The Complexity of Parallel Search[1]

Richard M. Karp[2]
Eli Upfal[3]
Avi Wigderson[4]

[1]Some of the results in this paper appear in preliminary form in [KUW 85a] and [KUW 85c].
[2]Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California. Research supported by NSF Grant #DCR-8411954.
[3]IBM Almaden Research Center. Research supported by a Weizmann Fellowship.
[4]Computer Science Department, Hebrew University, Jerusalem, Israel. Research supported by DARPA Grant #NOO039-83-C-1036 at the University of California at Berkeley.

ABSTRACT

This paper studies parallel search algorithms within the framework of independence systems. It is motivated by earlier work on parallel algorithms for concrete problems such as the determination of a maximal independent set of vertices or a maximum matching in a graph, and by the general question of determining the parallel complexity of a search problem when an oracle is available to solve the associated decision problem. Our results provide a parallel analogue of the self-reducibility process that is so useful in sequential computation.

An abstract independence system is specified by a ground set E and a family of subsets of E called the independent sets; it is required that every subset of an independent set be independent. We investigate parallel algorithms for determining a maximal independent set through oracle queries of the form "Is the set A independent?", as well as parallel algorithms for determining a maximum independent set through queries to a more powerful oracle called a rank oracle. We also study these problems for three special types of independence systems: matroids, graphic matroids and partition matroids.

We derive lower and upper bounds on the deterministic and randomized complexity of these problems. These bounds are sharp enough to give a clear picture of the processor-time trade-offs that are possible, to establish that randomized parallel algorithms are much more powerful than deterministic ones, and to show that even randomized algorithms cannot make effective use of extremely large numbers of processors.

1. INTRODUCTION

1.1 Motivation

This paper grew out of an attempt to understand the relative difficulty of decision problems and search problems in the context of parallel computation. In a decision problem one is asked to determine whether a certain object exists; the object might be a hamiltonian circuit, eulerian circuit or perfect matching in a given graph, or a satisfying assignment for a given boolean formula. In a search problem, on the other hand, one is required to produce the object when it exists. Thus the general form of a decision problem is "for a given input x, determine whether there exists a y such that the predicate $F(x,y)$ is true", whereas a search problem is of the form "for a given input x, determine whether there exists a y such that $F(x,y)$ is true and, if so, exhibit such a y".

In the realm of sequential computation the distinction between decision problems and search problems is relatively unimportant, because any search problem can be solved efficiently on the assumption that an efficient subroutine, or "oracle", is available for a closely related decision problem. The standard example is the construction of a truth-value assignment satisfying a given boolean formula, given a subroutine to test whether boolean formulas are satisfiable. Let the given formula be $F(x_1, x_2,...,x_n)$, where the x_i are variables which can assume the values 0 and 1. One call on the oracle determines whether the given formula is satisfiable. Assuming it is, a second call on the oracle is made to determine whether the formula $F(1, x_2,...,x_n)$, obtained by fixing x_1 to the value 1, is satisfiable. If so, the problem is reduced to finding a satisfying assignment for $F(1, x_2,...,x_n)$, a boolean formula with n-1 variables. If not, the formula $F(0, x_2,...,x_n)$ must be satisfiable, and so again the problem is reduced to finding a satisfying assignment for a formula with $n - 1$ variables. Each oracle call after the first fixes one more variable, and, if the original formula is satisfiable, a satisfying assignment is produced after $n + 1$ oracle calls.

A similar process can be defined for any search problem specified by a predicate $F(x,y)$. The string $x \in \{0,1\}^*$ specifies the problem instance, and the string $y \in \{0,1\}^*$ is a solution to that instance if the predicate $F(x,y)$ is true. We assume that, for each instance x, all solutions y are of a common length n, and that the value of n is easily determined from x. The process requires an oracle for the following decision problem: given the input x and a specification s of the bits that are to occur in certain positions of the string y, determine whether there exists a solution y satisfying the specification. Instances with solutions of length n can be solved with $n + 1$ successive calls on the oracle for this associated decision problem, and thus the sequential complexity of the search problem is greater than that of the decision problem by at most a factor of order n.

The process of solving a search problem by successive oracle calls is of special interest when the search problem enjoys a property called self-reducibility. The search problem associated with predicate $F(x,y)$ is self-reducible if the question "Does there exist a y such that $F(x,y)$ holds and y is consistent with specification s?" can be trivially converted to a question of the form "Does there exist a y' such that $F(x', y')$ holds?", where the length of the derived instance x' is no greater than the length of the original instance x, and the length of y' is equal to n minus the number of bits of the solution that are fixed by the specification s. In such cases the decision problems presented to the oracle involve the same predicate that defines the given search problem. For example, the satisfiability problem for boolean formulas is self-reducible because the question of whether a given boolean formula has a satisfying truth assignment in which certain variables are fixed at specified values, is easily converted to the question of whether a related boolean formula is satisfiable. Most search problems of practical interest are self-reducible. In the context of sequential computation, where the main issue is often whether a given search problem can be solved in polynomial time, the distinction between a self-reducible search problem and the corresponding decision problem is usually of secondary importance, since the self-reducibility process ensures that the search problem can be solved in polynomial time whenever the decision problem can.

In the context of parallel computation the distinction between search problems and decision problems is far more important because we are interested in algorithms that run in sublinear time. If we are trying for a sublinear algorithm then the self-reducibility process, with its $n + 1$ successive oracle calls, is not very promising. And in fact there are many cases where a decision problem is easy or even trivial to solve in parallel, but the corresponding search problem is challenging. For example, testing whether a connected graph is eulerian is easily done by checking whether all degrees are even, but finding an eulerian circuit is harder [AIS84]. The problem of deciding whether a graph contains a maximal independent set of vertices is vacuous, since the answer is always "yes", but constructing a fast parallel algorithm to find a maximal independent set has proved challenging

[KW84,L85]. Other examples where search problems and the associated decision problems seem to require very different approaches are biconnectivity [VT84], pattern matching [G84] and strong orientation [V84].

