construct another nondeterministic one-way AuxPDA accepting the set  $S' = \{0^n \# w \mid \text{there is a string } x \text{ such that } |wx| = n \text{ and } wx \in S\}$ . (This machine will store  $n$  in binary on its worktape and then simulate  $M$  on  $w$ . Then it will continue the simulation by guessing the  $n - |w|$  characters of  $x$ . Since  $M$  is one-way, only one bit of  $x$  needs to be stored at any time.) Thus  $S' \in P$ . By using  $S'$  it is now easy to construct the elements of  $S$ , bit by bit, and thus it follows that  $S$  is P-printable.

(This proof was pointed out to the first author by Osamu Watanabe, and simplifies the proof of the same result, which was presented in [All86b]).  $\square$

An interesting result somewhat related to Theorem 4 has recently been proved by Ibarra and Ravikumar; in [IR86] they show that all sparse CFLs are bounded.

The study of P-uniform circuit complexity is also related to the study of P-printable sets, and was part of the motivation for the present investigation of P-printable sets. A P-uniform family of circuits is a set of circuits  $\{C_n \mid n \geq 1\}$  such that the function  $n \rightarrow C_n$  is computable in time polynomial in  $n$ . P-uniform circuits are studied in [All85], [All86a], [BCH84], and [vzG84].

When studying circuit complexity classes, the class of functions that can be computed by very fast circuits (i.e., circuits of small depth) is of special interest. Thus, we are led to consider the class P-uniform NC (PUNC), the class of languages accepted by P-uniform circuits of depth  $\log^{O(1)} n$  [All86a].

Note that if  $\{C_n \mid n \geq 1\}$  is a P-uniform family of circuits, then it is a P-printable set. Thus in some sense, the sets in PUNC are those sets that can be computed very easily, relative to some P-printable set. For example, it is fairly easy to see that all P-printable sets are in PUNC.

Given this connection between P-printable sets and P-uniform circuit complexity, it is natural to consider the question of whether or not all sparse sets in PUNC are P-printable. It was shown in [All85], [All86a] that a set is in PUNC if and only if it is accepted by a logspace-bounded AuxPDA that moves its input head  $2^{\log^{O(1)} n}$  times. Thus both PUNC and the class of P-printable sets have characterizations in terms of AuxPDAs with restricted access to the input. On the other hand, we show in the next section that every sparse set in PUNC is P-printable if and only if every sparse set in P is P-printable.

**5. Some structural questions.** Questions relating to the existence and structure of non-P-printable sets are now presented.

While it is clear that all P-printable sets are sparse and in P, it is not known whether there are sparse sets in P that are not P-printable. The following theorem shows some equivalent conditions.

**THEOREM 5.** *The following are equivalent:*

- (1) *There is a sparse set in P that is not P-printable.*
- (2) *There is a sparse set in DLOG that is not P-printable.*
- (3) *There is a sparse set in FewP - P.*

*Proof.* If  $S$  is a sparse set in P that is not P-printable, then the set  $\{x \# 0^n \mid x \text{ is a prefix of a string of length } n \text{ in } S\}$  is a sparse set in FewP - P, thus proving [(1)  $\Rightarrow$  (3)]. [(3)  $\Rightarrow$  (2)] is proved by the observation that if  $S$  is sparse and in FewP - P, then the set of accepting computations of a machine accepting  $S$  is a sparse set in DLOG that is not P-printable. Because  $\text{DLOG} \subseteq P$ , the implication of (1) by (2) is clear.  $\square$

As a consequence of the fact that  $\text{DLOG} \subseteq \text{PUNC} \subseteq \text{P}$ , the above is also equivalent to the existence of a sparse set in  $\text{PUNC}$  that is not  $\text{P}$ -printable. This result provides a nice parallel to the result in [HY84] that there is a sparse set in  $\text{NP}$  that is not  $\text{P}$ -printable if and only if there is a sparse set in  $\text{NP-P}$ . While  $\text{NP}$  is the complexity class most closely related to the existence of non- $\text{P}$ -printable sparse sets in  $\text{NP}$ ,  $\text{FewP}$  is the complexity class most closely related to the existence of non- $\text{P}$ -printable sets in  $\text{P}$  and  $\text{DLOG}$ . As a side note, it is not known whether all sparse sets in  $\text{P}$  are in  $\text{PUNC}$ .

Turning now to the structure of non- $\text{P}$ -printable sets, we are interested in determining what they "look like." Can they be immune to the  $\text{P}$ -printable sets or must they have a  $\text{P}$ -printable subset? The following two theorems partially answer these questions.

**THEOREM 6.** *Let  $S$  be a set in  $\text{P}$  that is not  $\text{P}$ -printable, and let  $T(n)$  be a time-constructible function that grows faster than any polynomial. Then there is an infinite set  $A \in \text{DTIME}(T(n))$  such that  $A \subseteq S$  and  $A$  has finite intersection with every  $\text{P}$ -printable set.*

*Proof.* This is proved using techniques similar to those used in, e.g., [Orp86], where a result with a similar flavor concerning complexity cores was proved. Let  $M_1, M_2, \dots$  be an indexing of polynomial-time machines taking input from  $0^*$ ; each such machine can be viewed as taking  $0^n$  as input and producing a list of strings of length  $n$ . Thus  $\{M_i(0^*) \mid i \geq 1\}$  is a representation of all  $\text{P}$ -printable sets. Since the intersection of a set in  $\text{P}$  with a  $\text{P}$ -printable set is  $\text{P}$ -printable,  $S$  is not contained in  $M_i(0^*)$  for any  $i$ .

The idea of the proof is to find, for all  $i$ , a string  $x_i$  in  $S$  that is not in  $M_j(0^*)$  for any  $j \leq i$ . Such a string  $x_i$  must exist, or else  $S \subseteq M_1(0^*) \cup M_2(0^*) \cup \dots \cup M_i(0^*)$ , and thus  $S$  is  $\text{P}$ -printable.

Thus the function  $r(i) = \min \{n \mid \exists x \in S, |x| = n \text{ and } x \notin M_j(0^n) \text{ for all } j \leq i\}$  is total, monotone, and recursive. Let  $s$  be a total, monotone, recursive function that is greater than  $r$  and has the property that the function  $i \rightarrow s(i)$  can be computed in time linear in  $s(i)$ ; taking  $s$  to be the time complexity function for some machine computing  $r$  in unary will suffice.

We also require that our indexing satisfy the condition that machine  $M_i$  has running time bounded by  $T(n)/n$  on inputs of length  $n \geq i$ . Since  $T$  grows faster than any polynomial, this condition is easy to satisfy (see [Orp86]).

The following routine will run in time  $O(T(n))$  and will accept a subset  $A$  of  $S$  that has finite intersection with every  $\text{P}$ -printable set.

**begin** on input  $x$  of length  $n$

    Compute  $s(1), s(2), \dots$  until some  $i$  is found such that  $s(i) \leq n < s(i+1)$ .

    //This step takes  $O(n^2)$  time.

    Accept  $x$  iff  $x \in S - M_j(0^n)$  for all  $j \leq i$ .

    //This step takes  $\leq T(n)$  time, since (assuming without loss of generality that  $i < n$ ) each of the  $i$  simulations performed in this step requires at most  $T(n)/n$  steps.

**end**

It is clear that  $A \subseteq S$ , and that this routine runs in time  $O(T(n))$ . If  $x \in A$  then for some  $i$  such that  $s(i) \leq n < s(i+1)$ ,  $x$  is in  $S - M_j(0^n)$  for all  $j \leq i$ . Thus  $M_i(0^*)$  contains no elements in  $A$  of length  $\geq s(i)$ . It only remains to show that  $A$  is infinite.

Let  $i$  and  $i'$  be numbers such that  $s(i') \leq r(i) < s(i'+1)$ . By the definition of  $r$ , there is a string  $x_i \in S$  of length  $r(i)$  such that  $x_i \notin M_j(0^{|x_i|})$  for all  $j \leq i$ . In particular,  $x_i \notin M_j(0^{|x_i|})$  for all  $j \leq i'$ , since  $i' \leq i$ . Thus  $x_i \in A$  for all  $i$ , so  $A$  is infinite.  $\square$

It is natural to wonder whether the set  $A$  constructed in Theorem 6 can be made to be in  $\text{P}$ . A weaker form of this question asks whether every infinite set in  $\text{P}$  has an

infinite P-printable subset. While this is still undetermined, the following theorem, pointed out to the first author by David Russo, shows that the sets in NP have similar structure to the sets in P in this regard.

**THEOREM 7.** *Every infinite set in P has an infinite P-printable subset if and only if every infinite set in NP has an infinite P-printable subset.*

*Proof.* Assume that every infinite set in P has an infinite P-printable subset, and let  $L$  be an infinite set in NP. Thus there is an infinite set  $A$  in P such that  $L = \{x \mid \exists y, x \# y \in A\}$ . Without loss of generality, assume that there is some  $k$  such that  $x \# y \in A \Rightarrow |x \# y| = |x|^k$ . By assumption,  $A$  has an infinite P-printable subset  $S$ , so the set  $\{x \mid \exists y, x \# y \in S\}$  is an infinite P-printable subset of  $L$ .  $\square$

**Acknowledgments.** The first author acknowledges the influence of stimulating dialogue with his thesis advisor Kim King, Klaus Ambos-Spies, Ronald Book, Jin-Yi Cai, Juris Hartmanis, Steve Mahaney, Larry Ruzzo, David Russo, Alan Selman, and Osamu Watanabe.

The second author acknowledges the invaluable assistance of his advisor Alan Selman, and also Peter van Emde Boas.

#### REFERENCES

- [All85] E. ALLENDER, *Invertible functions*, Ph.D. thesis, Georgia Institute of Technology, Atlanta, GA, September 1985.
- [All86a] ———, *Characterizations of PUNC and precomputation*, in Proc. 13th International Colloquium on Automata, Languages, and Programming, Springer-Verlag, Berlin, New York, 1986, pp. 1–10.
- [All86b] ———, *The complexity of sparse sets in P*, in Proc. Conference on Structure in Complexity Theory, A. Selman, ed., Springer-Verlag, Berlin, New York, 1986, pp. 1–11.
- [All87] ———, *Some consequences of the existence of pseudorandom generators*, in Proc. 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 151–159.
- [BB86] J. BALCÁZAR AND R. BOOK, *On generalized Kolmogorov complexity*, Acta Inform., 23 (1986), pp. 679–688.
- [BCH84] P. BEAME, S. COOK, AND H. HOOVER, *Log depth circuits for division and related problems*, in Proc. 25th IEEE Symposium on Foundations of Computer Science, 1984, pp. 1–11.
- [BDG85] J. BALCÁZAR, J. DIAZ, AND J. GABARRÓ, *Uniform characterizations of non-uniform complexity measures*, Information and Control, 67 (1985), pp. 53–69.
- [Ber77] L. BERMAN, *Polynomial reducibilities and complete sets*, Ph.D. thesis, Cornell University, Ithaca, NY, 1977.
- [Bra77a] F.-J. BRANDENBERG, *The context sensitivity bounds of context sensitive grammars and languages*, in Proc. 4th International Colloquium on Automata, Languages, and Programming, Springer-Verlag, Berlin, New York, 1977, pp. 272–281.
- [Bra77b] ———, *On one-way auxiliary pushdown automata*, in Proc. 3rd GI Conference, Springer-Verlag, Berlin, New York, pp. 133–144.
- [Cha66] G. CHAITIN, *On the length of programs for computing finite binary sequences*, J. Assoc. Comput. Mach., 13 (1966), pp. 547–569.
- [Cha75] ———, *A theory of program size formally identical to information theory*, J. Assoc. Comput. Mach., 22 (1975), pp. 329–340.
- [Chy77] M. CHYTIL, *Comparison of the active visiting and the crossing complexities*, in Proc. 6th Conference on Mathematical Foundations of Computer Science, Springer-Verlag, Berlin, New York, 1977, pp. 272–281.
- [Coo71] S. COOK, *Characterizations of pushdown machines in terms of time-bounded computers*, J. Assoc. Comput. Mach., 19 (1971), pp. 175–183.
- [GS84] J. GROLLMANN AND A. SELMAN, *Complexity measures for public-key cryptosystems*, in Proc. 25th IEEE Symposium on Foundations of Computer Science, 1984, pp. 495–503.
- [GS85] A. GOLDBERG AND M. SIPSER, *Compression and ranking*, in Proc. 17th Annual ACM Symposium on Theory of Computing, 1985, pp. 440–448.
- [Har83] J. HARTMANIS, *Generalized Kolmogorov complexity and the structure of feasible computations*, in Proc. 24th IEEE Symposium on Foundations of Computer Science, 1983, pp. 439–445.



- [Hem87] L. HEMACHANDRA, *Counting in structural complexity theory*, Ph.D. thesis, Cornell University, Ithaca, NY, 1987.
- [HH86] J. HARTMANIS AND L. HEMACHANDRA, *On sparse oracles separating feasible complexity classes*, in Proc. 3rd Annual Symposium on Theoretical Aspects of Computer Science, Springer-Verlag, Berlin, New York, 1986, pp. 321-333.
- [HIS85] J. HARTMANIS, N. IMMERMANN, AND V. SEWELSON, *Sparse sets in NP-P: EXPTIME versus NEXPTIME*, Inform. Control, 65 (1985), pp. 158-181.
- [HU79] J. HOPCROFT AND J. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.
- [Huy85] D. HUYNH, *Non-uniform complexity and the randomness of certain complete languages*, Tech. Report TR 85-34, Iowa State University, Ames, IA, 1985.
- [Huy86] ———, *Resource-bounded Kolmogorov complexity of hard languages*, in Proc. Conference on Structure in Complexity Theory, A. Selman, ed., Springer-Verlag, Berlin, New York, 1986, pp. 184-195.
- [HY84] J. HARTMANIS AND Y. YESHA, *Computation times of NP sets of different densities*, Theoret. Comput. Sci., 34 (1984), pp. 17-32.
- [IR86] O. IBARRA AND B. RAVIKUMAR, *On sparseness, ambiguity, and other decision problems for acceptors and transducers*, in Proc. 3rd Annual Symposium on Theoretical Aspects of Computer Science, Springer-Verlag, Berlin, New York, 1986, pp. 171-179.
- [Ko86] K. KO, *On the notion of infinite pseudorandom sequences*, Theoret. Comput. Sci., 48 (1986), pp. 9-13.
- [Kol65] A. KOLMOGOROV, *Three approaches for defining the concept of information quantity*, Problems Inform. Transmission, 1 (1965), pp. 1-7.
- [KOSW86] K. KO, P. ORPONEN, U. SCHÖNING, AND O. WATANABE, *What is a hard instance of a computational problem?*, in Proc. Conference on Structure in Complexity Theory, A. Selman, ed., Springer-Verlag, Berlin, New York, 1986, pp. 197-217.
- [Lon86] L. LONGPRÉ, *Resource bounded Kolmogorov complexity, a link between computational complexity and information theory*, Ph.D. thesis, Cornell University, Ithaca, NY, 1986.
- [Mor82] S. MORAN, *On the accepting density hierarchy in NP*, SIAM J. Comput., 11 (1982), pp. 344-349.
- [Orp86] P. ORPONEN, *The structure of polynomial complexity cores*, Ph.D. thesis, University of Helsinki, Helsinki, Finland, 1986.
- [Rub86a] R. RUBINSTEIN, *Immunity for generalized Kolmogorov complexity classes*, Tech. Report NU-CCS-86-2, Northeastern University, Boston, MA, December 1986.
- [Rub86b] ———, *A note on sets with small generalized Kolmogorov complexity*, Tech. Report TR 86-4, Iowa State University, Ames, IA, March 1986.
- [Sip83] M. SIPSER, *A complexity theoretic approach to randomness*, in Proc. 15th ACM Symposium on Theory of Computing, 1983, pp. 330-335.
- [Val76] L. VALIANT, *Relative complexity of checking and evaluating*, Inform. Process. Lett., 5 (1976), pp. 20-23.
- [vzG84] J. VON ZUR GATHEN, *Parallel powering*, in Proc. 25th Annual ACM Symposium on Theory of Computing, 1984, pp. 31-36.
- [WB79] G. WECHSUNG AND A. BRANDSTADT, *A relation between space, return and dual return complexities*, Theoret. Comput. Sci., 9 (1979), pp. 127-140.
- [Wec80] G. WECHSUNG, *A note on the return complexity*, Elektron. Informationsverarb. Kybernet., 16 (1980), pp. 139-146.

## SEARCH IN AN ORDERED ARRAY HAVING VARIABLE PROBE COST\*

WILLIAM J. KNIGHT†

**Abstract.** Steiglitz and Parks ["What Is the Filter-Design Problem?," *Proc. 1986 Princeton Conference on Information Science and Systems*, B. W. Dickenson, ed., Princeton University, Dept. of Electrical Engineering, Princeton, NJ, 1986] have shown that a problem in filter design gives rise to a related problem of how to search an ordered array in which the cost of a probe into the array varies with the location being probed. In this paper we prove that if probing in location  $k$  has cost  $k^p$ , where  $p$  is a positive integer, then the expected cost of a successful or unsuccessful search for a target element is at least  $(p+1)^{-1} n^p \lg n + O(n^p)$ . We also prove the somewhat surprising fact that ordinary binary search has this expected cost. However, for the case  $p=1$  we describe what appears to be a marginally better search algorithm.

**Key words.** binary search, search trees, array search

**AMS(MOS) subject classification.** 68P10

**1. Introduction.** In [3] Steiglitz and Parks describe a filter-design problem whose solution involves the following situation. For each positive integer  $k$  between two prescribed limits there is an associated linear programming (l.p.) problem of complexity  $k$ , the feasibility of which is not known; when the l.p. problem is feasible for some value of  $k$ , then all larger values of  $k$  have feasible associated problems; it is required that we find the smallest  $k$  whose associated l.p. problem is feasible. An interesting question immediately arises: What strategy should we use to find the smallest  $k$ ? Recall that the expected amount of time required to solve an l.p. problem is proportional to its complexity. Thus, in effect, we are confronted by an array of no's and yes's, with all the no's to the left of the yes's, in which a probe into location  $k$  has an associated cost proportional to  $k$ , and we must search the array for the location of the first yes. Steiglitz and Parks remark that because the probe cost grows with  $k$ , binary search might be expected to be inferior (as measured by expected cost) to a strategy that probes somewhere to the left of the midpoint of the remaining array at each step of the search. Computation of the optimal strategies, however, suggested to them that binary search is surprisingly near optimal. In this paper we give mathematical proofs that confirm this conjecture, but we also describe a search algorithm that appears to be marginally better than binary search.

**2. Preliminaries.** Let us generalize the investigation to any situation in which we confront an array of problems of some kind, one for each array subscript. We assume that solving the  $k$ th problem requires, on the average, an amount of time given by some penalty function  $P(k)$ . Each such problem has a yes or no answer, and when the answer is yes for some  $k$  it is yes for all larger subscripts  $k$ . We must find the smallest  $k$  for which the answer is yes, and we want to know the optimal search strategy, as measured by expected cost. We have no prior information about the location of the first yes, and so we assume that it is equally likely to be found in any of the  $n$  locations or not to be found at all. That is, each possibility has probability  $1/(n+1)$ . There is no loss in generality in assuming that the subscripts range from 1 to  $n$ , since any other range can be handled by translation of the subscripts and of the variable  $k$  in  $P(k)$ . In the Steiglitz-Parks problem,  $P(k)$  is a linear polynomial.

---

\* Received by the editors October 10, 1986; accepted for publication (in revised form) January 16, 1988.

† Department of Mathematics and Computer Science, Indiana University, P.O. Box 7111, South Bend, Indiana 46634.

Every search strategy for this situation corresponds to a unique binary tree  $T_n$  with node labels  $1, 2, \dots, n$ , placed so that the inorder traversal of  $T_n$  finds the labels in increasing order. Conversely, every such binary tree corresponds to exactly one strategy. We shall call any such tree a *search tree*. The root of the tree gives the location of the first probe. We go left if we get a yes, right if we get a no, and repeat this with the corresponding subtree. Note that *we cannot stop until we reach an external node* (see [2, p. 239]), at which point we can identify the location of the first yes. Thus in this situation the search prescribed by the tree corresponds to an *unsuccessful search* of a conventional array, in which records with keys have been stored in increasing key order (see [2, p. 237]). It is this observation that prompted the title of this paper.

For a given probe penalty function  $P(k)$  the expected cost of an unsuccessful search using a search tree  $T_n$  is

$$(1) \quad \sum_{j=1}^{n+1} \frac{1}{n+1} \left\{ \sum_{\text{nodes } k \text{ from root to } j\text{th external node}} P(k) \right\},$$

where the inner sum in formula (1) is taken over all *internal* nodes  $k$  that lie on the path in  $T_n$  from the root down to the  $j$ th external node. We can write (1) in a more useful form by letting  $W_k(T_n)$  denote the number of internal nodes in that subtree of  $T_n$  whose root is  $k$ . The number of internal nodes in an extended binary tree is one less than the number of external nodes. It follows that  $W_k(T_n)$  is one less than the number of times that the term  $P(k)$  will occur when the value of formula (1) is computed. Thus the expected cost of our search, using  $T_n$ , is equal to

$$\frac{1}{n+1} \sum_{k=1}^n P(k) [W_k(T_n) + 1] = \frac{1}{n+1} \sum_{k=1}^n P(k) W_k(T_n) + \frac{1}{n+1} \sum_{k=1}^n P(k).$$

For computational convenience we discard the constant factor  $1/(n+1)$  and the constant term  $\sum P(k)$ , and we call the remaining sum the *search cost of  $T_n$*  for the given penalty function  $P(k)$ . We denote this by  $S_{P(k)}(T_n)$ . That is,

$$(2) \quad S_{P(k)}(T_n) = \sum_{k=1}^n P(k) W_k(T_n).$$

Thus, for example,  $S_1(T_n) = \sum W_k(T_n)$ ,  $S_{k+5}(T_n) = \sum (k+5) W_k(T_n)$ ,  $S_{\log(1+k)}(T_n) = \sum \log(1+k) W_k(T_n)$ , and so on. We note, parenthetically, that  $S_{P(k)}(T_n)$  is  $n$  times the expected cost of a *successful* search for a target record in a conventional ordered array having penalty  $P(k)$  for probing in location  $k$ .

Although our ultimate goal is to find all optimal search trees, extensive computer calculations indicate that even when  $P(k)$  is a rather tame function, the optimal trees do not organize themselves into simple patterns. For example, suppose  $P(k) = k$ . Then as  $n$  varies, the root of an optimal tree with  $n$  nodes wanders between  $n/4$  and  $n/2$  (approximately), with occasional sudden drops. For a fixed  $n$  there may be several optimal trees, the roots of which do not form a simple sequence. For example, when  $n = 33$  the optimal trees have roots 7, 10, and 11 (see the table at the end of the paper for more details). These cautionary findings suggest that it will not be easy to derive exact formulas for  $S_{P(k)}(T_n)$ , even when  $P(k)$  is simple, and in this paper we obtain mostly upper and lower bounds. For  $P(k) \equiv 1$ , however, we know the exact search cost in a *balanced* tree. Recall that a binary tree  $T$  of height  $h$  is called *balanced*, provided every external node of  $T$  is at level  $h$  or  $h+1$ .

LEMMA 1. Let  $T_n$  denote any balanced search tree with  $n$  nodes ( $n \geq 0$ ). Let  $p = \lfloor \lg(n+1) \rfloor$ , where as usual,  $\lg = \log_2$ . Then

$$S_1(T_n) = \sum_{k=1}^n W_k(T_n) = (n+1)(p+1) - 2^{p+1} + 1.$$

*Proof.* Let  $\text{Ext}(T_n)$  denote the external path length of  $T_n$  (see [2, pp. 239–243]). It is not hard to see that  $\sum W_k(T_n) = \text{Ext}(T_n) - n$ . By formula (5.10) in [2, p. 243]

$$\text{Ext}(T_n) - n = (n+1) \lg(n+1) + (n+1)[2 - \lg(n+1) + p - 2^{1-\lg(n+1)+p}] - n,$$

which reduces to the formula given in the lemma.  $\square$

The following corollary gives a lower bound for  $S_1(T_n)$  when  $T_n$  is any search tree, balanced or not. It is obtained from Lemma 1 by replacing  $p$  by  $\lg(n+1)$ .

LEMMA 2. For every search tree  $T_n$  with  $n$  nodes ( $n \geq 0$ ) we have

$$S_1(T_n) = \sum_{k=1}^n W_k(T_n) \geq (n+1)[\lg(n+1) - 1] + 1.$$

*Proof.* Let  $T_n$  be any search tree. First suppose  $T_n$  is not balanced. Then it is possible to move some leaf to a higher level (smaller level number). Doing so decreases the number of its ancestors, which decreases  $\sum W_k(T_n)$ . After a finite number of such leaf lifts, the tree is balanced. Thus it suffices to prove that the inequality holds for balanced trees. By Lemma 1 it then suffices to prove

$$(n+1)(p+1) - 2^{p+1} + 1 \geq (n+1)[\lg(n+1) - 1] + 1$$

for  $n = 0, 1, 2, \dots$ , where  $p = \lfloor \lg(n+1) \rfloor$ . This is equivalent to

$$(3) \quad (n+1)[p - \lg(n+1) + 2] \geq 2^{p+1}.$$

Let  $\theta = \lg(n+1) - p$ . Then  $0 \leq \theta < 1$ , and (3) is equivalent to

$$2^\theta(2 - \theta) \geq 2.$$

Denote the left side here by  $f(\theta)$  and note that  $f(0) = f(1) = 2$ . Moreover, the second derivative is

$$f''(\theta) = 2^\theta (\ln 2)[(-2)(1 - \ln 2) - \theta \ln 2] < 0$$

on the interval  $0 \leq \theta \leq 1$ , so  $f$  is concave down. These facts imply that  $f(\theta) \geq 2$  on the interval.  $\square$

There is a fundamental recursion formula that we shall use repeatedly. It arises from the following elementary fact: if  $T_n$  is a search tree with  $n$  nodes and root  $r$ , then the left subtree of  $r$  is a search tree with  $r-1$  nodes, while the right subtree is a search tree with  $n-r$  nodes, provided we decrease each node label by  $r$ . We denote the left subtree by  $T_{r-1}$ , and the right subtree with modified labels by  $T_{n-r}$ . Then for any probe penalty function  $P(k)$  whatsoever,

$$S_{P(k)}(T_n) = P(r) \cdot n + \sum_{k=1}^{r-1} P(k) \cdot W_k(T_n) + \sum_{k=r+1}^n P(k) \cdot W_k(T_n),$$

which gives us the important, perfectly general recursion formula

$$(4) \quad S_{P(k)}(T_n) = P(r) \cdot n + S_{P(k)}(T_{r-1}) + \sum_{k=1}^{n-r} P(k+r) \cdot W_k(T_{n-r}).$$

**3. Linear penalty functions.** In this section we derive upper and lower bounds for the search cost in an optimal tree when the probe penalty function is a linear polynomial.

These bounds are of interest in themselves, and they are also used to derive bounds for higher-order penalty functions. We begin with the special case where  $P(k) = k$  and where the search tree is *full*, by which we mean that it is balanced and has  $2^p - 1$  nodes for some integer  $p$ . Denote this tree by  $F_p$ .

LEMMA 3. For all integers  $p \geq 0$  and corresponding full binary trees  $F_p$  we have

$$(5) \quad S_k(F_p) = \sum_{k=1}^{2^p-1} k \cdot W_k(F_p) = 2(p-1)4^{p-1} + 2^{p-1}.$$

If we write  $n = 2^p - 1$  (the number of nodes in  $F_p$ ), then the formula takes the form

$$(6) \quad S_k(F_p) = \frac{1}{2}(n+1)^2[\lg(n+1) - 1] + \frac{1}{2}(n+1).$$

*Proof.* The proof is by induction. The formula is trivially true when  $p = 0$ . For  $p > 0$  we use (4), the fact that the root  $r$  of  $F_p$  is  $2^{p-1}$ , and the fact that the left and right subtrees of  $F_p$  are full trees with  $2^{p-1} - 1$  nodes. Thus

$$\begin{aligned} S_k(F_p) &= 2^{p-1}(2^p - 1) + S_k(F_{p-1}) + \sum_{k=1}^{2^{p-1}-2^{p-1}} (k + 2^{p-1})W_k(F_{p-1}) \\ &= 2^{2p-1} - 2^{p-1} + 2S_k(F_{p-1}) + 2^{p-1}S_1(F_{p-1}). \end{aligned}$$

By induction and (5),

$$S_k(F_{p-1}) = 2(p-2)4^{p-2} + 2^{p-2}.$$

By Lemma 1, with  $n = 2^{p-1} - 1$  and (consequently)  $p$  replaced by  $p - 1$ , we have

$$S_1(F_{p-1}) = 2^{p-1}p - 2^p + 1,$$

so

$$\begin{aligned} S_k(F_p) &= 2^{2p-1} - 2^{p-1} + 2[2(p-2)4^{p-2} + 2^{p-2}] + 2^{p-1}[2^{p-1}(p-2) + 1] \\ &= 2(4^{p-1}) - 2^{p-1} + (p-2)4^{p-1} + 2^{p-1} + 4^{p-1}(p-2) + 2^{p-1}. \end{aligned}$$

Algebraic reduction gives (5).  $\square$

Just as Lemma 1 suggests (2) of Lemma 2, so (6) of Lemma 3 suggests that for all search trees  $T_n$

$$(7) \quad S_k(T_n) \geq \frac{1}{2}(n+1)^2[\lg(n+1) - 1] + \frac{1}{2}(n+1).$$

Computer calculations verify (7) for all  $n$  up to 1084, but the best that the author can prove is the following slightly weaker inequality.

THEOREM 4. For every search tree  $T_n$  with  $n$  nodes ( $n \geq 0$ ) we have

$$(8) \quad S_k(T_n) \geq \frac{1}{2}(n+1)^2[\lg(n+1) - 1.070] + \frac{1}{2}(n+1).$$

*Proof.* We use induction to prove that for all search trees  $T_n$  with  $n$  nodes,

$$(9) \quad S_k(T_n) \geq \frac{1}{2}(n+1)^2[\lg(n+1) - K] + \frac{1}{2}(n+1)$$

for a constant  $K$  to be determined. When  $n = 0$  the formula holds, provided  $K \geq 1$ . Now suppose (9) holds for  $n = 0, 1, \dots, m - 1$ . Let  $T_m$  be a search tree with  $m$  nodes and root  $r$ . Then by (4) and induction we have

$$\begin{aligned} S_k(T_m) &= r \cdot m + S_k(T_{r-1}) + \sum_{k=1}^{m-r} (k+r) \cdot W_k(T_{m-r}) \\ &= r \cdot m + S_k(T_{r-1}) + S_k(T_{m-r}) + r \sum_{k=1}^{m-r} W_k(T_{m-r}) \\ &\geq r \cdot m + \frac{1}{2}r^2[\lg r - K] + \frac{1}{2}r + \frac{1}{2}(m-r+1)^2[\lg(m-r+1) - K] + \frac{1}{2}(m-r+1) \\ &\quad + r\{(m-r+1)[\lg(m-r+1) - 1] + 1\}, \end{aligned}$$

where the last line uses Lemma 2. We wish to prove that this expression is

$$\cong \frac{1}{2}(m+1)^2[\lg(m+1) - K] + \frac{1}{2}(m+1).$$

Write  $\lambda = r/(m+1)$ , so that  $0 < \lambda < 1$ . Then the inequality we seek to prove takes the form

$$\begin{aligned} & 2\lambda(m+1)m + \lambda^2(m+1)^2[\lg(\lambda(m+1)) - K] + \lambda(m+1) \\ & \quad + (1-\lambda)^2(m+1)^2[\lg((1-\lambda)(m+1)) - K] \\ & \quad + (1-\lambda)(m+1) + 2\lambda(m+1)\{(1-\lambda)(m+1)[\lg((1-\lambda)(m+1)) - 1] + 1\} \\ & \cong (m+1)^2[\lg(m+1) - K] + (m+1), \end{aligned}$$

which simplifies to

$$\begin{aligned} & \lambda^2(m+1)^2[\lg(m+1) + \lg \lambda] - \lambda^2(m+1)^2[\lg(m+1) + \lg(1-\lambda)] \\ & \quad + (m+1)^2 \lg(1-\lambda) + 2\lambda^2(m+1)^2 \cong -2K\lambda(1-\lambda)(m+1)^2, \end{aligned}$$

which further simplifies to

$$\frac{-2\lambda^2 - \lambda^2 \lg \lambda - (1-\lambda^2) \lg(1-\lambda)}{2\lambda(1-\lambda)} \leq K.$$

Mathematical analysis, combined with a computer search, shows that the maximum value of the expression on the left is less than 1.070 (for details, see the following paragraph). Thus we may take  $K$  to be 1.070.

Here is a sketch of the proof that the left side of the inequality above is less than 1.070. Denote the left side by  $f(\lambda)$  and use the identity  $\lg x = \ln x / \ln 2$  to write

$$f(\lambda) = \frac{-\lambda \lg \lambda}{2(1-\lambda)} + \frac{-2\lambda^2 - (1+\lambda)(1-\lambda) \ln(1-\lambda) / \ln 2}{2\lambda(1-\lambda)}.$$

We shall show that  $f(\lambda) < 1.070$  on each of the three intervals  $(0, 1/16)$ ,  $[1/16, 0.692]$ , and  $(0.692, 1)$ . First note that  $-\lambda \lg \lambda$  is increasing on the interval  $0 < \lambda < 1/e$ , and so is the fraction  $1/(2(1-\lambda))$ . Thus when  $0 < \lambda < 1/16$  we have

$$\frac{-\lambda \lg \lambda}{2(1-\lambda)} < \frac{-(1/16) \lg(1/16)}{2(15/16)} = \frac{2}{15}.$$

Also note that when  $|\lambda| < 1$  we have the Maclaurin expansions

$$-\ln(1-\lambda) = \lambda + \lambda^2/2 + \lambda^3/3 + \lambda^4/4 + \dots,$$

$$-(1-\lambda) \ln(1-\lambda) = \lambda - \lambda^2/(1 \cdot 2) - \lambda^3/(2 \cdot 3) - \lambda^4/(3 \cdot 4) + \dots.$$

Thus when  $0 < \lambda < 1$  we have the following three inequalities:

$$(10) \quad -\ln(1-\lambda) < \lambda + (\lambda^2/2)(1 + \lambda + \lambda^2 + \dots) = \lambda + (\lambda^2/2)/(1-\lambda),$$

$$(11) \quad -(1-\lambda) \ln(1-\lambda) < \lambda, \quad \text{and}$$

$$(12) \quad |\lambda + (1-\lambda) \ln(1-\lambda)| < (\lambda^2/2)(1 + \lambda + \lambda^2 + \dots) = (\lambda^2/2)/(1-\lambda).$$

From (11) we obtain

$$\frac{-2\lambda^2 - (1+\lambda)(1-\lambda) \ln(1-\lambda) / \ln 2}{2\lambda(1-\lambda)} < \frac{-2\lambda + (1+\lambda) / \ln 2}{2(1-\lambda)}.$$

In particular, when  $0 < \lambda < 1/16$  we have

$$f(\lambda) < \frac{2}{15} + \frac{0 + (17/16) / \ln 2}{2(15/16)} < 1.$$

Now consider  $f(\lambda)$  on the interval  $0.692 < \lambda < 1$ . First replace  $\lambda$  by  $1 - \lambda$  in (11) to get  $-\lambda \ln \lambda < 1 - \lambda$ , which gives us

$$\frac{-\lambda \ln \lambda}{2(1-\lambda)} < \frac{1}{2 \ln 2} = 0.7214.$$

Next use (10) and then  $1 - \lambda < 0.308$  to get

$$-(1 - \lambda^2) \ln (1 - \lambda) < \lambda + \frac{1}{2}(\lambda^2 - \lambda^3) < \lambda + \lambda^2(0.154).$$

This gives us

$$\frac{-2\lambda^2 - (1 - \lambda^2) \ln (1 - \lambda) / \ln 2}{2\lambda(1 - \lambda)} < \frac{-2\lambda^2 + [\lambda + \lambda^2(0.154)] / \ln 2}{2\lambda(1 - \lambda)} = \frac{0.7214 - 0.8889\lambda}{1 - \lambda}.$$

By elementary calculus, the last fraction above is decreasing when  $\lambda < 1$ , so on the interval  $0.692 < \lambda < 1$  its maximum value occurs at  $\lambda = 0.692$  and works out to be 0.3449. Thus  $f(\lambda) < 0.7214 + 0.3449 < 1.070$  on this interval. Finally, we investigate  $f(\lambda)$  on the interval  $1/16 \leq \lambda \leq 0.692$ . Here we have the derivative

$$f'(\lambda) = \frac{-1}{(1-\lambda)^2} + \frac{1}{2 \ln 2} \left[ \frac{-\ln \lambda}{(1-\lambda)^2} + \frac{\lambda + (1-\lambda) \ln (1-\lambda)}{\lambda^2(1-\lambda)} \right]$$

so by inequality (12) we have

$$\begin{aligned} |f'(\lambda)| &\leq \frac{1}{(1-\lambda)^2} + \frac{1}{2 \ln 2} \left[ \frac{|\ln \lambda|}{(1-\lambda)^2} + \frac{1}{2(1-\lambda)^2} \right] \\ &= \frac{1}{(1-\lambda)^2} \left[ 1 + \frac{|\lg \lambda|}{2} + \frac{1}{4 \ln 2} \right] \\ &< \frac{1}{(0.308)^2} \left[ 1 + \frac{|\lg (1/16)|}{2} + 0.3607 \right] < 35.43 \end{aligned}$$

because  $1/16 < \lambda < 0.692$ . It then follows from the Mean Value Theorem for derivatives that

$$(13) \quad f(\lambda) \leq f(b) + (35.43)(\lambda - b) \quad \text{when } 1/16 \leq b \leq \lambda \leq 0.692.$$

Computer examination of values of  $f(b)$  at finitely many points  $b$  spaced 0.00005 apart on the interval  $[1/16, 0.692]$  yields a maximum value  $f(0.36080) = 1.06822$ , so by (13)

$$f(\lambda) \leq 1.06822 + (35.43)(0.00005) < 1.070. \quad \square$$

Here is a lower bound for search cost for any linear penalty function  $P(k) = \alpha k + \beta$  in which the constants  $\alpha$  and  $\beta$  are nonnegative.

**COROLLARY 5.** *For all nonnegative constants  $\alpha$  and  $\beta$ , and for every search tree  $T_n$  with  $n$  nodes ( $n \geq 0$ ) we have*

$$S_{\alpha k + \beta}(T_n) \geq \frac{\alpha}{2}(n+1)^2[\lg(n+1) - 1.070] + \frac{\alpha}{2}(n+1) + \beta[(n+1) \lg(n+1) - n].$$

*Proof.* It easily follows from (2) that

$$S_{\alpha k + \beta}(T_n) = \alpha S_k(T_n) + \beta S_1(T_n).$$

Apply Theorem 4 and Lemma 2.  $\square$

We now prove that when  $P(k)$  is linear, ordinary binary search gives near-optimal results.

**THEOREM 6.** Let  $B_n$  denote the (balanced) search tree corresponding to binary search of an ordered array of length  $n$ . Then for all nonnegative constants  $\alpha$  and  $\beta$  we have

$$(14) \quad S_{\alpha k + \beta}(B_n) \leq \frac{\alpha}{2}(n+1)^2[\lg(n+1) - 0.9138] + \frac{\alpha}{2}(n+1) \\ + \beta[(n+1)(p+1) - 2^{p+1} + 1],$$

where  $p = \lfloor \lg(n+1) \rfloor$ .

*Proof.* It suffices to prove (14) in the case where  $\alpha = 1$  and  $\beta = 0$ ; the more general case follows from Lemma 1 and the equation in the proof of Corollary 5. We use induction on  $n$ . Unfortunately, the induction step leads to a rather messy inequality that turns out to be difficult to verify when  $n$  is small, but fairly easy when  $n$  is large enough that certain terms can be discarded. So as the first step of the proof we verify that inequality (14) holds for all  $n \leq 19999$ . This is done by using a computer and the recursion formula

$$(15) \quad S_k(B_n) = \begin{cases} m(2m) + S_k(B_{m-1}) + S_k(B_m) + m \cdot S_1(B_m) & \text{if } n = 2m, \\ (m+1)(2m+1) + 2 \cdot S_k(B_m) + (m+1)S_1(B_m) & \text{if } n = 2m+1 \end{cases}$$

derived from (4), to generate exact values for  $S_k(B_n)$ . Equation (15) arises from the fact that when  $n = 2m$  the root of  $B_n$  is  $m$  and the left and right subtrees are of the form  $B_{m-1}$  and  $B_m$ , but when  $n = 2m+1$  then both have the form  $B_m$ .

Now suppose (14) holds for  $n = 0, 1, \dots, 2m-1$ , where  $m \geq 10,000$ . We first examine the case where  $n = 2m$ , and later the case where  $n = 2m+1$ .

*Case 1.  $n = 2m$ .* Using (15), induction on (14), and Lemma 1 we find that

$$S_k(B_n) \leq 2m^2 + \frac{1}{2}m^2[\lg m - 0.9138] + \frac{1}{2}m + \frac{1}{2}(m+1)^2[\lg(m+1) - 0.9138] \\ + \frac{1}{2}(m+1) + m[(m+1)(p+1) - 2^{p+1} + 1],$$

where  $p = \lfloor \lg(m+1) \rfloor$ . We seek to prove that this expression is

$$\leq \frac{1}{2}(2m+1)^2[\lg(2m+1) - 0.9138] + \frac{1}{2}(2m+1) \quad \text{when } m \geq 10,000.$$

Write  $\lambda = 2^p/(m+1)$ . Then  $\frac{1}{2} < \lambda \leq 1$ , and the inequality we must prove can be written (after considerable algebra of the sort familiar from the proof of Theorem 4) in the form

$$m^2[\lg m - \lg(m+1)] + 2(0.9138)m(m+1) \\ \leq (2m+1)^2[\lg(2m+1) - \lg(m+1)] + 2m(m+1)[2\lambda - \lg \lambda] - 2m(3m+2).$$

By elementary calculus the minimum value of  $2\lambda - \lg \lambda$  on the interval  $\frac{1}{2} \leq \lambda \leq 1$  is 1.91393 (at  $\lambda = 1/(2 \ln 2)$ ). Since  $\lg m - \lg(m+1)$  is negative, it then suffices to prove that when  $m \geq 10,000$  we have

$$0.9138 \leq \frac{1}{2} \left( 2 + \frac{1}{m} \right) \left( \frac{2m+1}{m+1} \right) \lg \left( \frac{2m+1}{m+1} \right) - \frac{3m+2}{m+1} + 1.91393.$$

Since  $2 + 1/m > 2$  and  $(2m+1)/(m+1) \geq 1.9999$  and  $\lg((2m+1)/(m+1)) \geq 0.9999$  and  $-(3m+2)/(m+1) > -3$ , the inequality is satisfied.