It is natural to ask whether there is a useful parallel analogue of the self-reducibility process. To find a satisfying instance of a boolean formula this parallel self-reducibility process would execute a series of steps, each involving p simultaneous calls to a satisfiability oracle, where the parameter p indicates the allowed degree of parallelism. Each call would involve a formula derived from the original problem by making a partial truth assignment, in which some variables are set to 1 or 0, and the others are left free. But how should these partial truth assignments be chosen, so as to minimize the number of parallel steps for a given choice of p?

We have chosen to study a generalization of this problem in the context of independence systems. An independence system S is defined by a pair (E,I), where E is a finite set of n elements called the ground set and I is a family of subsets of E called the independent sets of S. The independent sets are closed under containment : if $A \subset B \subseteq E$ and $B \in I$, then $A \in I$. The subsets of E that are not independent are called dependent. A set $M \subseteq E$ is called a maximal independent set if it is an independent set and is not a proper subset of any independent set.

We study parallel algorithms whose goal is to construct a maximal independent set. At each step such an algorithm presents p queries to an independence oracle; each query is of the form "Is set A independent?". The algorithm terminates when a maximal independent set has been determined. It is easy to see that the parallel solution of a search problem, using oracle calls to determine whether solutions exist satisfying certain partial specifications, can be modelled within this framework. Let $F(\cdot,\cdot)$ be the predicate associated with a search problem, let x be an instance, and let n be the length of each solution associated with input x. We define an independence system with ground set $\{e_1,e_2,\ldots,e_n, \overline{e}_1,\overline{e}_2,\ldots,\overline{e}_n\}$ such that the maximal independent sets are in one-to-one correspondence with the solutions y. The maximal independent set corresponding to y is equal to $\{e_i | y_i = 1\} \cup \{\overline{e}_i | y_i = 0\}$. The set $\{e_{i_1},e_{i_2},\ldots,e_{i_k}, \overline{e}_{j_1},\overline{e}_{j_2},\ldots,\overline{e}_{j_k}\}$ is independent if and only if there exists a solution y satisfying $y_{i_1} = y_{i_2} = \cdots = y_{i_k} = 1$ and $y_{j_1} = y_{j_2} = \ldots = y_{j_k} = 0$.

In addition to the question of parallel self-reducibility, there were several other concrete problems that motivated us to study the problem of constructing a maximal independent set in an abstract independence system. One of these is the problem of constructing a maximal independent set of vertices in a graph. Here the ground set is the set of vertices in a graph, and a set is independent if no two of its vertices are adjacent. There is an obvious sequential algorithm which builds up a maximal independent set by inspecting the vertices one at a time. The independent set is initially empty, and each vertex in turn is added to it if doing so preserves independence. Parallel algorithms for this problem are investigated in ([KW84] [L85]). However, the algorithms given in those papers cannot be expressed within the present model, since they are not restricted to gathering information about the graph by queries to an independence oracle.

We shall also consider a second related problem motivated by our work on a parallel algorithm to construct a perfect matching in a graph G [KUW84]. Our algorithm was based on a fast parallel subroutine to solve the following problem: given H, a subgraph of G, and a set of edges A within H, determine the maximum, over all perfect matchings M in subgraph H, of the number of edges in the intersection of A with M (our algorithm has the property that, in every subroutine call, the graph H is known to possess a perfect matching). The problem of searching for a perfect matching using parallel calls on this subroutine can be generalized to an interesting parallel search problem involving an independence system. In order to pose this general problem we require some further definitions.

Let $S=(E,I)$ be an independence system. A set $M \subseteq E$ is called a maximum independent set if it is an independent set and no independent set has larger cardinality. The rank of a set $A \subseteq E$ is defined as the maximum cardinality of its intersection with a maximum independent set. The restriction of independence system S to a set $B \subseteq E$, denoted S/B, is an independence system defined as follows: the ground set of S/B is B, and a set $A \subseteq B$ is independent in S/B if and only if it is independent in S. When the identity of the independence system S is fixed, $\text{rank}_B(A)$ will denote the rank of A in S/B.

We shall be interested in parallel algorithms for the following problem: given an independence system $S = (E,I)$ and an oracle which answers questions of the form "What is $\text{rank}_B(A)$?", find a maximum independent set in S. The problem of constructing a maximum matching corresponds to the special case in which the ground set is the set of edges of a graph, and the independent sets are the matchings in a graph. The subroutine used in [KUW84] is nothing but a rank oracle for this independence system.

1.2 The Model of Computation

This paper is concerned with parallel algorithms for the following two problems:

The Maximal Independent Set Problem: given an independence system $S = (E,I)$, and an oracle which answers questions of the form "Is the set A independent?" find a maximal independent set in S.

The Maximum Independent Set Problem: given an independence system $S = (E,I)$ and an oracle which answers questions of the form "What is $\text{rank}_B\ (A)$?", find a maximum independent set in S.

Associated with each parallel algorithm for one of these problems is a positive integer p called the number of processors. An algorithm with p processors progresses in parallel steps; in each (parallel) step the algorithm presents p queries to its oracle. An algorithm may be either deterministic or randomized, and its oracle may be either an independence oracle or a rank oracle. In the case of an independence oracle the goal is to find a maximal independent set, and in the case of a rank oracle it is to find a maximum independent set. Thus we have four classes of algorithms, and, within each class, we are interested in the number of steps required by a p-processor algorithm operating on an independence system whose ground set contains n elements.