*Case 2.  $n = 2m+1$ .* Using (15), induction on (14), and Lemma 1 we find that

$$S_k(B_n) \leq (m+1)(2m+1) + (m+1)^2[\lg(m+1) - 0.9138] + (m+1) \\ + (m+1)[(m+1)(p+1) - 2^{p+1} + 1],$$

where  $p = \lfloor \lg(m+1) \rfloor$ . We seek to prove that this expression is

$$\leq \frac{1}{2}(2m+1+1)^2[\lg(2m+1+1) - 0.9138] + \frac{1}{2}(2m+1+1).$$



Write  $\lambda = 2^p/(m+1)$ . Again  $\frac{1}{2} < \lambda \leq 1$ , and the inequality we must prove reduces to

$$(16) \quad 0.9138 \leq 2\lambda - \lg \lambda - 1 - 1/(m+1) \quad \text{for } m \geq 10,000.$$

As in Case 1,  $2\lambda - \lg \lambda \geq 1.91393$ , and the minimum value of  $-1/(m+1)$  when  $m \geq 10,000$  is  $-0.0001$ , so the minimum value of the right side of (16) is at least  $1.91393 - 1 + 0.0001$ .  $\square$

Extensive computer calculation has shown that the value 0.9138 that appears in Theorem 6 for classical binary search trees  $B_n$  cannot be significantly improved. However, as we shall now prove, there is another class of balanced search trees for which it is possible to get a better value, i.e., a value closer to the 1.070 that appears in Theorem 4 and Corollary 5. This has the double advantage of giving us a better upper bound on the search cost in the elusive *optimal* search trees and also exhibiting a slightly better search algorithm than classical binary search. The search trees we want to investigate are formed in the following way: take any balanced tree with  $n$  nodes and height  $h$ . Move all the leaves  $h$  as far to the right as possible. This produces what we shall call the (unique) *right-heavy balanced tree* of  $n$  nodes, denoted by  $R_n$ .

**THEOREM 7.** *For all right-heavy balanced search trees  $R_n$  and all nonnegative constants  $\alpha$  and  $\beta$  we have the upper bound*

$$S_k(R_n) \leq \frac{\alpha}{2}(n+1)^2[\lg(n+1) - 0.975] + \frac{\alpha}{2}(n+1) + \beta[(n+1)(p+1) - 2^{p+1} + 1],$$

where  $p = \lfloor \lg(n+1) \rfloor$ .

*Proof.* As in the proof of Theorem 6 it suffices to consider the case  $\alpha = 1$  and  $\beta = 0$ . We use induction to prove that

$$(17) \quad S_k(R_n) \leq \frac{1}{2}(n+1)^2[\lg(n+1) - K] + \frac{1}{2}(n+1)$$

for a constant  $K$  to be determined. When  $n = 0$  the formula holds, provided  $K \leq 1$ . Now suppose (17) holds for  $n = 0, \dots, m-1$ . Consider the right-heavy balanced tree  $R_m$ . Let  $p = \lfloor \lg(m+1) \rfloor$ , from which it follows that

$$(18) \quad 2^p \leq m+1 < 2^{p+1}.$$

Think of  $R_m$  as the full binary tree  $F_p$  with  $2^p - 1$  nodes, to which  $m - 2^p + 1$  "surplus" nodes have been attached at level  $p$ , as far to the right as possible. By (18) the number of surplus nodes must be less than  $2^p$ . If the number of surplus nodes is less than half of  $2^p$ , then they will all be attached to the right subtree of the root, which will be the node  $2^{p-1}$ . The left subtree will be the full tree  $F_{p-1}$ . The right subtree will be identical to the tree obtained by building the unique right-heavy balanced tree  $R_{m-2^{p-1}}$  and then increasing each of its node labels by  $2^{p-1}$ . However, if the number of surplus nodes is at least half of  $2^p$ , then the surplus nodes will fill out a full row at the bottom of the right subtree, and any that remain will spill over onto the left subtree of the root, which in this case will be the node numbered  $m - (2^p - 1)$ . Then the left subtree will be the right-heavy balanced tree  $R_{m-2^p}$ . The right subtree will be identical to the tree obtained by building the full tree  $F_p$  and then increasing each of its node labels by  $m - 2^p + 1$ . The calculations are slightly different in these two cases, so we consider them separately.

*Case 1.* The surplus is less than half of  $2^p$ ; that is,

$$(19) \quad m - 2^p + 1 < 2^p/2.$$

By (18) and (19),  $\lfloor \lg(m - 2^{p-1}) \rfloor = p - 1$ . Using Lemmas 3 and 1, (4), induction on

(17), and the discussion preceding Case 1 we obtain

$$\begin{aligned} S_k(R_m) &= 2^{p-1}m + S_k(F_{p-1}) + S_k(R_{m-2^{p-1}}) + 2^{p-1} \cdot S_1(R_{m-2^{p-1}}) \\ &\leq 2^{p-1}m + 2(p-2)4^{p-2} + 2^{p-2} + \frac{1}{2}(m-2^{p-1}+1)^2[\lg(m-2^{p-1}+1) - K] \\ &\quad + \frac{1}{2}(m-2^{p-1}+1) + 2^{p-1}[(m-2^{p-1}+1)(p-1+1) - 2^p + 1]. \end{aligned}$$

We seek to prove that this expression is

$$\leq \frac{1}{2}(m+1)^2[\lg(m+1) - K] + \frac{1}{2}(m+1).$$

Write  $\lambda = 2^{p-1}/(m+1)$ . By (18) and (19),

$$2^p \leq m+1 < 2^p + 2^{p-1} = 3(2^{p-1}),$$

so  $\frac{1}{3} < \lambda \leq \frac{1}{2}$ . Substitution and reduction show that we must prove

$$(2\lambda - \lambda^2) \lg \lambda + (1 - \lambda)^2 \lg(1 - \lambda) + 4\lambda - 7\lambda^2 + K(2\lambda - \lambda^2) \leq 0.$$

Since  $2\lambda - \lambda^2 > 0$ , we see that this is equivalent to

$$K \leq \frac{7\lambda^2 - 4\lambda}{2\lambda - \lambda^2} - \lg \lambda - \frac{(1 - \lambda)^2}{2\lambda - \lambda^2} \lg(1 - \lambda).$$

Denote the right side by  $f(\lambda)$ . Then straightforward calculation, together with the triangle inequality, shows that  $|f'(\lambda)| < 15$  when  $\frac{1}{3} \leq \lambda \leq \frac{1}{2}$ . Computer examination of values of  $f(b)$  at finitely many points  $b$  spaced 0.0001 apart yields a minimum value  $f(0.4385) = 0.9768$ , so by the Mean Value Theorem for derivatives,  $f(\lambda) \geq 0.9768 - 15(0.0001) > 0.975$  when  $\frac{1}{3} \leq \lambda \leq \frac{1}{2}$ . Thus for Case 1 it suffices to make  $K \leq 0.975$ .

*Case 2.* The surplus is at least half of  $2^p$ ; that is,

$$(20) \quad m - 2^p + 1 \geq 2^p/2.$$

By (18) and (20),  $[\lg(m - 2^p + 1)] = p - 1$ . Using Lemmas 3 and 1, (4), induction over (17), and the discussion preceding Case 1 we obtain

$$\begin{aligned} S_k(R_m) &= (m - 2^p + 1)m + S_k(R_{m-2^p}) + S_k(F_p) + (m - 2^p + 1)S_1(F_p) \\ &\leq (m - 2^p + 1)m + \frac{1}{2}(m - 2^p + 1)^2[\lg(m - 2^p + 1) - K] + \frac{1}{2}(m - 2^p + 1) \\ &\quad + 2(p-1)4^{p-1} + 2^{p-1} \\ &\quad + (m - 2^p + 1)[(2^p - 1 + 1)(p + 1) - 2^{p+1} + 1]. \end{aligned}$$

We seek to prove that this expression is

$$\leq \frac{1}{2}(m+1)^2[\lg(m+1) - K] + \frac{1}{2}(m+1).$$

Write  $\lambda = 2^p/(m+1)$ . By (18) and (20) we have

$$2^p/2 + 2^p \leq m+1 < 2^{p+1},$$

so  $\frac{1}{2} < \lambda \leq \frac{2}{3}$ . Substitution and a lengthy reduction show that we must prove that

$$K \leq -\lg \lambda + \lg(1 - \lambda) - \frac{\lambda^2 - 4\lambda + 2 + \lg(1 - \lambda)}{2\lambda - \lambda^2}.$$

Denote the right side by  $f(\lambda)$ . A laborious but straightforward calculation shows that

$$f''(\lambda) = \frac{-3 - 2\lambda(1 - \lambda)^2}{\lambda^2(1 - \lambda)^2(2 - \lambda)^2 \ln 2} + \frac{[-4 - 2 \lg(1 - \lambda)][3(1 - \lambda)^2 + 1] + 4\lambda^3}{(2\lambda - \lambda^2)^3} < 0$$

when  $\frac{1}{2} < \lambda \leq \frac{2}{3}$ , so  $f$  is concave down on this interval. Since  $f(\frac{1}{2}) = 1$  and  $f(\frac{2}{3}) > 1$ , it follows that the minimum value of  $f$  is 1. Thus for Case 2 it suffices to make  $K \leq 1$ .  $\square$

The proof of Theorem 7 shows how we can write a program to carry out the search prescribed by a right-heavy balanced tree. Given an array of no's and yes's occupying locations  $i, i + 1, \dots, j$ , with all no's to the left of all yes's, do the following:

- (a) (Initialize.) Set  $L = i$  and  $U = j$ . (Throughout the algorithm the number of locations remaining to be searched is  $U - L + 1$ , corresponding to  $m$  in the proof.)
- (b) (Find probe location  $K$ .) If  $U < L$ , the search ends and the leftmost yes is in location  $L$  (of  $L = n + 1$ , there are no yes's); otherwise, calculate  $P = \lfloor \lg(U - L + 1) \rfloor$ ; if  $U - L + 2 < 3(2^{P-1})$ , set  $K = (L - 1) + 2^{P-1}$ , else set  $K = (L - 1) + (U - L + 1) - (2^P - 1)$ .
- (c) (Adjust search limits.) If the probe at  $K$  finds a no, set  $L = K + 1$  and return to step (b), else set  $U = K - 1$  and return to step (b).

The only difference between this algorithm and binary search is that in step (b), binary search does not calculate  $P$ , but simply sets  $K = \lfloor (L + U) / 2 \rfloor$ .

Theorems 6 and 7 suggest, but do not prove, that search using a right-heavy balanced tree is less costly in general than binary search (of course, the two schemes coincide when the length of the array has form  $n = 2^p - 1$ , which gives a full tree). Although the author has not been able to prove this conjecture, computer calculation of the exact values (using the recursion relations such as (15)) validates the conjecture up to array length  $n = 5,000$ . At best, right-heavy balanced trees appear to have a cost advantage of only about 2 percent on the average.

**4. Higher-order penalty functions.** In this section we state upper and lower bounds for the search cost in an optimal tree when the probe penalty function is given by a nonlinear polynomial. We begin with a lower bound for the simple case when  $P(k) = k^p$ .

**THEOREM 8.** *For each fixed positive integer  $p$  and for every search tree  $T_n$  with  $n$  nodes ( $n \geq 0$ ) we have*

$$(21) \quad S_{k^p}(T_n) \geq \frac{1}{p+1} (n+1)^{p+1} \lg(n+1) - (n+1)^{p+1}.$$

*Proof.* The proof is by induction on  $p$ . The case  $p = 1$  is covered by Theorem 4, which gives a somewhat larger lower bound. For higher values of  $p$  the proof begins along the lines of the proof of Theorem 4, using induction over subtrees. Formula (4) is invoked and the term  $P(k+r) = (k+r)^p$  is expanded by the Binomial Formula. The details involve arguments similar to but more complicated than those in the proof of Theorem 4, and can be found in [1].  $\square$

Next we prove that ordinary binary search is nearly optimal when  $P(k) = k^p$ .

**THEOREM 9.** *Let  $B_n$  denote the search tree corresponding to binary search in an ordered array of length  $n$ . Then for any fixed positive integer  $p$ ,*

$$(22) \quad S_{k^p}(B_n) \leq \frac{1}{p+1} (n+1)^{p+1} [\lg(n+1) - 0.9138] + \frac{1}{2} (n+1)^p.$$

*Proof.* The proof is by induction on  $p$ . Theorem 6 covers the case  $p = 1$ . For larger  $p$  we use formula (4) and the Binomial Formula to write

$$S_{k^p}(B_n) = \binom{n}{2}^p (n) + S_{k^p}(B_{n/2-1}) + \sum_{j=0}^p \binom{p}{j} \left(\frac{n}{2}\right)^{p-j} S_{k^p}(B_{n/2})$$

when  $n$  is even, and a similar formula when  $n$  is odd (cf. formula (15)). Now we use induction over subtrees. The details, which are lengthy, can be found in [1].  $\square$

Theorems 8 and 9 deal with penalty functions of the simple form  $P(k) = k^p$ . For more general penalty functions of the form  $P(k) = \alpha_p k^p + \alpha_{p-1} k^{p-1} + \dots + \alpha_1 k + \alpha_0$ , formula (2) gives

$$S_{P(k)}(T) = \alpha_p S_{k^p}(T) + \alpha_{p-1} S_{k^{p-1}}(T) + \dots + \alpha_1 S_k(T) + \alpha_0 S_1(T),$$

to which we can apply Theorems 8 and 9 and Lemmas 1 and 2.

**5. Computation of exact optimal trees and costs.** It is possible to compute recursively the optimal trees and their search costs for any prescribed penalty function  $P(k)$  whatsoever. Assume that the array to be searched is indexed from 1 to  $n$ . Introduce a "displacement" parameter into (2) as follows:

$$S_{P(k)}(T_n, d) = \sum_{k=1}^n P(k+d) W_k(T_n),$$

which gives the search for a search tree of  $n$  nodes indexed from  $1+d$  to  $n+d$ . Then (4) takes the form

$$(23) \quad S_{P(k)}(T_n, d) = P(r+d) \cdot n + S_{P(k)}(T_{r-1}, d) + S_{P(k)}(T_{n-r}, r+d).$$

Write  $M_{P(k)}(n, d) = \min_{T_n} S_{P(k)}(T_n, d)$ , where for a fixed pair  $(n, d)$  the minimum is taken over all search trees  $T_n$  with  $n$  nodes. By the fundamental idea of dynamic programming, (23) gives

$$(24) \quad M_{P(k)}(n, d) = \min_{1 \leq r \leq n} \{P(r+d) \cdot n + M_{P(k)}(r-1, d) + M_{P(k)}(n-r, r+d)\}.$$

This recursion relation, together with the base equation  $M_{P(k)}(0, d) = 0$  for all  $d$ , allows us to calculate, for any pair  $(n, d)$ , the roots and search cost of all corresponding

TABLE 1

$n$	Optimal cost	roots	$n$	Optimal cost	roots	$n$	Optimal cost	roots
0	0	0	29	1,763	14	125	47,451	62
1	1	1	30	1,909	15	126	48,327	63
2	4	1	31	2,063	9, 10	127	49,198	37
3	10	1, 2	32	2,216	10	128	50,057	37, 38
4	19	1, 2	33	2,376	7, 10, 11	—	—	—
5	31	2	34	2,542	7...12	140	61,010	46
6	47	3	35	2,713	9, 10	141	61,978	34...40
7	68	1, 2, 3, 4	36	2,890	10	142	62,948	36, 37
8	92	2, 3	—	—	—	143	63,926	37
9	120	3	61	9,527	30	—	—	—
10	153	3, 4	62	9,894	31	253	225,439	126
11	190	4	63	10,268	18, 19, 20	254	227,464	127
12	232	5	64	10,635	19, 20	255	229,432	74
13	279	6	65	11,009	20	256	231,401	74, 75
14	332	7	66	11,391	20, 21	257	233,379	75, 76
15	392	1...8	67	11,780	21, 22	—	—	—
16	454	3, 4, 5, 6	68	12,176	22	282	285,877	92, 93
17	521	4, 5, 6	69	12,581	15, 22, 23	283	288,094	69, 77, 78
18	593	5, 6	70	12,992	17...20	—	—	—
19	670	6	71	13,408	18, 19, 20	509	1,039,651	254
20	753	6, 7	72	13,831	19, 20	510	1,044,110	147, 148
—	—	—	—	—	—	511	1,048,544	148

optimal search trees. The entire trees can be constructed, provided we store for each  $(n, d)$  not only  $M_{P(k)}(n, d)$  but also the roots of all trees with this minimal cost. Note, however, that to calculate  $M_{P(k)}(n_0, d_0)$  by this method, we must calculate and store information for all  $(n, d)$  satisfying  $0 \leq n \leq n_0$  and  $0 \leq d \leq d_0 + n_0 - n$ . These enormous storage requirements can be dramatically reduced in some situations (e.g.,  $P(k) = k$ ) by storing, for fixed  $n$ , a linked list of cost-root pairs that are optimal over a wide range of  $d$  values. For more details, see [1].

Table 1 gives optimal search costs and roots of all the optimal trees for selected values of  $n$  when  $P(k) = k$ . These values were calculated using the recursive method outlined above.

**Acknowledgment.** The author gratefully acknowledges the guidance of Ed Reingold, who provided the author with the problem discussed in this paper.

#### REFERENCES

- [1] W. J. KNIGHT, *Search in an ordered table having variable probe cost*, Masters thesis, University of Illinois, Urbana, IL, 1986 (available from the Computer Science Library, Digital Computing Laboratory, Univ. of Illinois, Urbana, IL 61801).
- [2] E. M. REINGOLD AND W. J. HANSEN, *Data Structures*, Little Brown, Boston, 1983.
- [3] K. STEIGLITZ AND T. W. PARKS, *What Is the Filter-Design Problem?*, Proc. 1986 Princeton Conference on Information Sciences and Systems, B. W. Dickenson, ed., Princeton University Dept. of Electrical Engineering, Princeton, NJ, 1986.

## THE DECOMPOSITION OF A RECTANGLE INTO RECTANGLES OF MINIMAL PERIMETER\*

T. Y. KONG<sup>†</sup>, DAVID M. MOUNT<sup>‡</sup>, AND A. W. ROSCOE<sup>§</sup>

**Abstract.** We solve the problem of decomposing a rectangle  $R$  into  $p$  rectangles of equal area so that the maximum rectangle perimeter is as small as possible. This work has applications in areas such as flexible object packing and data allocation. Our solution requires only a constant number of arithmetic operations and integer square roots to characterize the decomposition, and linear time to print the decomposition. The discrete analogue of the problem in which the rectangle  $R$  is replaced by a rectangular array of lattice points is also considered, and three heuristic methods of solution are given. All of the heuristic methods operate by finding a discrete approximation to our optimal decomposition of  $R$ , but with different tradeoffs between the accuracy of the approximation and running time.

**Key words.** rectangle decomposition, flexible packing, digitization

**AMS(MOS) subject classifications.** 52A45, 68Q25, 68U05

**1. Introduction.** A fundamental problem in geometrical and combinatorial computing is how to decompose a large object into smaller objects subject to various constraints. By a *decomposition* of a region  $R$  we mean a finite set of closed regions whose union is  $R$  and whose interiors are pairwise disjoint. The regions of the decomposition need not be connected. Decomposition problems generally fall into one of two classes. In the first class, the objects have fixed dimensions, as in the knapsack and bin-packing problems [8], [11], [12]. In the second class the objects satisfy certain properties, as in decompositions of simple polygons into polygons that are star-shaped [2], convex [5], [17], triangular [7], or involve rectangles and rectilinear polygons [6], [9]. We consider a middle ground between these two classes in which the objects are of some specified area (or volume), but their shapes are not fully specified. In other words, the objects are flexible. The objective is to produce a decomposition in which the objects are not severely stretched; that is, they are nearly circular or square. More specifically, we consider the following problem involving the decomposition of a rectangle into rectangles of equal area or measure.

**RECTANGULAR DECOMPOSITION.** Given a rectangle  $R$  of height  $A$  and width  $B$ , and given an integer  $p$ , decompose  $R$  into  $p$  rectangles of equal area in such a way that the maximum rectangle perimeter is minimized.

Intuitively, the problem is to make the  $p$  rectangles in the decomposition of  $R$  as close to squares as we can. A special case of this problem in which  $R$  is a square was solved in [15]. The solution to the rectangular decomposition problem is a straightforward generalization of decomposition presented there, but the proof of optimality in the rectangular case is significantly more complex. A related problem with unconstrained areas was considered for the square in [1].

If we remove the restriction that  $R$  be decomposed into *rectangles* then we obtain another interesting problem. Define the *projection-perimeter* of a measurable plane set to be twice the sum of the lengths (measures) of its projections on the coordinate axes.

---

\* Received by the editors April 7, 1986; accepted for publication (in revised form) February 10, 1988.

<sup>†</sup> Department of Computer Science, Ohio University, Athens, Ohio 45701. This author gratefully acknowledges the support of the Air Force Office of Scientific Research under contract F-49620-85-K-0009.

<sup>‡</sup> Department of Computer Science, University of Maryland, College Park, Maryland 20742.

<sup>§</sup> Oxford University Computing Laboratory, Oxford, United Kingdom OX1 3QD.

**GENERAL DECOMPOSITION.** Given a rectangle  $R$  on the Cartesian plane with sides parallel to the coordinate axes of height  $A$  and width  $B$ , and given a positive integer  $p$ , decompose  $R$  into  $p$  regions of equal measure in such a way that the maximum projection-perimeter is minimized.

Here the terms *height* and *width* denote the length of the projection onto the  $y$ - and the  $x$ -axis, respectively.

In this paper we solve the rectangular decomposition problem. We also show that for certain values of  $A$ ,  $B$ , and  $p$ , the optimal rectangular decomposition is, in fact, a solution to the general decomposition problem. For all values of  $A$ ,  $B$ , and  $p$ , we prove that the optimal rectangular decomposition is a solution to the following problem, which can be thought of as a compromise between the rectangular and general decomposition problems. Define a *pseudorectangle* to be any set that is congruent to a Cartesian product  $P \times Q$ , where  $P$  and  $Q$  are measurable subsets of the real line. (In particular, every rectangle is also a pseudorectangle.)

**PSEUDORECTANGULAR DECOMPOSITION.** Given a rectangle  $R$  on the Cartesian plane with sides parallel to the coordinate axes, of height  $A$  and width  $B$ , and given a positive integer  $p$ , decompose  $R$  into  $p$  pseudorectangles of equal measure in such a way that the maximum projection-perimeter is minimized.

We consider a model of computation in which unit charge is assessed for  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\leq$ ,  $\lfloor \rfloor$  and integer square root on rational numbers representable using  $O(\log(p + A + B))$  bits of precision. In this model of computation our optimal rectangular decomposition can be characterized in constant time and output in  $O(p)$  time.

These decomposition problems and their higher-dimensional counterparts have a number of applications:

- *Flexible object packaging.* There are  $p$  sacks of fluid that are to be placed in a box of volume  $A$ , and each sack is to fit into a rectangular box of volume  $A/p$ . The dimensions of the partitions may vary, but to minimize the stress on each sack, it is desirable to make the boxes as nearly cubical as possible.

- *Circuit decomposition.* Given a large circuit, laid out on an  $A \times B$  board (for example, a mesh of computer processors), decompose the circuit by slicing the board into  $p$  rectangles of equal area. To minimize the interboard communications, the perimeter of each board should be as small as possible.

- *Flexible circuit layout.* In VLSI layout, a designer wants to place  $p$  functionally identical circuits on a rectangular chip of area  $A$ . Each circuit can be deformed into an arbitrary rectangle, as long as the area is equal to  $A/p$ . However, highly eccentric rectangles lead to long wire lengths. It is desirable to reduce the length of the longest side of each rectangle, which implies that its perimeter is minimized.

An analogue of the general decomposition problem can be posed on a rectangular array of lattice points.

**LATTICE DECOMPOSITION.** Given positive integers  $A$ ,  $B$ , and  $p$ , partition an  $A \times B$  array of lattice points into  $p$  subsets each containing at most  $\lceil AB/p \rceil$  points, in such a way that the maximum projection-perimeter of a subset is minimized.

This arises in the following data-allocation problem for parallel computation of tables.

- *Data allocation for parallel computers.* We wish to compute all of the values of a binary function  $f$  on the Cartesian product  $S \times T$ , where  $|S| = A$  and  $|T| = B$ . The computation is to be performed in parallel on  $p$  identical processing units. The function values are computed as follows. The  $i$ th processor computes the values of  $f$  on some subset  $W_i$  of  $S \times T$ . The sets  $W_i$ ,  $1 \leq i \leq p$ , form a partition of  $S \times T$ . To minimize computation time, each processor is assigned at most  $\lceil AB/p \rceil$  function values to

compute. Each processor has a small amount of local memory used for storing its operands. The objective is to minimize this storage. The amount of storage used by the  $i$ th processor is equal to the number of operands needed to compute the values of  $W_i$ , which equals one half of the projection-perimeter of  $W_i$ .

The data allocation problem was in fact the initial motivation for this work [13], [15]. Although we will not present an exact solution to the lattice decomposition problem, we will give heuristic methods based on the idea of approximating our solution to the pseudorectangular decomposition problem on the discrete lattice. The quality of these approximate solutions will be good when  $A$  and  $B$  are large relative to  $p$ .

This paper is organized as follows. In § 2 we solve the rectangular decomposition problem and prove that the optimal rectangular decompositions are also optimal solutions to the pseudorectangular decomposition problem. This work is based on two interesting combinatorial lemmas that give lower bounds on the amount of stretching and compressing that must occur when  $p$  rectangles of equal area are packed into an  $A \times B$  rectangle. In § 3 we describe three procedures for approximating, or *digitizing*, the geometric decomposition described in § 2 on an integer lattice of height  $A$  and width  $B$ . These digitization procedures provide different tradeoffs between desirable characteristics of the digitization and running time. Ideally, the digitization should respect the geometric solution and preserve areas as closely as possible. We say that a digitization is *equitable* if it has the property that the area of each digitized region is at most the ceiling of the area of the original region. We give an algorithm for computing an equitable digitization by reduction to the problem of finding a feasible flow in a graph. This algorithm runs in time polynomial in  $A+B+p$ . We give two efficient algorithms that produce approximations to an equitable digitization.

**2. Optimal decomposition of a rectangle.** Let  $R$  be a rectangle in the Cartesian plane, with sides parallel to the coordinate axes, of height  $A$  and width  $B$ . Let  $p$  be any positive integer. In this section we solve the problem of how to decompose  $R$  into  $p$  rectangles  $R_1, R_2, \dots, R_p$  of equal area in such a way that the maximum of the perimeters of the  $R_i$ 's is as small as possible. Our solution actually minimizes the maximum projection-perimeter over all decompositions of  $R$  into pseudorectangles of equal measure.

If the rectangle  $R$  is sufficiently thin relative to  $p$ , in particular if  $p \leq \max(A/B, B/A)$ , then it is easy to see that the optimal decomposition results by simply partitioning the longer side of  $R$  into  $p$  equal parts. So we shall henceforth assume that  $p > \max(A/B, B/A)$ .

From now on, whenever we refer to a pseudorectangle we will assume that it is so oriented that its sides are parallel to the coordinate axes. Unless otherwise stated, the term *projection* will mean a projection on one of the coordinate axes. The two projections of a pseudorectangle are generally not connected sets. If a projection is a finite union of disjoint open and closed intervals then by the *length* of the projection we mean the sum of the lengths of those intervals. More generally the length of a projection is taken to mean its (Lebesgue) measure. Analogously, when we refer to the *area* of an arbitrary, measurable plane set, we mean its measure. When we speak of the *perimeter* of a pseudorectangle we mean its projection-perimeter, which is twice the sum of the lengths of its projections.

The sum of the lengths of the projections of a pseudorectangle of given area  $q$  (in our case  $q = AB/p$ ) is a strictly increasing function of the length of the longer of the two projections. (For if the longer projection has length  $(\sqrt{q} + h)$  where  $h \geq 0$  then



the sum of projections is  $(\sqrt{q+h}) + q/(\sqrt{q+h})$ , which is easily shown to be a strictly increasing function of  $h$  when  $h \geq 0$ .) So if we define the *cost* of a pseudorectangle to be the length of its longer projection, and the *cost* of a decomposition of  $R$  into  $p$  pseudorectangles to be the cost of the most costly pseudorectangle in the decomposition, then our optimization problems are equivalent to the problems of finding minimal cost decompositions of  $R$  into  $p$  rectangles and into  $p$  pseudorectangles of equal area.

Define a *row(column)* of rectangles to be a set of rectangles whose sides are parallel to the coordinate axes, all of which have the same projection onto the  $y$ -axis ( $x$ -axis). For any integer  $n$  such that  $1 \leq n \leq p$ , define an  $n$ -row decomposition ( $n$ -column decomposition) of  $R$  to be a decomposition consisting of just  $n$  rows (columns) of rectangles, each of which contains either  $\lfloor p/n \rfloor$  congruent rectangles or  $\lceil p/n \rceil$  congruent rectangles. Figure 1 shows a 4-row decomposition in the case  $p = 23$ , and a 5-column decomposition in the case  $p = 27$ .

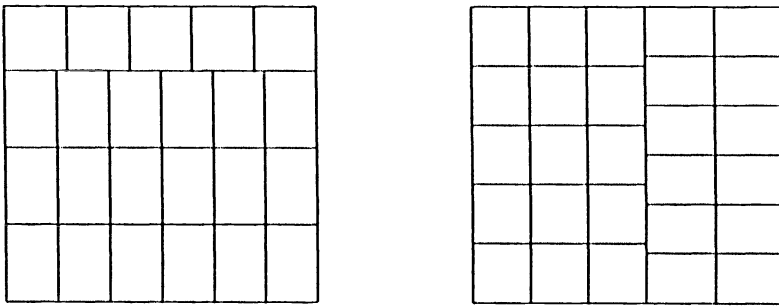


FIG. 1. A 4-row decomposition and a 5-column decomposition.

If  $p/n$  is an integer then the  $n$ -row decomposition is an  $n$ -by- $(p/n)$  array of congruent rectangles with sides  $A/n$  and  $Bn/p$ . If  $p/n$  is not an integer then an  $n$ -row decomposition has  $p - n \lfloor p/n \rfloor$  rows of rectangles, each of which contains exactly  $\lceil p/n \rceil$  congruent rectangles, and  $n \lceil p/n \rceil - p$  rows of rectangles, each of which contains exactly  $\lfloor p/n \rfloor$  congruent rectangles. If we regard two  $n$ -row decompositions that are related by a permutation of rows to be the same, then for each  $1 \leq n \leq p$  there is just one  $n$ -row decomposition of  $R$ . Thus, we may refer to “the”  $n$ -row decomposition of  $R$ . The cost of an  $n$ -row decomposition is the maximum of the side lengths:  $B/\lfloor p/n \rfloor$ ,  $A\lfloor p/n \rfloor/p$ ,  $B/\lceil p/n \rceil$ ,  $A\lceil p/n \rceil/p$ . Clearly the maximum will be either the first or last of these values. Analogously the cost of an  $n$ -column decomposition is  $\max(A/\lfloor p/n \rfloor, B\lceil p/n \rceil/p)$ .

Intuitively, a decomposition into pseudorectangles of equal area has minimum cost when the pseudorectangles have horizontal and vertical projections that are nearly equal. Ideally, the rectangle would be divided exactly into squares with side lengths  $\sqrt{AB/p}$ , i.e., into  $\sqrt{Ap/B}$  rows and  $\sqrt{Bp/A}$  columns. This is possible only when these square roots are integers. However, we will show that one of four decompositions based on the floors and ceilings of these square roots must be a minimal-cost decomposition.

These four decompositions are the  $h_1$ -row,  $h_2$ -row,  $k_1$ -column, and  $k_2$ -column decompositions of  $R$ , where  $h_1 = \lfloor \sqrt{Ap/B} \rfloor$ ,  $h_2 = \lceil \sqrt{Ap/B} \rceil$ ,  $k_1 = \lfloor \sqrt{Bp/A} \rfloor$ , and  $k_2 = \lceil \sqrt{Bp/A} \rceil$ . (The four quantities  $h_1, h_2, k_1, k_2$  all lie between 1 and  $p$ , because we are assuming that  $p > \max(A/B, B/A)$ .)

From now on we shall use the term *principal decomposition* to denote an  $h_1$ -row,  $h_2$ -row,  $k_1$ -column, or  $k_2$ -column decomposition of  $R$ . The main objective of this section is to prove the following theorem.

**THEOREM 2.1.** *At least one of the four principal decompositions of  $R$  is an optimal decomposition of  $R$  into pseudorectangles of equal area.*

This theorem provides us with a simple algorithm for finding an optimal decomposition. The algorithm performs  $O(1)$  arithmetic operations. Once it is determined which decomposition is to be used, it is an easy matter to output the boundaries of the rectangles in  $O(p)$  time.

*Outline of the Proof of Theorem 2.1.* The key results on which the proof is based are Lemmas 2.3 and 2.4. Each implies a good lower bound on the cost of any decomposition of  $R$  into pseudorectangles. The lower bounds are explicitly stated in Lemma 2.5.

Plainly, any decomposition of  $R$  that attains one of these two lower bounds must be optimal. This simple observation is used to derive a variety of sufficient conditions on  $A$ ,  $B$ , and  $p$  for one of the four principal decompositions to be an optimal decomposition (Lemmas 2.8, 2.9, and 2.10). We establish Theorem 2.1 by verifying that whatever the values of  $A$ ,  $B$ , and  $p$  are, at least one of these sufficient conditions is sure to be satisfied.  $\square$

We begin by dealing with a trivial special case.

**LEMMA 2.1.** *If  $h_1 = h_2$  and  $k_1 = k_2$  then the four principal decompositions are the same, and this decomposition is optimal.*

*Proof.* If  $h_1 = h_2 = h$  and  $k_1 = k_2 = k$  then  $h = \sqrt{Ap/B}$  and  $k = \sqrt{Bp/A}$ , so  $hk = p$ , and  $A/h = \sqrt{AB/p} = B/k$ . Thus, all four of the decompositions yield an  $h$ -by- $k$  array of equal squares. It is clear from our definition of optimality that this decomposition is optimal.  $\square$

Our next goal is to derive the lower bounds on the cost of decompositions of  $R$ . We state these bounds in Lemma 2.5. One of our two bounds follows from a well-known result that is usually attributed to Chebyshev:

**LEMMA 2.2 (Chebyshev).** *If  $x_1 \leq \dots \leq x_n$  and  $y_1 \geq \dots \geq y_n$ , then the arithmetic mean of the sequence  $x_1 y_1, \dots, x_n y_n$  does not exceed the product of the arithmetic mean of the  $x_i$  and the arithmetic mean of the  $y_i$ .*

*Proof.* For all  $i > j$  the product  $(x_i - x_j)(y_i - y_j)$  is nonpositive, so the sum of all such products is nonpositive. But this sum is precisely  $n^2(W - UV)$ , where  $U$  is the mean of the  $x_i$ ,  $V$  is the mean of the  $y_i$ , and  $W$  is the mean of the  $x_i y_i$ .  $\square$

The next two lemmas imply the lower bounds we seek.

**LEMMA 2.3.** *Let  $h$  and  $k$  be positive integers such that  $(h - 1)(k - 1) < p$ . Let  $\{R_i \mid 1 \leq i \leq p\}$  be a collection of pseudorectangles each of area  $AB/p$  contained in  $R$  whose interiors are pairwise disjoint. Then there is some pseudorectangle  $R_i$  whose shortest projection has length at most  $\max(A/h, B/k)$ .*

*Proof.* Let  $a_i$  be the length of the projection of  $R_i$  on the  $y$ -axis, and let  $b_i$  be the length of the projection of  $R_i$  on the  $x$ -axis.

If some vertical line meets at least  $h$  of the  $R_i$  then there is  $i$  such that  $a_i \leq A/h$ . If some horizontal line meets at least  $k$  of the  $R_i$  then there is  $i$  such that  $b_i \leq B/k$ . In either case we are done.

Suppose towards a contradiction, that all horizontal lines meet at most  $k - 1$  of the  $R_i$ , and all vertical lines meet at most  $h - 1$  of the  $R_i$ . This implies that the sum of the  $a_i$  is at most  $A(k - 1)$  and the sum of the  $b_i$  is at most  $B(h - 1)$ . Therefore the product of the mean of the  $a_i$  and the mean of the  $b_i$  is at most  $AB(h - 1)(k - 1)/(p^2)$ , which in turn is less than  $AB/p$  by the hypotheses of the lemma.

Without loss of generality, assume that the  $a_i$  are arranged in ascending order. Since for each  $i$ ,  $a_i b_i = AB/p$ , the  $b_i$  are in descending order. It follows, by Lemma 2.2, that the mean of the product  $a_i b_i$  is at most the product of the means, which we showed to be less than  $AB/p$ . However, since  $a_i b_i = AB/p$ , the mean of the products is equal to  $AB/p$ , a contradiction.  $\square$

LEMMA 2.4. *Let  $h$  and  $k$  be positive integers such that  $p < (h+1)(k+1)$ . Let  $\{R_i \mid 1 \leq i \leq p\}$  be a collection of (possibly disconnected) regions of area  $AB/p$  whose union is  $R$ . Then there is some region  $R_i$  whose longest projection has length at least  $\min(A/h, B/k)$ .*

*Proof.* Let  $a_i$  be the length of the projection of  $R_i$  on the  $y$ -axis, and let  $b_i$  be the length of the projection of  $R_i$  on the  $x$ -axis.

If some vertical line meets at most  $h$  of the  $R_i$ , then there is  $i$  such that  $a_i \geq A/h$ . If some horizontal line meets at most  $k$  of the  $R_i$ , then there is  $i$  such that  $b_i \geq B/k$ . In either case we are done. Thus, we may assume that each vertical line passing through  $R$  meets at least  $h+1$  of the  $R_i$ , and each horizontal line passing through  $R$  meets at least  $k+1$  of the  $R_i$ .

First, we show that there is some  $i$  such that

$$(1) \quad a_i + b_i \geq \min(A/h + Bh/p, B/k + Ak/p).$$

Our assumption implies that the sum of the  $a_i$  is at least  $A(k+1)$ , and that the sum of the  $b_i$  is at least  $B(h+1)$ . Thus there is some  $i$  such that  $a_i + b_i \geq (A(k+1) + B(h+1))/p$ .

Next, we show that  $(A(k+1) + B(h+1))/p \geq \min(A/h + Bh/p, B/k + Ak/p)$ . Suppose not. Then we derive a contradiction as follows. Since  $p < (h+1)(k+1)$  we have

$$(2) \quad p - h(k+1) \leq k,$$

$$(3) \quad p - k(h+1) \leq h.$$

It follows from  $(A(k+1) + B(h+1))/p < A/h + Bh/p$  and (2) (by routine manipulations) that

$$Bh < A(p - h(k+1)) \leq Ak.$$

Symmetrically, from  $(A(k+1) + B(h+1))/p < B/k + Ak/p$  and (3) we have

$$Ak < B(p - k(h+1)) \leq Bh,$$

giving the required contradiction. Thus (1) is proved. If  $a_i + b_i \geq A/h + Bh/p$ , then since  $a_i b_i = AB/p = (A/h)(Bh/p)$ , it follows that  $\max(a_i, b_i) \geq \max(A/h, Bh/p) \geq \min(A/h, B/k)$ , as claimed. By symmetry, the same is true if  $a_i + b_i \geq B/k + Ak/p$ .  $\square$

To state the lower bounds implied by the last two lemmas, define the following values where the variables  $h$  and  $k$  range over positive integers:

$$C = \min \{ \max(A/h, B/k) \mid (h-1)(k-1) < p \},$$

$$S = \max \{ \min(A/h, B/k) \mid p < (h+1)(k+1) \}.$$

The reason for the names  $C$  and  $S$  is that we found it helpful to visualize the “bad” pseudorectangles  $R_i$  and  $R_j$  in parts (i) and (ii) of the next lemma as a *compressed* and a *stretched* pseudosquare, respectively.

LEMMA 2.5. *In any decomposition of  $R$  into pseudorectangles  $R_1, \dots, R_p$  of equal area:*

(i) *There is an  $R_i$  whose shortest projection has length at most  $C$ ; hence the cost of the decomposition is at least  $AB/(pC)$ .*

(ii) *There is an  $R_j$  whose longest projection has length at least  $S$ ; hence the cost of the decomposition is at least  $S$ .*

*Proof.* Assertion (i) follows from Lemma 2.3 and assertion (ii) from Lemma 2.4.  $\square$

By Lemma 2.5(ii) any decomposition of  $R$  in which the longer projection of every pseudorectangle has length at most  $S$  is optimal. Also, since the length of the shorter projection of a pseudorectangle of a given area determines the length of the longer projection, it follows from Lemma 2.5(i) that any decomposition of  $R$  in which the shorter projection of every pseudorectangle has length at least  $C$  is optimal. (Thus Lemma 2.5 generalizes Propositions 1 and 2 in [15].)

The next step in the proof is to establish a variety of sufficient conditions on  $A, B$ , and  $p$  for one of the four principal decompositions to attain one of the lower bounds on cost stated in Lemma 2.5. As it turns out, these sufficient conditions are most conveniently stated in terms of the following four quantities:

$$\begin{aligned} h_0 &= \lfloor p/k_2 \rfloor, & k_0 &= \lfloor p/h_2 \rfloor, \\ h_3 &= \lceil p/k_1 \rceil, & k_3 &= \lceil p/h_1 \rceil. \end{aligned}$$

Before deriving the sufficient conditions, we prove two useful technical lemmas.

LEMMA 2.6. (i)  $h_0 \leq h_1 \leq h_2 \leq h_3$ ;

(ii)  $k_0 \leq k_1 \leq k_2 \leq k_3$ ;

(iii) *Either  $k_2 = k_3$  or  $h_0 = h_1$ ;*

(iv) *Either  $h_2 = h_3$  or  $k_0 = k_1$ .*

*Proof.* Plainly,  $p/k_2 = p/\lfloor \sqrt{Bp/A} \rfloor \leq p/\sqrt{Bp/A} = \sqrt{Ap/B}$ . Hence  $h_0 \leq h_1$ . Similarly,  $h_3 \geq h_2$ . So (i) holds, and, by symmetry, so does (ii). Next, observe that if  $h_1 k_2 \leq p$  then  $h_1$  is an integer such that  $h_1 \leq p/k_2$ , so  $h_1 \leq \lfloor p/k_2 \rfloor = h_0$ , whence (i) implies that  $h_1 = h_0$ . If on the other hand  $h_1 k_2 \geq p$  then by an analogous argument  $k_2 = k_3$ . So (iii) holds, and by symmetry so does (iv).  $\square$

LEMMA 2.7. (i) *If  $h_1 = h_2$  and  $k_1 < k_2$  then  $h_0 < h_1 = h_2 < h_3$ .*

(ii) *If  $k_1 = k_2$  and  $h_1 < h_2$  then  $k_0 < k_1 = k_2 < k_3$ .*

*Proof.* If  $h_1 = h_2$  and  $k_1 < k_2$  then  $\sqrt{Ap/B}$  is an integer but  $\sqrt{Bp/A}$  is not, so  $h_0 \leq p/\lfloor \sqrt{Bp/A} \rfloor < p/\sqrt{Bp/A} = \sqrt{Ap/B} = h_1$ , and similarly  $h_3 > h_2$ . This proves (i), and (ii) follows by symmetry.  $\square$

If all the rectangles in a principal decomposition are congruent, then by Lemma 2.5 that decomposition is optimal if the longest (shortest) side of the rectangles has length at most  $S$  (at least  $C$ ). This simple observation yields the following sufficient conditions for one of the principal decompositions to be optimal.