We formalize the concept of a parallel algorithm in terms of decision trees. As an illustration we deal here with the most complicated case: a randomized algorithm using a rank oracle. A (n,p) randomized decision tree with rank oracle is a rooted tree with three types of nodes: randomization nodes, query nodes and leaves. Every query node is labelled with p pairs of sets (A_1,B_1), (A_2,B_2), ... (A_p,B_p), where $A_i \subseteq B_i \subseteq \{1,2,...,n\}$, $i = 1,2,...,p$; the branches from such a node to it children are labelled with all possible joint values of the rank of A_i in S/B_i, $i = 1,2,...,p$. Every leaf is labelled with a subset of E.

Given an independence system S with ground set $\{1,2,...,n\}$, an execution of the algorithm corresponds to the traversal of a root-leaf path in the tree. Whenever a randomization node is encountered, one of its children is chosen at random (all choices being equally likely), and the edge to that child is traversed.

Whenever a query node is encountered, the p associated queries are executed, and the outcomes of the queries determine the branch to be followed. In order for the algorithm to be correct, it is required that any leaf which can be reached when the algorithm is executed on independence system S must be labelled with a maximum independent set in S.

Because of the randomization nodes, the path through decision tree H which is selected when the tree is executed on independence system S, is a random variable. Let $c(H,S)$, the expected cost of executing tree H on independence system S, be the expected number of query nodes occurring in the selected path. Let $T_{prob}^{rank}(n,p) = \min_H \max_S c(H,S)$, where H ranges over all (n,p) randomized decision trees with rank oracle, and S ranges over all independence systems with ground set $\{1,2,...,n\}$.

In a similar manner, we can define randomized algorithms with an independence oracle, and deterministic algorithms with a rank oracle or independence oracle. In the case of a deterministic algorithm the decision tree contains no randomization nodes, so that the same path is always selected when the algorithm is executed on a given independence system. In the case of an independence oracle, any computation with input $S = (E,I)$ must terminate at a leaf labelled with the name of a maximal (not necessarily maximum) independent set in S. We can define a complexity measure associated with each of the four classes of algorithms. These complexity measures will be denoted $T_{det}^{ind}(n,p)$, $T_{det}^{rank}(n,p)$, $T_{prob}^{ind}(n,p)$ and $T_{prob}^{rank}(n,p)$.

1.3 Summary of Theorems

In this section we summarize the main results of the paper and comment on their significance.

Section 2 is concerned with lower bounds on the deterministic complexity of both the maximal independent set problem and the maximum independent set problem. The main results of that section are as follows:

Theorem 3. $T_{det}^{ind}(n,p) = \Omega\left[\dfrac{n}{\log p}\right]$.

Theorem 4. $T_{det}^{rank}(n,p) = \Omega\left[\dfrac{n}{\log np}\right]$.

Section 3 is concerned with both lower and upper bounds on the randomized complexity of the maximal independent set problem. The main results are:

Theorem 6. $T^{ind}_{prob}(n, n^{\frac{3}{2}}) = O(\sqrt{n})$.

Theorem 7. $T^{ind}_{prob}(n,p) = \Omega\left[\frac{n}{\log np}\right]^{\frac{1}{3}}$.

Section 4 investigates the randomized complexity of the maximum independent set problem, and gives the following result:

Theorem 8. $T^{rank}_{prob}(n,n) = O(log^2 n)$.

In interpreting these theorems, it is useful to keep in mind that, when the size of the ground set is n, the time required for a sequential algorithm to solve any one of our basic problems is n. Comparing Theorems 6 and 8 with Theorems 3 and 4, we see that randomized parallel algorithms are much more powerful than deterministic ones. In the deterministic case, the speed-up obtained by using p processors is only logarithmic in p; this is true whether we are solving the maximal independent set problem using an independence oracle (Theorem 3) or the maximum independent set problem using a rank oracle (Theorem 4). On the other hand, a randomized algorithm with $p = n^{\frac{3}{2}}$ can solve the maximal independent set problem in $O(n^{\frac{1}{2}})$ steps using an independence oracle (Theorem 6); this represents a speed-up over the best sequential algorithm by a factor of $n^{\frac{1}{2}}$, and establishes that there does exist a useful parallel version of the self-reducibility process, provided that randomization is permitted. In the case of a rank oracle the speed-up achievable with a randomized algorithm is even more impressive; a running time of $O(log^2 n)$ can be achieved using n processors.

Theorem 7 is one of the most interesting results in the paper. It sets an absolute limit on the speed-up that a randomized algorithm using an independence oracle can achieve; as long as the number of processors is subexponential it is impossible to obtain an execution time much below $n^{\frac{1}{3}}$. This result, when contrasted with Theorem 8, shows that, in the context of randomized algorithms, a rank oracle is much more powerful than an independence oracle; Theorems 3 and 4 indicate that this is less true in the case of deterministic computation.

In Section 5 we consider matroids, a widely studied special class of independence systems. A matroid is an independence system $S = (E,I)$ such that, within every restriction S/B, the maximal and maximum independent sets coincide. When we consider algorithms whose inputs are restricted to matroids, the four complexity measures that arise are denoted $TMAT^{ind}_{\det}(n,p)$, $TMAT^{rank}_{\det}(n,p)$, $TMAT^{ind}_{prob}(n,p)$ and $TMAT^{rank}_{prob}(n,p)$.

The following theorem shows that, in the case of matroids, the rank oracle is extremely powerful.

Theorem 9. $TMAT^{rank}_{\det}(n,n) = 1$.

It turns out that the lower bound given in Theorem 7 holds even when the independence systems presented as input are restricted to matroids. Thus we have the following theorem.

Theorem 10. $TMAT^{ind}_{prob}(n,p) = \Omega\left[\frac{n}{\log np}\right]^{\frac{1}{3}}$.

However, when the inputs are restricted to matroids the complexity of deterministic computation with an independence oracle becomes reduced.

Theorem 11. $TMAT^{ind}_{\det}(n,n) = O(\sqrt{n})$.

Finally, in Section 6, we consider the very special case of graphic matroids. In a graphic matroid, the elements of the ground set are the edges of a graph, and a set of edges is independent if it contains no cycle of the graph; thus, the independent sets correspond to the forests in the graph. Let the prefix *TGRAPH* denote the complexity of an algorithm when the independence systems presented as input are required to be graphic. Then we have the following theorems.

Theorem 12. For every positive integer d, $TGRAPH^{ind}_{prob}(n, n^{2d+1}) = O(n^{\frac{2}{d}})$.

Theorem 13.

(i) For every integer $d \geq 4$, $TGRAPH^{ind}_{\det}(n, n^{2d+1}) = O(n^{\frac{3}{d}})$

(ii) $TGRAPH^{ind}_{\det}(n, n^{4\lceil \log_2 n \rceil + 1}) = O(log\ n)$.

These results show that much more impressive speed-ups are possible for graphic matroids than for general matroids.

2. LOWER BOUNDS FOR DETERMINISTIC ALGORITHMS

In this section we prove the following two lower bounds.

Theorem 3. $T_{\det}^{ind}(n,p) = \Omega\left[\frac{n}{\log p}\right]$.

Theorem 4. $T_{\det}^{rank}(n,p) = \Omega\left[\frac{n}{\log np}\right]$.

As a step toward the proof of these results we introduce a search problem called the group testing problem. This problem is of interest in its own right, and lower bounds on the complexity of parallel algorithms for its solution will easily translate into the lower bounds promised in Theorems 3 and 4.

2.1 The Group Testing Problem

An instance of the group testing problem is specified by a finite set U called the universe and a nonempty set $X \subseteq U$. The problem is to identify any element of the set X. Algorithms for solving this problem gather information by asking questions of the form "Is $A \cap X$ empty ?", where A is a subset of U. Associated with each algorithm is a positive integer N, specifying that the universe is the set $\{1,2,...,N\}$, and a positive integer p, the number of processors, which determines the number of queries that can be asked at a single step.

The definition of an algorithm for the group testing problem can be formalized in terms of decision trees as follows. A (N,p) probabilistic decision tree for the group testing problem is a rooted tree with three kinds of nodes: randomization nodes, query nodes and leaves. Each query node is labelled with a p-tuple $(A_1,A_2,...,A_p)$ of subsets of $\{1,2,...,N\}$, and it has 2^p children, corresponding to the possible outcomes of the queries "Is $A_i \cap X$ nonempty?", for $i = 1,2,...,p$. Each leaf is labelled with an element of $\{1,2,...,N\}$. The input to the algorithm is a nonempty set $X \subseteq \{1,2,...,N\}$. On input X, an execution of the algorithm corresponds to the traversal of a root-leaf path, with coin tossing determining the selection of branches from randomization nodes and the outcomes of queries determining the selection of branches from query nodes. In order for the algorithm to be correct it is required that, for all inputs X, every leaf that can be reached when the algorithm is ececuted on input X must be labelled with an element of X. Exactly as in the case of decision trees for the maximal or maximum independent set problem, a deterministic algorithm is one without randomization nodes. The cost of executing decision tree H on input X is denoted $\bar{c}\,(H,X)$, and it is equal to the expected number of query nodes visited while traversing the tree on input X. The randomized complexity of the group testing problem is defined as $\bar{T}_{prob}\,(N,p) = \min_H \max_S \bar{c}\,(H,X)$, where H ranges over (N,p) probabilistic decision trees for the group testing problem, and X ranges over the subsets of $\{1,2,...,N\}$. The deterministic complexity of the group testing problem, denoted $\bar{T}_{\det}\,(N,p)$, is defined similarly.

2.2 Complexity of the Deterministic Group Testing Problem

Theorem 1. For all N and p, $\frac{\log N}{\log(p+1)} \le \bar{T}_{\det}(N,p) \le \frac{\log N}{\log(p+1)} + 1$.

Proof: The upper bound is achieved by the following natural algorithm, which maintains a set A guaranteed to have a nonempty intersection with X. Initially $A = \{1,2,...,N\}$, and the algorithm terminates when A becomes a singleton set. At a general step the algorithm partitions A into $p + 1$ sets $A_1, A_2, ..., A_{p+1}$, each of which is of size at most $\lceil \frac{|A|}{p+1} \rceil$. It then tests in parallel whether $A_i \cap X$ is empty, for $i = 1,2,...,p$. The result of these tests identifies one of the sets $A_1, A_2, ..., A_{p+1}$ as having a nonempty intersection with X, and the lowest-indexed such set becomes the new A. The number of steps executed by this deterministic algorithm is at most $\lceil \log_{p+1} N \rceil$.

The lower bound is obtained by a simple adversary argument similar to one given in [FRW84]. The starting point for the lower bound is to observe that a complete description of the information available to the algorithm at any point during its execution can be specified by a collection of sets $\{\{L_i\}\}$, each of which is known to have a nonempty intersection with X, and a single set K which is known to have an empty intersection with X. The sets L_i are called positive sets, and K is called a negative set. The positive sets are all disjoint from K. Initially $\{1,2,...,N\}$ is the only positive set, and K is empty. When the result of a query becomes known, the positive sets L_i and the negative set K are updated as follows: if the query set is A and it is found that $A \cap X$ is nonempty,then $A \cap \bar{K}$ is added to the list of positive sets. On the other hand, if it is found that $A \cap X$ is empty, then the set K is replaced by $K \cup A$, and each positive set L_i is replaced by $L_i \cap \bar{A}$. When p queries are executed simultaneously, the information available to the algorithm can be updated by making updates corresponding to the individual queries, in an arbitrary order.