LEMMA 2.8. (i) *If  $k_0 = k_1$  and  $h_2 = h_3$  then the  $h_2$ -row and  $k_1$ -column decompositions are the same. If, in addition,  $A/h_2 \geq C$  or  $B/k_1 \leq S$  then this decomposition is optimal.*

(ii) *If  $h_0 = h_1$  and  $k_2 = k_3$  then the  $k_2$ -row and  $h_1$ -column decompositions are the same. If, in addition,  $B/k_2 \geq C$  or  $A/h_1 \leq S$  then this decomposition is optimal.*

*Proof.* Suppose  $k_0 = k_1$  and  $h_2 = h_3$ . The first hypothesis implies  $k_1 = \lfloor p/h_2 \rfloor \leq p/h_2$  and the second implies  $h_2 = \lceil p/k_1 \rceil \geq p/k_1$ . Hence  $k_1 h_2 = p$ , so the  $h_2$ -row and  $k_1$ -column decompositions are the same; the decomposition is an  $h_2$ -by- $k_1$  array of congruent rectangles with sides of length  $A/h_2$  and  $B/k_1$ . A side of length  $A/h_2$  is a shortest side, and a side of length  $B/k_1$  is a longest side, so if  $A/h_2 \geq C$  or  $B/k_1 \leq S$  then the decomposition is optimal by Lemma 2.5. This proves (i); (ii) follows by a symmetrical argument.  $\square$

A principal decomposition usually contains exactly two different kinds of rectangles (see Fig. 1). If it is clear that one kind of rectangle is costlier than the other, and that the longest (shortest) side of the costlier rectangles has length at most  $S$  (at

least  $C$ ), then by Lemma 2.5 the decomposition is optimal. The following lemma gives sufficient conditions for one of the principal decompositions to be optimal, based on this idea.

- LEMMA 2.9. (i) *If  $k_0 < k_1$  and  $B/k_0 \leq S$  then the  $h_2$ -row decomposition is optimal.*  
 (ii) *If  $k_3 > k_2$  and  $B/k_3 \geq C$  then the  $h_1$ -row decomposition is optimal.*  
 (iii) *If  $h_0 < h_1$  and  $A/h_0 \leq S$  then the  $k_2$ -column decomposition is optimal.*  
 (iv) *If  $h_3 > h_2$  and  $A/h_3 \geq C$  then the  $k_1$ -column decomposition is optimal.*

*Proof.* Suppose  $k_0 < k_1$  and  $B/k_0 \leq S$ . Now each rectangle in an  $h_2$ -row decomposition either has a side of length  $B/\lfloor p/h_2 \rfloor$  or has a side of length  $B/\lceil p/h_2 \rceil$ . But (since we are assuming  $k_0 < k_1$ )  $\lceil p/h_2 \rceil \leq k_0 + 1 \leq k_1 \leq \sqrt{Bp/A}$ , so a side of length  $B/\lceil p/h_2 \rceil$  is the longest side of a rectangle of area  $AB/p$ , and (a fortiori) the same is true of a side of length  $B/\lfloor p/h_2 \rfloor$ . As  $B/k_0 \leq S$ ,  $B/\lfloor p/h_2 \rfloor \leq S$ , and (a fortiori)  $B/\lceil p/h_2 \rceil \leq S$ . So the  $h_2$ -row decomposition is optimal by Lemma 2.5 (assertion (ii)). This proves part (i) of Lemma 2.9. Part (ii) is proved by an analogous argument by making substitutions that are order-inverting. That is,  $k_i$  is replaced by  $k_{3-i}$ ,  $h_i$  by  $h_{3-i}$ ,  $\leq$  by  $\geq$ ,  $\lfloor \cdot \rfloor$  by  $\lceil \cdot \rceil$ ,  $S$  by  $C$ , and so on. Parts (iii) and (iv) are symmetrical with (i) and (ii).  $\square$

If the longest side of one kind of rectangle in a principal decomposition has length at most  $S$ , while the shortest side of the other kind of rectangle has length at least  $C$ , then by Lemma 2.5 the decomposition is optimal. Hence we have the following sufficient conditions for optimality.

- LEMMA 2.10. (i) *If  $h_0 = h_1$ ,  $A/h_1 \leq S$ , and  $A/h_2 \geq C$ , then the  $k_2$ -column decomposition is optimal.*  
 (ii) *If  $h_3 = h_2$ ,  $A/h_1 \leq S$ , and  $A/h_2 \geq C$ , then the  $k_1$ -column decomposition is optimal.*  
 (iii) *If  $k_0 = k_1$ ,  $B/k_1 \leq S$ , and  $B/k_2 \geq C$ , then the  $h_2$ -row decomposition is optimal.*  
 (iv) *If  $k_3 = k_2$ ,  $B/k_1 \leq S$ , and  $B/k_2 \geq C$ , then the  $h_1$ -row decomposition is optimal.*

*Proof.* Suppose  $h_0 = h_1$ ,  $A/h_1 \leq S$ , and  $A/h_2 \geq C$ . Then  $\lfloor p/k_2 \rfloor = h_1$ . There are two cases: either  $\lceil p/k_2 \rceil = h_1$  or  $\lceil p/k_2 \rceil = h_1 + 1$ .

In the first case the  $k_2$ -column decomposition consists of  $k_2$  columns, each of which contains  $h_1$  congruent rectangles. Each of these rectangles has a side of length  $A/h_1$ , and (since by definition  $h_1 \leq \sqrt{Ap/B}$ ) this is the longest side of the rectangles. Hence  $A/h_1 \leq S$  implies that the decomposition is optimal (by assertion (ii) of Lemma 2.5).

In the second case we note that, by Lemma 2.7(i),  $h_0 = h_1$  implies that either  $h_1 < h_2$  or  $k_1 = k_2$ . So we may assume that  $h_1 < h_2$ , for if  $k_1 = k_2$  and  $h_1 = h_2$  then Lemma 2.10 is certainly true (by Lemma 2.1). By hypothesis  $\lceil p/k_2 \rceil = h_1 + 1$ , so  $\lceil p/k_2 \rceil = h_2$ . Now each rectangle in the  $k_2$ -column decomposition either has a side of length  $A/\lfloor p/k_2 \rfloor (= A/h_1)$  or has a side of length  $A/\lceil p/k_2 \rceil (= A/h_2)$ . Since  $h_1 \leq \sqrt{Ap/B} \leq h_2$ , a side of length  $A/h_1$  is a longest side, and a side of length  $A/h_2$  is a shortest side. Recalling that  $A/h_1 \leq S$  and  $A/h_2 \geq C$ , we see that Lemma 2.10 now follows from Lemma 2.5. This proves assertion (i); the other assertions follow by symmetrical arguments.  $\square$

Finally, we need to show that for all values of  $A$ ,  $B$ , and  $p$  at least one of the sufficient conditions for optimality holds. The proof is by case analysis, based on the following lemma.

- LEMMA 2.11. (i) *If  $h_1 < h_2$  then  $\min(A/h_1, B/k_0) \leq S$  and  $\max(A/h_2, B/k_3) \geq C$ .*  
 (ii) *If  $k_1 < k_2$  then  $\min(B/k_1, A/h_0) \leq S$  and  $\max(B/k_2, A/h_3) \geq C$ .*

*Proof.* Suppose  $h_1 < h_2$ . Then  $k_0 = \lfloor p/(h_1 + 1) \rfloor$  and  $k_3 = \lceil p/(h_2 - 1) \rceil$ . Hence,  $(h_1 + 1)(k_0 + 1) > p$  implying that  $\min(A/h_1, B/k_0) \leq S$ , by definition of  $S$ . Similarly,  $(h_2 - 1)(k_3 - 1) < p$ ; thus  $\max(A/h_2, B/k_3) \geq C$ . This proves (i); as usual, (ii) follows by a symmetrical argument.  $\square$

*Proof of Theorem 2.1.* If  $h_1 = h_2$  and  $k_1 = k_2$  then we are done by Lemma 2.1. Suppose  $h_1 = h_2$  and  $k_1 < k_2$ . Then by Lemma 2.7  $h_0 < h_1 = h_2 < h_3$ , and so, by Lemma 2.6(iii),  $k_2 = k_3$ . Hence, we deduce Theorem 2.1 by combining Lemma 2.11(ii) with Lemma 2.10(iv) and Lemma 2.9 ((iii) and (iv)). By symmetry, Theorem 2.1 holds if  $h_1 < h_2$  and  $k_1 = k_2$ .

Now suppose  $h_1 < h_2$  and  $k_1 < k_2$ . On applying Lemma 2.11(i) we see that there are four possibilities:

- (a)  $A/h_1 \leq S$  and  $B/k_3 \geq C$ .
- (b)  $B/k_0 \leq S$  and  $B/k_3 \geq C$ .
- (c)  $A/h_1 \leq S$  and  $A/h_2 \geq C$ .
- (d)  $B/k_0 \leq S$  and  $A/h_2 \geq C$ .

Case (a). Case (a) need not be considered separately as it is symmetrical with Case (d).

Case (b). If  $k_0 < k_1$  or  $k_3 > k_2$  then Theorem 2.1 follows from Lemma 2.9((i) and (ii)). Otherwise  $k_0 = k_1$  and  $k_3 = k_2$ , and Theorem 2.1 follows from Lemma 2.10((iii) or (iv)).

Case (c). If  $h_0 = h_1$  or  $h_2 = h_3$  then Theorem 2.1 follows from Lemma 2.10((i) and (ii)). Otherwise  $h_0 < h_1$  and  $h_2 < h_3$ , and, by Lemma 2.6(iii),  $k_2 = k_3$ . Now apply Lemma 2.11(ii) to get the following four subcases:

- (c1)  $B/k_1 \leq S$  and  $A/h_3 \geq C$ .
- (c2)  $A/h_0 \leq S$  and  $A/h_3 \geq C$ .
- (c3)  $B/k_1 \leq S$  and  $B/k_2 \geq C$ .
- (c4)  $A/h_0 \leq S$  and  $B/k_2 \geq C$ .

Since, in the present case,  $h_0 < h_1$  and  $h_2 < h_3$ , Theorem 2.1 follows from Lemma 2.9 ((iii) and (iv)) in Cases (c1), (c2), and (c4). Since, in the present case,  $k_2 = k_3$ , Theorem 2.1 follows from Lemma 2.10(iv) in case (c3).

Case (d). We again apply Lemma 2.11(ii) to get the same four subcases (c1)–(c4) now renamed as Subcases (d1)–(d4).

*Subcase (d1).* Recall that  $A/h_2 \geq C$  and  $B/k_0 \leq S$  (from (d)). Now if  $k_0 < k_1$  or  $h_2 < h_3$  then Theorem 2.1 follows from Lemma 2.9((i) and (iv)); if, on the other hand,  $k_0 = k_1$  and  $h_2 = h_3$  then Theorem 2.1 follows from Lemma 2.8(i).

*Subcase (d2).* Symmetric with Case (b) above.

*Subcase (d3).* Symmetric with Case (c) above.

*Subcase (d4).* We have  $B/k_2 \geq C$  and (from (d))  $B/k_0 \leq S$ . Hence, if  $k_0 < k_1$  then Theorem 2.1 follows from Lemma 2.9(i), while if  $k_0 = k_1$  then Theorem 2.1 follows from Lemma 2.10(iii).  $\square$

We have proved that at least one of four principal decompositions is an optimal decomposition of  $R$  into pseudorectangles. But we conjecture that an optimal decomposition of  $R$  into pseudorectangles is in fact an optimal decomposition of  $R$  into arbitrary sets of equal area. In other words, the conjecture is that Theorem 2.1 solves the general decomposition problem as well as the pseudorectangular decomposition problem. We end this section with a simple argument which shows that the conjecture is true if  $S \geq AB/(pC)$ .

Observe, first of all, that in the statement of Lemma 2.4 the  $R_i$ 's need not be pseudorectangles. So we see that a decomposition of  $R$  into arbitrary measurable sets of area  $AB/p$  must contain a set whose  $x$ - or  $y$ -projection has length at least  $S$ . As was explained in our outline of the proof, we established Theorem 2.1 by showing that at least one of the four principal decompositions attains one of the two lower bounds stated in Lemma 2.5. In other words we showed that (at least) one of the

principal decompositions either has cost  $S$  or has cost  $AB/(pC)$ . Now suppose  $S \cong AB/(pC)$ . Then by Lemma 2.5(ii) none of the principal decompositions of  $R$  can have cost  $AB/(pC)$ . Therefore one of the principal decompositions has cost  $S$ , so that the sum of the height and width of any rectangle in that decomposition is at most  $S + AB/(pS)$ . The fact that a rectangular decomposition of  $R$  has cost  $S$  also implies that  $S \cong \sqrt{AB/p}$ . But we have seen that a decomposition of  $R$  into  $p$  arbitrary sets of equal area must contain a set whose  $x$ - or  $y$ -projection has length at least  $S$ ; the sum of the  $x$ - and  $y$ -projections of that set must be at least  $S + AB/(pS)$ .

**3. Digitizing a rectangular decomposition.** As noted in the Introduction, the lattice decomposition problem, is a discrete analogue to the region decomposition problem. Recall that the problem is to partition an  $A \times B$  rectangular array of integer lattice points into  $p$  subsets each of size at most  $\lceil AB/p \rceil$  such that the maximum projection-perimeter is minimized. We do not know of an efficient solution to the lattice decomposition problem, but when  $A$  and  $B$  are large relative to  $p$  the results of the previous section can be used to find an approximate solution. The problem reduces to approximating the decomposition of an  $A \times B$  rectangle on the lattice, so that areas and projection-perimeters are very nearly preserved.

In this section we consider how to compute this approximation, which, to borrow a term from computer graphics and vision, we call *digitization*. We present three digitization algorithms that provide tradeoffs between running time and the quality of the digitization. The second digitization algorithm is quite general, and operates on any decomposition into convex polygons. The other digitizations are significantly more efficient, but operate on a special class of rectangular decompositions which we call *row-major* decompositions. Consider the  $A \times B$  rectangle,

$$R = \{(x, y) \mid 0 \leq x \leq B, 0 \leq y \leq A\}.$$

A decomposition of  $R$  into rectangles is called *row-major* if it is of the following form.

(1)  $R$  is partitioned into  $r$  rows by a set of horizontal line segments running from  $x = 0$  to  $x = B$ . Let  $0 = h_0 < h_1 < \dots < h_r = A$  be the  $y$ -values of these segments.

(2) Each row is further decomposed by vertical segments into some number of columns. For the row bounded by  $h_{i-1}$  and  $h_i$  let  $0 = v_{i,0} < v_{i,1} < \dots < v_{i,s_i} = B$  be the  $x$ -values of these vertical segments.

Note that the decomposition produced by the algorithm of § 2 is either row-major, or can be made so by transposing rows and columns. In this section, we assume that  $A$  and  $B$  are positive integers. We assume that the line segments defining the decomposition are described using rational numbers representable using  $O(\log(p + A + B))$  bits each.

Consider the  $A \times B$  rectangular array of integer *lattice points*  $L$  superimposed on the rectangle  $R$ . That is,  $L = \{(i, j) \mid 0 \leq i < A, 0 \leq j < B\}$ . For each  $(i, j) \in L$ , let  $S_{i,j}$  denote the open unit square consisting of the points  $(x, y)$  for  $i < x < i + 1$  and  $j < y < j + 1$ . These are called the *lattice squares*. The rectangle  $R$  consists of  $A$  horizontal rows and  $B$  vertical columns of squares. Our aim is to approximate a decomposition of  $R$  into  $p$  regions by a partition of the lattice squares into  $p$  subsets. Although digitization is common in applications from computer vision and graphics, the goals of our digitization are rather special. We seek a partition of squares satisfying the following criteria: (1) the number of lattice squares assigned to a given region of the decomposition is nearly equal to the area of the region, and (2) the projection-perimeters of a region and its corresponding subset of lattice squares are nearly equal. We formalize these criteria by defining two properties of digitizations that we seek to produce through our algorithms.

DEFINITION. (1) A digitization is *overlapping* if each region is assigned only lattice squares  $S_{i,j}$  that overlap the region.

(2) A digitization is *equitable* if the number of lattice squares assigned to a given region does not exceed the ceiling of the area of the region.

All three of the digitization algorithms presented here produce overlapping digitizations; however, the first and third algorithms do not necessarily produce equitable digitizations. Define the *absolute excess* of a digitization to be the maximum signed difference between the number of lattice squares assigned to a region and the true area of the region. (Note that the absolute excess may be negative.) An equitable digitization has an absolute excess less than 1. The *relative excess* is defined to be the maximum ratio of these two values. The first digitization algorithm produces a digitization with relative excess approaching unity as  $\min(A, B)$  approaches infinity. The second procedure produces an equitable digitization by reducing the digitization problem to the problem of finding a feasible flow in a graph. This algorithm is the least efficient of the three. The third algorithm, is a compromise between these two. Like the first algorithm, it is very efficient with respect to running time but produces a digitization that has an absolute excess less than 2 for all input parameters.

The amount of time required to compute the digitization can be measured as the amount of time required to describe the boundaries of the digitization as a sequence of line segments. Our first and third algorithms can print the boundaries in  $O(p)$  time, and hence are optimal with respect to this criterion. Our second algorithm runs in polynomial time in  $A + B + p$ , although we do not attempt to find the most efficient implementation.

Our first algorithm is a simple naive digitization based on point containment. Stated simply, the set of lattice squares assigned to a region are those whose lower left corners are contained in the region. If a lattice point  $(i, j)$  lies on the boundary between two or more regions, then the corresponding square is assigned arbitrarily to one of the regions that overlaps the square. This is easily seen to be an overlapping digitization. It is also easy to see that as  $A$  and  $B$  increase relative to  $p$ , then the relative excess of the digitization approaches 1.

Note that the digitization of a given rectangle can be computed in constant time. This digitization algorithm has the property of mapping rectangles to rectangular arrays of squares. The other two digitizations that we present do not have this property.

**3.1. Absolutely equitable digitization.** The second algorithm works by reducing the digitization problem to a graph flow problem. This algorithm can be applied to any decomposition of  $R$  into convex polygonal regions. Let  $R_1, R_2, \dots, R_p$  denote a decomposition of  $R$  into  $p$  polygonal regions.

We define a bipartite flow graph with upper and lower vertex capacities,  $G = (V, E, L, U)$ , with vertex set  $V$ , edge set  $E$ , and lower and upper capacity functions  $L$  and  $U$ .  $V$  consists of the following elements:

- *region vertices*  $r_1, r_2, \dots, r_p$ , one for each region of the decomposition, and
- *lattice vertices*  $s_{i,j}$ , one for each of the lattice squares,  $S_{i,j}$  for  $0 \leq i < A$  and  $0 \leq j < B$ .

The edge set,  $E$ , consists of the following pairs:

- an edge from each region vertex  $r_k$  to each lattice vertex  $s_{i,j}$  whenever the unit square  $S_{i,j}$  overlaps the region  $R_k$ ,

The vertex capacities are expressed as pairs,  $(L(v), U(v))$ , representing the lower and upper flow capacity for each vertex  $v$ . These capacities are:

- $(1, 1)$  for each lattice vertex, and
- $(\lfloor a \rfloor, \lceil a \rceil)$  for each region vertex  $r_k$ , where  $a$  is the area of region  $R_k$ .



For example, Fig. 2 shows a decomposition of  $R$  and the corresponding flow graph.

Let  $\Gamma(v)$  denote the neighbors of a vertex  $v$ . A *feasible flow* is an assignment  $f$  of nonnegative integers to each edge of  $E$ , such that, for each vertex  $v \in V$

$$L(v) \leq \sum_{w \in \Gamma(v)} f(v, w) \leq U(v).$$

We say that a digitization is *absolutely equitable* if the number of lattice squares assigned to each region is either the floor or ceiling of the area of the region. An absolutely equitable digitization is, in some sense, the most equitable digitization that we can hope to achieve. The connection between the digitization problem and the graph  $G$  is established in our next lemma, which follows immediately from our construction.

LEMMA 3.1. *There exists an overlapping, absolutely equitable digitization if and only if the graph  $G$  has a feasible flow.*

Although the relationship between the feasible flow problem and the problem of absolutely equitable digitization gives a method of computing the digitization, it is not obvious that such a digitization need exist. Our next result applies a generalization of Hall's well-known theorem on complete matchings in bipartite graphs to show that such a digitization exists for all decompositions.

THEOREM 3.1. *Given any decomposition of  $R$  into regions with disjoint interiors, an absolutely equitable digitization of the decomposition exists.*

*Proof.* We make a straightforward adaptation of an existing result from the study of feasible flows in graphs with lower- and upper-edge capacities [10, p. 81], [3, p. 88], which generalizes Hall's theorem on the existence of complete matchings in graphs [4]. This result states that a necessary and sufficient condition for the existence of nonnegative feasible flow in  $G$  is that for all subsets  $U$  of region vertices and all subsets,  $T$ , of lattice vertices we have

$$(4) \quad \sum_{v \in U} L(v) \leq \sum_{w \in \Gamma(U)} U(w), \quad \sum_{w \in T} L(w) \leq \sum_{v \in \Gamma(T)} U(v).$$

The first half of (4) follows by noting that the sum of areas of a set of disjoint regions  $U$  is no greater than the number of unit squares  $\Gamma(U)$  that cover the set. The second half follows by noting that the number of disjoint unit squares in a set  $T$  is no greater than the sum of areas of the regions  $\Gamma(T)$  covering  $T$ .  $\square$

COROLLARY. *If the regions of the decomposition have integer areas, then there exists an overlapping digitization with an absolute excess of zero.*

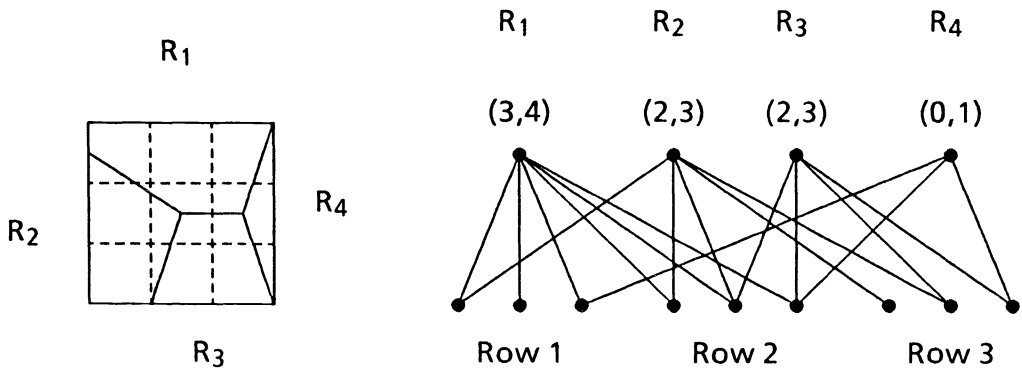


FIG. 2. The flow graph of a decomposition.

Once the graph  $G$  has been constructed, the equitable digitization can be computed in  $O(|V|^3)$  time by any known algorithm for finding feasible flows in graphs [3], [18]. The number of vertices  $V$  is  $p + AB$  and the magnitude of the capacities is at most  $AB$ .

In the case where  $p$  is small relative to  $A$  and  $B$  we might hope to find a bound on the size of the vertex set of  $G$  that is independent of  $A$  and  $B$ . To do this we reduce the number of lattice vertices as follows. The lattice vertices can be partitioned into equivalence classes according to the set of regions that they overlap, that is according to  $\Gamma$ . The individual lattice vertices forming each equivalence class can then be replaced by a single aggregate vertex representing the entire class. The capacity of this aggregate vertex is the sum of the capacities of the individual vertices. It can be shown that if  $R$  is decomposed into  $p$  convex polygonal regions then the number of equivalence classes is  $O(p^2)$  [14]. This reduction can be computed easily in  $O(pAB)$  time, and the digitization can be computed in  $O(p^6)$  time by a feasible flow algorithm.

**3.2. Nearly equitable digitization.** In the previous section we showed that absolutely equitable digitizations exist, although they are somewhat expensive to compute. Next we present an efficient digitization algorithm that does not provide us with an equitable digitization, but does achieve an absolute excess less than 2. Figure 3 shows the result of this digitization of a seven-region decomposition on a lattice of size  $13 \times 13$ . This digitization has the property that, given a lattice square, the rectangle to which it is assigned can be determined in  $O(1)$  time. The boundary of a digitized region can be described by a collection of line segments in  $O(1)$  time. This is possible because the algorithm works *locally*, digitizing each rectangle of the decomposition without knowledge of the digitization of any other rectangles, as opposed to the graph flow method which operates *globally*.

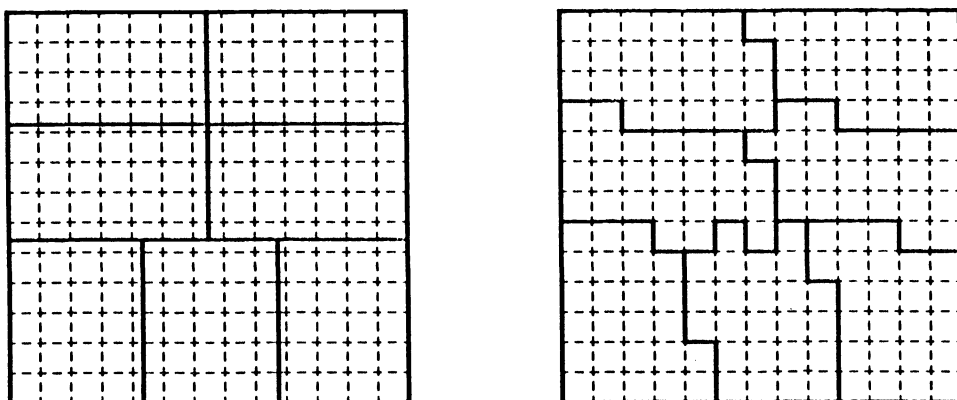


FIG. 3. A 7-region digitization on a  $13 \times 13$  lattice.

This algorithm makes extensive use of the fact that the decomposition is a row-major decomposition. The algorithm operates in two phases. First, for each adjacent pair of horizontal lines  $h_i$  and  $h_{i-1}$  the digitized region lying between  $h_i$  and  $h_{i-1}$  is determined. In the second phase, within the digitized region between  $h_i$  and  $h_{i-1}$ , we digitize the region lying between the vertical lines  $v_j$  and  $v_{j-1}$ . In our discussion, we will make use of the following easily verified identity. For all numbers  $s$  and  $t$ :

$$(5) \quad \lfloor s - t \rfloor \leq \lceil s \rceil - \lfloor t \rfloor \leq \lceil s - t \rceil.$$

For the first phase of the algorithm, we show how to digitize the region between the horizontal lines  $y = h_i$  and  $y = h_{i-1}$ ,  $1 \leq i \leq r$ . For each  $i$  we digitize the region lying below  $h_i$ , denoted  $H_i$ , and then define the digitized region between  $h_i$  and  $h_{i-1}$  to be the set difference  $H_i - H_{i-1}$ .  $H_i$  consists of all lattice squares lying strictly below  $h_i$ , that is,  $S_{x,y}$  where  $y < \lfloor h_i \rfloor$ , and some subset of the horizontal row of squares  $S_{x,y}$ , where  $y = \lfloor h_i \rfloor$ . Consider the set of decomposition vertices lying in this row of squares. For each such vertex  $(g, h)$ , construct the vertical lines  $x = \lfloor g \rfloor$  and  $x = \lceil g \rceil$ . Let  $G_i$  denote the set of  $x$ -intercepts of these lines, and let  $0 = g_{i,0} < g_{i,1} < \dots < g_{i,m} = B$ , the sorted elements of  $G_i$ . These lines partition this horizontal row of lattice squares into  $m$  contiguous blocks. See Fig. 4.

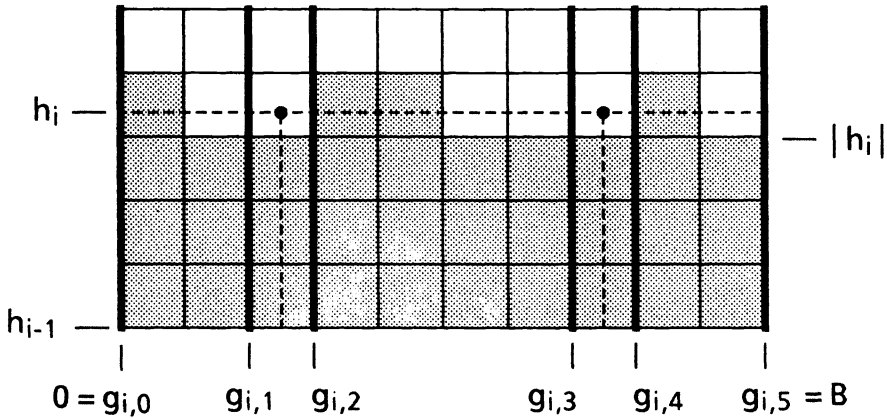


FIG. 4. Allocation below a horizontal line.

The portion of this row, bounded above by  $h_i$  and lying to the left of  $g_{i,j}$  has area  $(h_i - \lfloor h_i \rfloor)g_{i,j}$ . Let  $C_{i,j}$  denote the ceiling of this value.  $C_{i,j}$  is the desired total number of squares to allocate to the left of  $g_{i,j}$ . For the block bounded by  $g_{i,j-1}$  and  $g_{i,j}$ , we allocate the leftmost  $C_{i,j} - C_{i,j-1}$  lattice squares to  $H_i$ . For example, in Fig. 4,  $h_i - \lfloor h_i \rfloor = \frac{1}{3}$ ; hence the number of squares allocated to  $H_i$  lying between  $g_{i,2} = 3$  and  $g_{i,3} = 7$  is  $\lceil 7/3 \rceil - \lceil 3/3 \rceil = 2$ . The allocated squares are shaded in Fig. 4.

Using (5), we have

$$0 \leq C_{i,j} - C_{i,j-1} \leq \lceil (g_{i,j} - g_{i,j-1})(h_i - \lfloor h_i \rfloor) \rceil \leq g_{i,j} - g_{i,j-1}.$$

Hence, there are enough lattice squares between  $g_{i,j-1}$  and  $g_{i,j}$  to satisfy this allocation scheme. It is an easy consequence of our definition that  $H_{i-1} \subseteq H_i$ , for  $1 \leq i \leq r$ . Thus, the digitization of the row between  $h_i$  and  $h_{i-1}$  can be defined to be  $H_i - H_{i-1}$ . It is clear by our construction that each square of  $H_i - H_{i-1}$  overlaps the region between  $h_i$  and  $h_{i-1}$ . The next result states that the boundary of  $H_i$  can be computed in constant time within a fixed block of  $G_i$ .

CLAIM 3.1. For a lattice square  $S_{x,y}$ , where  $g_{i,j-1} \leq x < g_{i,j}$ , the membership of  $S_{x,y}$  in  $H_i$  can be tested in  $O(1)$  time. The boundary of  $H_i$  between  $g_{i,j-1}$  and  $g_{i,j}$  can be computed in  $O(1)$  time.

We now describe the second phase of the algorithm in which we complete the digitization of each rectangle by digitizing the vertical lines. Throughout, we will be considering the digitized region  $H_i - H_{i-1}$  for any fixed  $i$ ,  $1 \leq i \leq r$ . Let  $v_1, v_2, \dots, v_s$  denote the vertical segments between the horizontal lines  $h_i$  and  $h_{i-1}$ . Similar to the first phase, we determine the digitized region lying to the left of  $v_j$ , for each  $j$ , and then define the final digitized region by set difference. The process is slightly more

complex than the first phase because of the discontinuities in the boundary of  $H_i$  and  $H_{i-1}$ .

For an integer  $g, 0 \leq g \leq B$ , let  $L(g)$  denote the number of lattice squares in  $H_i - H_{i-1}$  that lie strictly to the left of the vertical line  $x = g$ . Intuitively,  $L(g)$  is a discrete approximation to the true area  $g(h_i - h_{i-1})$ . For example, in Fig. 4, the true area bounded by  $h_i = 3\frac{1}{3}$ ,  $h_{i-1} = 0$ , and  $g_{i,3} = 7$  is  $23\frac{1}{3}$  and  $L(g_{i,3}) = 24$ . In the special case that  $g$  is the floor or ceiling of a vertical line of the decomposition, then we can bound the value of  $L(g)$  as we now show.

CLAIM 3.2. *Let  $v_j$  be a vertical line of the decomposition between the horizontal lines  $h_i$  and  $h_{i-1}$ . Then*

- (i)  $L(\lfloor v_j \rfloor) \leq \lceil v_j(h_i - h_{i-1}) \rceil$ , and
- (ii)  $L(\lceil v_j \rceil) \geq \lfloor v_j(h_i - h_{i-1}) \rfloor$ .

*Proof.* By definition, both  $\lfloor v_j \rfloor$  and  $\lceil v_j \rceil$  are in  $G_i$  and  $G_{i-1}$ . From the definition of  $H_i$  and  $H_{i-1}$  it follows that  $L(\lfloor v_j \rfloor) = \lceil \lfloor v_j \rfloor h_i \rceil - \lceil \lfloor v_j \rfloor h_{i-1} \rceil$ . Part (i) follows by applying (5) and through simple manipulations. Part (ii) follows analogously.  $\square$

The digitized region to the left of  $v_j$ , denoted  $V_j$ , consists of all the squares  $S_{x,y} \in H_i - H_{i-1}$  for which  $x < \lfloor v_j \rfloor$  and a portion of the column of squares for which  $x = \lfloor v_j \rfloor$ . If  $v_j$  is not an integer, then the set of squares  $S_{x,y}$  with  $x \leq \lfloor v_j \rfloor$  is bounded on the right by the vertical line  $x = \lceil v_j \rceil$ . If  $v_j$  is an integer, then the overlapping constraint implies that we cannot allocate any squares to the right of  $v_j = \lceil v_j \rceil$ . Thus in either case the maximum number of such squares that can be allocated is  $L(\lceil v_j \rceil)$ . The digitization seeks to approximate the area bounded horizontally between  $h_i$  and  $h_{i-1}$  and on the left by  $v_j$ , that is,  $v_j(h_i - h_{i-1})$ . Thus, we define the size of  $V_j$ , denoted  $D_j$ , by combining these two values:

$$D_j = \min(\lceil v_j(h_i - h_{i-1}) \rceil, L(\lceil v_j \rceil)).$$

There are  $L(\lfloor v_j \rfloor)$  squares allocated strictly to the left of the column  $\lfloor v_j \rfloor$ , therefore, the number of squares  $S_{x,y}$  to be allocated where  $x = \lfloor v_j \rfloor$  is  $D_j - L(\lfloor v_j \rfloor)$ . We select the topmost squares of the column to be allocated. It follows from Claim 3.2(i) that  $0 \leq D_j - L(\lfloor v_j \rfloor)$ . It follows from the definition of  $D_j$  that there are enough squares in column  $\lfloor v_j \rfloor$  to satisfy this allocation. Finally, the digitized region between  $v_j$  and  $v_{j-1}$  is defined to be  $V_j - V_{j-1}$ . By definition of  $D_j$ , this digitization is overlapping.

For example, in Fig. 5, the true area bounded by  $h_i = 3\frac{1}{3}$ ,  $h_{i-1} = 0$ , and  $v_j = 7\frac{1}{2}$  is 25 and  $L(\lceil v_j \rceil) = 27$  and so  $D_j = 25$ . Since  $L(\lfloor v_j \rfloor) = 24$ , there is one square allocated in column  $\lfloor v_j \rfloor$ .

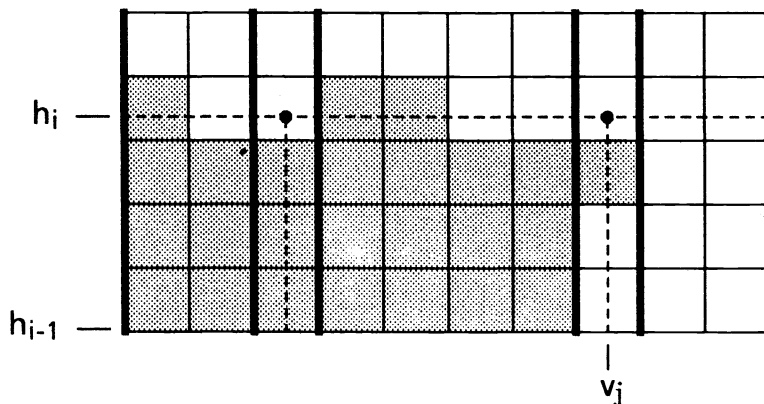


FIG. 5. Allocation to the left of a vertical line.

CLAIM 3.3. *The right-side boundary of  $V_j$  can be computed in  $O(1)$  time.*

*Proof.* The right-hand-side boundary is defined by the squares of column  $\lfloor v_j \rfloor$  that are in  $V_j$ . These squares can be computed in constant time once we know the topmost square of column  $\lfloor v_j \rfloor$  in  $H_i$ . However, since  $\lfloor v_j \rfloor \in G_i$ , the membership of this square in  $H_i$  can be determined in  $O(1)$  time by Claim 3.1.  $\square$

In summary, to digitize the rectangle bounded by horizontal lines  $h_i$  and  $h_{i-1}$  and vertical lines  $v_j$  and  $v_{j-1}$  we apply the first phase of the algorithm to digitize the horizontal region, and then apply the second phase to complete the digitization. We claim that this algorithm defines a digitization that is overlapping and has an absolute excess less than 2.

THEOREM 3.2. *Consider the digitization of each rectangle in a row-major decomposition of  $R$  described above.*

- (i) *The digitization defines a partition of the set of lattice squares in  $R$ .*
- (ii) *The digitization is overlapping.*
- (iii) *The digitization has an absolute excess less than 2.*

*Proof.* Parts (i) and (ii) follow from the preceding discussion. Details for both cases appear in [14]. To prove (iii), consider a digitized region bounded by vertical segments  $v_j$  and  $v_{j-1}$  between rows  $h_i$  and  $h_{i-1}$ . The size of the allocation is

$$\begin{aligned} D_j - D_{j-1} &= \min(\lceil v_j(h_i - h_{i-1}) \rceil, L(\lceil v_j \rceil)) - \min(\lceil v_{j-1}(h_i - h_{i-1}) \rceil, L(\lceil v_{j-1} \rceil)) \\ &\leq \lceil v_j(h_i - h_{i-1}) \rceil - \lfloor v_{j-1}(h_i - h_{i-1}) \rfloor \quad (\text{by Claim 3.2(ii)}) \\ &< \lceil (v_j - v_{j-1})(h_i - h_{i-1}) \rceil + 1 \quad (\text{by Equation (5)}) \\ &< (v_j - v_{j-1})(h_i - h_{i-1}) + 2. \end{aligned}$$

Thus, the digitization has absolute excess less than 2.  $\square$

The fact that the absolute excess may exceed 1 results from the min appearing in the definition of  $D_j$ . This seems to be an inherent consequence of the constraint that the digitization be overlapping and the locality exploited by the algorithm.

The running time of the algorithm follows from Claims 3.1 and 3.3 together with a few additional observations. By Claim 3.1, we can find the digitization of a rectangle, provided that the number of points in  $G_i$  is not too large. We can ignore all the vertices of the decomposition, except for those that appear within the set of unit squares that cover the rectangle, since the remaining vertices cannot affect the digitization here. The vertices to be considered will consist of the four corners of the rectangle, plus any other vertices along the horizontal edges of the rectangle. If the decomposition is generated by the algorithm presented in § 2, the widths of adjacent rectangles differ by at most a factor of  $\frac{1}{2}$ , from which it follows that the number of such vertices is never greater than 2. From this observation we have

CLAIM 3.4. *When the algorithm is applied to the  $n$ -row and  $n$ -column decompositions generated by the algorithm of § 2 and if  $p \leq AB$ , then we have the following:*

- (i) *The region containing a given lattice square can be determined in  $O(1)$  time.*
- (ii) *The boundary of a digitized region can be computed in  $O(1)$  time.*

**4. Further remarks.** We have given a simple algorithm for decomposing a rectangle into rectangles of equal area whose maximum perimeter is minimized. We have shown that this algorithm is optimal over the more general category of decompositions into pseudorectangles. We have also given an approximate solution to the discrete problem of partitioning grid squares into sets of equal size so that the maximum pseudoperimeter (sum of projections) is minimized.

There are a number of open questions remaining. In § 2, we showed that our decomposition is optimal over all decompositions into pseudorectangles. Is it true that

the decomposition is optimal even over all measurable, possibly disconnected, regions? The remark following Lemma 2.5 states that the algorithm is optimal (with respect to pseudoperimeter) for decomposition into arbitrary measurable sets, for certain input values. Also, the question of solving the rectangle decomposition problem in higher dimensions is open.

There is a wide class of similar partitioning problems that are related to the problems considered here. For example, given an  $A \times B$  rectangle  $R$  and a set of positive real numbers  $a_1, a_2, \dots, a_p$ , where  $\sum_i a_i = AB$ , decompose the rectangles into rectangles of area  $a_i$ , so that the maximum eccentricity (ratio of a rectangle's longer to shorter side) is minimized. It is also natural to consider alternate cost criteria, such as the sum of perimeters, rather than the maximum perimeter.

**Acknowledgments.** The authors thank Azriel Rosenfeld and Simon Kasif for drawing our attention to the data allocation problem, which was the motivation for this work. Stimulating conversations with Michael Werman are gratefully acknowledged. We would also like to thank the referees for their comments, which significantly improved the presentation.

## REFERENCES

- [1] N. ALON AND D. J. KLEITMAN, *Covering a square by small perimeter rectangles*, *Discrete Comput. Geom.*, 1 (1986), pp. 1-7.
- [2] D. AVIS AND G. T. TOUSSAINT, *An efficient algorithm for decomposing a polygon into star-shaped polygons*, *Pattern Recognition*, 13 (1981), pp. 395-398.
- [3] C. BERGE, *Graphs and Hypergraphs*, (translated by Edward Minieka), American Elsevier, New York, 1973.
- [4] B. BOLLOBÁS, *Graph Theory: An Introductory Course*, Springer-Verlag, Berlin, 1979.
- [5] B. CHAZELLE AND D. DOBKIN, *Decomposing a polygon into its convex parts*, in Proc. 11th Annual ACM Symposium on Theory of Computing, Atlanta, GA, 1979, pp. 38-48.
- [6] D. S. FRANZBLAU AND D. J. KLEITMAN, *An algorithm for constructing regions with rectangles: independence and minimum generating sets for collections of intervals*, in Proc. 16th Annual ACM Symposium on Theory of Computing, Washington, D.C., 1984, pp. 167-174.
- [7] M. R. GAREY, D. S. JOHNSON, F. P. PREPARATA, AND R. E. TARJAN, *Triangulating a simple polygon*, *Inform. Process Lett.*, 7 (1978), pp. 175-179.
- [8] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.
- [9] T. GONZALEZ AND S-Q. ZHENG, *Bounds for partitioning rectilinear polygons*, in Proc. Symposium on Computational Geometry, 1985, pp. 281-287.
- [10] C. BERGE, *The Theory of Graphs and its Applications*, Translated by A. Doig, John Wiley, New York, 1962. (In French.)
- [11] M. HOFRI, *Two-dimensional packing: expected performance of simple level algorithms*, *Inform. Control*, 45 (1980), pp. 1-17.
- [12] R. KARP, M. LUBY, AND A. MARCHETTI-SPACCAMELA, *Probabilistic analysis of multidimensional bin packing problems*, in Proc. 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 289-298.
- [13] S. KASIF AND R. KLETTE, *A data allocation problem for SIMD computers*, Tech. Report CAR-TR-11 Center for Automation Research, University of Maryland, College Park, MD, 1983.
- [14] T. Y. KONG, D. M. MOUNT, AND A. W. ROSCOE, *The decomposition of a rectangle into rectangles of minimal perimeter*, Tech. Report CAR-TR-169, Center for Automation Research, College Park, MD, 1986.
- [15] T. Y. KONG, D. M. MOUNT, AND M. WERMAN, *The decomposition of a square into rectangles of minimal perimeter*, *Discrete Appl. Math.*, 16 (1987), pp. 239-243.
- [16] S. R. LAY, *Convex Sets and their Applications*, John Wiley, New York, 1982.
- [17] J. O'ROURKE, *The complexity of computing minimum convex covers for polygons*, in Proc. 20th Annual Allerton Conference on Communications, Control and Computing, Urbana, IL, 1982, pp. 75-84.
- [18] R. E. TARJAN, *Data Structures and Network Algorithms*, CBMS-NSF Regional Conference Series in Applied Mathematics 44, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

## THE BOOLEAN HIERARCHY I: STRUCTURAL PROPERTIES\*

JIN-YI CAI†, THOMAS GUNDERMANN‡, JURIS HARTMANIS§, LANE A. HEMACHANDRA¶,  
VIVIAN SEWELSON<sup>1</sup>, KLAUS WAGNER<sup>2</sup>, AND GERD WECHSUNG‡

**Abstract.** In this paper, we study the complexity of sets formed by boolean operations (union, intersection, and complement) on NP sets. These are the sets accepted by trees of hardware with NP predicates as leaves, and together these form the boolean hierarchy.