The algorithm cannot terminate until it creates a positive singleton set. To see this, suppose that the algorithm concluded that element a was contained in X, at a time when no positive set was a singleton. Then the algorithm might be incorrect, since the available information is consistent with the possibility that X consists of all elements of $\{1,2,...,N\}$ except a and the elements of K.

The adversary strategy is based on the idea of measuring the progress of the algorithm by the minimum size of a positive set. This minimum size is initially n and, as we have just pointed out, it is equal to 1 when the algorithm terminates. We shall show that if, at the beginning of a step, the minimum size of a positive set is t, and the p queries $A_1, A_2, ..., A_p$ are then simultaneously executed, then the adversary will be able to specify the outcomes of these queries so that, after the information is updated, the size of each positive set is at least $\lceil \frac{t}{p+1} \rceil$.

In choosing his answers to the queries $A_1, A_2, ..., A_p$ the adversary maintains a set W of unanswered queries. Initially, $W = \{A_1, A_2, ..., A_p\}$. The adversary then does the following

```
while $W \neq \varphi$ do
    if $W$ contains a query whose outcome is determined by the present
    information state
    then give the unique consistent answer to all such queries and
    delete them from $W$
    else
    let $A$ be the query in $W$ such that $A \cap \bar{K}$ is of minimum
    cardinality
    if $|A \cap \bar{K}| \geq \frac{t}{p+1}$
    then, for every query $B \in W$ give the answer $B \cap X \neq \varphi$ and
    adjoin $B \cap \bar{K}$ to the list of positive sets
    else give the answer $A \cap X = \varphi$, set $K$ to $K \cup A$ and replace each
    positive set $L$ by $L \cap \bar{A}$.
```

It is easy to check that the outcomes specified for the queries are consistent and that the updates to the list of positive sets and the negative set K correctly reflect the outcomes of the queries. The fact that, after the step, each positive set is of size at least $\frac{t}{p+1}$, is a consequence of the following observations:

- at the beginning of the step, each positive set is of size at least t;
- whenever a query is answered affirmatively, the set added to the list of positive sets is of size at least $\frac{t}{p+1}$;
- whenever a query is answered negatively, the number of elements deleted from any positive set is at most $\frac{t}{p+1}$;
- the number of queries answered is p;
- all negative answers precede all positive answers.

This completes the proof of Theorem 1.

2.3 Lower Bounds from the Group Testing Model

The following theorem establishes a relationship between the deterministic group testing problem and the problems of finding a maximal independent set deterministically using an independence oracle and finding a maximum independent set deterministically using a rank oracle. This relationship yields lower bounds for the complexity of the latter two problems.

Theorem 2. Let $N = \binom{n}{\frac{n}{2}}$. Then

(a) $T^{ind}_{\text{det}}(n,p) \geq \hat{T}_{\text{det}}(N,p)$

(b) $T^{rank}_{\text{det}}(n,p) \geq \hat{T}_{\text{det}}(N,np)$

Proof: Let h be any bijection from the set $\{1,2,...,N\}$ onto the $\frac{n}{2}$-element subsets of $\{1,2,...,n\}$. This bijection induces a one-to-one correspondence between the instances X of the group testing problem with universe $\{1,2,...,N\}$ and the independence systems with ground set $\{1,2,...,n\}$ and all their maximal independent sets of size $\frac{n}{2}$. The independence system $S(X)$ corresponding to instance X of the group testing problem has as its maximal independent sets those $\frac{n}{2}$-element sets that correspond to the elements of X under the bijection h.

Let H be any (n,p) randomized decision tree with independence oracle for the maximal independent set problem. Then, using the bijection h, we can derive a

(N,p) randomized decision tree H' for the group testing problem with universe $\{1,2,...,N\}$. The execution of H' on instance X simulates the execution of H on instance $S(X)$ step-by-step. Whenever H asks the query "Is A an independent set ?", H' asks the query "Is $D \cap X$ nonempty ?", where D is the set of elements of $\{1,2,...,N\}$ which correspond under the bijection H to $\frac{n}{2}$-element sets which contain A. Clearly, the two queries have the same answer, and H' will determine an element of X at exactly the same step when H determines a maximal independent set in $S(X)$. Thus the complexity of the deterministic group testing problem with universe $\{1,2,...,N\}$ and p processors is no greater than the complexity of the deterministic problem of finding a maximal independent set in a n-element independence system using p processors. This completes the proof of (a).

The proof of (b) uses the same correspondence between instances of the group testing problem and instances of the maximum independent set problem. Let R be any (n,p) randomized decision tree with rank oracle for the maximum independent set problem. We define a corresponding (N,np) randomized decision tree R' for the group testing problem with universe $\{1,2,...,N\}$, such that, for every step executed by R on instance $S(x)$, R' executes two steps on instance X. At any step where R asks the query "What is $\text{rank}_B(A)$?", R' first asks n queries of the form "Is $D_i \cap X$ nonempty?", where D_i is the set of elements of $\{1,2,...,N\}$ which correspond under the bijection h to $\frac{n}{2}$-element subsets of E which contain at least i elements of B. If R' receives the n-tuple of answers "$D_i \cap X$ is nonempty, $i = 1,2,...,r$" and "$D_i \cap X$ is empty, $i = r+1,...,n$" it knows that, in the independence system $S(X)/B$, the size of a maximum independent set is r. At the second step in the simulation of a single step of R, R' asks r queries of the form "Is $C_i \cap X$ empty?", $i + 1,2,...,r$, where C_i is the set of elements of $\{1,2,...,n\}$ which correspond under the bijection h to $\frac{n}{2}$-element subsets of E which contain exactly r elements of B and at least i elements of A. If R' receives the n answers "$C_i \cap X$ is nonempty, $i = 1,2,...,s$" and "$C_i \cap X$ is empty", $i = s+1, s+2,...,r$, then it goes on to continue the simulation of R as if R had received the answer "$\text{rank}_B(A) = s$". Thus R' executes twice as many steps on instance X as R executes on instance $S(X)$. It follows that the complexity of deterministic group testing with a N-element universe and np processors is not more than twice as large as the complexity of the deterministic maximal independent set problem with p processors and a rank oracle, on instances where the ground set is of size n. This completes the proof of part (b).