We present many results about the structure of the boolean hierarchy: separation and immunity results, natural complete languages, and structural asymmetries between complementary classes.

We show that in some relativized worlds the boolean hierarchy is infinite, and that for every  $k$  there is a relativized world in which the boolean hierarchy extends exactly  $k$  levels. We prove natural languages, variations of VERTEX COVER, complete for the various levels of the boolean hierarchy. We show the following structural asymmetry: though no set in the boolean hierarchy is  $D^P$ -immune, there is a relativized world in which the boolean hierarchy contains  $\text{co}D^P$ -immune sets.

Thus, this paper explores the structural properties of the boolean hierarchy. A companion paper [J. Cai et al., *SIAM J. Comput.* 18 (1989), to appear] uses the boolean hierarchy to extend known results on small circuits [R. Karp and R. Lipton, *Proc. 12th Annual Symposium on the Theory of Computation*, 1980, pp. 302–309], sparse sets in NP-P [J. Hartmanis, N. Immerman, and V. Sewelson, *Proc. 15th Annual Symposium on the Theory of Computation*, 1983, pp. 382–391], and counting classes [A. Blass and Y. Gurevich, *Inform. and Control*, 55 (1982), pp. 80–88].

**Key words.** polynomial-time hierarchy, boolean hierarchy, relativized complexity classes, oracles, immunity, complete languages, structural complexity

**AMS(MOS) subject classifications.** 68Q15, 03D15

### 1. Introduction and overview of results.

**1.1. Introduction.** The class NP has long been central to computational complexity theory.  $D^P$ , the closure of  $\text{NP} \cup \text{coNP}$  under intersections, has recently been intensively studied [PY82], [PW85], [CM85]. In this paper, we study the natural completion of such structures—the closure of NP under boolean operations.

This closure, the boolean hierarchy (BH), has a clear “physical” interpretation. The sets in the boolean hierarchy are exactly those representable by *hardware over NP*. Each boolean hierarchy language is accepted by a hardware tree connecting NP machines (Fig. 1).

---

\* Received by the editors October 29, 1986; accepted for publication (in revised form) March 4, 1988. Preliminary versions of portions of this paper were presented at the 1985 International Conference on Fundamentals of Computation Theory [W85a] and the 1986 Structure in Complexity Theory Conference [CH86], [CH85].

† Computer Science Department, Cornell University, Ithaca, New York 14853. Present address, Department of Computer Science, Yale University, New Haven, Connecticut 06520. The research of this author was supported by a Sage Fellowship and National Science Foundation grants DCR-8301766 and DCR-8520597.

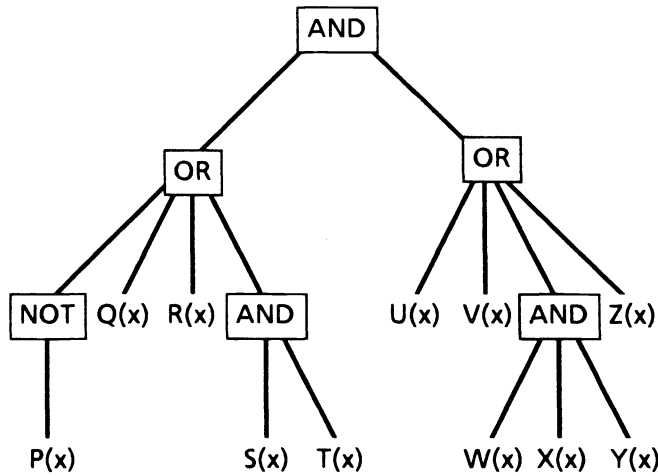
‡ Mathematics Section, Friedrich-Schiller University, Jena, German Democratic Republic.

§ Computer Science Department, Cornell University, Ithaca, New York 14853. The research of this author was supported by National Science Foundation grants DCR-8301766 and DCR-852097.

¶ Computer Science Department, Cornell University, Ithaca, New York 14853. Present address, Department of Computer Science, Columbia University, New York, New York 10027. The research of this author was supported by a Fannie and John Hertz Foundation Fellowship, National Science Foundation grants DCR-8301766 and DCR-8520597, and a Hewlett-Packard Corporation equipment grant.

<sup>1</sup> Department of Mathematics and Computer Science, Dartmouth College, Hanover, New Hampshire 03755.

<sup>2</sup> Institute for Mathematics, University of Augsburg, Augsburg, Federal Republic of Germany.



Each predicate is an NP predicate.

FIG. 1. Hardware over NP.

We present many results about the boolean hierarchy: separations and immunity results, natural complete languages, and structural asymmetries between complementary classes. Throughout, we emphasize the structure of the boolean hierarchy and its relationship to more familiar classes.

The boolean hierarchy appears in passing in many papers [PZ82], [R85], and was defined explicitly by Wagner and Wechsung [W85a]. Early results noted the location of the boolean hierarchy, i.e.,  $BH \subseteq P^{\#P}$  (in fact, a single call to a  $\#P$  oracle suffices), due to Papadimitriou and Zachos [PZ82] and  $BH \subseteq PP$ , due to Russo [R85]. This paper broadly investigates the structure of the boolean hierarchy itself. Related work of Köbler, Schöning, and Wagner [KSW86] explores the connection between the boolean hierarchy and sets that bounded truth-table reduce to SAT.

The boolean hierarchy has played a crucial role in the study of terseness and bounded query classes [AG87]. An important recent result is Kadin's proof that if the boolean hierarchy collapses then the polynomial hierarchy collapses [K87a]. This provides evidence that levels of the boolean hierarchy may be distinct.

**1.2. Definitions.** In § 2 we define the levels of the boolean hierarchy (Fig. 2). Each level of the boolean hierarchy consists of sets represented by a certain fixed structure of boolean operators on NP sets:

$$\begin{aligned}
 NP(0) &= P, \\
 NP(2i+1) &= \{L_A \cup L_B \mid L_A \in NP(2i), L_B \in NP\}, \\
 NP(2i+2) &= \{L_A \cap \overline{L_B} \mid L_A \in NP(2i+1), L_B \in NP\}, \\
 BH &= \bigcup_{i \geq 0} NP(i), \\
 coNP(i) &= \{S \mid \overline{S} \in NP(i)\}.
 \end{aligned}$$

For example,

$$NP(1) = NP, \quad NP(2) = D^P = \{L_1 \cap \overline{L_2} \mid L_1, L_2 \in NP\},$$



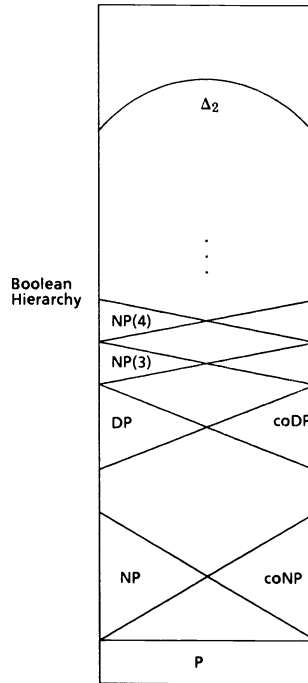


FIG. 2. Complexity hierarchies.

$$NP(3) = \{(L_1 \cap \overline{L_2}) \cup L_3 \mid L_1, L_2, L_3 \in NP\},$$

and

$$NP(4) = \{((L_1 \cap \overline{L_2}) \cup L_3) \cap \overline{L_4} \mid L_1, L_2, L_3, L_4 \in NP\}.$$

Sets in the boolean hierarchy have normal forms. Each can be written as a finite union of  $D^P$  sets. Indeed, the ubiquitous boolean hierarchy has many characterizations.

LEMMA. *The following are equivalent:*

- (1)  $T \in BH$ .
- (2)  $T$  is a finite union of  $D^P$  sets.
- (3)  $T$  is a finite intersection of  $coD^P$  sets.
- (4)  $T$  is in the boolean closure of  $NP$ .
- (5)  $T \equiv_{\text{bounded truth-table}}^P SAT$ .

**1.3. Relativized separations of the boolean hierarchy.** In § 3 we explore the relativized structure of the boolean hierarchy and present strong separation results. We show relativized worlds in which the boolean hierarchy is infinite. By comparison, oracles making the polynomial hierarchy infinite (see Yao [Y85]) are indirect, quite involved, and took many years to discover.

It follows from the relativizations of § 3 that  $BH$  is a natural complexity class that may lack many-one complete languages. Indeed, the boolean hierarchy in relativized worlds distinguishes between the power of many-one and bounded truth-table reducibilities; though the boolean hierarchy lacks many-one complete languages in some relativizations, it always has bounded truth-table complete sets.

To prove our infinite separation of the boolean hierarchy, we introduce the “sawing” technique. This method forms the basis of all the relativized results in the

boolean hierarchy of [CGII]. This technique places successively more stringent requirements on oracle extensions, while seeking to *force* a desired behavior.

Indeed, we can prove probabilistic separations. A well-known result of Bennett and Gill shows that  $P^A \neq NP^A$  with probability one (over randomly chosen  $A$ ) [BG81]. We can show that the boolean hierarchy is infinite with probability 1 [C86].

Heller's proof of Theorem 2 in [H84] shows that with appropriate relativization the polynomial hierarchy can be collapsed to  $D^P$  while separating NP from coNP. Recently, Ko [K87b] showed that for every  $k$  there is an oracle for which the polynomial hierarchy extends exactly  $k$  levels. For each  $k$ , we display worlds where the boolean hierarchy extends exactly  $k$  levels, with PSPACE collapsing to the  $k+1$ st level. In such worlds, understanding the boolean hierarchy is a prerequisite to understanding the structure of standard feasibly computable complexity classes, as the structure of the polynomial hierarchy degenerates exactly in the boolean hierarchy.

**THEOREM.** (1) *There exists recursive  $B$ , for all  $i \geq 1$ ,  $[NP^B(i) \neq NP^B(i+1)]$ .*

(2) *For all  $i \geq 0$ , there exists recursive  $B$   $[NP^B(1) \neq NP^B(2) \neq \dots \neq NP^B(i) \neq NP^B(i+1) = PSPACE^B]$ .*

**COROLLARY.** *Though the boolean hierarchy always has Turing (and even bounded truth-table) complete sets, there is an oracle  $B$  for which  $BH^B$  has no many-one complete sets.*

*Proof of the corollary.* SAT is bounded truth-table complete for the boolean hierarchy. If the boolean hierarchy has a many-one complete set, it is finite, i.e., it collapses to the level in which the complete set resides. Thus the oracle  $B$  from (1) of the above theorem guarantees that  $BH^B$  has no many-one complete sets.  $\square$

Since the method of (1) of the above theorem is central to the work of this paper, we now give a quick, informal proof. A much more detailed proof of a stronger version of this theorem appears as Theorem 3.1.1. However, the interested reader is urged to first peruse the following short proof.

*Proof of (1) of the theorem.* In the standard way, if we can separate a generic level of the boolean hierarchy by diagonalization, then by interleaving diagonalizations we can construct a single oracle, simultaneously separating the entire hierarchy. For concreteness, we show an oracle  $B$  so  $NP^B(17) \neq coNP^B(17)$ . In particular, we show that  $L_B$ , defined below, is in  $coNP^B(17)$ - $NP^B(17)$ :

$$L_B = \{0^{17j} | (\forall x, |x| = 17j)[x \notin B] \text{ AND} \\ \{(\exists x, |x| = 17j + 1)[x \in B] \text{ OR} \\ \{(\forall x, |x| = 17j + 2)[x \notin B] \text{ AND} \\ \{ \dots \text{ OR} \\ \{(\forall x, |x| = 17j + 16)[x \notin B] \} \dots \} \} \\ \}.$$

*Stage 0.* Let  $B_0 = \emptyset$ . We will have  $B = \cup B_i$ .

*Stage  $j$ .* Let  $H$  be the  $j$ th NP(17) machine. It is composed of nine NP machines ( $N_i$ 's) and eight coNP machines ( $A_j$ 's) so that (for simplicity, we use low numbers for the machines instead of indexing them):

$$L(H^B) =_{\text{def}} L(N_1^B) \cup \{L(A_1^B) \cap \{L(N_2^B) \cup \{ \dots \cup \{L(A_8^B) \cap L(N_9^B)\} \dots \}\}.$$

Without loss of generality each of the machines runs at most  $n^j + j$  steps. Recall that a coNP machine is a nondeterministic machine that rejects if and only if at least one

of its nondeterministic paths rejects. Choose  $w_j$  so large that we have the following: (1) no string of length  $\cong w_j$  has ever been queried or set before; (2)  $w_j > 18w_{j-1}$ ; and (3)  $17((w_j)^j + j) \ll 2^{w_j}$ . We now use our method of successive restriction (the “sawing” technique); we see-saw between pressuring NP machines to accept and pressuring coNP machines to reject. Throughout, we obey the constraint that in stage  $j$  we never add to our oracle any string of length less than  $w_j$ .

*Case 1.* There exists an extension  $C$  of  $B_{j-1}$  so that  $N_1^C(0^{17w_j})$  accepts (otherwise, go to Case 2). Freeze into  $B_j$  an accepting path of the machine; thus we know that  $H^B(0^{17w_j})$  accepts. Add an untouched string of length  $17w_j$  to  $B_j$ . Now we have ensured that  $0^{17w_j} \in L(H^B) - L_B$ . Thus, machine  $H$  fails to accept  $L_B$ .

*Case 2k* ( $1 \leq k \leq 8$ ). There exists an extension  $C$  of  $B_{j-1}$  so that  $C$  contains no length  $17w_j, \dots, 17w_j + 2k - 2$  strings and  $A_k^C(0^{17w_j})$  rejects (otherwise, go to case  $2k + 1$ ). Freeze into  $B_j$  a rejecting path of the machine; thus we know that  $H^B(0^{17w_j})$  rejects. Add an untouched string of length  $17w_j + 2k - 1$  to  $B_j$ . Now we have ensured that  $0^{17w_j} \in L_B - L(H^B)$ . Thus, machine  $H$  fails to accept  $L_B$ .

*Case 2k + 1* ( $1 \leq k \leq 8$ ). There exists an extension  $C$  of  $B_{j-1}$  so that  $C$  contains no length  $17w_j, 17w_j + 1, \dots, 17w_j + 2k - 1$  strings and  $N_{k+1}^C(0^{17w_j})$  accepts (otherwise, go to case  $2k + 2$ ). Freeze into  $B_j$  an accepting path of the machine; thus we know that  $H^B(0^{17w_j})$  accepts. Add an untouched string of length  $17w_j + 2k$  to  $B_j$ . Now we have ensured that  $0^{17w_j} \in L(H^B) - L_B$ . Thus, machine  $H$  fails to accept  $L_B$ .

*Case 18.* If we have made it to this case, we know that for any extension  $C$  of  $B_{j-1}$  that has no length  $17w_j, 17w_j + 1, \dots, 17w_j + 15$  strings,  $L(H^C)$  rejects  $0^{17w_j}$ . Let  $B_j := B_{j-1}$ , which safely avoids strings for the forbidden lengths. Thus  $0^{17w_j} \in L_B - L(H^B)$ .

In each case, we have seen that  $L(H^B) \neq L_B$ . Thus, by the preceding discussion, we are done.  $\square$

**1.4. Immunity results.** In § 4 we study the immunity structure of the boolean hierarchy. With appropriate relativization, we display a  $\Delta_2$  set that is BH-immune. Also, no set in the boolean hierarchy can be either NP bi-immune or coNP bi-immune. We show that no boolean hierarchy set is NP (2)-immune, yet there are worlds where the boolean hierarchy has a coNP (2)-immune set. Thus NP (2) and coNP (2) are structurally asymmetric. Such asymmetry between complementary classes is rare in the realm of feasible computations. The most notable previous example is the asymmetry between NP and coNP of Hartmanis, Immerman, and Sewelson [HIS83], which is extended in a companion paper [CGII].

**THEOREM.** *There is a recursive oracle  $A$  for which  $\Delta_2^A$  has a BH<sup>A</sup>-immune set.*

**THEOREM.** (1) *No set in the boolean hierarchy is NP bi-immune or coNP bi-immune.*

(2) *No set in the boolean hierarchy is NP (2)-immune, yet there is a relativized world in which NP (2) has coNP (2)-immune sets.*

**1.5. Complete languages for the boolean hierarchy.** Though certain complexity classes may lack complete languages [S82], [HI85], [HH86b], most standard classes have complete languages. Complete languages mark the upper limit of the complexity of a class, and are useful in proving new languages complete for the class [GJ79].

In § 5 we present natural complete languages for the levels of the boolean hierarchy. Related work on complete languages appears in [W86]. We show variations of SAT that provide canonical complete languages for the levels of the boolean hierarchy. For example,  $SAT(3) = \{\langle f_1, f_2, f_3 \rangle \mid (f_1 \in SAT \wedge f_2 \in UNSAT) \vee f_3 \in SAT\}$  is easily seen to be NP (3)-complete, and we can similarly define  $SAT(k)$ , complete for NP ( $k$ ), by mimicking the definitions of § 2. By reductions from these languages, we show many problems

to be complete for the levels of the boolean hierarchy. For example, the following theorem shows the completeness of versions of graph colorability.

**THEOREM.** COLORABILITY ( $k$ ) is NP ( $k$ )-complete, where

$$\text{COLORABILITY } (k) = \{G \mid \chi(G) \text{ is odd} \wedge 3k \leq \chi(G) \leq 4k\}, \quad k \text{ even}$$

$$\text{COLORABILITY } (k) = \{G \mid (\chi(G) \text{ is odd} \wedge 3k \leq \chi(G) \leq 4k) \vee \chi(G) < 3k\}, \quad k \text{ odd.}$$

**2. Definitions and representations.**

**2.1. The boolean hierarchy: definitions and algebraic characterizations.** In this section, we make precise the notation of fixing a certain amount of logic hardware over NP.

**DEFINITION 2.1.1.**

**Operator C:**

$$C(L_1) = L_1.$$

$$C(L_1, \dots, L_k) = \begin{cases} C(L_1, \dots, L_{k-1}) \cup L_k & \text{if } k \text{ is odd} \\ C(L_1, \dots, L_{k-1}) \cap \overline{L_k} & \text{if } k \text{ is even} \end{cases} \quad \text{for } k > 1.$$

**Operator D:**

$$D(L_1) = L_1.$$

$$D(L_1, \dots, L_k) = L_1 - D(L_2, \dots, L_k) \quad \text{for } k > 1.$$

**Operator E:**

$$E(L_1) = L_1.$$

$$E(L_1, L_2) = L_1 - L_2.$$

$$E(L_1, \dots, L_k) = \begin{cases} E(L_1, \dots, L_{k-1}) \cup L_k & \text{if } k \text{ is odd} \\ E(L_1, \dots, L_{k-2}) \cup (L_{k-1} - L_k) & \text{if } k \text{ is even} \end{cases} \quad \text{for } k > 2.$$

Thus,  $C(L_1, \dots, L_n)$  represents an alternating sum of the languages

$$(C^*) \quad L_1 - L_2 + L_3 \cdots + (-1)^{n-1} L_n$$

(where the expression is associated to the left, and  $+$ ,  $-$  stand for set theoretic union and difference, respectively). Similarly,  $D(L_1, \dots, L_n)$  represents the nested difference of the languages:

$$(D^*) \quad L_1 - (L_2 - (\dots (L_{n-1} - L_n) \dots)),$$

and  $E$  represents sums of differences of consecutive pairs:

$$(E^*) \quad E(L_1, \dots, L_n) = \begin{cases} (L_1 - L_2) \cup \dots \cup (L_{n-2} - L_{n-1}) \cup L_n & \text{if } n \text{ is odd,} \\ (L_1 - L_2) \cup \dots \cup (L_{n-1} - L_n) & \text{if } n \text{ is even.} \end{cases}$$

We consider the language classes generated by applying  $C$  to a fixed number of NP languages.

**DEFINITION 2.1.2.** For all  $n \geq 1$ , define

$$NP(n) = \{C(L_1, \dots, L_n) \mid L_i \in NP, 1 \leq i \leq n\}.$$

In particular,  $NP(1) = NP$  and  $NP(2) = D^P$ . For any oracle set  $X$ , the relativized version  $NP^X(n)$  is naturally defined.

These normal forms are analogous to normal forms developed by Hausdorff [H14], as noted in [W85a]. For detailed proofs see [H14], [W85a], [CH85].

**THEOREM 2.1.1** [H14], [W85a]. Each of the following classes is precisely  $NP(n)$ :

(1)  $\{C(L_1, \dots, L_n) \mid L_i \in NP, 1 \leq i \leq n\}$ .

(1')  $\{C(L_1, \dots, L_n) \mid L_1 \supseteq \dots \supseteq L_n, L_i \in NP, 1 \leq i \leq n\}$ .

- (2)  $\{\mathbf{D}(L_1, \dots, L_n) \mid L_i \in \text{NP}, 1 \leq i \leq n\}$ .
- (2')  $\{\mathbf{D}(L_1, \dots, L_n) \mid L_1 \supseteq \dots \supseteq L_n, L_i \in \text{NP}, 1 \leq i \leq n\}$ .
- (3)  $\{\mathbf{E}(L_1, \dots, L_n) \mid L_i \in \text{NP}, 1 \leq i \leq n\}$ .
- (3')  $\{\mathbf{E}(L_1, \dots, L_n) \mid L_1 \supseteq \dots \supseteq L_n, L_i \in \text{NP}, 1 \leq i \leq n\}$ .

DEFINITION 2.1.3.  $\text{coNP}(n) = \{\bar{L} \mid L \in \text{NP}(n)\}$ .

If we start from  $\text{coNP}$ , then the following theorem can be proved similarly.

THEOREM 2.1.1' [H14], [W85a]. *The following classes are identical:*

- (1)  $\{\mathbf{C}(L_1, \dots, L_n) \mid L_i \in \text{coNP}, 1 \leq i \leq n\}$ .
- (1')  $\{\mathbf{C}(L_1, \dots, L_n) \mid L_1 \supseteq \dots \supseteq L_n, L_i \in \text{coNP}, 1 \leq i \leq n\}$ .
- (2)  $\{\mathbf{D}(L_1, \dots, L_n) \mid L_i \in \text{coNP}, 1 \leq i \leq n\}$ .
- (2')  $\{\mathbf{D}(L_1, \dots, L_n) \mid L_1 \supseteq \dots \supseteq L_n, L_i \in \text{coNP}, 1 \leq i \leq n\}$ .
- (3)  $\{\mathbf{E}(L_1, \dots, L_n) \mid L_i \in \text{coNP}, 1 \leq i \leq n\}$ .
- (3')  $\{\mathbf{E}(L_1, \dots, L_n) \mid L_1 \supseteq \dots \supseteq L_n, L_i \in \text{coNP}, 1 \leq i \leq n\}$ .

Using Theorem 2.1.1' we can prove the following theorem, which will be useful later. (The proof is left to the reader.)

THEOREM 2.1.2.

$$\{\mathbf{C}(\bar{L}_1, \dots, \bar{L}_n) \mid \bar{L}_i \in \text{coNP}\} = \begin{cases} \text{NP}(n) & \text{if } n \text{ is even,} \\ \text{coNP}(n) & \text{if } n \text{ is odd.} \end{cases}$$

THEOREM 2.1.3 (Weak Hierarchy Theorem). *For every*

$$m < n, \text{NP}(m) \cup \text{coNP}(m) \subseteq \text{NP}(n) \cap \text{coNP}(n).$$

*Proof of Theorem 2.1.3.* Using normal form (3) from Theorem 2.1.1, we get  $\text{NP}(n) \subseteq \text{NP}(n+1)$ , since  $\mathbf{E}(L_1, \dots, L_n) = \mathbf{E}(L_1, \dots, L_n, \emptyset)$ . Using normal form (1), we have  $\mathbf{C}(L_1, \dots, L_n) = \mathbf{C}(\Sigma^*, \bar{L}_1, \bar{L}_2, \dots, \bar{L}_n)$ ; hence  $\text{NP}(n) \subseteq \text{coNP}(n+1)$ . Therefore,  $\text{NP}(n) \subseteq \text{NP}(n+1) \cap \text{coNP}(n+1)$ . Since the right-hand side is closed under complementation, the theorem follows.  $\square$

DEFINITION 2.1.4.  $\text{BH} = \bigcup_{k \geq 1} \text{NP}(k)$ . This defines the *boolean hierarchy*.

THEOREM 2.1.4.  $\text{BH} = \bigcup_{k \geq 1} \{\mathbf{C}(\bar{L}_1, \dots, \bar{L}_n) \mid \bar{L}_i \in \text{coNP}\}$ .

Whether the boolean hierarchy collapses (at some level, or entirely, or not at all) is unknown. We note that  $\text{P} = \text{NP} \Rightarrow \text{P} = \text{BH}$ ; hence a proof that the boolean hierarchy does not collapse entirely would entail  $\text{P} \neq \text{NP}$ . On the other hand, even assuming  $\text{P} \neq \text{NP}$ , a proof that the boolean hierarchy is infinite would still subsume such major open problems as  $\text{D}^{\text{P}} \neq \text{coD}^{\text{P}}$ . Therefore, settling whether the boolean hierarchy is a true hierarchy will be extremely difficult. Kadin [K87a] has shown that if the boolean hierarchy collapses then the polynomial hierarchy collapses, which suggests that the boolean hierarchy may be proper.

Note that  $\text{NP}(k) = \text{coNP}(k)$  implies that  $\text{NP}(k) = \text{BH}$ , using the normal forms. As a final remark, the boolean hierarchy sits between the first and second levels of the polynomial hierarchy, i.e.  $\text{NP} \subseteq \text{BH} \subseteq \text{P}^{\text{SAT}} = \Delta_2^{\text{P}}$ .

OBSERVATION 2.1.1 (Downward Separation). For all  $k \geq 1$ ,

$$\text{NP}(k) = \text{coNP}(k) \Rightarrow \text{NP}(k) = \text{BH}.$$

**2.2. The boolean hierarchy is the boolean closure of NP.** The boolean hierarchy is the boolean closure of NP. To prove this claim, we simply note that any amount of hardware over NP can be reshaped into a finite union of  $\text{D}^{\text{P}}$  sets. Thus two levels of hardware on top of  $\text{NP} \cup \text{coNP}$  are as powerful as  $n$  levels. Though every hardware tree has an equivalent two-level tree, we may need many more gates on the shallow tree.

LEMMA 2.2.1. *If  $S$  is expressible by hardware over NP predicates (i.e., if  $S$  is in the closure of NP with respect to {union, intersection, complement}), then  $S$  is a finite union of  $\text{D}^{\text{P}}$  sets and a finite intersection of  $\text{coD}^{\text{P}}$  sets.*

*Proof of Lemma 2.2.1.* To show  $S$  is the finite union of  $D^P$  sets, rewrite the tree describing  $S$  in disjunctive normal form over the predicates and their negations. We get the two-level logic tree of (Fig. 3), the leaves of which are NP and coNP predicates. Since any intersection of sets from  $NP \cup coNP$  is in  $D^P$ , we can write  $S$  as the finite union of  $D^P$  sets. Similarly, using conjunctive normal form,  $S$  is the finite intersection of  $coD^P$  sets.  $\square$

**THEOREM 2.2.1.** *The boolean hierarchy is the closure of NP under boolean operations.*

*Proof of Theorem 2.2.1.* Clearly every set in the boolean hierarchy is in the boolean closure of NP. By our lemma, every set  $S$  in the boolean closure of NP is the finite union of  $D^P$  sets. Thus for some  $k$  and  $\{L_i \in NP\}$ ,  $S$  is  $E(L_1, \dots, L_{2k})$ . Thus  $S$  is in  $NP(2k)$  by Theorem 2.1.1, so  $S$  is in the boolean hierarchy.  $\square$

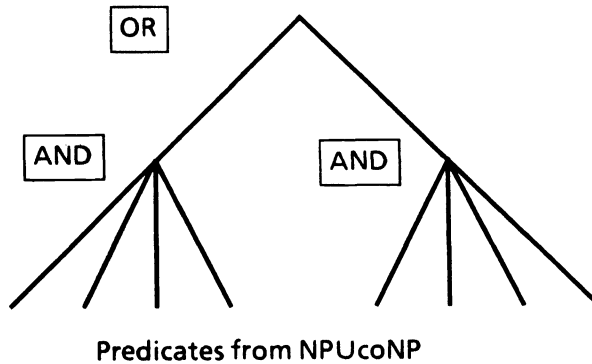


FIG. 3. Hardware tree of depth two.

The above theorems are NP versions of general theorems of Hausdorff about boolean closures of set rings [H14], [W85a].

We can give another characterization of our boolean hierarchy. The bounded truth-table reducibility of recursion theory has the following analogue among polynomial time computable classes [Y83]:

$A \leq_{btt}^P B \Leftrightarrow$  there are polynomial time transducers  $M_1$  and  $M_2$  and an integer  $n$  so that for every  $x$ ,  $M_1(x)$  prints a finite set of at most  $n$  elements and  $M_2(x)$  prints a finite collection of finite sets, such that  $x \in A \Leftrightarrow B \cap M_1(x)$  is present in  $M_2(x)$ .

We have the following theorem, which states that the boolean hierarchy is exactly the sets that are bounded truth-table reducible to SAT.

**THEOREM 2.2.2.**  $BH = \{T \mid T \leq_{btt}^P SAT\}$ .

*Proof of Theorem 2.2.2.*  $\Rightarrow (BH \subseteq \{T \mid T \leq_{btt}^P SAT\})$ : Let  $A \in BH$ , then there are sets  $L_1 \supseteq L_2 \supseteq \dots \supseteq L_k$  so that  $A = C(L_1, L_2, \dots, L_k)$ , where  $L_i \in NP$ . Let  $f_i$  be a many-one  $p$ -time reduction from  $L_i$  to SAT, a complete language for NP. Now for all  $x$ ,  $M_1(x)$  prints  $\{f_1(x), \dots, f_k(x)\}$  and  $M_2(x)$  prints the following sets  $\{f_1(x), \dots, f_{2i-1}(x)\}$ , where  $2i - 1 \leq k$ .

$\Leftarrow (BH \supseteq \{T \mid T \leq_{btt}^P SAT\})$ : Let  $A \leq_{btt}^P SAT$  via  $M_1$  and  $M_2$ . Without loss of generality,  $M_1(x)$  always prints  $k$  element sets. Let  $M_1(x) = \{f_1(x), \dots, f_k(x)\}$ , and  $L_i = \{x \mid f_i(x) \in SAT\}$ . Then  $L_i \in NP$ . Now it is easy to construct a union of  $2^k$  many  $D^P$  sets that equals  $A$ , using the definition of  $A \leq_{btt}^P SAT$ .  $\square$

**3. Separations.** Is the boolean hierarchy infinite or finite? If it is finite, how many levels does it extend before it collapses? This section shows that all reasonable answers

to these questions can be obtained in appropriately relativized worlds. In § 3.1 we give a detailed relativization that makes the boolean hierarchy infinite. In § 3.2 we show that, for each  $k$ , the boolean hierarchy can extend exactly  $k$  levels, with everything up to PSPACE collapsing into the  $(k + 1)$ st level.

**3.1. An infinite relativized boolean hierarchy.** This section proves the following strong separation theorem. As proven in § 1.3, a corollary of this theorem is that in some relativized worlds the boolean hierarchy does not have many-one complete sets—the fact that it *does* have bounded truth-table complete sets notwithstanding. The following proof combines the sawing argument of § 1.3 with other relativized separation techniques. Readers not interested in the details of oracle constructions may skip to § 3.2.

**THEOREM 3.1.1.** *There exists a recursive oracle  $X$  separating the entire boolean hierarchy. Specifically,*

$$\begin{aligned}
 & \subset \text{NP}^X \subset \\
 \text{P}^X \subset \text{NP}^X \cap \text{coNP}^X & \neq \text{NP}^X \cup \text{coNP}^X \subset \text{NP}^X(2) \cap \text{coNP}^X(2) \cdots \\
 & \subset \text{coNP}^X \subset \\
 & \subset \text{NP}^X(k) \subset \\
 \cdots \subset \text{NP}^X(k) \cap \text{coNP}^X(k) & \neq \text{NP}^X(k) \cup \text{coNP}^X(k) \\
 & \subset \text{coNP}^X(k) \subset \\
 & \subset \text{NP}^X(k+1) \cap \text{coNP}^X(k+1) \cdots .
 \end{aligned}$$

We first describe the strategies for  $X$  to achieve each of these separations.

(1)  $\text{P}^X \neq \text{NP}^X \cap \text{coNP}^X$  is done in the usual way: Allocate two “segments” of strings with exponentially many for each  $n$ . Diagonalize each  $p$ -time deterministic machine over a tally set  $L_1^\cap$ , where “ $1^n \in L_1^\cap$ ” is determined by the existence of some strings in the appropriate “segment.” To insure that  $L_1^\cap \in \text{NP}^X \cap \text{coNP}^X$ , we impose certain global constraints on  $X$ :  $X$  will have strings from exactly one segment of the two allocated, corresponding to any particular  $n$ . The oracle set  $X$  is constructed in stages  $\{X_s\}$ ; an extension of  $X_s$  is termed “admissible” only if it satisfies all these global constraints. (There are other constraints to be discussed later.)

To diagonalize, we pick  $n$  sufficiently large, and fix an extension  $\hat{X}_s$  of  $X_s$  that is “admissible” except for length  $n$ , where  $\hat{X}_s$  has no strings from either segments. Run the next  $p$ -time machine  $P_i$  on  $1^n$ , using  $\hat{X}_s$  to answer queries, and “freeze” to  $\bar{X}$  any string that is queried negatively. Since at most polynomially many strings are “frozen” when  $P_i$  halts, we may, according to whether  $P_i$  accepts or rejects  $1^n$ , choose the appropriate segment to enumerate a string (corresponding to length  $n$ ) into  $X$ . Then extend  $X_s$  according to  $\hat{X}_s$  so that, the computation of  $P_i$  on  $1^n$  is preserved. Thus, the final  $X$  will be “admissible” and  $\text{P}^X \neq \text{NP}^X \cap \text{coNP}^X$ .

The other requirements are of the following two types:

$$\text{NP}^X(k) \neq \text{coNP}^X(k), \quad \text{NP}^X(k) \cup \text{coNP}^X(k) \neq \text{NP}^X(k+1) \cap \text{coNP}^X(k+1)$$

for all  $k \geq 1$ .

Note that this is enough since

$$\begin{aligned}
 \text{NP}^X(k) \neq \text{coNP}^X(k) & \Rightarrow \text{NP}^X(k) \cap \text{coNP}^X(k) \subset \text{NP}^X(k) \subset \\
 & \subset \text{coNP}^X(k) \subset \\
 & \text{NP}^X(k) \cup \text{coNP}^X(k).
 \end{aligned}$$

Furthermore, for requirements  $\text{NP}^X(k) \neq \text{coNP}^X(k)$  it is enough to satisfy for all odd  $k$ .

(2) Strategies to insure  $NP^X(k) \neq coNP^X(k)$  for odd  $k$ . We will illustrate this with the case  $k=3$ . Let us fix three separate “segments”  $S_i$ ,  $1 \leq i \leq 3$ , each having an  $n$ th “block”  $S_i^n$  with exponentially many strings for every  $n$ . Define

$$L_3 = \{1^n \mid (\exists s_1 \wedge \neg \exists s_2) \vee \exists s_3\},$$

where “ $\exists s_i$ ” stands for  $\exists s_i \in X \cap S_i^n$ . Clearly  $L_3 \in NP^X(3)$ . We want to make sure that for each triple  $\langle N_i, N_j, N_k \rangle$ , the  $coNP^X(3)$  language  $W_3 = (L(N_i^X) \cup L(N_j^X)) \cap L(N_k^X)$  is not  $L_3$ ; hence  $L_3 \notin coNP^X(3)$ .

The following generic step embodies a “sawing” argument that is the key to our separation results, and can be readily generalized to the  $k > 3$  case.

Pick  $n$  sufficiently large so that the  $S_i^n$ ’s are all “clean,” in the sense that no string from these blocks has been committed to either  $X$  or  $\bar{X}$ .

*Case 1.* If  $N_k$  accepts  $1^n$  under some “admissible” extension of the current  $X_s$ , then we “freeze” an accepting computation and extend  $X_s$  “admissibly” (“freeze” those used negatively in the computation to  $\bar{X}$ ). Then enumerate an “unfrozen”  $s_3 \in S_3^n$  into  $X$ , (if there is none in  $X$  so far). We will see that there will always be some  $s_3$  “unfrozen,” if  $n$  is large, and  $s_3 \in X$  does not violate any global constraints. Thus  $1^n \notin W_3$  but  $1^n \in L_3$ .

*Case 2.*  $N_k$  never accepts  $1^n$  under any “admissible” extension. We will say an extension is *without*  $T$  if  $X \cap T^n = \emptyset$ . Consider all “admissible” extensions without  $S_3$  (i.e., extensions in which  $X \cap S_3^n = \emptyset$ ).

*Case 2.1.* If  $N_j$  accepts  $1^n$  under some such extension, then “freeze” an accepting computation and extend  $X_s$  “admissibly” without  $S_3$ . Enumerate an “unfrozen”  $s_2 \in S_2^n$  into  $X$  (if none exists in  $X$  so far). Thus  $1^n \in W_3$ , but  $1^n \notin L_3$ .

*Case 2.2.*  $N_j$  never accepts  $1^n$  under any admissible “without  $S_3$ ” extension. We consider all “admissible” extensions that are “without  $S_3$  and without  $S_2$ .”

*Case 2.2.1.* If  $N_i$  accepts  $1^n$  under one of such extension, “freeze” it. Extend  $X_s$  “admissibly” without  $S_3, S_2$ ; put an “unfrozen”  $s_1 \in S_1^n$  in  $X$ . Thus  $1^n \notin W_3$  but  $1^n \in L_3$ .

*Case 2.2.2.*  $N_i$  never accepts  $1^n$  under any “admissible without  $S_3, S_2$ ” extension. Then extend  $X_s$  “admissibly” without  $S_3, S_2, S_1$ . Thus  $1^n \in W_3$ , but  $1^n \notin L_3$ .

Hence, the final  $X$  will be admissible and  $NP^X(3) \neq coNP^X(3)$ .

(3) Strategies for  $NP^X(k) \cup coNP^X(k) \neq NP^X(k+1) \cap coNP^X(k+1)$ , for all  $k \geq 1$ .

We illustrate with  $k=2$ . Fix six separate “segments”  $S_i, S'_i, i=1, 2, 3$  as before. Denote  $S_i^n, S'_i^n$  their  $n$ th blocks. We impose the following global constraints on  $X$ :

For each  $n$ , either

(I) 
$$[(\exists s_1 \wedge \neg \exists s_2) \vee \exists s_3] \wedge [(\neg \exists s'_1 \vee \exists s'_2) \wedge \neg \exists s'_3]$$

or

(II) 
$$[(\neg \exists s_1 \vee \exists s_2) \wedge \neg \exists s_3] \wedge [(\exists s'_1 \wedge \neg \exists s'_2) \vee \exists s'_3]$$

where “ $\exists s_i$ ” “ $\exists s'_i$ ” are shorthands as before. For example, “ $\exists s_i$ ” means “ $\exists s_i \in S_i^n \cap X$ .”

Define

$$L_3^\cap = \{1^n \mid (\exists s_1 \wedge \neg \exists s_2) \vee \exists s_3\}.$$

Clearly, if  $X$  satisfies the above constraints, then  $L_3^\cap \in NP^X(3) \cap coNP^X(3)$ .



We need to insure for each tuple  $\langle N_i, N_j \rangle$  that neither  $L(N_i^X) \cap \overline{L(N_j^X)}$  nor  $L(N_i^X) \cup L(N_j^X)$  equals  $L_3^\cap$ , hence  $L_3^\cap \notin \text{NP}^X(2) \cup \text{coNP}^X(2)$ .

To insure this, we use the following variant of the “sawing” argument discussed before.

*Step (1).* For  $\overline{L(N_i^X)} \cup L(N_j^X)$ . Pick an  $n$  large enough so that the corresponding “blocks”  $S_i^n, S_i'^n$  are all “clean.”

Temporarily ignore global constraints (I), (II) for this particular  $n$ ; consider all “admissible” extensions of  $X_s$  without  $S_3, S_3'$  (that is, for which  $X \cap S_3^n = \emptyset$  and  $X \cap S_3'^n = \emptyset$ ).

*Case 1.* If  $N_j$  accepts  $1^n$  under such an extension, then we “freeze” an accepting computation. Extend  $X_s$  “admissibly,” and for this  $n$ , add an “unfrozen” member of  $S_2^n$  and an “unfrozen” member of  $S_3'^n$  to  $X$ , thus satisfying (II); therefore  $1^n \in L(N_j^X)$ , but  $1^n \notin L_3^\cap$ .

*Case 2.*  $N_j$  never accepts  $1^n$  under any such extension. Consider all “admissible” (again ignore (I), (II) for this  $n$ ) extensions of  $X_s$  without  $S_3, S_3', S_2, S_2'$ .

*Case 2.1.* If  $N_i$  accepts  $1^n$  for such an extension, then “freeze” one accepting computation and extend  $X_s$  “admissibly.” For this  $n$ , add to  $X$  an  $s_1$  and an  $s_2'$  “unfrozen” from the appropriate blocks  $S_1^n$  and  $S_2'^n$ , respectively. Thus satisfy (I), and  $1^n \notin \overline{L(N_i^X)} \cup L(N_j^X)$ , but  $1^n \in L_3^\cap$ .

*Case 2.2.*  $N_i$  never accepts  $1^n$  under such extensions. Then it rejects when we extend  $X_s$  “admissibly” without  $S_1$  and with some  $s_1' \in S_1'^n$ , and “without  $S_3, S_3', S_2, S_2'$ .” Then (II) is satisfied. Now  $1^n \in L(N_i^X)$ , but  $1^n \notin L_3^\cap$ .

This completes the diagonalization of Step (1).

*Step (2)* is similar, except we do it for  $L(N_i^X) \cap \overline{L(N_j^X)}$ .

A formal proof follows.

*Proof of Theorem 3.1.1.* We use a standard pairing function  $\langle \cdot, \cdot \rangle$ , for which  $|\langle x, y \rangle| \sim |x| + |y|$ . When convenient, we identify binary string  $x$  with integer  $1x$ . Define  $\langle a, b, c \rangle = \langle a, \langle b, c \rangle \rangle$ , etc.

Let  $\{P_i\}$  ( $\{N_i\}$ ) be standard enumerations of deterministic (nondeterministic) TMs with polynomial time clocks  $p_i$ .