Theorem 2 allows us to derive lower bounds on the complexity of the deterministic maximal independent set problem from lower bounds for the deterministic group testing problem, as follows.

Theorem 3. $T_{\det}^{ind}(n,p) = \Omega\left(\frac{n}{\log p}\right)$

Proof: By Theorem 2,

$$T_{\det}^{ind}(n,p) \geq \vec{T}_{\det}\left(\binom{n}{\frac{n}{2}}, p\right)$$

and, by Theorem 1,

$$\vec{T}_{\det}\left(\binom{n}{\frac{n}{2}}, p\right) \geq \log_{p+1}\binom{n}{\frac{n}{2}} p$$

But

$$\log_{p+1}\left(\binom{n}{\frac{n}{2}} p\right) = \Omega\left(\frac{n}{\log p}\right).$$

This completes the proof.

Theorem 4. $T_{\det}^{rank}(n,p) = \Omega\left(\frac{n}{\log np}\right)$

Proof: By Theorem 2,

$$2\, T_{\det}^{rank}(n,p) \geq \vec{T}_{\det}\left(\binom{n}{\frac{n}{2}}, np\right)$$

and, by Theorem 1,

$$\vec{T}_{\det}\left(\binom{n}{\frac{n}{2}}, np\right) \geq \log_{np+1}\left(\binom{n}{\frac{n}{2}}, np\right)$$

But

$$\log_{np+1} \begin{bmatrix} n \\ \frac{n}{2} \end{bmatrix} = \Omega \left[\frac{n}{\log np} \right]$$

This completes the proof.

3.3 The Randomized Complexity of the Group Testing Problem

Theorem 5. $\hat{T}_{prob}\,(N, \log N) = O(1)$.

The proof requires a probabilistic lemma.

Lemma 1. Let U be a set with 2^m elements and let X be a nonempty subset of U. Let a random linear ordering be imposed on the set X. Then, with probability greater than 1/2, there exists a nonnegative integer a such that exactly one of the first 2^a elements in the ordering is contained in X.

Proof: The result is clearly true if $|X| = 1$. Assume that $|X| \geq 2$ and condition on the position in the ordering of the second occurrence of an element of X. If this second occurrence is at b, and 2^a is the largest power of 2 less than b, then the probability that the first occurrence of an element of X is among the first 2^a positions, and hence that there is exactly one element of x among the first 2^a positions, is $\frac{2^a}{b-1}$, which is greater than 1/2. This completes the proof.

Proof of Theorem 5: We can assume without loss of generality that the size of the universe is of the form 2^m; if not, we can add an appropriate number of dummy elements, none of which lie in X. We present an algorithm which, in three steps, using $m + 1$ processors, determines an element of X with probability greater than 1/2. Assume that the elements of the universe are arranged in a random permutation. At the first step, the algorithm asks queries of the form "Is $S_k \cap X \neq \varphi$?", $k = 0,1,\ldots,m$, where S_k consists of the first 2^k elements in the ordering. If a is the first index for which the answer is "Yes" then, by Lemma 1, $|S_a \cap X| = 1$ with probability greater than 1/2. At the second step, assuming that S_a contains exactly one element of X, the algorithm identifies that element. For this purpose the algorithm establishes an arbitrary one-to-one correspondence between the 2^a elements of S_a and the 2^a a-tuples of zeros and ones. Then, in parallel, the algorithm presents the queries "Is $A_i \cap X \neq \varphi$?", for $i = 1,2,\ldots,a$, where A_i consists of those elements of S_a whose corresponding a-tuples contain 1 in the i^{th} position. Assuming that S_a contains exactly one element of X, the i^{th} query determines the i^{th} bit in the a-tuple associated with that element, and thus the a queries combine to identify the element. At the third step, the algorithm checks whether the selected element lies in X; with probability $> 1/2$ the answer will be "Yes". By repeating this three-step process until an element of X is determined, the algorithm solves the randomized group testing problem in less than six steps on the average.

4. THE RANDOMIZED COMPLEXITY OF THE MAXIMAL INDEPENDENT SET PROBLEM

In this section we prove lower and upper bounds on $T_{prob}^{ind}(n,p)$.

4.1 An Upper Bound

We begin by presenting a randomized algorithm using an independence oracle for the construction of a maximal independent set. The algorithm maintains a partition of the ground set E into three sets, IN, OUT and F; the set IN consists of elements that will appear in the final maximal independent set, OUT consists of elements that will not appear in that set, and F consists of elements about which the algorithm is still undecided.

Algorithm A

```
$IN \leftarrow \varphi$; $OUT \leftarrow \varphi$; $F \leftarrow E$
while $F \neq \varphi$ do
begin
    Select a random permutation $a_1,a_2,\ldots,a_{|F|}$ of the elements of $F$;
    in parallel, for $i = 1,2,\ldots,\lfloor\sqrt{n}\rfloor$, test whether $\{a_1,a_2,\ldots,a_i\} \cup IN$
    is independent;
    $k \leftarrow \max \{i | \{a_1,a_2,\ldots,a_i\} \cup IN$ is independent$\}$
    $IN \leftarrow IN \cup \{a_1,a_2,\ldots,a_k\}$
    $OUT \leftarrow OUT \cup \{a \in F | \{a_1,a_2,\ldots a_k\} \cup IN \cup \{a\}$ is dependent$\}$
    $F \leftarrow E - (IN \cup OUT)$
end
output $\leftarrow IN$
```

We shall investigate the expected number of elements deleted from F at each iteration. Let $T = (F,I')$ be the independence system with ground set F in which a set $A \subseteq F$ is independent if and only if $A \cup IN$ is independent in the original independence system (E,I). For $i = 0,1,...,|F|$, let q_i be the probability that an i-element set drawn at random from F is independent in T. For $j = 0,1,...,|F| - 1$ consider the following experiment: draw a j-element set A at random from F and then draw an element a at random from $F - A$. Let p_j be the conditional probability that $A \cup \{a\}$ is independent in T, given that A is independent in T. Then, for $i = 1,2,...,|F|$, $q_i = \prod_{j=0}^{i-1} p_j$. Let $m = |F|$.