Define  $S_{i,j,k}^n = \{\langle i, j, k, x \rangle \mid |x| = n\}$ . Note that  $S_{i,j,k}^n \cap S_{i',j',k'}^n = \emptyset$ , unless they are identical. Define propositions  $p_{i,j}^n \equiv S_{i,j,1}^n \cap X \neq \emptyset$ ,  $q_{i,j}^n \equiv S_{i,j,2}^n \cap X \neq \emptyset$ . We also use the operator  $C$  on propositions in a natural way.

FACT.  $\{1^n \mid C(q_1, \dots, q_t)\} \in \text{NP}(t)$ , provided  $\{1^n \mid q_s\} \in \text{NP}$ ,  $1 \leq s \leq t$ .

Define  $L_k = \{1^n \mid C(q_{k,1}^n, \dots, q_{k,k}^n)\}$  for odd  $k$ . Then  $L_k \in \text{NP}^X(k)$ .

We will impose the following global constraints  $GC_{k,n}$  on  $X$ :

$$(*) \quad C(p_{k,1}^n, \dots, p_{k,k}^n) \Leftrightarrow \neg C(p_{k,k+1}^n, \dots, p_{k,2k}^n).$$

Define  $L_k^\cap = \{1^n \mid C(p_{k,1}^n, \dots, p_{k,k}^n)\}$ .

FACT. If  $X$  satisfies  $GC_{k,n}$ , then  $L_k^\cap \in \text{NP}^X(k) \cap \text{coNP}^X(k)$ .

We have the following requirements (relativized to  $X$ ):

$$R_{\langle 0, i \rangle}: L(P_i) \neq L_1^\cap, \quad \text{for } k \geq 1,$$

$$R_{\langle 2k-1, i \rangle}: \overline{C(L(N_{i_1}), \dots, L(N_{i_{2k-1}}))} \neq L_{2k-1}, \quad \text{where } i = \langle i_1, \dots, i_{2k-1} \rangle,$$

$$R_{\langle 2k, i \rangle}: C(L(N_{i_1}), \dots, L(N_{i_k})) \neq L_{k+1}^\cap \quad \text{and} \quad \overline{C(L(N_{i_1}), \dots, L(N_{i_k}))} \neq L_{k+1}^\cap,$$

where  $i = \langle i_1, \dots, i_k \rangle$ .

Each  $S^n_{i,j,k}$  is called a “block.” Of importance to us are the blocks concerned in  $p^n_{i,j}$  and  $q^n_{i,j}$ . Specifically,  $S^n_{i,j,1}$  for  $1 \leq j \leq 2i$  and  $S^n_{i,j,2}$  for odd  $i$ ,  $1 \leq j \leq i$ . We call  $\cup_{j=1}^{2k} S^n_{k,j,1}$  and  $\cup_{j=1}^k S^n_{k,j,2}$  “large blocks.” Moreover, the former is called a “global constraint block,”  $GCB_{k,n}$ .

We further suppose a well-ordering  $\ll$  (of type  $\omega$ ) is defined, which “holds together” each block and large block (i.e., it induces a well-ordering on the block quotient space). This can be easily accomplished, e.g., imagine all integers on a line, then bring everything in  $S^n_{i,j,k}$  to just to the right of its leftmost member. That is, we will say that  $x_1 \ll x_2$  if (1) the smallest integer in the block to which  $x_1$  belongs is less than the smallest integer in the block to which  $x_2$  belongs, or (2)  $x_1$  and  $x_2$  are in the same block and  $x_1 < x_2$ . Also, do this for the large blocks.

$X$  is constructed in stages:  $X = \cup X_s$ . At the end of stage  $s$ , we have an initial (under  $\ll$ ) segment  $X_s$  of  $X$ . Membership in  $X$  or  $\bar{X}$  of any string in a block or a large block is determined in a single stage. If we define  $\text{block}(X_s)$  = the set of blocks on which  $X$  has been defined by stage  $s$ , then  $\text{block}(X_s)$  is an initial segment in the block quotient space and

$$(*) \quad X_s \cap \cup \text{block}(X_s) = X \cap \cup \text{block}(X_s).$$

An *admissible extension* of  $X_s$  is a set with  $X_s$  as initial segment (condition  $(*)$ ) and satisfies all  $GCB_{k,n}$ . We will maintain the condition that for each  $GCB_{k,n}$ , if  $GCB_{k,n} \cap \cup \text{block}(X_s) \neq \emptyset$ , then  $GCB_{k,n}$  is satisfied by  $X_s$ ; thus such admissible extensions always exist.

We say a block  $B$  is *clean* if  $B \cap \cup \text{block}(X_s) = \emptyset$ .

CONSTRUCTION OF  $X$ .

$$X_0 = \emptyset \quad s = 0; \quad n_0 = 0.$$

*Stage  $s + 1 = \langle 0, i \rangle + 1$ .* We satisfy  $R_{\langle 0, i \rangle}$ .

Let  $n_{s+1} = \min \{n \mid n > n_s \wedge GCB_{1,n} \text{ is clean} \wedge 2^n > p_i(n)\}$ . (Recall that  $p_i(\cdot)$  is the run time of deterministic polynomial time machine  $P_i$ .)

Take an extension  $\hat{X}_s$  of  $X_s$  satisfying  $\hat{X}_s \cap GCB_{1,n_{s+1}} = \emptyset$  and otherwise admissible. Run  $P_i$  on  $1^{n_{s+1}}$  using  $\hat{X}_s$ . “Freeze” into  $\bar{X}$  all strings not in  $\hat{X}_s$  that are queried by  $P_i$ . According to whether  $P_i(1^{n_{s+1}})$  accepts or rejects, enumerate an unfrozen  $x$  from  $S^n_{1,2,1}$  or  $S^n_{1,1,1}$ , respectively, into  $\hat{X}_s$ . (This satisfies  $GC_{1,n_{s+1}}$ .) Let  $B$  be the  $\ll$ -last (large) block that was queried, or  $GCB_{1,n_{s+1}}$ , whichever is later.

Set  $X_{s+1}$  to  $\hat{X}_s$  up to  $B$ .  $s := s + 1$ .

*Stage  $s + 1 = \langle 2k - 1, i \rangle + 1$ .* We satisfy  $R_{\langle 2k - 1, i \rangle}$ .

(Recall that  $i = \langle i_1, \dots, i_{2k-1} \rangle$ .)

Let  $n_{s+1} = \min \{n \mid n > n_s \wedge \cup_{j=1}^{2k-1} S^n_{2k-1,j,2} \text{ is clean} \wedge 2^n > \sum_{j=1}^{2k-1} p_{i_j}(n)\}$ .

$saw := 2k - 1$ ;  $done := false$ ;

Repeat

Consider all admissible extensions of  $X_s$ , satisfying  $\neg q^n_{2k-1,j}$ , for  $saw < j \leq 2k - 1$ . (If  $saw = 2k - 1$ , the previous condition is void.)

If  $N_{saw}(1^{n_{s+1}})$  has an accepting computation under such an extension  $\hat{X}_s$ , we will “freeze” an accepting computation as follows:

Extend  $X_s$  according to  $\hat{X}_s$  up to the  $\ll$ -last block for which there exists a queried string (round up to a large block if the block is a part of it) or  $\cup_{j=1}^{2k-1} S^n_{2k-1,j,2}$ , whichever is later.

Then enumerate an  $s_{saw} \in S_{2k-1, saw, 2}^{n_{s+1}}$  into  $X$  that is not queried negatively in the computation (if there does not exist one in  $X$  already).

$done := true$ ;

Else,  $saw := saw - 1$ .

Until ( $saw = 0$ ) or ( $done$ ).

If  $saw = 0$ , take any admissible extension  $\hat{X}_s$  of  $X_s$  satisfying  $\neg q_{2k-1, j}^{n_{s+1}}$ , for  $1 \leq j \leq 2k-1$ , then extend  $X_s$  according to  $\hat{X}_s$  up to large block  $\bigcup_{j=1}^{2k-1} S_{2k-1, j, 2}^{n_{s+1}}$ .

$s := s + 1$ .

**Stage  $s+1 = \langle 2k, i \rangle + 1$ .** We satisfy  $R_{\langle 2k, i \rangle}$ .

**Step 1.** Diagonalize  $\mathbf{C}(L(N_{i_1}), \dots, L(N_{i_k}))$ .

Let  $n_{s+1} = \min \{n \mid n > n_s \wedge GCB_{k+1, n} \text{ is clean} \wedge 2^n > \sum_{j=1}^k p_{i_j}(n)\}$

$saw := k$ ;  $done := false$ ;

**Repeat**

Consider all admissible extensions of  $X_s$  (except temporarily ignore  $GC_{k+1, n_{s+1}}$ ) satisfying  $(\neg p_{k+1, j}^{n_{s+1}}) \wedge (\neg p_{k+1, k+1+j}^{n_{s+1}})$ , for  $saw < j \leq k+1$ .

If  $N_{i_{saw}}(1^{n_{s+1}})$  has an accepting computation under such an extension  $\hat{X}_s$ , we will “freeze” an accepting computation as follows:

Extend  $X_s$  according to  $\hat{X}_s$  up to the  $\ll$ -last block for which there exists a queried string (round up to a large block if the block is part of it) or  $GCB_{k+1, n_{s+1}}$ , whichever is later.

Then enumerate an  $s_{saw+1} \in S_{k+1, saw+1, 1}^{n_{s+1}}$  and  $s'_{saw} \in S_{k+1, k+1+saw, 1}^{n_{s+1}}$  into  $X$  that are not queried negatively in the computation (if there do not exist such two strings in  $X$  already).

$done := true$ ;

Else,  $saw := saw - 1$

Until ( $saw = 0$ ) or ( $done$ ).

If  $saw = 0$  take any admissible extension of  $X_s$  that has no string from  $S_{k+1, j, 1}^{n_{s+1}}$ , for  $1 < j \leq k+1$ , and no string from  $S_{k+1, k+1+j, 1}^{n_{s+1}}$ , for  $1 \leq j \leq k+1$ , and has an  $s_1 \in S_{k+1, 1, 1}^{n_{s+1}}$  (this satisfies  $GC_{k+1, n_{s+1}}$ ). Then extend  $X_s$  up to large block  $GCB_{k+1, n_{s+1}}$ .

**Step 2.** Diagonalize  $\mathbf{C}(L(N_{i_1}), \dots, L(N_{i_k}))$ .

We omit the details since this step is symmetric to Step 1.

**End of construction**

Let us prove that stage  $\langle 2k-1, i \rangle + 1$  does satisfy requirement  $R_{\langle 2k-1, i \rangle}$ . The other parts are left to the reader.

Suppose the Repeat loop ends with  $saw > 0$ . Then clearly for some admissible extension  $\hat{X}_s$  of  $X_s$  satisfying  $\neg q_{2k-1, j}^{n_{s+1}}$ ,  $saw < j \leq 2k-1$ ,  $N_{i_{saw}}$  accepts  $1^{n_{s+1}}$ . Meanwhile  $N_{i_j}$ ,  $saw < j \leq 2k-1$ , reject  $1^{n_{s+1}}$  for all such admissible extensions of  $X_s$ .

There are at most  $p_{i_{saw}}(n_{s+1})$  strings being queried on the accepting computation path. Thus an  $s_{saw} \in S_{2k-1, saw, 2}^{n_{s+1}}$  can be found and enumerated into  $X$ . Therefore the accepting computation is preserved, and  $q_{2k-1, saw}^{n_{s+1}}$  is satisfied. Hence for the final  $X$ ,  $1^{n_{s+1}} \in \mathbf{C}(L(N_{i_{saw}}), \dots, L(N_{i_{2k-1}}))$ .

If  $saw$  is odd, then  $1^{n_{s+1}} \in L_{2k-1} - \overline{\mathbf{C}(L(N_{i_1}), \dots, L(N_{i_{2k-1}}))}$ .

If  $saw$  is even, then  $1^{n_{s+1}} \in \overline{\mathbf{C}(L(N_{i_1}), \dots, L(N_{i_{2k-1}}))} - L_{2k-1}$ .

Similarly, suppose the Repeat loop ends with  $saw = 0$ , then none of the  $N_{i_j}$ ,  $1 \leq j \leq 2k-1$ , will accept  $1^{n_{s+1}}$  for any admissible extensions with  $\neg q_{2k-1, j}^{n_{s+1}}$ ,  $1 < j \leq 2k-1$ . Since the final  $X$  has no string in  $\bigcup_{j=1}^{2k-1} S_{2k-1, j, 2}^{n_{s+1}}$ , we have  $1^{n_{s+1}} \in \overline{\mathbf{C}(L(N_{i_1}), \dots, L(N_{i_{2k-1}}))}$ , and  $1^{n_{s+1}} \notin L_{2k-1}$ .  $\square$

**3.2. Relativized boolean hierarchies extending exactly  $k$  levels.** In some relativized worlds, the boolean hierarchy separates at the  $k$ th level, yet PSPACE falls into the

$(k+1)$ st level. This fact opens the possibility that in the real world the structure of feasible computations degenerates exactly in the boolean hierarchy.

**THEOREM 3.2.1.** *For each  $k \geq 1$ , there is a recursive oracle  $A$  so that  $\text{NP}^A(k) \neq \text{coNP}^A(k)$ , yet  $\text{NP}^A(k+1) = \text{coNP}^A(k+1) = \text{PSPACE}^A$ .*

We state a proof for the case  $k=2$ , and show the form of the general proof. Our strategy is simple. While coding to ensure that  $\text{PSPACE}^A \subseteq \text{NP}^A(3)$ , we diagonalize so that  $\text{D}^P(A) \neq \text{coD}^P(A)$ .

*Proof of Theorem 3.2.1 ( $k=2$  case).* We display a recursive oracle  $A$  for which  $\text{D}^P(A) \neq \text{coD}^P(A)$  and  $\text{NP}^A(3) = \text{PSPACE}^A$ .

Let  $M_i$  be an enumeration of  $\text{PSPACE}$  machines. Without loss of generality,  $M_i$  runs in space at most  $n^i + i$  and queries strings of length at most  $n^i + i$ . By adopting polynomial length queries, we follow the standard model of relativized  $\text{PSPACE}$ [FSS81].

Let  $U = \{M_i \# x \# 1^{|x|^{i+1}} \mid M_i^A(x) \text{ accepts}\}$ . Note that  $U$  is  $\text{PSPACE}^A$  complete. We will encode to ensure that  $U \in \text{NP}^A(3)$ . In particular we will assure that if  $y$  is of length  $i$ :

$$(*) \quad y \in U \Leftrightarrow [(\exists z, |yz| = 5i+1)[yz \in A] \wedge (\forall z, |yz| = 5i+2)[yz \notin A]] \\ \vee (\exists z, |yz| = 5i+3)[yz \in A].$$

Let  $L = \{0^i \mid (\exists z, |z| = i)[z \in A]\} \cap \{0^i \mid (\forall z, |z| = i+1)[z \notin A]\}$ . Clearly  $L \in \text{D}^P(A)$ . We show that no  $\text{coD}^P(A)$  machine accepts  $L$ . The  $k$ th  $\text{coD}^P$  machine,  $H_k$ , will be such that  $L(H_k^A) = L(N_j^A) \cup L(A_l^A)$ ,  $\langle j, l \rangle = k$  and  $\{N_i\}, \{A_i\}$  are enumerations of  $\text{NP}$  and  $\text{coNP}$  machines, respectively. That is, to “run”  $H_k$ , we “run” both  $N_j$  and  $A_l$  and accept if either accepts.

Our oracle is built in stages. The following constraint applies throughout the construction: In stage  $5i+j$ ,  $1 \leq j \leq 5$ , no string of length less than  $5i+1$  can be added to the oracle.

#### ORACLE CONSTRUCTION.

*Stage “ $5i+1, 5i+2, 5i+3$ ” (Encoding).* For each  $y$  of length  $i$ , assure that equation  $(*)$  is satisfied. A possible problem is that our ability to control strings of lengths  $5i+1, 2, 3$  may be constrained by the effects of some previous diagonalization. Various cases of the diagonalizations put various types of constraints on us. Fortunately, for each type of constraint we retain the freedom of coding both possibilities,  $y \notin U$  or  $y \in U$ . After we list the diagonalization cases, we give a recipe for codings done under each case.

*Stage “ $5i+4, 5i+5$ ” (Diagonalization).* If strings of length greater than or equal to  $5i+1$  have been queried by any previous diagonalization phase, do nothing during this stage. Otherwise, let  $H_k$  be the first  $\text{coD}^P$  machine that still might accept  $L$ . Let us say  $k = \langle j, l \rangle$ ; so, informally speaking,  $H_k = N_j \cup A_l$ . If either  $N_j$  or  $A_l$  has run time (on inputs of length  $5i+4$ ) more than  $2^{\sqrt{i}}$ , do nothing.

Otherwise, run  $N_j(0^{5i+4})$  with, as oracles, all extensions of  $A$  that add no new string having length congruent to  $3 \pmod 5$ .

*Case 1.*  $N_j^A(0^{5i+4})$  accepts on some such extension. Freeze in  $A$  all elements along some accepting path. So  $H_k^A(0^{5i+4})$  accepts. Make it a liar by adding some unconstrained length  $5i+5$  string to  $A$ . We win, since  $0^{5i+4} \in L(H_k^A)$ , but  $0^{5i+4} \notin L$ .

*Case 2.*  $N_j^A(0^{5i+4})$  rejects on all such extensions. Then, as long as we make sure all strings of length congruent to  $3 \pmod 5$  remain out of  $A$ ,  $H_k$  is simply an overworked  $A_l$  trying to accept a hard language,  $L$ . So, run  $A_l(0^{5i+4})$

on all oracle extensions that (1) add no new string with length congruent to 3 mod 5, and (2) have no strings of lengths  $5i+2$  or  $5i+5$ .

- Case 2a.  $A_i^A(0^{5i+4})$  rejects on some such extension. Freeze in  $A$  elements along some rejecting path. So  $H_k^A(0^{5i+4})$  rejects. Make it a liar by putting an unconstrained length  $5i+4$  string into  $A$  (recall that by the conditions imposed, we have no length  $5i+5$  strings). We win, as  $0^{5i+4} \notin L(H_k^A)$ , but  $0^{5i+4} \in L$ .
- Case 2b.  $A_i^A(0^{5i+4})$  accepts on all such extensions. Then, in particular, it accepts on the extension that puts *no* strings of length  $5i+4$  or  $5i+5$  into  $A$ .  $H_k^A(0^{5i+4})$  accepts but  $0^{5i+4} \notin L$ , so we have again made  $H_k$  a liar.

End of cases

Finally, we must provide the promised coding strategies. Since our diagonalizations only take place when out of the range of any previous diagonalization, each time we code we are constrained by at most one case of the diagonalization. Figure 4 shows these strategies.

For example, consider trying to code  $y \in U$ , when our last diagonalization succeeded on Case 2a. Case 2 obligates us to keep strings of length congruent to 3 mod 5 out of  $A$ . Also, it may have fixed some strings of length  $5i+1$  in and out of  $A$ , and it may have fixed some strings of length  $5i+2$  out of  $A$ . Nonetheless, we have no problem here. We code  $y \in U$  by placing an unconstrained length  $5i+1$  string into  $A$ . Now we have no length  $5i+2$  strings in  $A$ ; by (\*) the new length  $5i+1$  string codes the fact that  $y \in U$ . Note that it does not interfere with the diagonalization.

Case	To code $y \in U$		To code $y \notin U$						
1	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">X O</td></tr> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">X O</td></tr> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">☒ O</td></tr> </table>	X O	X O	☒ O	$5i+1$ $5i+2$ $5i+3$	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">X O</td></tr> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">X O ☒</td></tr> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">O</td></tr> </table>	X O	X O ☒	O
X O									
X O									
☒ O									
X O									
X O ☒									
O									
2a	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">☒ O X</td></tr> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">O</td></tr> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">EMPTY</td></tr> </table>	☒ O X	O	EMPTY	$5i+1$ $5i+2$ $5i+3$	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">X O</td></tr> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">☒ O</td></tr> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">EMPTY</td></tr> </table>	X O	☒ O	EMPTY
☒ O X									
O									
EMPTY									
X O									
☒ O									
EMPTY									
2b	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">☒</td></tr> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">EMPTY</td></tr> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">EMPTY</td></tr> </table>	☒	EMPTY	EMPTY	$5i+1$ $5i+2$ $5i+3$	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">EMPTY</td></tr> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">EMPTY</td></tr> <tr><td style="border: 1px solid black; padding: 2px; text-align: center;">EMPTY</td></tr> </table>	EMPTY	EMPTY	EMPTY
☒									
EMPTY									
EMPTY									
EMPTY									
EMPTY									
EMPTY									

KEY: X strings may be fixed in A  
 O strings may be fixed out of A  
 ☒ string we add  
 EMPTY all strings fixed out of A

FIG. 4. Encoding strategies under various restrictions.

This is a proof of the case  $k = 2$ . The general proof follows the same outline. To separate exactly  $k$  levels, we put  $L$  in  $NP^A(k) - coNP^A(k)$  and code  $U$  into  $NP^A(k + 1)$ . We will have  $k + 1$  encoding stages and  $k$  diagonalization stages.  $\square$

**4. Immunity results.** We first prove a theorem that shows the difficulty of finding immunity relations within the boolean hierarchy: No BH set is NP bi-immune or coNP bi-immune. Jockusch first proved the analogue for recursively enumerable sets of the following theorem.

**THEOREM 4.1.** *No set in the boolean hierarchy can be NP bi-immune. That is, for each  $S \in BH$  there is an infinite set  $L \in NP$  such that either  $L \subseteq S$  or  $L \subseteq \bar{S}$ . Similarly, no set in the boolean hierarchy is coNP bi-immune.*

*Proof of Theorem 4.1.* By definition,  $BH = \bigcup_{k \geq 1} NP(k)$ . We prove the first part; the second part is similar, using  $BH = \bigcup_{k \geq 1} \{C(\bar{L}_1, \dots, \bar{L}_k) \mid \bar{L}_i \in coNP\}$ .

Our proof is by induction on the level of  $S$  in the boolean hierarchy. Our base case is that  $S$  is in NP. If  $S$  is infinite, set  $L = S$ . If  $S$  is finite, set  $L = \bar{S}$ . In either case, we see that  $S$  is not bi-immune.

Suppose  $S \in NP(k)$ ,  $k > 1$ , and (inductively) no set in  $NP(k - 1)$  is NP bi-immune. By definition, there are NP sets  $L_i$ ,  $1 \leq i \leq k$ , for which  $S = C(L_1, \dots, L_k)$ . If  $k$  is odd, then  $S = S_1 \cup L_k$ , where  $S_1 \in NP(k - 1)$ . If  $L_k$  is infinite or  $S_1$  has an infinite NP subset, then so does  $S$ . Otherwise,  $L_k$  is finite and  $\bar{S}_1$  has an infinite NP subset  $X$ . Then  $\bar{S} = \bar{S}_1 \cap \bar{L}_k \supseteq X - L_k$ .  $X - L_k$  is an infinite NP set, so we are done.

A dual argument can be applied for  $k$  even.  $\square$

Is it possible for a set in a higher level of the boolean hierarchy to be immune to lower levels? We show the following theorem.

**THEOREM 4.2.** *There is a recursive oracle  $X$  such that  $NP^X(2)$  has a  $coNP^X(2)$ -immune set.*

*Proof of Theorem 4.2.* Define  $L = \{1^n \mid (\exists 0x \in X, |x| = n) \wedge \neg(\exists 1y \in X, |y| = n)\}$ . Clearly  $L$  is in  $NP^X(2)$ . We now diagonalize to ensure that  $L$  is  $coNP^X(2)$ -immune.

$X$  is constructed in stages. At each stage we keep a finite list of  $coNP(2)$  machines, represented as pairs of NP machines, that have not been “kicked out.” Recall  $coNP(2) = \{\bar{L}_i \cup L_j \mid L_i, L_j \in NP\}$ . At stage  $s$ , get a large  $n$ ; we achieve the following objective:

*Either* some machine on the list accepts  $1^n$  (this machine’s accepting computation will be preserved) and we will arrange to have  $1^n \notin L$  or no machine on the list accepts (this fact will also be preserved) and we will arrange to have  $1^n \in L$ .

In the former case, the “mistaken” machines, those we have fooled, are deleted from the list. In the latter, we append to the list one more  $\langle N_i, N_j \rangle$ . From stage  $s$  to  $s + 1$ , we will arrange to have some  $1y$  of the “skipped” length so that  $L$  is not enlarged.

Hence every  $\bar{L}(N_i) \cup L(N_j)$  once on the list will not accept any more members of  $L$ , unless it is “kicked out” later. Since at any moment the list is finite, we can only “kick out” a finite number of  $\langle N_i, N_j \rangle$ ’s consecutively. Thus the second possibility does occur infinitely often, so  $L$  is infinite and every  $\bar{L}(N_i) \cup L(N_j)$  is eventually considered. Therefore  $L$  only has finite subsets from  $coNP^X(2)$ .

The strategy to achieve that objective is the following:

Assume at some point there are  $k$  pairs  $\langle N_i, N_j \rangle$ ,  $1 \leq i \leq k$ , on the list. Construct NP machines  $N_\sigma, N_\tau$  such that

$$\bar{L}(N_\sigma) \cup L(N_\tau) = \bigcup_{1 \leq i \leq k} (\bar{L}(N_i) \cup L(N_j)).$$

Consider all extensions that contain no string previously frozen to  $\bar{X}$ , see if  $N_\tau$  accepts  $1^n$  under some such extension, and if so,

- Case 1. Freeze one accepting computation path of that extension. Choose a  $1y$  unfrozen with  $|y| = n$ ; put  $1y$  into  $X$ . Thus  $1^n \notin L$  and we may kick out all mistaken pairs (there is at least one). Otherwise,
- Case 2. Consider all extensions that have no string of the form  $1y$ ,  $|y| = n$ ; see if  $N_\sigma$  accepts  $1^n$  on one of these extensions. If so,
- Case 2a. Freeze one accepting computation and add an unfrozen string  $0x$ ,  $|x| = n$  to  $X$ . Thus  $1^n \in L$  and no pair on the list accepts  $1^n$ . Otherwise,
- Case 2b. Do nothing. At length  $n$ , we have neither  $0x$  nor  $1y$ ; hence  $1^n \notin L$ , but some  $\overline{L(N_i)}$  said yes to  $1^n$ .  $\square$

On the other hand, it follows from our normal form theorem that the situation is drastically different for NP(2). By Theorem 2.1.1, every set in the boolean hierarchy can be expressed as a finite union of NP(2) sets. Clearly, this implies the following theorem.

**THEOREM 4.3.** *No set in the boolean hierarchy is NP(2)-immune.*

Thus we have a *structural asymmetry among complementary classes*: though no boolean hierarchy set is  $D^P$ -immune, there are worlds where the boolean hierarchy does have  $\text{co}D^P$ -immune sets.

The argument used to prove Theorem 4.3 yields more. In fact, for every  $L$

$$L \text{ is NP(2)-immune} \Leftrightarrow L \text{ is BH-immune.}$$

The proof is immediate. If  $L$  has an infinite subset  $M$  from the boolean hierarchy then  $M$  in turn must have an infinite subset from NP(2). Thus  $L$  has an infinite NP(2) subset.

The above observation can be refined to prove the following theorem.

**THEOREM 4.4.** *There is a recursive oracle  $X$  so that  $\Delta_2^{P,X}$  has a set that is  $\text{BH}^X$ -immune.*

*Proof of Theorem 4.4.* Define predicates  $p_k^n \equiv \text{“}\exists x, |x| = n, 1^k 0x \in X\text{”}$ ,  $1 \leq k \leq 2n$ . (Note that  $1^k 0x = 1^l 0y \Rightarrow k = l, x = y$ .)

Consider  $L_\Delta = \{1^n \mid \mathbf{C}(p_1^n, \dots, p_{2n}^n)\}$ . Clearly  $L_\Delta \in \Delta_2^{P,X}$ .

We will construct  $X$ , so that  $L_\Delta$  is  $\text{BH}^X$ -immune.

$X$  is constructed in stages;  $X = \bigcup X_s$ . At each stage we keep a finite list of BH machines, represented by finite sequences of NP machine indices.  $\text{NP}(2k)$  consists of unions of  $k$  NP(2) sets. We may construct at stage  $s$  a finite sequence of NP indices  $\langle i_1, \dots, i_t \rangle$ , so that  $\mathbf{C}(L(N_{i_1}), \dots, L(N_{i_t}))$  is precisely the union of those on the list, relativized to any oracle set. Pick  $n$  sufficiently large, then apply our “sawing” argument to diagonalize.

CONSTRUCTION OF  $X$ .

$$X_0 = \emptyset; \quad X'_0 = \emptyset; \quad \text{list}_0 = \emptyset; \quad n_0 = 0; \quad s = 0.$$

Stage  $s + 1$ :

Construct  $\langle i_1, \dots, i_t \rangle$ , so that for every  $X$ ,

$$\mathbf{C}(L(N_{i_1}^X), \dots, L(N_{i_t}^X)) = \bigcup L(M^X);$$

the union is over all  $M \in \text{list}_s$ , represented as finite sequences of NP indices via  $\mathbf{C}$ . Let  $p$  be the sum of time bounds of  $N_{i_1}, \dots, N_{i_t}$ . Without loss of generality,  $t$  is even.

Let  $n_{s+1} = \min \{n \mid n > n_s \text{ and } 2n > t \text{ and } (\forall m \geq n)[m \text{ is “clean”}] \text{ and } (\forall m \geq n)[2^m > p(m)]\}$ .

$\forall n: n_s < n < n_{s+1}$ , keep  $1^n$  out of  $L_\Delta$  as follows:

{Find  $1^{2n} 0x \notin X'_s, |x| = n$ , set  $X_s := X_s \cup \{1^{2n} 0x\}$ .}

Set  $X'_s := X'_s \cup \{1^k 0x \mid |x| = n_{s+1}, t + 1 < k \leq 2n_{s+1}\}$

(This effectively strips away the last  $2n_{s+1} - (t + 1)$  quantifiers in the definition of  $L_\Delta$  for  $1^{n_{s+1}}$ .)

For some  $10x \notin X'_s$ ,  $|x| = n_{s+1}$  set  $X_s := X_s \cup \{10x\}$ .

$saw := 0$ ;  $done := false$ ;

repeat

{If there is an extension  $Y$  of  $X_s$ , so that  $X_s \subseteq Y$ ,  $Y \cap X'_s = \emptyset$ , and  $N_{i_t - saw}^Y(1^{n_{s+1}})$  has an accepting computation, we will freeze one accepting computation as follows:

$\{X_s := X_s \cup \{\text{those in } Y \text{ that are queried}\}$

$X'_s := X'_s \cup \{\text{those not in } Y \text{ that are queried}\}$

Pick a  $1^{t+1-saw}0x \notin X'_s$ ,  $|x| = n_{s+1}$ , and add it to  $X_s$ .

$done := true$ .

}

else  $\{X'_s := X'_s \cup \{1^{t+1-saw}0x \mid |x| = n_{s+1}\}$ ;  $saw := saw + 1\}$

}

until ( $saw = t$ ) or ( $done = true$ )

If  $saw$  is even then appended the next BH machine to  $list_s$

else delete all  $M \in list_s$  with  $1^{n_{s+1}} \in L(M^{X_s})$  (there must be one)

$X_{s+1} := X_s$ ,  $X'_{s+1} := X'_s$ ,  $list_{s+1} := list_s$ ,  $s := s + 1$

End of stage  $s + 1$

End of construction

We leave the proof that the construction works to the reader.  $\square$

**5. Complete languages for the boolean hierarchy.** Complete languages are important in the understanding of complexity class; a complete language marks the upper limit of a class's complexity. In this section we drive *natural* complete languages for the levels of the boolean hierarchy. However, our first complete languages will be universal and canonical ones.

**THEOREM 5.1.** For each  $k$ ,  $NP(k)$  has a complete language.

*Proof of Theorem 5.1.* Fix  $k \geq 1$ , we claim that the universal language

$$U = \{\langle i_1, i_2, \dots, i_k, x, 0^{p_i(|x|)} \rangle \mid x \in C(L(N_{i_1}), \dots, L(N_{i_k})), i = \langle i_1, \dots, i_k \rangle\},$$

is complete for  $NP(k)$  with respect to polynomial time reductions, where  $p_i(\cdot)$  is a sum of the polynomial time bounds for  $N_{i_1}, \dots, N_{i_k}$ . Clearly  $U$  is in  $NP(k)$ , owing to the padding  $0^{p_i(|x|)}$ . Given any  $L \in NP(k)$ , there exist  $i_1, \dots, i_k$ , so that  $L = C(L(N_{i_1}), \dots, L(N_{i_k}))$ . Let  $p_i(\cdot)$  be a sum of time bounds of  $N_{i_j}$ 's, then  $x \rightarrow \langle i_1, \dots, i_k, x, 0^{p_i(|x|)} \rangle$  is a one-to-one polynomial time reduction from  $L$  to  $U$ .  $\square$

Languages such as the one constructed in Theorem 5.1 are unnatural. Natural complete languages more forcefully demonstrate the usefulness of language classes. Natural complete languages exist for every level of the boolean hierarchy.

Define  $SAT(k) = \{\langle f_1, \dots, f_k \rangle \mid C(p_1, \dots, p_k)\}$ , where  $p_i \equiv \text{"}f_i \text{ is satisfiable.}"$  By the proof of Cook's theorem [C71] we have the following theorem.

**THEOREM 5.2.**  $SAT(k)$  is  $NP(k)$ -complete.

We continue our quest for natural complete languages, and display a family of variants of graph colorability that are complete for the levels of the boolean hierarchy.

**THEOREM 5.3.**  $COLORABILITY(k)$  is  $NP(k)$ -complete, where

$$COLORABILITY(k) = \{G \mid \chi(G) \text{ is odd} \wedge 3k \leq \chi(G) \leq 4k\}, \quad k \text{ even},$$

$$COLORABILITY(k) = \{G \mid (\chi(G) \text{ is odd} \wedge 3k \leq \chi(G) \leq 4k) \vee \chi(G) < 3k\}, \quad k \text{ odd}.$$



*Sketch of the proof of Theorem 5.3.* We reduce SAT( $k$ ) to COLORABILITY( $k$ ). Using the normal forms of boolean hierarchy sets [CH85, Props. 2.1.6, 2.1.3] we can efficiently transform  $\langle f_1, \dots, f_k \rangle$  to  $\langle g_1, \dots, g_k \rangle$  so that (1)  $\langle f_1, \dots, f_k \rangle \in \text{SAT}(k) \Leftrightarrow g(f) = \langle g_1, \dots, g_k \rangle \in \text{SAT}(k)$ , and (2) the  $g_i$  have the nice property that

$$(**) \quad (\forall i < j) \quad [g_j \in \text{SAT} \Rightarrow g_i \in \text{SAT}].$$

If  $G_i = (V_i, E_i)$ ,  $i = 1, 2$ , define  $G_1 \times G_2 = (V, E)$ , where  $V = V_1 \cup V_2$ , and  $E = E_1 \cup E_2 \cup \{\langle \nu_1, \nu_2 \rangle \mid \nu_i \in V_i\}$ . Clearly  $\chi(G_1 \times G_2) = \chi(G_1) + \chi(G_2)$ . Given  $\langle f_1, \dots, f_k \rangle$ , we transform this to  $\langle g_1, \dots, g_k \rangle$ , as explained in the previous paragraph. Let  $h$  be the polynomial time reduction from formulas to graphs of Cai and Meyer [CM85]. Crucially,  $\chi(h(F)) = 3$  if  $f \in \text{SAT}$  and  $\chi(h(F)) = 4$  if  $f \notin \text{SAT}$ . Let  $G_i = h(g_i)$ . Now, since the  $g_i$  satisfy (\*\*), the graph  $G = (\dots (G_1 \times G_2) \times G_3 \dots) \times G_k$  is just what we want;  $\langle f_1, \dots, f_k \rangle \in \text{SAT}(k) \Leftrightarrow G \in \text{COLORABILITY}(k)$ .  $\square$

Now, let  $f$  be a boolean formula. Then  $f$  is satisfiable  $\Leftrightarrow f' = (f \vee y) \wedge \bar{y}$  is satisfiable where  $y$  is a new variable. Furthermore,  $f'' = (f \vee y) \wedge z$  is always satisfiable. Using these observations, we can apply the standard transformation from SAT to 3SAT to Vertex Cover (VC) [GJ79]; we get a polynomial time transformation  $f \rightarrow (G_f, l_f)$ , so that if  $f$  is satisfiable, then the minimum size vertex cover for  $G_f$  is  $l_f$ ; otherwise, that minimum number is  $l_f + 1$ .

For each  $k \geq 0$ , define

$$\text{VC}(2k+2) = \{\langle G, l \rangle \mid \text{The minimum size VC of } G \text{ is } l+2i+1, 0 \leq i \leq k\},$$

$$\text{VC}(2k+1) = \{\langle G, l \rangle \mid \text{The minimum size VC of } G \text{ is } \leq l,$$

$$\text{or equal to } l+2i \text{ for some } 0 < i \leq k\}.$$

**THEOREM 5.4.** VC( $k$ ) is NP( $k$ )-complete.

*Proof of Theorem 5.4.* Start with formulae  $f_1, \dots, f_k$ . Using the normal forms of the boolean hierarchy [CH85, Props. 2.1.6, 2.1.3] we can without loss of generality assume that for all  $i < j$ ,  $[f_j \in \text{SAT} \Rightarrow f_i \in \text{SAT}]$  (see the proof of Theorem 5.3). Then apply the transformation described before to each  $f_i$  separately:  $f_i$  maps to  $\langle G_i, l_i \rangle$ . We get a graph  $G$  having  $k$  distinct components  $G_i$ ,  $1 \leq i \leq k$ . Let  $l = \sum l_i$ .

Clearly the minimum size of a vertex cover for  $G$  is  $l+j$ , where  $j$  = the number of  $f_i$ 's that are unsatisfiable. Hence,  $\langle f_1, \dots, f_k \rangle \in \text{SAT}(k) \Leftrightarrow \langle G, l \rangle \in \text{VC}(k)$ .  $\square$

We can prove similar completeness results for several variants of the Vertex Cover problem, such as independent set, clique, and others.

**6. Conclusions.** This paper explores the structure of the boolean hierarchy. The hierarchy responds flexibly to relativized control: there are worlds where the hierarchy is infinite and for every  $k$  there are worlds where it extends exactly  $k$  levels. It has only recently been discovered that the polynomial hierarchy can be similarly controlled [Y85], [K87b]. In relativized worlds the boolean hierarchy may have no complete languages, a behavior like that of UP, BPP, and  $\text{NP} \cap \text{coNP}$  [S82], [HI85], [HH86b] but unlike that of P, NP, and PSPACE [GJ79]. There are natural complete languages for the levels of the hierarchy and we have seen, via immunity, that the boolean hierarchy displays structural asymmetries between complementary classes.

These results supply a foundation of structural understanding for the boolean hierarchy. A companion paper [CGII] applies the hierarchy to extend the results of Karp and Lipton [KL80] on sparse oracles for NP, of Hartmanis, Immerman, and Sewelson [HIS83] on sparse sets in NP-P, and of Blass and Gurevich [BG82] of the weakness of counting classes.

**Acknowledgments.** We thank Professors R. Book, P. Odifreddi, U. Schöning, and S. Zachos for informative discussions.

## REFERENCES

- [AG87] A. AMIR AND W. GASARCH, *Polynomial terse sets*, in Proc. 2nd Annual Conference on Structure in Complexity Theory, 1987, pp. 22–27.
- [BG81] C. BENNETT AND J. GILL, *Relative to a random oracle,  $P^A \neq NP^A \neq coNP^A$  with probability 1*, SIAM J. Comput., 10 (1981), pp. 96–113.
- [BG82] A. BLASS AND Y. GUREVICH, *On the unique satisfiability problem*, Inform. and Control, 55 (1982), pp. 80–88.
- [C86] J. CAI, *On some most probable separations of complexity classes*, Ph.D. thesis, Computer Science Department, Cornell University, Ithaca, NY, 1986.
- [C71] S. COOK, *The complexity of theorem-proving procedures*, in Proc. 3rd Annual Symposium on the Theory of Computation, 1971, pp. 151–158.
- [CGII] J. CAI, T. GUNDERMANN, J. HARTMANIS, L. HEMACHANDRA, V. SEWELSON, K. WAGNER, AND G. WECHSUNG, *The boolean hierarchy II: applications*, SIAM J. Comput., to appear.
- [CH85] J. CAI AND L. HEMACHANDRA, *The boolean hierarchy: hardware over NP*, Tech. Report TR85-724, Computer Science Department, Cornell University Ithaca, NY, 1985.
- [CH86] ———, *The boolean hierarchy: hardware over NP*, Structure in Complexity Theory, A. Selman, ed., Lecture notes in Computer Science 223, Springer-Verlag, Berlin, New York, 1986, pp. 105–124.
- [CM85] J. CAI AND G. MEYER, *Graph minimal uncolorability is  $D^P$ -complete*, Tech. Report TR85-688, Computer Science Department, Cornell University, Ithaca, NY, June 1985; SIAM J. Comput., 16 (1987), pp. 259–277.
- [FSS81] M. FURST, J. SAXE, AND M. SIPSER, *Parity, circuits, and the polynomial-time hierarchy*, in Proc. 22nd Annual Symposium on Foundations of Computer Science, 1981, pp. 260–270.
- [GJ79] M. GAREY AND D. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
- [H14] F. HAUSDORFF, *Grundzüge der Mengenlehre*, Leipzig, 1914.
- [H84] H. HELLER, *Relativized polynomial hierarchies extending two levels*, Math. Systems Theory, 17 (1984), pp. 71–84.
- [HH86a] J. HARTMANIS AND L. HEMACHANDRA, *On sparse oracles separating feasible complexity classes*, in Proc. 3rd Annual Symposium on Theoretical Aspects of Computer Science (STACS '86), Lecture Notes in Computer Science 210, Springer-Verlag, Berlin, New York, 1986, pp. 321–333.
- [HH86b] ———, *Complexity classes without machines: on complete sets for UP*, in Proc. 13th Colloquium on Automata, Languages and Programming (ICALP '86), Lecture Notes in Computer Science, Springer-Verlag, Berlin, New York, 1986, pp. 123–135.
- [HI85] J. HARTMANIS AND N. IMMERMANN, *On complete problems for  $NP \cap coNP$* , in Proc. 12th Colloquium on Automata, Languages and Programming (ICALP '85), Lecture Notes in Computer Science 194, Springer-Verlag, Berlin, New York, 1985.
- [HIS83] J. HARTMANIS, N. IMMERMANN, AND V. SEWELSON, *Sparse sets in NP-P: EXPTIME versus NEXPTIME*, in Proc. 15th Annual Symposium on the Theory of Computing, 1983, pp. 382–391.
- [K87a] J. KADIN, *The polynomial hierarchy collapses if the boolean hierarchy collapses*, Tech. Report TR87-843, Computer Science Department, Cornell University, Ithaca, NY, 1987; this issue, pp. 1263–1282.
- [K87b] K. KO, *Relativized polynomial time hierarchies having exactly K levels*, in Proc. 20th Annual Symposium on the Theory of Computing, May, 1988, pp. 245–253.
- [KL80] R. KARP AND R. LIPTON, *Some connections between nonuniform and uniform complexity classes*, in Proc. 12th Annual Symposium on the Theory of Computation, 1980, pp. 302–309.
- [KSW86] J. KÖBLER, U. SCHÖNING, AND K. WAGNER, *The difference and the truth-table hierarchies for NP*, Tech. Report TR 7-86, EWH Koblenz, 1986.
- [PW85] C. PAPADIMITRIOU AND D. WOLFE, *The complexity of facets resolved*, in Proc. 26th Annual Symposium on Foundations of Computer Science, 1985.
- [PY82] C. PAPADIMITRIOU AND M. YANNAKAKIS, *The complexity of facets (and some facets of complexity)*, in Proc. 14th Annual Symposium on the Theory of Computation, 1982, pp. 255–260.

- [PZ82] C. PAPANIMITRIOU AND S. ZACHOS, *Two remarks on the complexity of counting*, MIT-LCS Tech. Report MIT/LCS/TM-228, Massachusetts Institute of Technology, Cambridge, MA, August 1982, 6th GI Conference on Theoretical Computer Science, Springer-Verlag, Berlin, New York, 1983.
- [R85] D. RUSSO, *Structural properties of complexity classes*, Ph.D. thesis; Department of Mathematics, University of California, Santa Barbara, CA March 1985.
- [S82] M. SIPSER, *On relativization and the existence of complete sets*, in Proc. 9th Colloquium on Automata, Languages and Programming (ICALP '82), Lecture Notes in Computer Science 140, Springer-Verlag, Berlin, New York, 1982, pp. 523–531.
- [W85a] G. WECHSUNG, *On the boolean closure of NP*, in Proc. 1985 International Conference on Fundamentals of Computation Theory, Lecture Notes in Computer Science, Springer-Verlag, Berlin, New York, 1985, pp. 485–493. (An unpublished precursor of this paper was coauthored by K. Wagner.)
- [W85b] ———, *On the boolean closure of NP*, Tech. Report N/85/44, Friedrich-Schiller-Universität, Jena, December 1985.
- [W85c] ———, *More about the closure of NP*, Tech. Report N/85/43, Friedrich-Schiller-Universität, Jena, December 1985.
- [W86] K. WAGNER, *More complicated questions about maxima and minima, and some closures of NP*, in Proc. 13th Colloquium on Automata, Languages and Programming (ICALP '86), Lecture Notes in Computer Science, Springer-Verlag, Berlin, New York, 1986.
- [Y83] Y. YESHA, *On certain polynomial-time truth-table reducibilities of complete sets to sparse sets*, SIAM J. Comput., 12 (1983), pp. 411–425.
- [Y85] A. YAO, *Separating the polynomial-time hierarchy by oracles*, in Proc. 26th Annual Symposium on Foundations of Computer Science, 1985.

## ON FINDING LOWEST COMMON ANCESTORS: SIMPLIFICATION AND PARALLELIZATION\*

BARUCH SCHIEBER† AND UZI VISHKIN‡

**Abstract.** We consider the following problem. Suppose a rooted tree  $T$  is available for preprocessing. Answer on-line queries requesting the lowest common ancestor for any pair of vertices in  $T$ . We present a linear time and space preprocessing algorithm that enables us to answer each query in  $O(1)$  time, as in Harel and Tarjan [*SIAM J. Comput.*, 13 (1984), pp. 338–355]. Our algorithm has the advantage of being simple and easily parallelizable. The resulting parallel preprocessing algorithm runs in logarithmic time using an optimal number of processors on an EREW PRAM. Each query is then answered in  $O(1)$  time using a single processor.

**Key words.** parallel algorithms, tree algorithms, lowest common ancestors

**AMS(MOS) subject classifications.** 05C05, 68Q10, 68Q20, 68R10

**1. Introduction.** We consider the following problem. Given a rooted tree  $T(V, E)$  for preprocessing, answer on-line LCA queries of the form, “Which vertex is the Lowest Common Ancestor (LCA) of  $x$  and  $y$ ?” for any pair of vertices  $x, y$  in  $T$ . (Let us denote such a query  $LCA(x, y)$ .) We present a preprocessing algorithm that runs in linear time and linear space on the serial RAM model. (For the definition of a random access machine (RAM) model see, e.g., [1].) Given this preprocessing, we show how to process each such LCA query in constant time.

We also consider parallelization of our algorithm. The model of parallel computation used is the exclusive-read exclusive-write (EREW) parallel random access machine (PRAM). A PRAM employs  $p$  synchronous processors all having access to a common memory. An EREW PRAM does not allow simultaneous access by more than one processor to the same memory location for either read or write purposes. See [11] for a survey of results concerning PRAMs.

Let  $\text{Seq}(n)$  be the fastest known worst-case running time of a sequential algorithm, where  $n$  is the length of the input for the problem at hand. A parallel algorithm that runs in  $O(\text{Seq}(n)/p)$  time using  $p$  processors is said to have *optimal speedup* or, more simply, to be *optimal*. A primary goal in parallel computation is to design optimal algorithms that also run as fast as possible.

Our preprocessing algorithm is easily parallelized to obtain an optimal parallel preprocessing algorithm that runs in  $O(\log n)$  time using  $n/\log n$  processors on an EREW PRAM, where  $n$  is the number of vertices in  $T$ . Parallelizing the query processing is straightforward, provided read conflicts are allowed:  $k$  queries can be processed in  $O(1)$  time using  $k$  processors.

---

\* Received by the editors March 3, 1987; accepted for publication (in revised form) February 22, 1988. This research was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contract DE-AC02-76ER03077.

† Department of Computer Science, School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel 69978. Present address, T. J. Watson Research Center, IBM Research Division, Yorktown Heights, New York 10598.

‡ Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. The research of this author was supported by National Science Foundation grant NSF-CCR-8615337, Office of Naval Research grant N00014-85-K-0046 and the Foundation for Research in Electronics, Computers and Communication, administered by the Israeli Academy of Sciences and Humanities.

In their extensive paper [5], Harel and Tarjan gave a serial algorithm for the same problem. The performance of their algorithm is the same as ours. However, our algorithm has two advantages: (1) It is considerably simpler in both the preprocessing stage and the query processing and (2) It leads to a simple parallel algorithm. Below, we discuss similarities and differences with respect to [5]. *Similarities*: Both algorithms use two basic observations: (1) It is possible to answer LCA queries in simple paths in constant time and (2) It is possible to answer LCA queries in complete binary trees in constant time. Both algorithms pack information regarding several vertices into a single  $O(\log n)$  bits number. *Differences*: The subtler part of both algorithms is to show how to use the above two observations for answering an LCA query. In this part, our approach is completely different. In the preprocessing stage we compute a mapping from the vertices of the input tree  $T$  to the vertices of a complete binary  $B$ . The mapping has two properties: (i) All the vertices of  $T$  mapped into the same vertex in  $B$  form a path and (ii) For each vertex  $v$  in  $T$ , the descendants of  $v$  are mapped into descendants of the image of  $v$  in  $B$ . This mapping, together with some additional information, enables us to answer an LCA query in constant time. In [5], on the other hand, the vertices of the input tree  $T$  are mapped to the vertices of an arbitrary tree of logarithmic height, called the compressed tree. The preprocessing consists of a quite involved manipulation of this compressed tree. This manipulation includes partitioning the compressed tree into three plies and preprocessing each ply separately and also embedding the compressed tree in a complete binary tree.

Consider a dynamic LCA problem which, interspersed with the LCA queries, are on-line deletions and insertions of edges. Reference [5] also gives algorithms for some cases of this problem. We do not consider this problem in the present paper.

Our parallel algorithm improves on the following results. Tsin [9] gave two parallel algorithms for the LCA problem. In his first algorithm both the preprocessing stage and the query processing take logarithmic time with a linear number of processors. In his second algorithm the preprocessing stage takes  $O(\log n)$  time using  $n^2$  processors and processing a query takes  $O(1)$  time using a single processor. Vishkin [12] includes a parallel algorithm for the LCA problem. The processing of an LCA query takes logarithmic time (as in the first algorithm of Tsin). The preprocessing stage takes  $O(\log n)$  time using  $n/\log n$  processors (as in the present paper).

Observe that using our parallel preprocessing algorithm we can process  $k$  off-line LCA queries in  $O(\log n)$  time using  $(n+k)/\log n$  processors, provided read conflicts are allowed. This affects the performance of parallel algorithms for three problems: (1) Given an undirected graph, orient its edges so that the resulting digraph is strongly connected (if such orientation is possible) [12]. (2) Computing an open-ear decomposition and  $st$ -numbering of a biconnected graph [8]. Using the new parallel connectivity and list-ranking algorithms of [3], it has become possible to solve each of these problems in logarithmic time using an optimal number of processors only when  $m \geq n \log n$ , where  $n$  is the number of vertices and  $m$  is the number of edges in the input graph. Our off-line LCA computation enables us to extend the range of optimal speedup logarithmic time parallel algorithms for these problems to sparser graphs, where  $m \geq n \log^* n$  as in the above connectivity algorithm. (3) Approximate string matching [6]. The new parallel suffix tree construction of [7] together with the present parallel LCA computation lead to a considerable simplification of the parallel algorithm of [6]. This simplification has already been described in [2].

The paper is organized as follows. Section 2 gives a high-level description of the algorithm. Section 3 describes the preprocessing stage. In § 4 we show how to process LCA queries in  $T$  using the outcome of the preprocessing stage. Section 5 presents parallelization of our preprocessing stage.

**2. High-level description.** The entire algorithm is based on the following two observations (made also in [5]): (1) Had our input tree been a simple path, it would have been possible to preprocess it (by way of computing the distance of each vertex from the root, as explained below) and later answer each LCA query in constant time. (2) Had our input tree been a complete binary tree, it would have been possible to preprocess it (by way of computing its inorder number, as explained below) and later to answer each LCA query in constant time.

The preprocessing stage assigns a number  $\text{INLABEL}(v)$  to each vertex  $v$  in  $T$ . Motivated by observation (1), these numbers satisfy the following *Path-Partition Property*: The  $\text{INLABEL}$  numbers partition the tree  $T$  into paths, called  $\text{INLABEL}$  paths. Each  $\text{INLABEL}$  path consists of the vertices that have the same  $\text{INLABEL}$  number.

Let  $B$  be the smallest complete binary tree having at least  $n$  vertices. Our description identifies each vertex in  $B$  by its inorder number. Motivated by observation (2), the  $\text{INLABEL}$  numbers also satisfy the following *Descendance-Preservation Property*: The  $\text{INLABEL}$  numbers map each vertex  $v$  in  $T$  into the vertex  $\text{INLABEL}(v)$  in  $B$ , such that the descendants of  $v$  are mapped into descendants of  $\text{INLABEL}(v)$  in  $B$  ( $v$  is considered both a descendant and an ancestor of itself).

Consider a vertex  $v$  in  $T$ . By the Descendance-Preservation Property all the ancestors of  $v$  are mapped into ancestors of  $\text{INLABEL}(v)$ . This implies that there are at most  $\log n$  distinct numbers among the  $\text{INLABEL}$  numbers of all the ancestors of  $v$ . Later, we show how to record all these  $\text{INLABEL}$  numbers using a single string of  $\log n$  bits. In the preprocessing stage we compute this string, for each vertex  $v$  in  $T$ , into  $\text{ASCENDANT}(v)$ .

In the preprocessing stage we also compute the table  $\text{HEAD}$ . It contains the highest vertex in every  $\text{INLABEL}$  path.

Section 4 describes how to process a query  $\text{LCA}(x, y)$  for any pair of vertices  $x, y$  in  $T$ . The processing breaks into two cases. The simpler case is where  $x$  and  $y$  belong to the same  $\text{INLABEL}$  path. In the preprocessing stage we compute for each vertex  $v$  in  $T$  its distance from the root into  $\text{LEVEL}(v)$ . So,  $\text{LCA}(x, y)$  is simply the vertex among  $x$  and  $y$  that is closer to the root. The more complicated case is where  $\text{INLABEL}(x) \neq \text{INLABEL}(y)$ . We proceed in four steps. In the first step, we find the  $\text{LCA}$  of  $\text{INLABEL}(x)$  and  $\text{INLABEL}(y)$  in the complete binary tree  $B$ , denoted by  $b$ . Let  $z = \text{LCA}(x, y)$  in  $T$ . In the second step, we find  $\text{INLABEL}(z)$ .  $\text{INLABEL}(z)$  is the lowest ancestor of  $b$  in  $B$  that is the  $\text{INLABEL}$  number of a common ancestor of  $x$  and  $y$  in  $T$ . For this, we use information provided by  $\text{ASCENDANT}(x)$  and  $\text{ASCENDANT}(y)$ . In the third and fourth steps we find  $z$  in the  $\text{INLABEL}$  path defined by  $\text{INLABEL}(z)$ . In the third step, we find the lowest ancestor of  $x$ , denoted  $\hat{x}$ , and the lowest ancestor of  $y$ , denoted  $\hat{y}$ , in the path defined by  $\text{INLABEL}(z)$  in  $T$ . This is done in an indirect fashion. Consider the path in  $B$  from  $\text{INLABEL}(z)$  to  $\text{INLABEL}(x)$ . We derive from  $\text{ASCENDANT}(x)$  the first  $\text{INLABEL}$  number (i.e., vertex of  $B$ ) of an ancestor of  $x$  in this path. Table  $\text{HEAD}$  gives the highest ancestor of  $x$  in  $T$  having this  $\text{INLABEL}$  number. Finally,  $\hat{x}$  is the father in  $T$  of this ancestor. We find  $\hat{y}$  similarly. In the fourth step we find  $z$ , which is simply the vertex among  $\hat{x}$  and  $\hat{y}$  that is closer to the root.

**3. The preprocessing stage.** The outcome of the preprocessing stage consists of labels that are assigned to the vertices of  $T$  and a look-up table, called  $\text{HEAD}$ . The label of each vertex  $v$  in  $T$  consists of three numbers:  $\text{INLABEL}(v)$ ,  $\text{ASCENDANT}(v)$ , and  $\text{LEVEL}(v)$ .

We start with computing  $\text{INLABEL}(v)$ , for each vertex  $v$  in  $T$ . This is done in two steps. After a discussion of these two steps we show how to implement them.

Let  $\text{PREORDER}(v)$  be the serial number of  $v$  in preorder traversal of  $T$  and  $\text{SIZE}(v)$  be the number of vertices in the subtree rooted at  $v$ . Definition of preorder traversal can be found, e.g., in [1, pp. 54–55].

*Step 1.* Compute  $\text{PREORDER}(v)$  and  $\text{SIZE}(v)$ .

We note that the  $\text{PREORDER}$  numbers of the vertices in the subtree rooted at  $v$  range between  $\text{PREORDER}(v)$  and  $\text{PREORDER}(v) + \text{SIZE}(v) - 1$ , and therefore, the closed interval  $[\text{PREORDER}(v), \text{PREORDER}(v) + \text{SIZE}(v) - 1]$  is called the *interval of  $v$* .

In Step 2 we consider the binary representation of the (integer) numbers in the interval of  $v$ . We remark that throughout this paper we alternately refer to numbers and to their binary representations. No confusion will arise.

*Step 2.* Find the (integer) number that has the maximal number of rightmost “0” bits in the interval of  $v$ . This number is assigned to  $\text{INLABEL}(v)$ .

For an example of computations described in this section see Fig. 3.1.

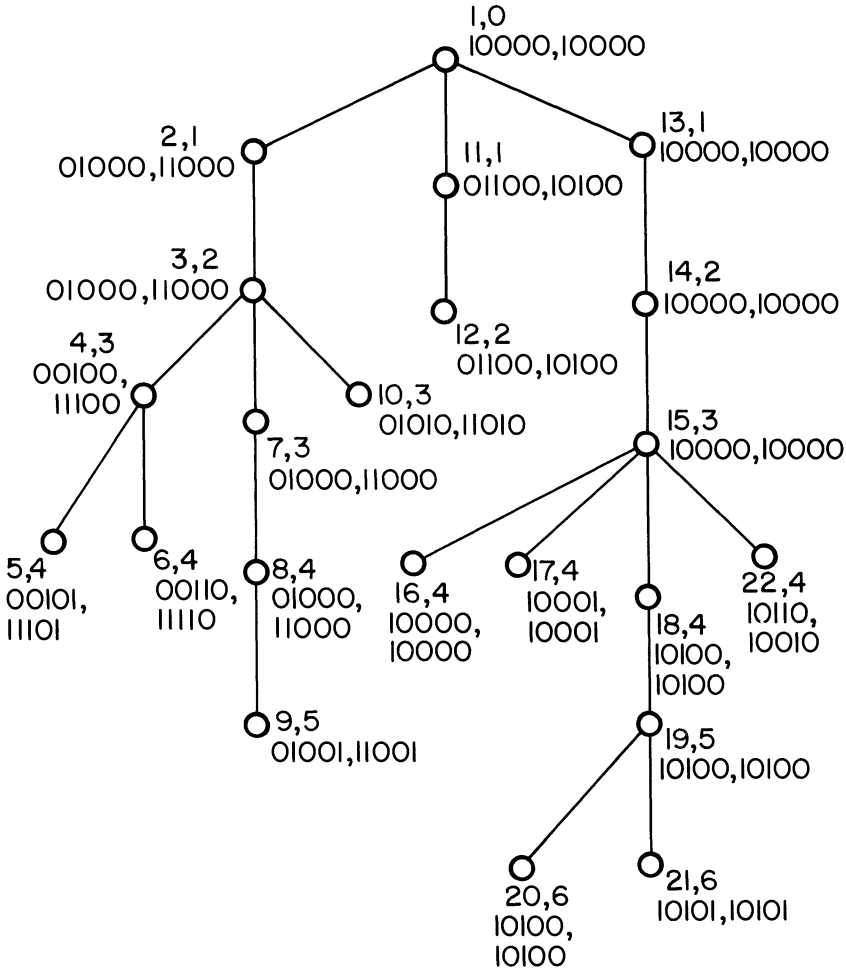


FIG. 3.1. Example. A tree with four numbers: PREORDER, LEVEL, INLABEL, and ASCENDANT at each vertex. (The last two numbers are given in binary representation.)

**Discussion.** We show that the INLABEL numbers satisfy the two properties defined in the high-level description of the previous section.

LEMMA 1. *The INLABEL numbers satisfy the Path-Partition Property.*

*Proof.* Observe that the intervals of the sons of  $v$  must be pairwise disjoint. Therefore, INLABEL( $v$ ) belongs to the interval of at most one son of  $v$ . Denote such a son by  $u$ . By the selection of the INLABEL numbers (Step 2), INLABEL( $u$ ) = INLABEL( $v$ ) (if  $u$  exists), and for any other son  $w$  of  $v$ , INLABEL( $w$ )  $\neq$  INLABEL( $v$ ). This implies the *Path-Partition Property* of the INLABEL numbers.  $\square$

LEMMA 2. *The INLABEL numbers satisfy the Descendance-Preservation Property.*

*Proof.* Let  $d$  be any descendant of  $v$  in  $T$ . We show that INLABEL( $d$ ) is a descendant of INLABEL( $v$ ) in the complete binary tree  $B$ . (Recall that our description identifies each vertex in  $B$  by its inorder number, thus proving the lemma.) Consider any two vertices  $b$  and  $c$  in  $B$ . We first give a necessary and sufficient condition for  $c$  to be a descendant of  $b$  in  $B$  and then show that INLABEL( $d$ ) and INLABEL( $v$ ) satisfy this condition. Let  $l = \lfloor \log n \rfloor + 1$  and  $i$  be the number of rightmost “0” bits in  $b$ . That is,  $b$  consists of  $l - i$  leftmost bits followed by a single “1” and  $i$  “0”s.

CLAIM. *A vertex  $c$  is a descendent of  $b$  if and only if (1) the  $l - i$  leftmost bits of  $c$  are the same as the  $l - i$  leftmost bits of  $b$ , and (2) the number of rightmost “0” bits in  $c$  is at most  $i$ .*

*Proof.* Let  $b_L$  and  $b_R$  be the left and right sons of  $b$ , respectively. It is not difficult to see the following: (i)  $b_L$  consists of the  $l - i$  leftmost bits of  $b$  followed by a single “0”, a single “1”, and  $i - 1$  “0”s; and (ii)  $b_R$  consists of the  $l - i$  leftmost bits of  $b$  followed by two “1”s and  $i - 1$  “0”s. These facts readily imply both directions of our claim.

For an example of a complete binary tree and its inorder numbering see Fig. 3.2.

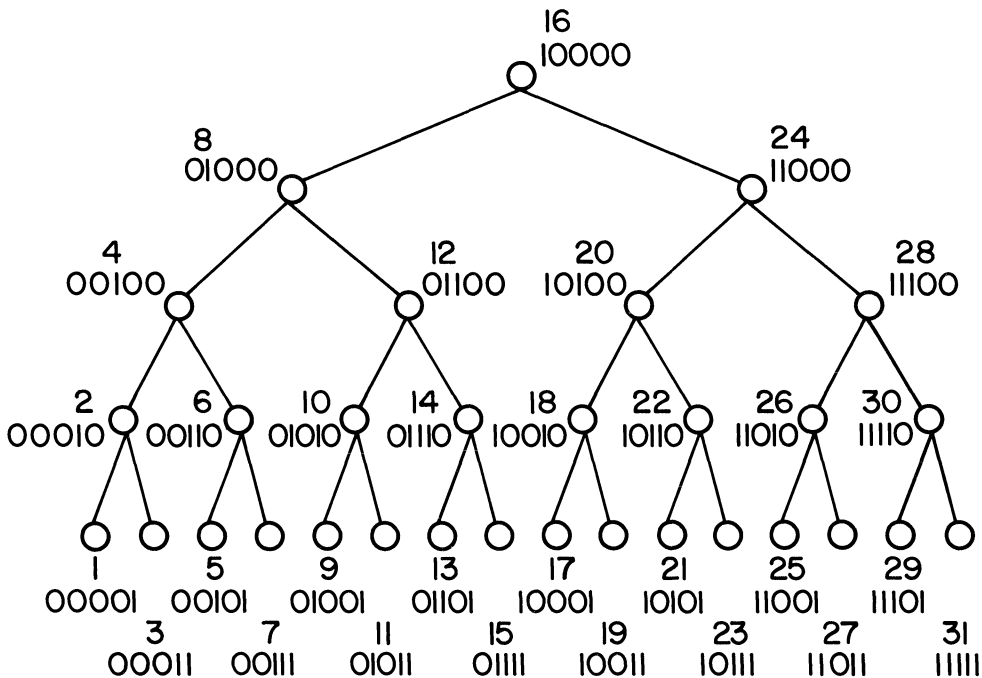


FIG. 3.2. Example. Inorder numbering of the complete binary tree with 31 vertices. (The numbers are given also in binary representation.)

<sup>1</sup> The base of all logarithms in this paper is two.



We return to the proof of Lemma 2. Let  $i$  be the number of rightmost “0” bits in  $\text{INLABEL}(v)$ . Since  $\text{INLABEL}(d)$  belongs to the interval of  $v$  and  $\text{INLABEL}(v)$  has the maximal number of rightmost “0” bits in this interval, the number of rightmost “0” bits in  $\text{INLABEL}(d)$  is at most  $i$ . The  $l-i$  leftmost bits are the same for all numbers in the interval. In particular, the  $l-i$  leftmost bits in  $\text{INLABEL}(d)$  are the same as the  $l-i$  leftmost bits in  $\text{INLABEL}(v)$ . This implies that  $\text{INLABEL}(d)$  is the descendant of  $\text{INLABEL}(v)$  in  $B$ . Lemma 2 follows.  $\square$

**Implementation.** Step (1) is implemented in linear time and linear space, using preorder traversal of  $T$ . Given  $\text{PREORDER}(v)$  and  $\text{SIZE}(v)$ , for each vertex  $v$  in  $T$ , Step (2) is implemented in constant time per vertex in two substeps.

*Step 2.1.* Compute  $\lfloor \log [(\text{PREORDER}(v)-1) \text{ xor } (\text{PREORDER}(v) + \text{SIZE}(v)-1)] \rfloor$  into  $i$ . Let us explain this. The bitwise logical exclusive OR (denoted **xor**) of  $\text{PREORDER}(v)-1$  and  $\text{PREORDER}(v)+\text{SIZE}(v)-1$  assigns “1” to each bit in which  $\text{PREORDER}(v)-1$  and  $\text{PREORDER}(v)+\text{SIZE}(v)-1$  differ. The floor of the (base two) logarithm gives the index of the leftmost bit of difference (counting from the rightmost bit whose index is zero). Note that the bit-indexed  $i$  must be “0” in  $\text{PREORDER}(v)-1$  and “1” in  $\text{PREORDER}(v)+\text{SIZE}(v)-1$ , since the second number is larger.

Step 2.2 shows how to “compose”  $\text{INLABEL}(v)$ . For this, we need two observations. (1) The  $l-i+1$  leftmost bits of  $\text{INLABEL}(v)$  are the same as the  $l-i+1$  leftmost bits in  $\text{PREORDER}(v)+\text{SIZE}(v)-1$ . (2) The  $i$  other bits in  $\text{INLABEL}(v)$  are “0”s.

*Step 2.2.* Compute  $2^i \lfloor (\text{PREORDER}(v)+\text{SIZE}(v)-1)/2^i \rfloor$  into  $\text{INLABEL}(v)$ . This assigns the  $l-i+1$  leftmost bits in  $\text{PREORDER}(v)+\text{SIZE}(v)-1$  to the  $l-i+1$  leftmost bits in  $\text{INLABEL}(v)$  and “0”s to the other bits of  $\text{INLABEL}(v)$ .

*Remark.* The above computation is based on  $\text{PREORDER}$  numbering of the vertices of  $T$ . This numbering has the property that the numbers assigned to the subtree rooted at any vertex of  $T$  provide a consecutive series of integers. In fact, any alternative numbering having this property (e.g.,  $\text{POSTORDER}$ ,  $\text{INORDER}$ ) will produce  $\text{INLABEL}$  numbers that will be suitable for our preprocessing stage.

We proceed to the computation of the  $\text{ASCENDANT}$  numbers. The general idea is that for each vertex  $v$ , the single number  $\text{ASCENDANT}(v)$  will record the  $\text{INLABEL}$  numbers of *all* the ancestors of  $v$  in  $T$ . We observe that, from the viewpoint of vertex  $v$  the  $\text{INLABEL}$  number of each of its ancestors can be fully specified by the index of its rightmost “1”. This is so because the bits that are to the left of this “1” are the same as their respective bits in  $\text{INLABEL}(v)$ . Like the  $\text{INLABEL}$  numbers,  $\text{ASCENDANT}(v)$  is also an  $(l+1)$ -bit number. Denote the binary representation of  $\text{ASCENDANT}(v)$  by the binary sequence  $A_l(v), \dots, A_0(v)$ . We set  $A_i(v) = 1$  only if  $i$  is the index of a rightmost “1” in the  $\text{INLABEL}$  number of an ancestor of  $v$  in  $T$ . To compute the  $\text{ASCENDANT}$  numbers, we scan the vertices of  $T$  from its root  $r$  down to its leaves (use, for instance, Breadth-First Search). We start with  $\text{ASCENDANT}(r) = 2^l$ . Consider an internal vertex  $v$  in  $T$  and let  $F(v)$  be the father of  $v$  in  $T$ . If  $\text{INLABEL}(v) = \text{INLABEL}(F(v))$  then we assign  $\text{ASCENDANT}(F(v))$  to  $\text{ASCENDANT}(v)$ ; otherwise, we assign  $\text{ASCENDANT}(F(v)) + 2^i$  to  $\text{ASCENDANT}(v)$ , where  $i$  is the index of the rightmost “1” in  $\text{INLABEL}(v)$ . It can be easily verified that  $i$  is given by  $\log(\text{INLABEL}(v) - [\text{INLABEL}(v) \text{ and } (\text{INLABEL}(v)-1)])$ , where **and** denotes bitwise logical AND.

Recall that  $\text{LEVEL}(v)$ , for each vertex  $v$  in  $T$ , is the distance, counting edges, of the path from  $v$  to the root  $r$ . Computation of the  $\text{LEVEL}$  numbers is straightforward and can be done using, e.g., Breadth-First Search.

Recall that Fig. 3.1 gives an example of the labels.

We conclude by describing how to compute the Table HEAD. HEAD( $k$ ) contains the vertex closest to the root in the path consisting of all vertices whose INLABEL number is  $k$ . HEAD( $k$ ) is sometimes called the *head* of the INLABEL path  $k$ . Computation of the table HEAD is trivial. For each vertex  $v$ , such that  $\text{INLABEL}(v) \neq \text{INLABEL}(F(v))$  we assign  $v$  to HEAD( $\text{INLABEL}(v)$ ). This, again, takes linear time and linear space.

*A general implementation remark.* The time bounds of both the preprocessing stage and the query processing depend on the ability to perform multiplication, division, powers-of-two computation, bitwise AND, base-two discrete logarithm, and bitwise exclusive OR in constant time. If these operations are not part of the machine's repertoire, look-up tables for each missing operation are prepared in linear time and linear space as part of the preprocessing stage. These tables will be used to perform the missing operations in  $O(1)$  operations in the repertoire.

**4. Processing LCA queries.** In this section we show how to answer LCA queries using the outcome of the preprocessing stage.

Consider a query LCA( $x, y$ ), for any pair of vertices  $x, y$  in  $T$ . (To illustrate the presentation the reader is referred to Fig. 3.1.) There are two cases.

*Case A.*  $\text{INLABEL}(x) = \text{INLABEL}(y)$ . It must be that  $x$  and  $y$  are in the same INLABEL path. We conclude that LCA( $x, y$ ) is  $x$  if  $\text{LEVEL}(x) \leq \text{LEVEL}(y)$  and  $y$  otherwise.

*Case B.*  $\text{INLABEL}(x) \neq \text{INLABEL}(y)$ . Let  $z$  be LCA( $x, y$ ). We find  $z$  in four steps:

*Step 1.* Find  $b$ , the LCA of INLABEL( $x$ ) and INLABEL( $y$ ) in the complete binary tree  $B$ , as follows. Let  $i$  be the index of the rightmost "1" in  $b$ . Since  $b$  is a common ancestor of INLABEL( $x$ ) and INLABEL( $y$ ) in  $B$ ,  $i$  must satisfy the following two conditions. (1) The  $l-i$  leftmost bits in INLABEL( $x$ ) and in INLABEL( $y$ ) are the same as these bits in  $b$ . (2) The index of the rightmost "1" in INLABEL( $x$ ) and in INLABEL( $y$ ) is at most  $i$ . Since  $b$  is the *lowest* common ancestor of INLABEL( $x$ ) and INLABEL( $y$ ) in  $B$ ,  $i$  is the *minimum* index satisfying both conditions. We distinguish three *cases*.

*Case (1).* INLABEL( $x$ ) is an ancestor of INLABEL( $y$ ). Let  $i_1$  be the index of the rightmost "1" in INLABEL( $x$ ). Note that in this case the  $l-i_1$  leftmost bits in INLABEL( $x$ ) and in INLABEL( $y$ ) are the same and that the index of the rightmost "1" in INLABEL( $y$ )  $< i_1$ . Hence,  $i$  equals  $i_1$ .

*Case (2).* INLABEL( $y$ ) is an ancestor of INLABEL( $x$ ). Similar to Case (1),  $i$  is the index of the rightmost "1" in INLABEL( $y$ ).

*Case (3).* Not cases (1) and (2). In this case  $i$  is the *minimum* index such that the  $l-i$  leftmost bits in INLABEL( $x$ ) and INLABEL( $y$ ) are the same.

We can deal with all three cases at once by simply taking  $i$  to be the maximum among the following: the index of the leftmost bit in which INLABEL( $x$ ) and INLABEL( $y$ ) differ; the index of the rightmost "1" in INLABEL( $x$ ); and the index of the rightmost "1" in INLABEL( $y$ ).  $b$  consists of the  $l-i$  leftmost bits in INLABEL( $x$ ) (or INLABEL( $y$ )) followed by a single "1" and  $i$  "0"s.

In Step 2 we find INLABEL( $z$ ) (where  $z$  is LCA( $x, y$ )). The Descendance-Preservation Property of the INLABEL numbers implies that INLABEL( $z$ ) is a common ancestor of INLABEL( $x$ ) and INLABEL( $y$ ). Notice that INLABEL( $z$ ) is not necessarily  $b$ , the *lowest* common ancestor of INLABEL( $x$ ) and INLABEL( $y$ ). This is so because the vertices in  $T$  mapped into  $b$  are not necessarily ancestors of  $x$

or  $y$ . However, it is not difficult to see that  $\text{INLABEL}(z)$  is the lowest ancestor of  $b$  in  $B$  that is the  $\text{INLABEL}$  number of an ancestor of both  $x$  and  $y$  in  $T$ .

*Step 2.* Find  $\text{INLABEL}(z)$ . For this we find the index of the rightmost “1” in  $\text{INLABEL}(z)$ , denoted by  $j$ . Since  $z$  is a common ancestor of  $x$  and  $y$  in  $T$ ,  $A_j(x) = 1$  and  $A_j(y) = 1$ . Since  $\text{INLABEL}(z)$  is the *lowest* ancestor of  $b$  that is a common ancestor of  $x$  and  $y$ , the index  $j$  must be the index of the *rightmost* “1” in  $A_l(x), \dots, A_i(x)$  and  $A_l(y), \dots, A_i(y)$ .  $\text{INLABEL}(z)$  consists of the  $l-j$  leftmost bits of  $\text{INLABEL}(x)$  (or  $\text{INLABEL}(y)$ ) followed by a single “1” and  $j$  “0”s.

In the next steps we find  $z$ , the lowest vertex in the path defined by  $\text{INLABEL}(z)$  that is a common ancestor of  $x$  and  $y$  in  $T$ . For this we find  $\hat{x}$ , the lowest ancestor of  $x$  in the path defined by  $\text{INLABEL}(z)$  and  $\hat{y}$ , the lowest ancestor of  $y$  in this same path.  $z$  is the highest vertex among these two vertices.

*Step 3.* Find  $\hat{x}$  and  $\hat{y}$ . We show how to find  $\hat{x}$ .  $\hat{y}$  is found similarly. If  $\text{INLABEL}(x) = \text{INLABEL}(z)$  then  $\hat{x} = x$  and nothing has to be done. Suppose  $\text{INLABEL}(x) \neq \text{INLABEL}(z)$ . We set the following intermediate goal, as the main step toward finding  $\hat{x}$ : Find the son of  $\hat{x}$  that is also an ancestor of  $x$ . Denote the vertex that we search by  $w$  and let  $k$  be the index of the rightmost “1” in  $\text{INLABEL}(w)$ . It is not difficult to verify that  $k$  is the index of the leftmost “1” in  $A_{j-1}(x), \dots, A_0(x)$ . So, we find  $k$ . Clearly,  $\text{INLABEL}(w)$  consists of the  $l-k$  leftmost bits of  $\text{INLABEL}(x)$  followed by a single “1” and  $k$  “0”s. Observe that  $w$  is the head of its  $\text{INLABEL}$  path (since the  $\text{INLABEL}$  number of its father  $\hat{x}$  is different from  $\text{INLABEL}(w)$ ). Therefore,  $w$  is  $\text{HEAD}(\text{INLABEL}(w))$  and our intermediate goal is achieved. Finally,  $\hat{x}$  is the father of  $w$ .

*Step 4.*  $\text{LCA}(x, y)$  is  $\hat{x}$  if  $\text{LEVEL}(\hat{x}) \leq \text{LEVEL}(\hat{y})$  and  $\hat{y}$  otherwise.

In the rest of this section we give additional implementation details required for the above processing.

*Step 1.* To find  $i$ , the index of the rightmost “1” in  $b$ , we do the following.

*Step 1.1.* Find  $i_1$ , the index of the rightmost “1” in  $\text{INLABEL}(x)$ , and  $i_2$ , the index of the rightmost “1” in  $\text{INLABEL}(y)$ . To find  $i_1$  we compute  $i_1 := \log(\text{INLABEL}(x) - [\text{INLABEL}(x) \text{ and } (\text{INLABEL}(x) - 1)])$ , as in the  $\text{ASCENDANT}$  numbers computation of the previous section.  $i_2$  is found similarly.

*Step 1.2.* Find  $i_3$ , the index of the leftmost bit in which  $\text{INLABEL}(x)$  and  $\text{INLABEL}(y)$  differ. To find  $i_3$  we compute  $i_3 := \lfloor \log[\text{INLABEL}(x) \text{ xor } \text{INLABEL}(y)] \rfloor$ . This is similar to Step 2.1 in the  $\text{INLABEL}$  numbers computation of the previous section.

$i$  is the maximum among  $i_1$ ,  $i_2$ , and  $i_3$ . Given  $i$ ,  $b$  can be computed similarly to Step 2.2 in the  $\text{INLABEL}$  numbers computation.

*Step 2.* To find  $j$  we do the following steps.

*Step 2.1.* Compute the bitwise logical AND of  $\text{ASCENDANT}(x)$  and  $\text{ASCENDANT}(y)$  into  $\text{COMMON}$ .

*Step 2.2.* Compute  $2^i \lfloor \text{COMMON} / 2^i \rfloor$  into  $\text{COMMON}_i$ .  $\text{COMMON}_i$  lists all the “1”s in both  $A_l(x), \dots, A_i(x)$  and  $A_l(y), \dots, A_i(y)$ .

*Step 2.3.*  $j$  is the index of the rightmost “1” in  $\text{COMMON}_i$ . To find  $j$  we compute  $j := \log(\text{COMMON}_i - [\text{COMMON}_i \text{ and } (\text{COMMON}_i - 1)])$ , as in the  $\text{ASCENDANT}$  numbers computation of the previous section.

The implementation of Step 3 uses the same techniques.

**5. The parallel preprocessing algorithm.** In this section we describe the parallel version of our preprocessing stage. It runs in  $O(\log n)$  time using  $n/\log n$  processors.

We make the following assumption regarding the representation of the input tree  $T$ . Its  $n - 1$  edges are given in an array, where the incoming edges of each vertex are grouped successively. By our definition of the tree  $T$ , its edges are directed towards the root.

*Computing the labels in parallel.* To compute the labels of the vertices in  $T$  we apply the Euler tour technique for computing tree functions, which was given in [10] and [12]. We will implement it, however, using the  $O(\log n)$  time optimal parallel list-ranking algorithm of [3]. This list-ranking algorithm is designed for an EREW PRAM. It is based on expander graphs and its  $O(\log n)$  time bound hides a constant that is not very small. We note that [4] recently gave an alternative list-ranking algorithm with the same time and processor efficiencies. This alternative algorithm is designed for a PRAM that allows simultaneous access to the same memory location for both read and write purposes (called CRCW PRAM). It is simpler and its  $O(\log n)$  time bound requires a small constant.

Below, we first recollect the construction required for the Euler tour technique. We then show how to use it for computing the labels. The only reason we were forced to present anew the Euler tour technique is that the computation of the ASCENDANT numbers has not appeared elsewhere.

*Step 1.* For each edge  $(v \rightarrow u)$  in  $T$  we add its antiparallel edge  $(u \rightarrow v)$ . Let  $H$  denote the new graph.

Since the indegree and outdegree of each vertex in  $H$  are the same,  $H$  has an Euler path that starts and ends in  $r$ . Step 2 computes this path into the vector of pointers  $D$ , where for each edge  $e$  of  $H$ ,  $D(e)$  will have the successor edge of  $e$  in the Euler path.

*Step 2.* For each vertex  $v$  of  $H$  we do the following: (Let the outgoing edges of  $v$  be  $(v \rightarrow u_0), \dots, (v \rightarrow u_{d-1})$ .)  $D(u_i \rightarrow v) := (v \rightarrow u_{(i+1) \bmod d})$ , for  $i = 0, \dots, d - 1$ . Now  $D$  has an Euler circuit. The “correction”  $D(u_{d-1} \rightarrow r) := \text{end-of-list}$  (where the outdegree of  $r$  is  $d$ ) gives an Euler path which starts and ends in  $r$ .

We show how to use the Euler path in order to find  $\text{PREORDER}(v)$ ,  $\text{PREORDER}(v) + \text{SIZE}(v) - 1$ , and  $\text{LEVEL}(v)$  for each vertex  $v$  in  $T$ .

*Step 3.* We assign two weights:  $W_1(e)$  and  $W_2(e)$  to each edge  $e$  in the Euler path as follows. (1)  $W_1(e) = 1$  if  $e$  is directed from  $r$  (that is, if  $e$  is not a tree edge), and  $W_1(e) = 0$  otherwise. (2)  $W_2(e) = 1$  if  $e$  is directed from  $r$ , and  $W_2(e) = -1$  otherwise.

*Step 4.* We apply twice an optimal logarithmic time parallel list-ranking algorithm to find for each  $e$  in  $H$  its (weighted) distance from the *start* of the Euler path: The first application is relative to the weights  $W_1$  and the result is stored in  $\text{DISTANCE}_1(e)$ ; the second application is relative to the weights  $W_2$  and the result is stored in  $\text{DISTANCE}_2(e)$ . Consider a vertex  $v \neq r$  and let  $u$  be its father in  $T$ .  $\text{PREORDER}(v)$  is  $\text{DISTANCE}_1(u \rightarrow v) + 1$ ,  $\text{PREORDER}(v) + \text{SIZE}(v) - 1$  is  $\text{DISTANCE}_1(v \rightarrow u) + 1$ , and  $\text{LEVEL}(v)$  is  $\text{DISTANCE}_2(u \rightarrow v)$ . (These claims can be readily verified by the reader.)

*Step 5.* Given  $\text{PREORDER}(v)$  and  $\text{PREORDER}(v) + \text{SIZE}(v) - 1$  for each vertex  $v$  in  $T$  we compute  $\text{INLABEL}(v)$  in constant time using  $n$  processors as in the serial algorithm.

Next, we show how to use the Euler path in order to find  $\text{ASCENDANT}(v)$  for each vertex  $v$  in  $T$ .

*Step 6.* We assign a (new) weight  $W(e)$  to each edge  $e$  in the Euler path as follows. For each vertex  $v \neq r$  we do the following. Let  $u$  be the father of  $v$  in  $T$  and let  $i$  be the index of the rightmost “1” in  $\text{INLABEL}(v)$ . If  $\text{INLABEL}(v) \neq \text{INLABEL}(u)$ , we assign  $W(u \rightarrow v) = 2^i$  and  $W(v \rightarrow u) = -2^i$ . The weight of all other edges is set to zero.

*Step 7.* We again apply a parallel list-ranking algorithm to find for each  $e$  in  $H$  its (weighted) distance from the start of the Euler path. Consider a vertex  $v \neq r$  and let  $u$  be its father in  $T$ .  $\text{ASCENDANT}(v)$  is the distance of the edge  $(u \rightarrow v)$  plus  $2^l$ . Clearly,  $\text{ASCENDANT}(r) = 2^l$ .

We note that, given the labels, the table HEAD can be computed in constant time using  $n$  processors.

*Complexity.* Each of steps 4 and 7 needs  $n/\log n$  processors and  $O(\log n)$  time. Each of Steps 1, 2, 3, 5, 6 and the computation of HEAD needs  $n$  processors and  $O(1)$  time and can be readily simulated by  $n/\log n$  processors in  $O(\log n)$  time. Thus, the parallel preprocessing stage can be done in a total  $O(\log n)$  time using  $n/\log n$  processors.

**Acknowledgments.** We are grateful to Noga Alon and Yael Maon for stimulating discussions. We also thank the anonymous referee for drawing our attention to [9].