Now consider the behavior of the algorithm on a permutation $a_1,a_2,<..,a_{|F|}$ of F. For any $s \leq \lfloor\sqrt{n}\rfloor$ such that $\{a_1,a_2,...,a_s\}$ is independent in T, the algorithm will place the elements $a_1,a_2,...,a_s$ into IN and will place in OUT all elements $a \in F - \{a_1,a_2,...,a_s\}$ such that $\{a_1,a_2,...,a_s\} \cup \{a\}$ is dependent . If the permutation is chosen at random then the probability that $\{a_1,a_2,...,a_s\}$ is independent is q_s, and, given that $\{a_1,a_2,...,a_s\}$ is independent, the expected number of elements $a \in F - \{a_1,a_2,...,a_s\}$ such that $\{a_1,a_2,...,a_s\} \cup \{a\}$ is dependent is $p_s\,(m - s)$. It follows that, for any $s \leq \lfloor\sqrt{n}\rfloor$, the expression $q_s(s + (1 - p_s)\,(m - s))$ gives a lower bound on the expected number of elements deleted from F at the next iteration. We shall show that, no matter what the values of the conditional probabilities p_j are, there is an s such that $q_s(s + (1 - p_s)\,(m - s))$ is at least $e^{-1}\sqrt{m} + O(1)$. We distinguish two cases:

Case 1: for $j = 0,1,...\lfloor\sqrt{m}\rfloor$, $p_j > \exp(-\frac{1}{\lfloor\sqrt{m}\rfloor})$. Choose $s = \lfloor\sqrt{m}\rfloor$. Then

$$q_s \geq \prod_{i=0}^{s-1} \exp(-\frac{1}{\lfloor\sqrt{m}\rfloor}) > e^{-1}$$

and hence

$$q_s(s + (1 - p_s)) > e^{-1}\lfloor\sqrt{m}\rfloor = e^{-1}\sqrt{m} + O(1).$$

Case 2: there exists a $j < \lfloor\sqrt{m}\rfloor$ such that $p_j \leq \exp(-\frac{1}{\lfloor\sqrt{m}\rfloor})$. Let s be the least such j. Then

$$q_s = \prod_{j=0}^{s-1} p_j \geq \left[\exp(-\frac{1}{\lfloor\sqrt{m}\rfloor})\right]^s \geq e^{-1},$$

and hence

$$q_s(s + (1 - p_s)(m - s)) \geq e^{-1}(s + (1 - e^{-\frac{1}{\lfloor\sqrt{m}\rfloor}})(m - s))$$
$$\geq e^{-1} m(1 - e^{-\frac{1}{\lfloor\sqrt{m}\rfloor}}) = e^{-1}\sqrt{m} + O(1).$$

We have now shown that, at an iteration step in which the number of elements in F is m, the expected reduction in the cardinality of F is $\geq e^{-1}\sqrt{m} + O(1)$. From this it is easy to show that the expected number of iterations required to reduce $|F|$ from its initial value n to its final value 1 is $O(\sqrt{n})$. The probabilistic analysis needed to justify this plausible conclusion is discussed in the Appendix.

Since each execution of the while loop in algorithm A can be executed in a constant number of steps using $n^{\frac{3}{2}}$ processors, we have proved the following theorem.

Theorem 6. $T_{prob}^{rank}(n, n^{\frac{3}{2}}) = O(\sqrt{n})$

4.2 A Lower Bound

The following theorem gives a lower bound which sets a limit on the extent to which the performance of Algorithm A can be improved.

Theorem 7. $T_{prob}^{ind}(n,p) = \Omega\left(\frac{n}{\log np}\right)^{\frac{1}{3}}.$

Proof: We use the following observation, which states that a lower bound on the randomized complexity of a problem is given by the minimum, over all deterministic algorithms, of the expected time required to solve the same problem when the instances are drawn from a fixed probability distribution. This observation appears to have been first stated explicitly by Yao [Y77], in a context more general than the present one. Let H be a (n,p) deterministic decision tree with an independence oracle. Let D be a probability distribution over the independence systems with ground set $\{1,2,...,n\}$, and let $T(H, D)$ be the expected number of (parallel) steps executed by decision tree H on an instance drawn from D. Then $T_{prob}^{ind}(n,p) \geq \min_H T(H, D)$

We shall pick a probability distribution D and show that, for all (n,p) deterministic decision trees H, the expected number of steps executed by H on instances drawn from D is bounded below by a certain function $f(n,p)$ which grows as fast as $(\frac{n}{\log np})^{\frac{1}{3}}$. The distribution D is concentrated on independence systems of the following form: the n-element ground set E is partitioned into $2t$ sets $A_1, A_2, \ldots, A_{2t}$, each of size $\frac{n}{2t}$, where t is a parameter to be specified later. A set B is independent if and only if, for all i, $i = 1,2,\ldots,2t$, $|B \cap A_i| \leq \frac{in}{4t^2}$. All independence systems of this type are taken to be equally likely in the distribution D.