#### REFERENCES

- [1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [2] A. APOSTOLICO, C. ILIOPOULOS, G. M. LANDAU, B. SCHIEBER, AND U. VISHKIN, *Parallel construction of a suffix tree with applications*, Algorithmica Special Issue on Parallel and Distributed Computing, to appear.
- [3] R. COLE AND U. VISHKIN, *Approximate and exact parallel scheduling with applications to list, tree and graph problems*, in Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 478–491.
- [4] ———, *Faster optimal parallel prefix sums and list ranking*, TR 56/86, the Moise and Frida Eskenasy Institute of Computer Science, Tel Aviv University, Tel Aviv, Israel, 1986.
- [5] D. HAREL AND R. E. TARJAN, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.
- [6] G. M. LANDAU AND U. VISHKIN, *Introducing efficient parallelism into approximate string matching*, in Proc. 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 220–230.
- [7] G. M. LANDAU, B. SCHIEBER, AND U. VISHKIN, *Parallel construction of a suffix tree*, in Proc. 14th Internat. Colloquium on Automata Language and Programming, Lecture Notes in Computer Science 267, Springer-Verlag, Berlin, New York, 1987, pp. 314–325.
- [8] Y. MAON, B. SCHIEBER, AND U. VISHKIN, *Parallel ear decomposition search (EDS) and st-numbering in graphs*, Theoret. Comput. Sci., 47 (1986), pp. 277–298.
- [9] Y. H. TSIN, *Finding lowest common ancestors in parallel*, IEEE Trans. Comput., 35 (1986), pp. 764–769.
- [10] R. E. TARJAN AND U. VISHKIN, *An efficient parallel biconnectivity algorithm*, SIAM J. Comput., 14 (1985), pp. 862–874.
- [11] U. VISHKIN, *Synchronous parallel computation—a survey*, TR-71, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, NY, 1983.
- [12] ———, *On efficient parallel strong orientation*, Inform. Process. Lett., 20 (1985), pp. 235–240.

## THE POLYNOMIAL TIME HIERARCHY COLLAPSES IF THE BOOLEAN HIERARCHY COLLAPSES\*

JIM KADIN†

**Abstract.** It is shown that if the Boolean hierarchy (BH) collapses, then there exists a sparse set  $S$  such that  $\text{co-NP} \subseteq \text{NP}^S$ , and therefore the polynomial time hierarchy (PH) collapses to  $\text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$ , a subclass of  $\Delta_3^{\text{P}}$ . Since the BH is contained in  $\text{P}^{\text{NP}}$ , these results relate the internal structure of  $\text{P}^{\text{NP}}$  to the structure of the PH as a whole. Other conditions that imply the collapse of the BH (and the collapse of the PH in turn) include  $\text{D}^{\text{P}} = \text{co-D}^{\text{P}}$ ,  $\text{P}^{\text{NP}[k]} = \text{P}^{\text{NP}[k+1]}$  for any  $k$ , and  $\text{P}^{\text{NP}\parallel[k]} = \text{P}^{\text{NP}\parallel[k+1]}$  for any  $k$ .  $\text{P}^{\text{NP}[i]}$  is the class of languages recognizable in polynomial time with at most  $i$  queries to an oracle from NP, and  $\text{P}^{\text{NP}\parallel[i]}$  is the class of languages recognizable with at most  $i$  parallel queries to an oracle from NP.

**Key words.** polynomial time hierarchy, Boolean hierarchy, polynomial time Turing reductions, oracle access, nonuniform algorithms, sparse sets

**AMS(MOS) subject classifications.** 68Q15, 03D15, 03D20

**1. Introduction.** Many different complexity hierarchies have recently been shown to collapse. Hemachandra showed that the strong exponential hierarchy collapses to its second level [7]. Several proofs appeared showing that the logarithmic space and linear space hierarchies collapse to their second levels [15], [23], [20]. Then Immerman and Szelepcsényi independently showed that nondeterministic space classes are closed under complementation which implies that the logarithmic and linear space hierarchies collapse all the way to their first levels, nondeterministic logarithmic and linear space, respectively [9], [22]. These results have greatly increased our understanding of resource-bounded computation and have simplified the complexity theory world.

In this paper we present strong evidence that three other lesser-known hierarchies, the Boolean hierarchy (BH), the query hierarchy (QH), and the parallel query hierarchy ( $\text{QH}_{\parallel}$ ), do not collapse. We show that if the BH, QH, or  $\text{QH}_{\parallel}$  collapses, then the polynomial time hierarchy (PH) collapses to  $\text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$ , the class of languages recognizable in deterministic polynomial time with  $O(\log n)$  queries to an oracle for  $\text{NP}^{\text{NP}}$ .  $\text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$  is a subclass of  $\Delta_3^{\text{P}}$ , the third level of the PH. If the PH is an infinite hierarchy as most researchers believe, then the BH, QH, and  $\text{QH}_{\parallel}$  are also infinite hierarchies.

The BH, QH, and  $\text{QH}_{\parallel}$  contain  $\text{NP} \cup \text{co-NP}$  and intertwine to form a rich structure inside  $\text{P}^{\text{NP}}$ . By relating these hierarchies to the PH, our results provide a deeper understanding of the PH. We see that the internal structure of  $\text{P}^{\text{NP}}$  and issues concerning how polynomial time machines can access the information from NP oracles are tied to the structure of the PH as a whole. This is also a downward structural result. We show that the structure of the BH above NP and co-NP affects the structure of the set of unsatisfiable Boolean formulas and thus co-NP itself.

The QH is  $\bigcup_k \text{P}^{\text{NP}[k]}$ , where  $\text{P}^{\text{NP}[k]}$  is the class of languages recognizable in deterministic polynomial time with at most  $k$  queries to an oracle in NP.

The  $\text{QH}_{\parallel}$  is  $\bigcup_k \text{P}^{\text{NP}\parallel[k]}$ , where  $\text{P}^{\text{NP}\parallel[k]}$  is the class of languages recognizable in

\* Received by the editors November 30, 1987; accepted for publication (in revised form) February 26, 1988. This research was supported by National Science Foundation research grant DCR-8520597.

† Department of Computer Science, Upson Hall, Cornell University, Ithaca, New York 14853.

deterministic polynomial time with at most  $k$  parallel queries to an oracle in NP. A machine makes its queries in parallel if it computes all the strings that are to be queried before it receives any query answers from the oracle. The  $QH_{\parallel}$  equals  $BTT[NP]$ , the class of languages that are polynomial time bounded truth-table reducible to NP sets. Polynomial time bounded truth-table reducibilities are defined in [14].

The BH, defined by Cai, Gundermann, Hartmanis, Hemachandra, Sewelson, Wagner, and Wechsung [4] is a generalization of the class  $D^P$ .  $D^P$  was defined by Papadimitriou and Yannakakis to fully capture the complexity of several natural problems related to NP-optimization problems [19]. A language is in  $D^P$  if it is the difference of two NP languages:

$$D^P \stackrel{\text{def}}{=} \{L_1 - L_2 \mid L_1, L_2 \in NP\}.$$

Equivalently,  $L \in D^P$  if  $L = L_1 \cap \overline{L_2}$ , where  $L_1, L_2 \in NP$ . TSP facets, exact clique, and graph minimal uncolorability are  $\leq_m^P$ -complete for  $D^P$  [18], [19], [5].

The class  $\text{co-}D^P$  consists of the languages whose complements are in  $D^P$ . A language  $L$  is in  $\text{co-}D^P$  if  $L = \overline{L_1} \cup L_2$ , where  $L_1, L_2 \in NP$ . It is unknown whether  $D^P = \text{co-}D^P$ .

The first level of the BH,  $BH(1)$ , is defined to be NP. The second level,  $BH(2)$ , is  $D^P$ . Generalizing  $D^P$ , which is defined by one Boolean operator over NP predicates, the subsequent levels of the BH are defined by allowing more Boolean operations over NP predicates. See § 3 for a precise definition.  $BH \stackrel{\text{def}}{=} \bigcup_k BH(k)$ . For each  $k$ ,  $\text{co-}BH(k)$  is the class of languages whose complements are in  $BH(k)$ . It is unknown whether any level of the BH is closed under complementation, but it is shown in [4] that for all  $k$ ,

$$BH(k) = \text{co-}BH(k) \iff BH = BH(k).$$

The levels of the BH, QH, and  $QH_{\parallel}$  intertwine (see § 3). Therefore  $BH = QH = QH_{\parallel} = BTT[NP]$ , and either all three hierarchies collapse or all three are infinite hierarchies. All three hierarchies are contained in  $P^{NP}$ , the  $\Delta_2^P$  level of the PH:

$$NP \cup \text{co-NP} \subseteq P^{NP[1]} \subseteq D^P \subseteq P^{NP[2]} \subseteq P^{NP[2]} \subseteq \dots \subseteq QH = QH_{\parallel} = BH \subseteq P^{NP}.$$

Our proof that the collapse of these hierarchies causes the PH to collapse depends on the structure of the BH. We show that if the BH collapses, then there exist nonuniform NP algorithms for the languages in  $\text{co-NP}$ . That is, every language in  $\text{co-NP}$  can be recognized by an NP machine with access to a sparse oracle. A set  $S$  is sparse if the number of strings in  $S$  of each length is bounded by a polynomial.

**THEOREM.** *For all  $k$ , if  $BH(k) = \text{co-}BH(k)$ , then there exists a sparse set  $S \in NP^{NP}$  such that  $\text{co-NP} \subseteq NP^S$ .*

By the results of Yap in [26], the existence of a sparse set  $S$  such that  $\text{co-NP} \subseteq NP^S$  implies  $PH \subseteq NP^{NP^{NP}}$ , the  $\Sigma_3^P$  level of the PH. We relativize the main result in [10] to show that if the sparse set is in  $NP^{NP}$ , then the PH collapses further.

**THEOREM.** *If there exists a sparse set  $S \in NP^{NP}$  such that  $\text{co-NP} \subseteq NP^S$ , then  $PH \subseteq P^{NP^{NP}[O(\log n)]}$ .*

These two theorems together imply our main result.

**THEOREM.** *For all  $k$ , if  $BH(k) = \text{co-}BH(k)$ , then  $PH \subseteq P^{NP^{NP}[O(\log n)]}$ .*

**COROLLARY.** *If  $D^P = \text{co-}D^P$ , then  $PH \subseteq P^{NP^{NP}[O(\log n)]}$ .*

Since the collapse of the QH or the  $QH_{\parallel}$  implies the collapse of the BH, we also have the following corollaries.

COROLLARY. For all  $k$ , if  $P^{NP[k]} = P^{NP[k+1]}$ , then  $PH \subseteq P^{NP^{NP}[O(\log n)]}$ .

COROLLARY. For all  $k$ , if  $P^{NP\parallel[k]} = P^{NP\parallel[k+1]}$ , then  $PH \subseteq P^{NP^{NP}[O(\log n)]}$ .

Therefore the structure of the PH as a whole is closely related to the structure of  $P^{NP}$  and to the question of whether deterministic polynomial time machines can compute more if they are allowed to make more NP queries.

Our proof also shows that the structure within  $P^{NP}$  is related to co-NP itself. The existence of the sparse set  $S$  such that  $co-NP \subseteq NP^S$  is really quite a strong statement about the structure of unsatisfiable Boolean formulas. Indeed, the set  $S$  that we construct consists mainly of key unsatisfiable formulas, a constant number of each length, such that knowing these key formulas provides an NP algorithm for recognizing all unsatisfiable formulas of that length. In particular, if  $D^P = co-D^P$ , then there exists one key string of each length.

These results also relativize to the other  $\Delta_i^P$  levels of the PH. We can define a Boolean hierarchy and query hierarchy within each  $P^{\Sigma_i^P}$  and conclude that

$$PH \text{ collapses} \iff \exists i, k, P^{\Sigma_i^P[k]} = P^{\Sigma_i^P[k+1]}.$$

This work is also related to Krentel's results for classes of functions. He showed that for functions  $f(n)$  and  $g(n)$  with  $f(n) \leq \frac{1}{2} \log n$  and  $f(n) < g(n)$ , if  $FP^{NP[f(n)]} = FP^{NP[g(n)]}$ , then  $P = NP$  [13].  $FP^{NP[h(n)]}$  is the class of functions computable in polynomial time with  $h(n)$  queries to an oracle for NP. In particular,

$$FP^{NP[k]} = FP^{NP[k+1]} \implies P = NP.$$

Our result that

$$P^{NP[k]} = P^{NP[k+1]} \implies PH \text{ collapses}$$

has a weaker hypothesis and a weaker, yet similar, conclusion.

**2. Preliminaries.** The reader is assumed to be familiar with P, NP, oracle Turing machines, the NP-complete set SAT, the set of satisfiable Boolean formulas, and  $\overline{SAT}$ , the set of unsatisfiable Boolean formulas. All of these topics are covered in [8]. We also assume the reader is familiar with the PH defined in [21].

For any Turing machine  $N$ ,  $|N|$  is the size of the state transition table of  $N$ .

For any string  $x$ ,  $|x|$  is the length of  $x$ .

For any set of strings  $C$ ,  $C^{=n}$  is the set of strings in  $C$  of length  $n$ .  $C^{\leq n}$  is the set of strings in  $C$  of length less than or equal to  $n$ .

We write  $\|C\|$  for the cardinality of any set  $C$ .

For any two sets of strings  $B$  and  $C$ , the *disjoint union* of  $B$  and  $C$  is

$$B \oplus C \stackrel{\text{def}}{=} \{0x \mid x \in B\} \cup \{1x \mid x \in C\}.$$

For  $k$  sets  $B_1, \dots, B_k$ , this extends to

$$B_1 \oplus \dots \oplus B_k \stackrel{\text{def}}{=} \bigcup_i \{\underline{i}x \mid x \in B_i\},$$

where  $\underline{i}$  is the bit pattern of  $\lceil \log k \rceil$  bits representing  $i$  in binary.



DEFINITION 2.1. *A set of strings  $S$  is sparse if there exists a polynomial  $p(\cdot)$  such that for all  $n$ , the number of strings in  $S$  of length  $n$  is at most  $p(n)$ , i.e.,  $\|S^{\leq n}\| \leq p(n)$ .*

DEFINITION 2.2. *For any set of strings  $S$ , the census function of  $S$  gives the number of strings in  $S$  of length at most  $n$ :  $\text{Census}_S(n) \stackrel{\text{def}}{=} \|S^{\leq n}\|$ .*

There is a standard technique of using the prefixes of strings in a sparse set to find the strings in the set. For any sparse set  $S$ , let

$$\text{prefix}(S) \stackrel{\text{def}}{=} \{y\#^i \mid i \geq 0, \exists w \text{ with } |w|=i \text{ and } yw \in S\},$$

where “#” is a new alphabet symbol. The set  $\text{prefix}(S)$  is the set of *marked prefixes* of  $S$ . For every  $n$ ,  $\text{prefix}(S)$  contains at most  $n$  times as many strings of length  $n$  as  $S$  does. Therefore  $\text{prefix}(S)$  is sparse if  $S$  is. A deterministic polynomial time machine with an oracle for  $\text{prefix}(S)$  can generate all the strings in  $S^{\leq n}$  on input  $1^n$ . The machine builds up each string one character at a time by asking the oracle about longer and longer prefixes. A machine can actually generate all the strings in  $\text{prefix}(S)$  up to a given length this way. If we add all the strings in  $\text{prefix}(S)$  to  $S$ , we say  $S$  is *prefix marked*.

DEFINITION 2.3. *A set  $S$  is  $P^B$ -printable for an oracle  $B$  if there exists a  $P^B$  machine that prints all the strings in  $S^{\leq n}$  on input  $1^n$ .*

DEFINITION 2.4. *A set  $S$  is self-P-printable if  $S$  is  $P^S$ -printable.*

Thus if a sparse set  $S$  is prefix marked, then  $S$  is self-P-printable. Note that  $S \subseteq \text{prefix}(S)$ , and clearly  $\text{NP}^S \subseteq \text{NP}^{\text{prefix}(S)}$ .

**3. The hierarchies.** The query hierarchy, QH, and the parallel query hierarchy,  $\text{QH}_{\parallel}$ , arise quite naturally from questions about the structure of  $\text{P}^{\text{NP}}$  and the power of NP queries in general.

DEFINITION 3.1. *For  $k \geq 1$ ,  $\text{P}^{\text{NP}[k]}$  is the class of languages recognized by deterministic polynomial time oracle machines that make at most  $k$  queries to an oracle for NP.*

DEFINITION 3.2.  $\text{QH} \stackrel{\text{def}}{=} \bigcup_k \text{P}^{\text{NP}[k]}$ .

Unrestricted  $\text{P}^{\text{NP}}$  and  $\text{P}^{\text{NP}[k]}$  machines make their queries in a serial fashion: the answers to previous queries can be used to determine which string to query next. An alternative is to consider machines that are required to make their queries all at once, in parallel. Parallel query machines write down a list of query strings on their oracle tape and then enter the query state. The oracle replaces the query strings with a bit vector representing the answers to the queries. No further queries are permitted. Thus parallel query machines must compute all their queries in polynomial time without access to any oracle.

DEFINITION 3.3.  $\text{P}^{\text{NP}\parallel}$  is the class of languages recognizable by deterministic polynomial time machines that make (polynomially many) parallel queries to an oracle for NP.

DEFINITION 3.4. *For  $k \geq 1$ ,  $\text{P}^{\text{NP}\parallel[k]}$  is the class of languages recognized by deterministic polynomial time oracle machines that make at most  $k$  parallel queries to an oracle for NP.*

DEFINITION 3.5.  $\text{QH}_{\parallel} \stackrel{\text{def}}{=} \bigcup_k \text{P}^{\text{NP}\parallel[k]}$ .

Clearly,  $\text{P}^{\text{NP}\parallel} \subseteq \text{P}^{\text{NP}}$ , and for all  $k$ ,  $\text{P}^{\text{NP}\parallel[k]} \subseteq \text{P}^{\text{NP}[k]}$ .

Polynomial time truth-table and bounded truth-table reducibilities were defined in [14]. It is clear from the definitions that  $\text{P}^{\text{NP}\parallel} = \text{TT}[\text{NP}]$ , the class of languages polynomial time truth-table reducible to a language in NP. Similarly,  $\text{QH}_{\parallel} =$

BTT[NP], the class of languages that are bounded truth-table reducible to an NP language.

A related class is the set of languages recognizable with  $O(\log n)$  queries to an NP oracle.

DEFINITION 3.6.  $P^{NP[O(\log n)]}$  is the class of languages recognizable in deterministic polynomial time with  $O(\log n)$  queries to an NP oracle.

Clearly,  $QH \subseteq P^{NP[O(\log n)]}$ .

Since SAT is NP-complete, any queries made to an oracle for an NP language can be translated to queries to SAT without changing the number of queries made during the computation. Thus  $P^{SAT[k]} = P^{NP[k]}$ ,  $P^{SAT\parallel[k]} = P^{NP\parallel[k]}$ ,  $P^{SAT[O(\log n)]} = P^{NP[O(\log n)]}$ , etc.

It is easy to see that  $P^{NP[O(\log n)]} \subseteq P^{NP\parallel}$ . Given any  $P^{NP[O(\log n)]}$  machine and input  $x$ , we can run the machine on  $x$  trying both answers to each query and see what queries it asks with each answer sequence. Since the machine makes only  $O(\log n)$  queries no matter how the queries are answered, the number of different answer sequences and the number of possible queries is bounded by a polynomial in  $|x|$ . Therefore a polynomial time machine can generate all the possible queries and then determine if the  $P^{NP[O(\log n)]}$  machine accepts  $x$  by asking all the queries in parallel. Hence  $P^{NP[O(\log n)]} \subseteq P^{NP\parallel}$ . By a similar argument,  $P^{NP[k]} \subseteq P^{NP\parallel[2^k]}$ , and hence the QH and  $QH_{\parallel}$  intertwine, and  $QH = QH_{\parallel}$ .

Exploiting the NP-completeness of SAT, Hemachandra showed that  $P^{NP\parallel} \subseteq P^{NP[O(\log n)]}$  [7]. Therefore  $P^{NP\parallel} = P^{NP[O(\log n)]}$ .

In summary,

$$NP \cup \text{co-NP} \subseteq P^{NP[1]} \subseteq P^{NP[2]} \subseteq \dots \subseteq QH = QH_{\parallel} = \text{BTT[NP]} \\ \subseteq P^{NP[O(\log n)]} = P^{NP\parallel} \subseteq P^{NP}.$$

The BH is defined in [4]. It is a generalization of the class  $D^P$ , defined by Papadimitriou and Yannakakis [19].  $D^P$  has stirred a lot of interest because it contains many languages related to NP optimization problems. There are several ways of defining the BH and many normal forms for each level. We use the definition that most explicitly shows the structure that we will exploit later.

DEFINITION 3.7.  $BH(1) \stackrel{\text{def}}{=} NP$ . For  $i > 0$ ,

$$BH(2i) \stackrel{\text{def}}{=} \{L \mid L = L' \cap \overline{L_2}, \text{ where } L' \in BH(2i - 1), L_2 \in NP\},$$

$$BH(2i + 1) \stackrel{\text{def}}{=} \{L \mid L = L' \cup L_2, \text{ where } L' \in BH(2i), L_2 \in NP\}.$$

DEFINITION 3.8. For all  $i \geq 1$ ,  $\text{co-BH}(i)$  is the class of languages whose complements are in  $BH(i)$ .

DEFINITION 3.9.  $BH \stackrel{\text{def}}{=} \bigcup_k BH(k)$ .

Let  $L_i \in NP$ , then these definitions unwind as follows:

$$\begin{aligned} BH(1) &= NP, \\ BH(2) &= \{L_1 \cap \overline{L_2}\} (= D^P), \\ BH(3) &= \{(L_1 \cap \overline{L_2}) \cup L_3\}, \\ BH(4) &= \{((L_1 \cap \overline{L_2}) \cup L_3) \cap \overline{L_4}\}, \end{aligned}$$

$$\begin{aligned}
 \text{co-BH}(1) &= \text{co-NP}, \\
 \text{co-BH}(2) &= \{\overline{L_1} \cup L_2\} (= \text{co-D}^P), \\
 \text{co-BH}(3) &= \{(\overline{L_1} \cup L_2) \cap \overline{L_3}\}, \\
 \text{co-BH}(4) &= \{((\overline{L_1} \cup L_2) \cap \overline{L_3}) \cup L_4\}, \\
 &\vdots
 \end{aligned}$$

Many results about the BH are proven in [4]. For instance,

- (1)  $\text{BH}(k) \cup \text{co-BH}(k) \subseteq \text{BH}(k+1) \cap \text{co-BH}(k+1)$ .
- (2) The BH, QH, and  $\text{QH}_{\parallel}$  intertwine. Therefore

$$\text{BH} = \text{QH} = \text{QH}_{\parallel} = \text{BTT}[\text{NP}].$$

- (3) The BH has the upward collapse property:

$$\text{BH}(k) = \text{co-BH}(k) \iff \text{BH}(k) = \text{BH}.$$

The QH and  $\text{QH}_{\parallel}$  also have upward collapse properties:

$$\begin{aligned}
 \text{P}^{\text{NP}[k]} = \text{P}^{\text{NP}[k+1]} &\iff \text{P}^{\text{NP}[k]} = \text{QH}, \quad \text{and} \\
 \text{P}^{\text{NP}\parallel[k]} = \text{P}^{\text{NP}\parallel[k+1]} &\iff \text{P}^{\text{NP}\parallel[k]} = \text{QH}_{\parallel}.
 \end{aligned}$$

One way to see these properties is through the detailed examination of the intertwining of the hierarchies in [1], [2]. In [2], the following relationships are proven:

$$\begin{aligned}
 \text{BH}(k) \cup \text{co-BH}(k) &\subseteq \text{P}^{\text{NP}\parallel[k]} \subseteq \text{BH}(k+1) \cap \text{co-BH}(k+1), \\
 \text{BH}(2^k - 1) \cup \text{co-BH}(2^k - 1) &\subseteq \text{P}^{\text{NP}[k]} \subseteq \text{BH}(2^k) \cap \text{co-BH}(2^k).
 \end{aligned}$$

Therefore, if any two levels of the QH or  $\text{QH}_{\parallel}$  collapse, some level of the BH is closed under complementation, which implies that all three hierarchies collapse.

The intertwining, the upward collapse properties, and the fact that  $\text{P}^{\text{NP}[k]}$  and  $\text{P}^{\text{NP}\parallel[k]}$  are closed under complementation imply the following theorem.

**THEOREM 3.10.** *The following are equivalent:*

- (1) BH collapses.
- (2) QH collapses.
- (3)  $\text{QH}_{\parallel}$  collapses.
- (4)  $\exists k \text{ BH}(k) = \text{co-BH}(k)$ .
- (5)  $\exists j \text{ P}^{\text{NP}[j]} = \text{P}^{\text{NP}[j+1]}$ .
- (6)  $\exists j \text{ P}^{\text{NP}\parallel[j]} = \text{P}^{\text{NP}\parallel[j+1]}$ .
- (7)  $\exists j, k \text{ BH}(k) = \text{P}^{\text{NP}[j]}$ .
- (8)  $\exists j, k \text{ BH}(k) = \text{P}^{\text{NP}\parallel[j]}$ .
- (9)  $\exists j, k \text{ BH}(k) \cup \text{co-BH}(k) = \text{P}^{\text{NP}[j]}$ .
- (10)  $\exists j, k \text{ BH}(k) \cup \text{co-BH}(k) = \text{P}^{\text{NP}\parallel[j]}$ .

The equivalence of conditions (1) - (4) is also noted in [2].

To see that the structure of the BH is related to the structure of the PH, we need to understand the structure of the  $\leq_m^P$ -complete sets defined in [4]. For each  $k$ , a complete language,  $L_{\text{BH}(k)}$ , can be defined in terms of SAT and  $\overline{\text{SAT}}$ .

**DEFINITION 3.11.**  $L_{\text{BH}(1)} \stackrel{\text{def}}{=} \text{SAT}$ . For  $i > 0$ ,

$$\begin{aligned}
 L_{\text{BH}(2i)} &\stackrel{\text{def}}{=} \{(F_1, \dots, F_{2i}) \mid (F_1, \dots, F_{2i-1}) \in L_{\text{BH}(2i-1)} \text{ and } F_{2i} \in \overline{\text{SAT}}\}, \\
 L_{\text{BH}(2i+1)} &\stackrel{\text{def}}{=} \{(F_1, \dots, F_{2i+1}) \mid (F_1, \dots, F_{2i}) \in L_{\text{BH}(2i)} \text{ or } F_{2i+1} \in \text{SAT}\}.
 \end{aligned}$$

Similarly, for each  $k$ , we can define a complete language for  $\text{co-BH}(k)$ .

DEFINITION 3.12.  $L_{\text{co-BH}(1)} \stackrel{\text{def}}{=} \overline{\text{SAT}}$ . For  $i > 0$ ,

$$L_{\text{co-BH}(2i)} \stackrel{\text{def}}{=} \{ (F_1, \dots, F_{2i}) \mid (F_1, \dots, F_{2i-1}) \in L_{\text{co-BH}(2i-1)} \text{ or } F_{2i} \in \text{SAT} \},$$

$$L_{\text{co-BH}(2i+1)} \stackrel{\text{def}}{=} \{ (F_1, \dots, F_{2i+1}) \mid (F_1, \dots, F_{2i}) \in L_{\text{co-BH}(2i)} \text{ and } F_{2i+1} \in \overline{\text{SAT}} \}.$$

Unwinding these definitions a bit:

$$L_{\text{BH}(1)} \stackrel{\text{def}}{=} \text{SAT},$$

$$L_{\text{BH}(2)} \stackrel{\text{def}}{=} \text{SAT} \& \overline{\text{SAT}}$$

$$\stackrel{\text{def}}{=} \{ (F_1, F_2) \mid F_1 \in \text{SAT} \text{ and } F_2 \in \overline{\text{SAT}} \},$$

$$L_{\text{BH}(3)} \stackrel{\text{def}}{=} (\text{SAT} \& \overline{\text{SAT}}) \mid \text{SAT}$$

$$\stackrel{\text{def}}{=} \{ (F_1, F_2, F_3) \mid (F_1 \in \text{SAT} \text{ and } F_2 \in \overline{\text{SAT}}) \text{ or } F_3 \in \text{SAT} \},$$

$$L_{\text{BH}(4)} \stackrel{\text{def}}{=} ((\text{SAT} \& \overline{\text{SAT}}) \mid \text{SAT}) \& \overline{\text{SAT}}$$

$$\stackrel{\text{def}}{=} \{ (F_1, F_2, F_3, F_4) \mid ((F_1 \in \text{SAT} \text{ and } F_2 \in \overline{\text{SAT}}) \text{ or } F_3 \in \text{SAT}) \text{ and } F_4 \in \overline{\text{SAT}} \},$$

$$\vdots$$

$$L_{\text{co-BH}(1)} \stackrel{\text{def}}{=} \overline{\text{SAT}},$$

$$L_{\text{co-BH}(2)} \stackrel{\text{def}}{=} \overline{\text{SAT}} \mid \text{SAT}$$

$$\stackrel{\text{def}}{=} \{ (F_1, F_2) \mid F_1 \in \overline{\text{SAT}} \text{ or } F_2 \in \text{SAT} \},$$

$$L_{\text{co-BH}(3)} \stackrel{\text{def}}{=} (\overline{\text{SAT}} \mid \text{SAT}) \& \overline{\text{SAT}}$$

$$\stackrel{\text{def}}{=} \{ (F_1, F_2, F_3) \mid (F_1 \in \overline{\text{SAT}} \text{ or } F_2 \in \text{SAT}) \text{ and } F_3 \in \overline{\text{SAT}} \},$$

$$L_{\text{co-BH}(4)} \stackrel{\text{def}}{=} ((\overline{\text{SAT}} \mid \text{SAT}) \& \overline{\text{SAT}}) \mid \text{SAT}$$

$$\stackrel{\text{def}}{=} \{ (F_1, F_2, F_3, F_4) \mid ((F_1 \in \overline{\text{SAT}} \text{ or } F_2 \in \text{SAT}) \text{ and } F_3 \in \overline{\text{SAT}}) \text{ or } F_4 \in \text{SAT} \},$$

$$\vdots$$

Note the structure of  $L_{\text{BH}(k)}$  and  $L_{\text{co-BH}(k)}$ . For even  $k$ 's, the  $k$ -tuples in  $L_{\text{BH}(k)}$  have the form “ $(k-1\text{-tuple}) \& \overline{\text{SAT}}$ ”, where the  $k-1$ -tuple is in  $L_{\text{BH}(k-1)}$ , and the  $k$ th component is in  $\overline{\text{SAT}}$ . The  $k$ -tuples in  $L_{\text{co-BH}(k)}$  have the form “ $(k-1\text{-tuple}) \mid \text{SAT}$ ”, where the  $k-1$ -tuple is in  $L_{\text{co-BH}(k-1)}$ , or the  $k$ th component is in  $\text{SAT}$ . If  $k$  is odd,  $L_{\text{BH}(k)}$  and  $L_{\text{co-BH}(k)}$  have a similar structure, but  $\text{BH}(i)$  and  $\text{co-BH}(i)$  are reversed. It will become clear that this is the key that relates the collapse of the BH to the collapse of the PH.

For all  $k$ ,  $L_{\text{BH}(k)}$  and  $L_{\text{co-BH}(k)}$  are complements. Hence any function  $f$  that reduces  $L_{\text{BH}(k)}$  to  $L_{\text{co-BH}(k)}$  also reduces  $L_{\text{co-BH}(k)}$  to  $L_{\text{BH}(k)}$ .

**4. Generating the sparse oracle.** In this section we show that the collapse of the BH forces the existence of a sparse set  $S \in \text{NP}^{\text{NP}}$  such that  $\text{co-NP} \subseteq \text{NP}^S$ . By the results of Yap, the existence of a sparse set  $S$  such that  $\text{co-NP} \subseteq \text{NP}^S$  implies that

the PH collapses to  $\Sigma_3^P$  [26]. In § 5 we will show that the fact that the sparse set is in  $NP^{NP}$  actually implies that the PH collapses to  $P^{NP^{NP^{O(\log n)}}$ , the class of languages recognizable in polynomial time with  $O(\log n)$  queries to an oracle in  $NP^{NP}$ .

The basic idea is that if for some  $k$ ,  $BH(k) = \text{co-BH}(k)$ , and thus

$$L_{\text{co-BH}(k)} \leq_m^P L_{\text{BH}(k)},$$

then  $\overline{\text{SAT}}$  can be reduced to SAT by a polynomial time function that accesses a sparse oracle. The crux of the proof is in the construction of the sparse set. We will illustrate the main idea of the construction by showing how the argument works in the case where  $\text{BH}(2) = \text{co-BH}(2)$  ( $D^P = \text{co-D}^P$ ).

Suppose that  $\text{BH}(2) = \text{co-BH}(2)$ . Then  $L_{\text{BH}(2)} \leq_m^P L_{\text{co-BH}(2)}$ . That is, there is some polynomial time function  $h(\cdot, \cdot)$  such that

$$\text{SAT} \& \overline{\text{SAT}} \leq_m^P \overline{\text{SAT}} \mid \text{SAT} \text{ via } h.$$

The key is that  $h$  maps a conjunction to a disjunction. Both conditions of the conjunction are met if just one of the disjuncts is met. In the easy case, if the second component of the output of  $h$  is a satisfiable formula, then the first component of the input is satisfiable, and the second component is *not satisfiable*. This gives rise to an NP algorithm for recognizing some of  $\overline{\text{SAT}}$ .

Consider a formula  $F$  of length  $n$ . Suppose there exists a formula  $F'$  of length  $n$  such that

$$h(F', F) = (G', G), \text{ where } G \in \text{SAT}.$$

Then  $(G', G) \in \overline{\text{SAT}} \mid \text{SAT}$ , which implies  $F \in \overline{\text{SAT}}$  (and  $F' \in \text{SAT}$ ). Therefore an NP machine can recognize that  $F \in \overline{\text{SAT}}$  by guessing  $F'$ , computing  $h(F', F)$ , and verifying that  $G \in \text{SAT}$ . We call formulas such as  $F$  *easy* since an NP algorithm can recognize that  $F \in \overline{\text{SAT}}$ .

Let  $N_{\text{easy}}$  be the NP machine that executes this algorithm.  $L(N_{\text{easy}}) \subseteq \overline{\text{SAT}}$ , and clearly there is a polynomial time reduction from  $L(N_{\text{easy}})$  to SAT. Call this reduction  $h_{\text{easy}}$ . If all the strings in  $\overline{\text{SAT}}^n$  are easy, then  $h_{\text{easy}}$  reduces  $\overline{\text{SAT}}^n$  to SAT.

If not all the unsatisfiable strings of length  $n$  are easy, then some of them must be *hard*. A formula  $F$  of length  $n$  is *hard* if  $F \in \overline{\text{SAT}}$  and for all  $F'$  of length  $n$ ,

$$h(F', F) = (G', G), \text{ where } G \in \overline{\text{SAT}}.$$

Hard strings have a very useful property: given any hard string  $\hat{F}$  of length  $n$ ,  $h$  can be used to reduce  $\overline{\text{SAT}}^n$  to SAT. If  $F$  is any formula of length  $n$ , then

$$h(F, \hat{F}) = (G, \hat{G}), \text{ where } \hat{G} \in \overline{\text{SAT}}.$$

Since  $h$  is a reduction from  $\text{SAT} \& \overline{\text{SAT}}$  to  $\overline{\text{SAT}} \mid \text{SAT}$ ,

$$(F, \hat{F}) \in \text{SAT} \& \overline{\text{SAT}} \iff (G, \hat{G}) \in \overline{\text{SAT}} \mid \text{SAT}.$$

Since  $\hat{G} \notin \text{SAT}$ ,

$$(F, \hat{F}) \in \text{SAT} \& \overline{\text{SAT}} \iff G \in \overline{\text{SAT}}.$$

Since  $\hat{F} \in \overline{\text{SAT}}$ ,

$$F \in \text{SAT} \iff G \in \overline{\text{SAT}},$$

but this is equivalent to  $F \in \overline{\text{SAT}} \iff G \in \text{SAT}$ .

Let  $h_{\text{hard}(\hat{F})}$  be the polynomial time function that takes an input  $F$ , computes  $h(F, \hat{F}) = (G, \hat{G})$ , and outputs  $G$ . If  $|\hat{F}| = n$  and  $\hat{F}$  is hard, then  $h_{\text{hard}(\hat{F})}$  reduces  $\overline{\text{SAT}}^n$  to SAT. Note how each hard formula of length  $n$  is really a *key* string that gives rise to an NP algorithm for recognizing  $\overline{\text{SAT}}^n$  (accept  $F$  if  $h_{\text{hard}(\hat{F})}(F) \in \text{SAT}$ ).

Now it should be clear how to define a sparse set  $S$  and function  $g$  such that  $g^S$  reduces  $\overline{\text{SAT}}$  to SAT. For each length  $n$ , if some unsatisfiable formula of length  $n$  is hard, put the lexicographically least string and all of its marked prefixes into  $S$ . Then  $S$  is prefix marked so it is self-P-printable. Then  $g^S$  on input  $F$  of length  $n$  computes as follows:

- (1) Generate all the strings in  $S^n$ .
- (2) If  $S^n = \phi$ , then output  $h_{\text{easy}}(F)$ .
- (3) Otherwise, let  $\hat{F}$  be the hard string of length  $n$  in  $S$  (the only string in  $S^n$  that is not marked as a prefix), and output  $h_{\text{hard}(\hat{F})}(F)$ .

We can generalize this argument to show that there exists such a sparse set if any level of the BH is closed under complementation. The main lemma is that for any  $k$ , if there exist a sparse  $S$  and polynomial time function  $g$  such that  $g^S$  reduces  $L_{\text{BH}(k)}$  to  $L_{\text{co-BH}(k)}$ , then there exist another sparse set and function that reduces  $L_{\text{BH}(k-1)}$  to  $L_{\text{co-BH}(k-1)}$ . So if  $\text{BH}(k) = \text{co-BH}(k)$ , then for all  $i \leq k$ , there exist a sparse  $S$  and function  $g$  such that  $g^S$  reduces  $L_{\text{BH}(i)}$  to  $L_{\text{co-BH}(i)}$ . Thus if the BH collapses,  $\overline{\text{SAT}}$  can be reduced to SAT by a polynomial time function that accesses a sparse oracle. This implies  $\text{co-NP} \subseteq \text{NP}^S$  for some sparse set  $S$ , and therefore the PH collapses (see § 5).

The key that makes this work is the fine structure of the BH. Recall that a function is a reduction from  $L_{\text{BH}(k)}$  to  $L_{\text{co-BH}(k)}$  if and only if it is a reduction from  $L_{\text{co-BH}(k)}$  to  $L_{\text{BH}(k)}$ . Hence a reduction from  $L_{\text{BH}(k)}$  to  $L_{\text{co-BH}(k)}$  always maps  $k$ -tuples of the form

$$“(L_{\text{BH}(k-1)}) \& \overline{\text{SAT}}” \quad \text{to} \quad “(L_{\text{co-BH}(k-1)}) | \text{SAT}”,$$

or if  $k$  is odd:

$$“(L_{\text{co-BH}(k-1)}) \& \overline{\text{SAT}}” \quad \text{to} \quad “(L_{\text{BH}(k-1)}) | \text{SAT}”.$$

Thus we can define “easy” strings and “hard” strings as above and give a similar argument to define a reduction from  $L_{\text{BH}(k-1)}$  to  $L_{\text{co-BH}(k-1)}$ .

We need one technical refinement of the definitions of  $L_{\text{BH}(k)}$  and  $L_{\text{co-BH}(k)}$  before we prove the lemma. We will require that all  $k$  formulas in the  $k$ -tuples in  $L_{\text{BH}(k)}$  and  $L_{\text{co-BH}(k)}$  have the same length. This restricted version is still  $\leq_m^P$ -complete since the shorter formulas in any  $k$ -tuple can be padded up to the length of the longest formula in time that is polynomial in the length of the original  $k$ -tuple.

We will also need to talk about subsets of  $L_{\text{BH}(k)}$  and  $L_{\text{co-BH}(k)}$  for which all the formulas in each tuple have a fixed length.

DEFINITION 4.1. *For all  $k$  and  $n$ ,  $L_{\text{BH}(k),n}$  ( $L_{\text{co-BH}(k),n}$ ) is the finite subset of  $L_{\text{BH}(k)}$  ( $L_{\text{co-BH}(k)}$ ) where each  $k$ -tuple contains only formulas of length  $n$ .*

Clearly, for all  $k$ , any function  $f$  that reduces  $L_{\text{BH}(k)}$  to  $L_{\text{co-BH}(k)}$  also reduces  $L_{\text{BH}(k),n}$  to  $L_{\text{co-BH}(k)}$  and  $L_{\text{co-BH}(k),n}$  to  $L_{\text{BH}(k)}$ .

LEMMA 4.2. *For all  $k > 1$ , if there exist  $S_k, g_k$  where  $S_k$  is sparse and  $g_k^{S_k}$  is a polynomial time reduction of  $L_{\text{BH}(k)}$  to  $L_{\text{co-BH}(k)}$ , then there exist  $S_{k-1}, g_{k-1}$  where  $S_{k-1}$  is sparse and  $g_{k-1}^{S_{k-1}}$  is a polynomial time reduction of  $L_{\text{BH}(k-1)}$  to  $L_{\text{co-BH}(k-1)}$ .*

*Proof.* Suppose  $g_k^{S_k}$  reduces  $L_{\text{BH}(k)}$  to  $L_{\text{co-BH}(k)}$ . Assume, without loss of generality, that  $S_k$  is prefix marked so that  $S_k$  is self-P-printable.

As above, for each length  $n$ , there are two cases to distinguish: the case where all the unsatisfiable strings of length  $n$  are “easy” and the case where at least one is “hard.”

DEFINITION 4.3. *A formula  $F$  is  $k$ -easy if there exist  $F_1, \dots, F_{k-1}$  with  $|F_i| = |F|$  such that*

$$g_k^{S_k}(F_1, \dots, F_{k-1}, F) = (G_1, \dots, G_{k-1}, G), \text{ where } G \in \text{SAT}.$$

DEFINITION 4.4. *A formula  $F$  of length  $n$  is  $k$ -hard if  $F \in \overline{\text{SAT}}$  and for all  $F_1, \dots, F_{k-1}$  with  $|F_i| = n$ ,*

$$g_k^{S_k}(F_1, \dots, F_{k-1}, F) = (G_1, \dots, G_{k-1}, G), \text{ where } G \in \overline{\text{SAT}}.$$

Suppose  $F$  is  $k$ -easy. If  $k$  is even, and

$$g_k^{S_k}(F_1, \dots, F_{k-1}, F) = (G_1, \dots, G_{k-1}, G),$$

where  $G \in \text{SAT}$ , then

$$(G_1, \dots, G_{k-1}, G) \in L_{\text{co-BH}(k)}.$$

Since  $g_k^{S_k}$  is a reduction,

$$(F_1, \dots, F_{k-1}, F) \in L_{\text{BH}(k)},$$

which implies  $F \in \overline{\text{SAT}}$ . If  $k$  is odd, the same reasoning holds with  $\text{BH}(k)$  and  $\text{co-BH}(k)$  reversed. Therefore if  $F$  is  $k$ -easy,  $F \in \overline{\text{SAT}}$ , and there is an  $\text{NP}^{S_k}$  algorithm for recognizing  $F$  as unsatisfiable. On input  $F$ , guess  $F_1, \dots, F_{k-1}$ , compute  $g_k^{S_k}(F_1, \dots, F_{k-1}, F)$ , and accept if the last component of the output of  $g_k$  is satisfiable.

Call the machine that executes this  $\text{NP}^{S_k}$  algorithm  $N_{k\text{-easy}}$ ;

$$L(N_{k\text{-easy}}^{S_k}) \subseteq \overline{\text{SAT}}.$$

Since  $S_k$  is  $\text{P}^{S_k}$ -printable, there exists a polynomial time reduction  $r(\cdot)$  such that  $r^{S_k}$  reduces  $L(N_{k\text{-easy}}^{S_k})$  to  $\text{SAT}$ . On input  $F$ ,  $r^{S_k}$  first generates the subset  $S'_k$  of strings in  $S_k$  up to the length  $N_{k\text{-easy}}$  can ask on input  $F$ . Then since it is an  $\text{NP}$  question whether or not  $N_{k\text{-easy}}(F)$  accepts using oracle  $S'_k$ ,  $r$  can map  $\langle N_{k\text{-easy}}, F, S'_k \rangle$  to a formula that is satisfiable if and only if  $N_{k\text{-easy}}$  accepts.

Case 1. Suppose all the unsatisfiable formulas of length  $n$  are  $k$ -easy. Then

$$L(N_{k\text{-easy}}^{S_k})^n = \overline{\text{SAT}}^n,$$

and for all  $F$  of length  $n$ ,

$$r^{S_k}(F) \in \text{SAT} \iff F \in \overline{\text{SAT}}.$$

For such  $n$ 's, we can use  $r^{S_k}$  (and the fact that we can combine formulas with logical AND's and OR's) to reduce  $L_{\text{co-BH}(k-1), n}$  to  $\text{SAT}$ . Since  $\text{SAT} \leq_m^{\text{P}} L_{\text{BH}(k-1)}$ , we

can use  $r^{S^k}$  to reduce  $L_{\text{co-BH}(k-1),n}$  to  $L_{\text{BH}(k-1)}$ . Call the reduction that uses  $r$  in this way  $g_{k,\text{easy}}^{S^k}$ . For example, suppose  $k = 4$ , then

$$\begin{aligned} L_{\text{co-BH}(3)} &= (\overline{\text{SAT}} \mid \text{SAT}) \& \overline{\text{SAT}}, \quad \text{and} \\ L_{\text{BH}(3)} &= (\text{SAT} \& \overline{\text{SAT}}) \mid \text{SAT}. \end{aligned}$$

If there are no 4-hard strings of length  $n$ ,  $r^{S^4}$  reduces  $\overline{\text{SAT}}^n$  to SAT. Define

$$g_{4-\text{easy}}^{S^4}(F_1, F_2, F_3) \stackrel{\text{def}}{=} (\hat{G}_1, \hat{G}_2, (r^{S^4}(F_1) \vee F_2) \wedge r^{S^4}(F_3)),$$

where  $\hat{G}_1$  is FALSE padded up to the appropriate length, and  $\hat{G}_2$  is TRUE padded up to the appropriate length. Then

$$\begin{aligned} g_{4-\text{easy}}^{S^4}(F_1, F_2, F_3) \in (\text{SAT} \& \overline{\text{SAT}}) \mid \text{SAT} &\iff ((r^{S^4}(F_1) \vee F_2) \wedge r^{S^4}(F_3)) \in \text{SAT} \\ &\iff (F_1, F_2, F_3) \in (\overline{\text{SAT}} \mid \text{SAT}) \& \overline{\text{SAT}}. \end{aligned}$$

Similarly, if  $k$  is odd, then we can use  $r^{S^k}$  (and logical AND's and OR's) to reduce

$$L_{\text{BH}(k-1),n} \rightarrow \text{SAT} \rightarrow L_{\text{co-BH}(k-1)}.$$

Therefore, if all the formulas in  $\overline{\text{SAT}}^n$  are  $k$ -easy, then

$$g_{k,\text{easy}}^{S^k} \text{ reduces } L_{\text{co-BH}(k-1),n} \rightarrow L_{\text{BH}(k-1)} \text{ and } L_{\text{BH}(k-1),n} \rightarrow L_{\text{co-BH}(k-1)}.$$

For such lengths  $n$  and  $k - 1$ -tuples  $(F_1, \dots, F_{k-1})$  of formulas of length  $n$ , we will define  $S_{k-1}$  and  $g_{k-1}$  so that

$$g_{k-1}^{S_{k-1}}(F_1, \dots, F_{k-1}) = g_{k,\text{easy}}^{S^k}(F_1, \dots, F_{k-1}).$$

*Observation.* A subtle fact that we will use later is that, by defining  $g_{k-1}$  this way, if all the unsatisfiable formulas of length  $n$  are  $k$ -easy, then all the unsatisfiable formulas of length  $n$  are also  $k - 1$ -easy. That is, since  $g_{k-1}$  encodes all the  $k - 1$  formulas it takes as its input into the *last* component of its output, for any  $F \in \overline{\text{SAT}}^n$ , there exist  $(F_1, \dots, F_{k-2})$  all of length  $n$  such that

$$g_{k-1}^{S_{k-1}}(F_1, \dots, F_{k-2}, F) = (G_1, \dots, G_{k-2}, G), \text{ where } G \in \text{SAT}.$$

If  $k$  is even, any  $(F_1, \dots, F_{k-2}) \in L_{\text{co-BH}(k-2),n}$  will be such a tuple. If  $k$  is odd, any  $(F_1, \dots, F_{k-2}) \in L_{\text{BH}(k-2),n}$  will be such a tuple. Therefore, if all the unsatisfiable formulas of length  $n$  are  $k$ -easy, they are also all  $k - 1$ -easy. In other words, if there are no  $k$ -hard strings of length  $n$ , then there are no  $k - 1$ -hard strings of length  $n$ .

*Case 2.* Suppose there exists a formula  $\hat{F}$  of length  $n$  such that  $\hat{F}$  is  $k$ -hard. Suppose  $k$  is even. Let  $F_1, \dots, F_{k-1}$  be any formulas of length  $n$ , and let

$$g_k^{S^k}(F_1, \dots, F_{k-1}, \hat{F}) \stackrel{\text{def}}{=} (G_1, \dots, G_{k-1}, \hat{G}).$$

Then since  $\hat{F} \in \overline{\text{SAT}}$ ,

$$(F_1, \dots, F_{k-1}) \in L_{\text{BH}(k-1),n} \iff (F_1, \dots, F_{k-1}, \hat{F}) \in L_{\text{BH}(k),n}.$$



Since  $g_k^{S_k}$  is a reduction,

$$(F_1, \dots, F_{k-1}, \hat{F}) \in L_{\text{BH}(k),n} \iff (G_1, \dots, G_{k-1}, \hat{G}) \in L_{\text{co-BH}(k)}.$$

Since  $\hat{F}$  is  $k$ -hard,  $\hat{G} \notin \text{SAT}$ , and therefore

$$(G_1, \dots, G_{k-1}, \hat{G}) \in L_{\text{co-BH}(k)} \iff (G_1, \dots, G_{k-1}) \in L_{\text{co-BH}(k-1)}.$$

If  $k$  is odd, the same reasoning holds with  $\text{BH}(i)$  and  $\text{co-BH}(i)$  reversed. Hence there is a polynomial time function which, using  $S_k$ , can reduce  $L_{\text{BH}(k-1),n}$  to  $L_{\text{co-BH}(k-1)}$  (and  $L_{\text{co-BH}(k-1),n}$  to  $L_{\text{BH}(k-1)}$ ). Call this function  $g_{k\text{-hard}}(\hat{F})$ . On input  $F_1, \dots, F_{k-1}$ ,  $g_{k\text{-hard}}^{S_k}(\hat{F})$  computes  $g_k^{S_k}(F_1, \dots, F_{k-1}, \hat{F})$  and outputs the first  $k - 1$  components.

In summary, if all the strings of length  $n$  are  $k$ -easy, then  $g_{k\text{-easy}}^{S_k}$  reduces  $L_{\text{co-BH}(k-1),n}$  to  $L_{\text{BH}(k-1)}$ . If some formula  $\hat{F}$  of length  $n$  is  $k$ -hard, then  $g_{k\text{-hard}}^{S_k}(\hat{F})$  reduces  $L_{\text{co-BH}(k-1),n}$  to  $L_{\text{BH}(k-1)}$ .

Thus  $S_{k-1}$  can be defined as  $S_k$  plus enough strings to indicate which reduction works for each length  $n$ . Let  $\text{Lexhard}_k$  be the set consisting of the lexicographically least  $k$ -hard string of each length (if there is a  $k$ -hard string of that length). Then the set of marked prefixes of  $\text{Lexhard}_k$ ,  $\text{prefix}(\text{Lexhard}_k)$ , is self-P-printable. A polynomial time machine with access to oracle  $\text{prefix}(\text{Lexhard}_k)$  can determine if there are any  $k$ -hard strings of a given length, and it can get hold of the lexicographically least one if there are any. Therefore  $\text{prefix}(\text{Lexhard}_k)$  provides enough information to determine which reduction works for each length  $n$ .

Let

$$S_{k-1} \stackrel{\text{def}}{=} S_k \oplus \text{prefix}(\text{Lexhard}_k).$$

$S_{k-1}$  is sparse since  $S_k$  and  $\text{prefix}(\text{Lexhard}_k)$  are both sparse. Then  $g_{k-1}^{S_{k-1}}$  on input  $F_1, \dots, F_{k-1}$  (all of length  $n$ ) works as follows:

- (1) Generate all the strings in  $\text{prefix}(\text{Lexhard}_k)^{=n}$ .
- (2) If  $\text{Lexhard}_k^{=n} = \phi$ , output  $g_{k\text{-easy}}^{S_k}(F_1, \dots, F_{k-1})$ .
- (3) Otherwise let  $\hat{F}$  be the  $k$ -hard string in  $\text{Lexhard}_k^{=n}$ , and output

$$g_{k\text{-hard}}^{S_k}(\hat{F})(F_1, \dots, F_{k-1}).$$

For each  $n$ ,  $g_{k-1}^{S_{k-1}}$  reduces  $L_{\text{co-BH}(k-1),n}$  to  $L_{\text{BH}(k-1)}$  and hence reduces  $L_{\text{co-BH}(k-1)}$  to  $L_{\text{BH}(k-1)}$  and vice versa.  $\square$

**COROLLARY 4.5.** *For all  $k$ , if  $\text{BH}(k) = \text{co-BH}(k)$ , then there exist  $S, g$  such that  $S$  is sparse and  $g^S$  is a polynomial time reduction from  $\overline{\text{SAT}}$  to  $\text{SAT}$ .*

*Proof.* If  $\text{BH}(k) = \text{co-BH}(k)$ , then there exists a polynomial time reduction (that needs no oracle) from  $L_{\text{BH}(k)}$  to  $L_{\text{co-BH}(k)}$ . By  $k - 1$  repetitions of the above lemma, we can construct  $S$  and  $g$ .  $\square$

**LEMMA 4.6.** *If  $S$  is sparse and  $g^S$  is a polynomial time reduction from a language  $L$  to  $\text{SAT}$ , then  $L \in \text{NP}^S$ .*

*Proof.*  $L = L(\text{NP}^S)$ , where on input  $x$ ,  $N$  computes  $g^S(x) = y$  and accepts if  $y \in \text{SAT}$ .  $\square$

*Remark.* At first glance, the existence of a sparse  $S$  such that  $g^S$  reduces  $L$  to  $\text{SAT}$  seems to be a stronger condition than the existence of a sparse set  $S$  such that

$L \in \text{NP}^S$ . An  $\text{NP}^S$  machine can access  $S$  nondeterministically while the function  $g$  must ask queries deterministically. However, since any sparse set can be made self-P-printable by adding all of its marked prefixes, the following conditions are actually equivalent:

- (1) There exist sparse  $S$  and function  $g$  such that  $g^S$  is a polynomial time reduction from  $L$  to SAT.
- (2) There exists sparse  $S$  such that  $L \in \text{NP}^S$ .

Corollary 4.5 and Lemma 4.6 together imply the following theorem.

**THEOREM 4.7.** *For all  $k$ , if  $\text{BH}(k) = \text{co-BH}(k)$ , then there exists a sparse set  $S$  such that  $\overline{\text{SAT}} \in \text{NP}^S$  (i.e.,  $\text{co-NP} \subseteq \text{NP}^S$ ).*

By Theorem 4.7 and Yap’s result, we now know that the collapse of the BH implies the collapse of the PH (to  $\text{NP}^{\text{NP}^{\text{NP}}}$ ). In the rest of this section, we show that the sparse set  $S$  is in  $\text{NP}^{\text{NP}}$ . In § 5, we will prove that this fact implies  $\text{PH} \subseteq \text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$ .

We need the following easy lemma.

**LEMMA 4.8.** *If  $L \in \text{co-NP}$ , then*

$$L' \stackrel{\text{def}}{=} \{x \mid x \text{ is the lexicographically least string in } L^{|x|}\}$$

is in  $\text{co-NP}$ .

*Proof.* Since  $L \in \text{co-NP}$ , there exist a polynomial time predicate  $R(\cdot, \cdot)$  and a polynomial  $p(\cdot)$  such that

$$x \in L \iff \forall y \text{ with } |y| \leq p(|x|), R(x, y).$$

Then

$$x \in L' \iff \forall y, u, v \text{ such that } |y| \leq p(|x|), |u|=|x|, |v| \leq p(|x|), \\ R(x, y), \text{ and } R(u, v) \implies u \geq x \text{ lexicographically.}$$

Thus  $L' \in \text{co-NP}$ .  $\square$

Now we can show that the sparse oracle is in  $\text{NP}^{\text{NP}}$ .

**LEMMA 4.9.** *The sparse oracle  $S$  constructed in Corollary 4.5 is in  $\text{NP}^{\text{NP}}$ .*

*Proof.* We show that the set  $S$  defined by repeated applications of Lemma 4.2 is in  $\text{NP}^{\text{NP}}$ . Suppose  $\text{BH}(k) = \text{co-BH}(k)$  with  $g_k$  a polynomial time reduction from  $L_{\text{BH}(k)}$  to  $L_{\text{co-BH}(k)}$ . We define the set  $S$  from the hard strings at each level. For  $i$  backing down from  $k$  to 2, we define  $\text{Hard}_i$  to be the set of  $i$ -hard strings.  $\text{Lexhard}_i$  is the set consisting of the lexicographically least  $i$ -hard string of each length, and

$$S \stackrel{\text{def}}{=} \bigoplus_{i=2}^k \text{prefix}(\text{Lexhard}_i).$$

By backward induction, we will show that for each  $i$ ,  $\text{Lexhard}_i \in \text{NP}^{\text{NP}}$ . Therefore  $\text{prefix}(\text{Lexhard}_i) \in \text{NP}^{\text{NP}}$  which implies  $S \in \text{NP}^{\text{NP}}$ .

*Base Case.*  $\text{Lexhard}_k \in \text{co-NP} \subseteq \text{NP}^{\text{NP}}$ . Recall that a string  $F$  is  $k$ -easy if there exist  $F_1, \dots, F_{k-1}$  of length  $|F|$  such that

$$g_k(F_1, \dots, F_{k-1}, F) = (G_1, \dots, G_{k-1}, G), \text{ where } G \in \text{SAT}.$$

Hence the set of  $k$ -easy strings is in NP. Since

$$\text{Hard}_k = \overline{\text{SAT} \cup \{F \mid F \text{ is } k\text{-easy}\}},$$

$\text{Hard}_k \in \text{co-NP}$ . Then by Lemma 4.8,  $\text{Lexhard}_k \in \text{co-NP}$ .

*Inductive Step.* For  $i$  with  $2 \leq i < k$ ,  $\text{Lexhard}_i \in \text{NP}^{\text{NP}}$  if for all  $j$  with  $i < j \leq k$ ,  $\text{Lexhard}_j \in \text{NP}^{\text{NP}}$ . As we observed in the “easy” case of Lemma 4.2, there are no

$i$ -hard strings of length  $n$  if there are no  $i + 1$ -hard strings of length  $n$ . Therefore we know that a formula  $F$  of length  $n$  can be in  $\text{Lexhard}_i$  only if there are strings in  $\text{Lexhard}_{i+1}^{=n}, \dots, \text{Lexhard}_k^{=n}$ . If we are given the strings  $\hat{F}_{i+1}, \dots, \hat{F}_k$  that are in  $\text{Lexhard}_{i+1}^{=n}, \dots, \text{Lexhard}_k^{=n}$ , respectively, then we have enough information to compute  $g_i^{S_i}$ , the reduction from  $L_{\text{BH}(i),n}$  to  $L_{\text{co-BH}(i)}$  in polynomial time. Thus given  $\hat{F}_{i+1}, \dots, \hat{F}_k$ , a co-NP question can determine if  $F \in \text{Hard}_i$ :

$$“\forall F_1, \dots, F_{i-1}, g_i^{S_i}(F_1, \dots, F_{i-1}, F) = (G_1, \dots, G_{i-1}, G), \text{ where } G \in \overline{\text{SAT}}?”.$$

Similarly, a co-NP question can determine if  $F \in \text{Lexhard}_i$ .

Assuming the inductive hypothesis, on input  $F$ , an  $\text{NP}^{\text{NP}}$  machine can guess  $\hat{F}_{i+1}, \dots, \hat{F}_k$ , all of length  $|F|$ , and verify that  $\hat{F}_{i+1}, \dots, \hat{F}_k$  are in  $\text{Lexhard}_{i+1}^{=n}, \dots, \text{Lexhard}_k^{=n}$ , respectively. Then the  $\text{NP}^{\text{NP}}$  machine can verify that  $F \in \text{Lexhard}_i$  with one more query to NP.

Therefore by induction, the sparse set  $S$  is in  $\text{NP}^{\text{NP}}$ .  $\square$

**THEOREM 4.10.** *For all  $k$ , if  $\text{BH}(k) = \text{co-BH}(k)$ , then there exists a sparse set  $S \in \text{NP}^{\text{NP}}$  such that  $\text{co-NP} \subseteq \text{NP}^S$ .*

**5. Collapsing the polynomial time hierarchy.** We complete the proof that the collapse of the BH implies  $\text{PH} \subseteq \text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$  by showing that if there exists a sparse set  $S \in \text{NP}^{\text{NP}}$  such that  $\text{co-NP} \subseteq \text{NP}^S$ , then  $\text{PH} \subseteq \text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$ .

This result is really a relativization of the following theorem proved in [10].

**THEOREM 5.1.** *If there exists a sparse set  $S \in \text{NP}$  such that  $\text{co-NP} \subseteq \text{NP}^S$ , then  $\text{PH} \subseteq \text{P}^{\text{NP}[O(\log n)]}$ .*

The proof uses a technique called oracle replacement which is embedded in the proofs of the sparse oracle results of Karp and Lipton [12], Mahaney [17], Long [16], and Yap [26].

For oracles  $A$  and  $B$ , oracle replacement is a technique for mapping questions about whether an  $\text{NP}^B$  machine accepts an input to questions about whether some  $\text{NP}^A$  machine accepts an input. Let  $N$  be an  $\text{NP}^B$  machine. If we have two  $\text{NP}^A$  machines, one that accepts  $B$  and one that accepts  $\overline{B}$ , then we can construct an  $\text{NP}^A$  machine  $N_{\text{new}}$  such that

$$L(N^B) = L(N_{\text{new}}^A).$$

$N_{\text{new}}^A$  simulates  $N^B$ . When  $N$  would query  $B$ ,  $N_{\text{new}}$  runs the  $B$  and  $\overline{B}$  recognizers on the query string to determine the oracle answer. Thus, if there exist  $\text{NP}^A$  recognizers for  $B$  and  $\overline{B}$ , we can map  $\text{NP}^B$  questions to  $\text{NP}^A$  questions.

The technique can be used even if we cannot necessarily find  $\text{NP}^A$  machines that accept all of  $B$  and  $\overline{B}$ . For a given input  $x$ , if we can find  $\text{NP}^A$  machines that accept  $B^{\leq m}$  and  $\overline{B}^{\leq m}$ , where  $m$  is a bound on the length of the queries that  $N$  can make on input  $x$ , then the same trick works.

We will use the technique to map  $\text{NP}^S$  questions to  $\text{NP}^{\text{SAT}}$  questions. If  $S \in \text{NP}^{\text{NP}}$ , then there exists an  $\text{NP}^{\text{SAT}}$  machine that recognizes  $S$ . We will show that if  $S$  is also sparse, then a  $\text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$  machine can generate  $\text{NP}^{\text{SAT}}$  machines that recognize initial segments of  $\overline{S}$ . Hence a  $\text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$  machine can use oracle replacement to map a question of the form

$$“x \in L(N^S)?”$$

to a question of the form

$$“x \in L(N_{\text{new}}^{\text{SAT}})?”.$$

Then with one more query, the  $P^{NP^{NP}[O(\log n)]}$  machine can determine if  $x \in L(N_{new}^{SAT})$ . Therefore if  $S \in NP^{NP}$  and  $S$  is sparse, then  $NP^S \in P^{NP^{NP}[O(\log n)]}$ .

We need one more lemma before we prove the theorem.

LEMMA 5.2. *If  $S$  is sparse and prefix marked, and  $co-NP \subseteq NP^S$ , then  $NP^S = co-NP^S$ .*

*Proof.* Recall that if  $S$  is prefix marked, then  $S$  is  $P^S$ -printable. Let  $N$  be any  $NP^S$  machine. An  $NP^S$  algorithm for accepting  $x$ , if  $x \notin L(N^S)$ , starts by writing down the strings in  $S^{\leq m}$ , where  $m$  is a bound on the length of the queries  $N$  can make on input  $x$ . This can be done in polynomial time since  $S$  is  $P^S$ -printable. Of course  $x \in L(N^S) \iff x \in L(N^{S^{\leq m}})$ . With  $S^{\leq m}$  in hand, it is an NP question whether  $x \in L(N^{S^{\leq m}})$ , and thus  $\langle N, S^{\leq m}, x \rangle$  can be mapped to a Boolean formula  $F$  such that

$$F \in SAT \iff x \in L(N^{S^{\leq m}}).$$

Since  $\overline{SAT} \in NP^S$ , an  $NP^S$  machine can accept  $x$  if  $x \notin L(N^S)$ .  $\square$

THEOREM 5.3. *If there exists a sparse set  $S \in NP^{NP}$  such that  $co-NP \subseteq NP^S$ , then  $PH \subseteq P^{NP^{NP}[O(\log n)]}$ .*

*Proof.* If  $S \in NP^{NP}$  and  $co-NP \subseteq NP^S$ , then  $prefix(S) \in NP^{NP}$ , and  $co-NP \subseteq NP^{prefix(S)}$ . Therefore, without loss of generality, we can assume  $S$  is prefix marked. Then by the previous lemma,  $NP^S = co-NP^S$ . This implies that the polynomial time hierarchy relative to  $S$ ,  $PH^S$ , is contained in  $NP^S$  [26]. Hence

$$PH \subseteq PH^S \subseteq NP^S.$$

We will show  $PH \subseteq P^{NP^{NP}[O(\log n)]}$  by showing  $NP^S \subseteq P^{NP^{NP}[O(\log n)]}$ . This containment follows from oracle replacement.

Using the technique from [10], we will show that since  $P^{NP^{NP}[O(\log n)]}$  has the power to compute the census function of  $S$ , on an input of length  $m$ , a  $P^{NP^{NP}[O(\log n)]}$  machine can generate an  $NP^{SAT}$  machine,  $N_{\overline{S},m}$ , that accepts  $\overline{S}^{\leq m}$ .

The machines that accept  $\overline{S}^{\leq m}$  are based on the  $NP^{SAT}$  machine,  $N_{pseudo}$ , that accepts the *pseudo-complement* of  $S$ . The pseudo-complement of sparse sets was an important concept used in Mahaney's proof that the existence of a sparse  $\leq_m^P$ -complete set for NP implies  $P = NP$  [17]. On input  $(0^m, 1^j, y)$ ,  $N_{pseudo}^{SAT}$  guesses  $j$  strings of length at most  $m$ , verifies that the strings are in  $S$ , and accepts if  $|y| \leq m$  and  $y$  is not one of the guessed strings. If  $|y| \leq m$ , and  $j = Census_S(m)$ , then

$$(0^m, 1^j, y) \in L(N_{pseudo}^{SAT}) \iff y \in \overline{S}^{\leq m}.$$

Let  $c = Census_S(m)$ . Then  $N_{\overline{S},m}$  is  $N_{pseudo}$  with the first two arguments  $(0^m, 1^c)$  held constant. That is, on input  $y$ ,  $N_m$  writes  $(0^m, 1^c, y)$  on its tape and then runs like  $N_{pseudo}^{SAT}$ . Clearly  $L(N_{\overline{S},m}^{SAT}) = \overline{S}^{\leq m}$ .

On input  $1^m$ , a  $P^{NP^{NP}[O(\log n)]}$  machine can generate  $N_{\overline{S},m}$  since it can compute  $Census_S(m)$  by binary search. The binary search ranges over the numbers  $0 \cdots g(m)$ , where  $g(\cdot)$  is the polynomial that bounds  $Census_S$ . Each iteration of the binary search is a query of the form

“does  $N_{census}^{SAT}(0^m, 1^k)$  accept?”.

$N_{census}$  is the  $\text{NP}^{\text{SAT}}$  machine that takes an input  $(0^m, 1^k)$ , guesses  $k$  strings of length at most  $m$ , and accepts if they are all in  $S$ . Clearly,  $N_{census}^{\text{SAT}}$  accepts if and only if  $\text{Census}_S(m) \geq k$ . Thus the census value can be computed with  $O(\log g(m))$  queries to  $\text{NP}^{\text{SAT}}$ , and  $O(\log g(m))$  is  $O(\log m)$ . Therefore on input  $1^m$ , a  $\text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$  machine can generate  $N_{\bar{S},m}$ , an  $\text{NP}^{\text{SAT}}$  machine that accepts  $\bar{S}^{\leq m}$ .

These machines,  $\{N_{\bar{S},m}\}$ , that accept initial segments of  $\bar{S}$  can be used with  $N_S^{\text{SAT}}$ , the  $\text{NP}^{\text{SAT}}$  machine that accepts  $S$ , to replace the  $S$  oracle of any  $\text{NP}^S$  machine.

Let  $L = L(N^S)$  be any  $\text{NP}^S$  language. Let  $x$  be any string,  $|x| = n$ . Let  $m = p(n)$ , where  $p(\cdot)$  is the polynomial that bounds the running time of  $N$ . On input  $x$ , the  $\text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$  machine that recognizes  $L$  first generates  $N_{\bar{S},m}$ . Then using  $N_{\bar{S},m}$  and  $N_S$  to replace the oracle of  $N^S$ , it generates a machine  $N_{new}$  such that

$$x \in L(N_{new}^{\text{SAT}}) \iff x \in L(N^S).$$

Hence one more  $\text{NP}^{\text{NP}}$  query determines if  $x \in L(N^S)$ .

Therefore  $\text{NP}^S \subseteq \text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$ , and  $\text{PH} \subseteq \text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$ .  $\square$

Finally, we have all the pieces, and we have proved our main theorem.

**THEOREM 5.4.** *For all  $k$ , if  $\text{BH}(k) = \text{co-BH}(k)$ , then  $\text{PH} \subseteq \text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$ .*

Since the  $\text{BH}$ ,  $\text{QH}$ , and  $\text{QH}_{\parallel}$  are intertwined, we have the following corollaries.

**COROLLARY 5.5.** *If the  $\text{QH}$  collapses, then  $\text{PH} \subseteq \text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$ .*

**COROLLARY 5.6.** *If the  $\text{QH}_{\parallel}$  collapses, then  $\text{PH} \subseteq \text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$ .*

Since  $\text{P}^{\text{NP}[O(\log n)]} = \text{P}^{\text{NP}\parallel} = \text{TT}[\text{NP}]$ ,  $\text{BH} = \text{BTT}[\text{NP}]$ , and the  $\text{BH}$  collapses if  $\text{P}^{\text{NP}[O(\log n)]} = \text{BH}$ , we also have the following corollaries.

**COROLLARY 5.7.** *If  $\text{P}^{\text{NP}[O(\log n)]} = \text{BH}$ , then  $\text{PH} \subseteq \text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$ .*

**COROLLARY 5.8.** *If  $\text{P}^{\text{NP}\parallel} = \text{QH}_{\parallel}$ , then  $\text{PH} \subseteq \text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$ .*

**COROLLARY 5.9.** *If  $\text{TT}[\text{NP}] = \text{BTT}[\text{NP}]$ , then  $\text{PH} \subseteq \text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$ .*

Since Hemachandra’s result that  $\text{P}^{\text{NP}[O(\log n)]} = \text{P}^{\text{NP}\parallel} = \text{TT}[\text{NP}]$  relativizes to every level of the  $\text{PH}$ , we also have the following corollary.

**COROLLARY 5.10.** *If the  $\text{BH}$  collapses, then  $\text{PH} \subseteq \text{TT}[\text{NP}^{\text{NP}}]$ .*

Wagner has generalized Theorem 5.4 to show that for well-behaved functions  $r(n) < \log n$ , if  $r(n)$  queries to  $\text{NP}$  are as powerful as  $r(n) + 1$  queries, then the  $\text{PH}$  collapses to  $\text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}$  [24]. Thus the techniques we have developed here apply to slow-growing functions in addition to constants. Let  $\text{P}^{\text{SAT}[r(n)]}$  and  $\text{P}^{\text{SAT}\parallel[s(n)]}$  be the classes of languages recognizable by machines that make no more than  $r(n)$  serial queries and no more than  $s(n)$  parallel queries, respectively.

**THEOREM 5.11 (WAGNER).** *For functions  $r(n) < \log n$ ,  $s(n) \leq n$ ,*

$$\begin{aligned} \text{P}^{\text{SAT}[r(n)]} = \text{P}^{\text{SAT}[r(n)+1]} &\implies \text{PH} \subseteq \text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}, & \text{and} \\ \text{P}^{\text{SAT}\parallel[s(n)]} = \text{P}^{\text{SAT}\parallel[s(n)+1]} &\implies \text{PH} \subseteq \text{P}^{\text{NP}^{\text{NP}}[O(\log n)]}. \end{aligned}$$

*Proof sketch.* The proof is a generalization of the “easy-hard” argument of Lemma 4.2. In the constant case (Lemma 4.2), there is a fixed  $k$  such for each  $n$ , we can map

$$L_{\text{co-BH}(k),n} \text{ to } L_{\text{BH}(k)}.$$

Then by percolating down the  $k$  levels of the  $\text{BH}$ , we can define a sparse set with at most  $k$  hard strings of length  $n$  that lets us reduce  $\bar{\text{SAT}}$  to  $\text{SAT}$ .

In the generalization, instead of having one fixed level from which we can start to define the sparse set, for each length  $n$ , there is some  $m$  which is polynomial in  $n$  such that we can map

$$L_{\text{co-BH}(m),n} \text{ to } L_{\text{BH}(m)}.$$

Therefore for each length, we can define the sparse set by percolating down polynomially many levels.  $\square$

**6. Extensions and related results.**

**Other levels of the polynomial time hierarchy.** This work can be viewed as a study of oracle access mechanisms within the PH. The results given above generalize for every  $\Delta_i^P$  in the PH. Recall

$$\Delta_i^P \stackrel{\text{def}}{=} P^{\Sigma_{i-1}^P}.$$

A Boolean hierarchy and query hierarchy,  $P^{\Sigma_{i-1}^P[k]}$ , can be defined within each  $\Delta_i^P$ . If these hierarchies collapse, then there exists a sparse set  $S$  such that  $\Pi_{i-1}^P \subseteq \Sigma_{i-1}^{P,S}$ , and this collapses the PH to  $\Delta_{i+1}^P$ . In other words, for any  $\Delta_i^P$  and  $k$ , if  $k$  queries are as powerful as  $k + 1$  queries, the PH collapses.

**THEOREM 6.1.** *The PH collapses  $\iff$  there exist  $k, i$ ,  $P^{\Sigma_i^P[k]} = P^{\Sigma_i^P[k+1]}$ .*

In contrast, for each  $\Sigma_i^P$  in the PH, one query is enough. Recall

$$\Sigma_i^P \stackrel{\text{def}}{=} NP^{\Sigma_{i-1}^P}.$$

It is easy to prove from the quantifier structure of the PH [25] that for all  $i$ ,  $NP^{\Sigma_i^P} = NP^{\Sigma_i^P[1]}$ . This is a generalization of the result in [6] that  $NP^{NP} = NP^{NP[1]}$ . Thus for all the NP levels of the PH, one query is enough, yet if one query is enough at any of the P levels, then the PH collapses.

**Small machines for  $\overline{\text{SAT}}$ .** This work can also be viewed as a downward structural result. If the BH collapses, then the languages in co-NP are forced to have a certain structure.

A language  $L$  has *small NP (P) machines* if there exists a sequence of NP (P) machines  $\{N_i\}$  such that:

- (1)  $L(N_i) = L^{\leq i}$ .
- (2) There is a polynomial  $p_{\text{size}}(\cdot)$  such that for all  $i$ ,  $|N_i| \leq p_{\text{size}}(i)$ .
- (3) There is a polynomial  $p_{\text{run}}(\cdot)$  such that for all  $i$  and  $x$  with  $|x| \leq i$ ,  $p_{\text{run}}(i)$  bounds the running time of  $N_i(x)$ .

Small NP machines characterize nonuniform NP algorithms in much the same way that polynomial size circuits characterize nonuniform P algorithms. We show that a language  $L$  has small NP machines if and only if there exists a sparse  $S$  such  $L \in NP^S$ . As we saw in Theorem 5.3, this characterization ties in nicely with the oracle replacement technique. In fact, these two concepts are used in [11] to unify all the sparse oracle results of Karp and Lipton [12], Mahaney [17], Long [16], Yap [26], and Kadin [10].

**THEOREM 6.2.** *For all  $L$ , there exists a sparse set  $S$  such that  $L \in NP^S \iff$  there exist small NP machines for  $L$ .*

*Proof.* ( $\implies$ ) Let  $S$  be sparse, and let  $N$  be the NP machine such that  $L = L(N^S)$ .

Let  $p_1(\cdot)$  be a polynomial that bounds  $\|S^{\leq n}\|$ .

Let  $p_2(\cdot)$  be a polynomial that bounds the running time of  $N$ .

Define  $N_i$  to be  $N$  with  $S^{\leq p_2(i)}$  encoded as a table in  $N_i$ 's states and a routine to look up strings in the table in place of  $N$ 's query state. If the table is sorted by increasing length, then on an input  $x$  of length  $n \leq i$ , only the portion of the table containing strings of length less than or equal to  $p_2(n)$  has to be searched to find a query answer. This portion of the table contains at most  $p_1(p_2(n))$  strings, each of at most  $p_2(n)$  characters long. So each query takes  $N_i$  roughly  $p_2(n)p_1(p_2(n))$  steps to answer. Therefore

$$\begin{aligned} p_{run}(n) &= p_2(n)(p_2(n)p_1(p_2(n))), & \text{and} \\ p_{size}(i) &= |N| + p_2(i)p_1(p_2(i)) + c, \end{aligned}$$

which is the size of  $N$  plus the size of the table and lookup routine. Note that because the table is sorted,  $p_{run}$  actually bounds the running time of  $N_i$  on all inputs.

( $\Leftarrow$ ) Let  $\{N_i\}$  be the small machines for  $L$  with polynomials  $p_{run}$  and  $p_{size}$  as defined above. We can assume  $p_{size}$  is monotonically increasing. Add dummy states to each  $N_i$  so that it is exactly  $p_{size}(i)$  characters long. Let the sparse oracle  $S$  consist of the machines  $N_i$  themselves.  $S$  is sparse since it has at most one string of each length.

The  $NP^S$  machine  $N$  for recognizing  $L$  works as follows. On input  $x$  with  $|x|=n$ ,  $N$  guesses an NP machine of length  $p_{size}(n)$  and verifies with one query to  $S$  that it has guessed  $N_n$ . It then simulates  $N_n$  on  $x$ .  $\square$

Since the collapse of the BH implies the existence of a sparse set  $S$  such that  $co-NP \subseteq NP^S$ , we have the following corollary.

**COROLLARY 6.3.** *For all  $k$ , if  $BH(k) = co-BH(k)$ , then every language in  $co-NP$  has small NP machines.*

If  $\overline{SAT}$  has small NP machines, then there is a sequence of NP machines that recognize initial segments of  $\overline{SAT}$ . The sequence is uniform in the sense that a single polynomial bounds the running time of all the machines, and the difference in size from one machine to the next grows polynomially. The bound on the size of the machines implies that the machines do not simply contain tables of all the satisfiable or unsatisfiable strings of a certain length since there is no polynomial that bounds the number of such strings. Thus each machine in the sequence really executes some NP algorithm to recognize unsatisfiable strings. The existence of such machines would imply that nondeterminism is useful in recognizing  $\overline{SAT}$  and that  $co-NP$  is somehow close to NP.

In many ways, the collapse of the BH is similar to NP being closed under complementation. If the BH collapses, then there exists a polynomial time function relative to a sparse oracle that reduces  $\overline{SAT}$  to SAT, there exists a *nonuniform* NP algorithm for  $\overline{SAT}$ , and the PH collapses — in other words, after two quantifier alternations, polynomially bounded universal and existential quantifiers are equivalent.

Theorem 6.2 also goes through in the deterministic case.

**THEOREM 6.4.** *For all  $L$ , there exists a sparse set  $S$  such that  $L \in P^S \iff$  there exist small P machines for  $L$ .*

*Proof.* The proof is parallel to the proof of Theorem 6.2 except that when the small machines are put into the sparse oracle, a P machine cannot guess the small P machine for  $L$ . The prefixes of the small P machines must be encoded into the sparse oracle so that a deterministic machine can get hold of the small machines.  $\square$

There is a well-known result of Meyer published in [3] that relates sparse oracles for NP and polynomial size circuits for NP:

$$\exists \text{ sparse } S \text{ with } NP \subseteq P^S \iff NP \text{ has polynomial size circuits.}$$

Hence we know that the following are equivalent:

- (1) There exists sparse  $S$  such that  $SAT \in P^S$ .
- (2) There exist small P machines for SAT.
- (3) There exist polynomial size circuits for SAT.

**7. Conclusion.** The BH, QH, and  $QH_{||}$  intertwine to form a rich structure inside  $P^{NP}$ . This structure is strongly related to the structure of the PH itself.

Looking the other way, if these hierarchies within  $P^{NP}$  collapse, then there is a surprising structure among the unsatisfiable strings of each length: a small number of key unsatisfiable formulas must exist. These formulas can be put into a sparse oracle  $S$  such that  $co-NP \subseteq NP^S$ , or they can be encoded into a sequence of small NP machines that accept initial segments of  $\overline{SAT}$ .

There are still some open problems related to these results:

- (1) What about the collapse of  $P^{NP}$  to  $P^{NP^{O(\log n)}}$  or other classes above  $P^{NP^{O(\log n)}}$ ? It would seem that such a collapse should have some drastic implications. Krentel has shown that the collapse of the corresponding function classes,  $FP^{NP^{O(\log n)}} = FP^{NP}$ , implies  $P = NP$  [13].
- (2) The existence of the key (hard) unsatisfiable strings used in this paper seems unlikely. An interesting approach for further research would be to consider what kind of relationships of this type can and cannot exist among Boolean formulas. What other conditions (say collapses above  $P^{NP^{O(\log n)}}$ ) imply similar interplay among formulas?

**Acknowledgments.** I am grateful to Professor Juris Hartmanis for his support and guidance. It was partly through discussions with him that the formulation of the collapse in terms of sparse sets evolved. I am also grateful to Stuart Allen, Lane Hemachandra, Steve Mitchell, Geoff Smith, and Vijay Vazirani for helpful discussions.

## REFERENCES

- [1] A. AMIR AND W. GASARCH, *Polynomial terse sets*, Proc. 2nd Structure in Complexity Theory Conference, 1987, pp. 22–27.
- [2] R. BEIGEL, *Bounded queries to SAT and the Boolean hierarchy*, Tech. Rep. TR 87-07, Department of Computer Science, The Johns Hopkins University, Baltimore, Maryland, June 1987.
- [3] L. BERMAN AND J. HARTMANIS, *On isomorphisms and density of NP and other complete sets*, SIAM J. Comput., 6 (1977), pp. 305–322.
- [4] J. CAI, T. GUNDERMANN, J. HARTMANIS, L. HEMACHANDRA, V. SEWELSON, K. WAGNER, AND G. WECHSUNG, *The Boolean hierarchy I: Structural properties*, SIAM J. Comput., 17 (1988).
- [5] J. CAI AND G. MEYER, *Graph minimal uncolorability is  $D^P$ -complete*, SIAM J. Comput., 16 (1986), pp. 259–277.
- [6] M. FURST, J. SAXE, AND M. SIPSER, *Parity, circuits, and the polynomial-time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–27.
- [7] L. A. HEMACHANDRA, *The strong exponential hierarchy collapses*, Proc. 19th ACM Symposium on Theory of Computing, 1987, pp. 110–122.
- [8] J. HOPCROFT AND J. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Massachusetts, 1979.
- [9] N. IMMERMAN, *Nondeterministic space is closed under complement*, Proc. 3rd Structure in Complexity Theory Conference, 1988, pp. 112–115.



- [10] J. KADIN,  $P^{NP[\log n]}$  and sparse Turing complete sets for NP, Proc. 2nd Structure in Complexity Theory Conference, 1987, pp. 33–40; J. Comput. System Sci., to appear.
- [11] ———, *Restricted Turing Reducibilities and the Structure of the Polynomial Time Hierarchy*, Ph.D. thesis, Cornell University, Ithaca, New York, February 1988.
- [12] R. KARP AND R. LIPTON, *Some connections between nonuniform and uniform complexity classes*, Proc. 12th ACM Symposium on Theory of Computing, 1980, pp. 302–309.
- [13] M. KRENTEL, *The complexity of optimization problems*, Proc. 18th ACM Symposium on Theory of Computing, 1986, pp. 69–76.
- [14] R. LADNER, N. LYNCH, AND A. SELMAN, *A comparison of polynomial time reducibilities*, Theoret. Comput. Sci., 1 (1975), pp. 103–124.
- [15] K. LANGE, B. JENNER, AND B. KIRSIG, *The logarithmic hierarchy collapses:  $A\Sigma_2^L = A\Pi_2^L$* , Proc. 14th International Colloquium on Automata, Languages, and Programming (ICALP), Lecture Notes in Computer Science 267, Springer-Verlag, New York, 1987, pp. 531–541.
- [16] T. LONG, *A note on sparse oracles for NP*, J. Comput. System Sci., 24 (1982), pp. 224–232.
- [17] S. MAHANEY, *Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis*, J. Comput. System Sci., 25 (1982), pp. 130–143.
- [18] C. PAPADIMITRIOU AND D. WOLFE, *The complexity of facets resolved*, Proc. 26th IEEE Annual Symposium on Foundations of Computer Science, 1985, pp. 74–78.
- [19] C. PAPADIMITRIOU AND M. YANNAKAKIS, *The complexity of facets (and some facets of complexity)*, J. Comput. System Sci., 28 (1984), pp. 244–259.
- [20] U. SCHÖNING AND K. WAGNER, *Collapsing oracle hierarchies, census functions and logarithmically many queries*, Tech. Rep. 140, Institut für Mathematik, University of Augsburg, Augsburg, West Germany, April 1987.
- [21] L. STOCKMEYER, *The polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1977), pp. 1–22.
- [22] R. SZELEPCSÉNYI, *The method of forcing for nondeterministic automata*, Bull. European Assoc. Theoret. Comput. Sci., 33 (1987), pp. 96–100.
- [23] S. TODA,  $\Sigma_2SPACE(n)$  is closed under complement, J. Comput. System Sci., 35 (1987), pp. 145–152.
- [24] K. WAGNER, *Number-of-query hierarchies*, Tech. Rep. 158, Institut für Mathematik, University of Augsburg, Augsburg, West Germany, October 1987.
- [25] C. WRATHALL, *Complete sets and the polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1977), pp. 23–33.
- [26] C. YAP, *Some consequences of non-uniform conditions on uniform classes*, Theoret. Comput. Sci., 26 (1983), pp. 287–300.