We shall derive a lower bound on the expected number of steps required for a deterministic algorithm to find a maximal independent set in an independence system drawn from D. Since we are proving a lower bound we may assume that certain extra information is provided to the algorithm free of charge in the course of its execution; at the i^{th} step this information consists of the identity of the set A_i. We may also assume that the algorithm is required to work correctly only for independence systems that have nonzero probability in the distribution D. Thus, we work with a modified definition of a deterministic (n,p) decision tree, in which a query node at distance i from the root returns not only the answers to the p queries presented to the independence oracle, but also the identity of the set A_i, and in which the algorithm is only required to be correct on instances with nonzero probability. Let H be such a decision tree. A computation path in H defines the two sequences of events $O_1, O_2,\ldots,$ and $Q_1, Q_2,\ldots,$ where O_i is the event defined by the answers of the independence oracle in the first $i - 1$ steps, and Q_i is the event defined by the assignments of elements to the sets $A_1, A_2,\ldots,A_{i-1}$. The probability of an input instance at the start of step i of the computation is its conditional probability in D given the event $O_i \cap Q_i$.

Because the identity of $A_1, A_2,\ldots,A_{j-1}$ is determined before the j^{th} step, we may assume that, if B is an oracle query presented at the j^{th} step, then $B \subseteq A_j \cup A_{j=1} \cup \ldots A_{2t}$. We may also assume that the algorithm terminates by presenting a query at some step j of the form "Is B independent?", where B is a maximal independent subset of $A_j \cup A_{j+1} \cup \ldots \cup A_{2t}$. Since the algorithm is in a position to present such a query as soon as it has determined a maximal independent set in the independence system, the requirement that the algorithm must explicitly ask such a "verification query" extends its execution time by at most one step.

We say that an oracle query B at step j is local if either

(i) $|B \cap A_j| > \frac{jn}{4t^2}$ (so that B is a dependent set) or

(ii) B is independent and, for every $i > j$, $|B \cap \bigcup_{k \geq i} A_k| \leq (j + \frac{1}{4})(2t - i + 1)\frac{n}{4t^2}$.

Note that the outcome of a local query B at step j is determined by the cardinality of $B \cap A_j$.

The probabilistic calculations required for this proof are based on the following bounds on the tail of the hypergeometric distribution. Let n,M and N be positive integers with $M \leq N$. Let $p = \frac{M}{N}$. Then, for $0 < \beta < 1$

$$\sum_{k=0}^{\lfloor (1-\beta)np \rfloor} \frac{\binom{M}{k}\binom{N-M}{n-k}}{\binom{N}{n}} \leq \exp\left[-\frac{\beta^2 np}{2}\right]$$

and

$$\sum_{k=\lceil (1+\beta)np \rceil}^{n} \frac{\binom{M}{k}\binom{N-M}{n-k}}{\binom{N}{n}} \leq \exp\left[-\frac{\beta^2 np}{3}\right]$$

Analogous bounds are derived for the binomial distribution in [AV 79]. The present bounds can be obtained by a similar derivation based on an inequality of Hoeffding [H63]; see also the discussion in [C79]. We shall call these bounds the $H-AV$ bounds.

Let E denote the event "at least one of the queries in the i^{th} step is not local". Then

$$Prob[E|O_i \cap Q_i] = \frac{Prob[E \cap O_i|Q_i]}{Prob[O_i|Q_i]} \leq \frac{Prob[E|Q_i]}{Prob[O_i|Q_i]}$$

We shall derive a lower bound on $Prob\ [O_i|Q_i]$. Let B be a query presented to the independence oracle in some step $j < i$. Then, by our hypothesis that the queries at the first $i - 1$ steps are local, B is a local query. There are two cases to consider:

(i) $|B \cap A_j| > \frac{jn}{4t^2}$. Then, at step j, the oracle responds that B is dependent. The dependence of B is implied by Q_i, since the event Q_i determines the set A_j. Thus the probability that B is dependent given Q_i is 1.

(ii) The oracle has responded that B is independent and, since B is a local query, $|B \cap \bigcup_{k \geq i} A_k| \leq (j + \frac{1}{4})(2t - i + 1)\frac{n}{4t^2}$. The probability that B fails to be independent given Q_i is bounded above by the probability that a set of $(i - \frac{3}{4})(2t - i + 1)\frac{n}{4t^2}$ elements drawn at random from $A_i \cup A_{i+1} \cup ..., \cup A_t$ has an intersection of size at least $\frac{in}{4t^2}$ with some set A_k. For each choice of A_k this probability is bounded above by the probability that a hypergeometric random variable with mean $(i - \frac{3}{4})\frac{n}{4t^2}$ is greater than or equal to $\frac{in}{4t^2}$. Using the $H-AV$ bound we find that this probability is bounded above by $\exp\left(-(i - \frac{3}{4})\frac{n}{4t^2} \cdot \frac{1}{3}\left[\frac{\frac{3}{4}}{i - \frac{3}{4}}\right]^2\right)$. As i ranges from 1 to t this quantity remains bounded above by $e^{-\frac{3n}{128t^3}}$.

The number of queries presented during the first $i - 1$ steps is $p(i - 1)$, which is less than pt. For a given query B, the oracle's response that B is independent could be false only if, for some k, $|B \cap A_k| > \frac{kn}{4t^2}$. But, since B is a local query, we have shown that the probability of this event is bounded above by $e^{-\frac{3n}{128t^3}}$. Hence, the conditional probability, given Q_i, that some query during the first $i - 1$ steps has an outcome different from the "normal" outcome indicated by the event O_i is bounded above by $2pt^2 e^{-\frac{3n}{128t^3}}$, and it follows that $Prob[O_i | Q_i] \geq 1 - 2pt^2 e^{-\frac{3n}{128t^3}}$.

Next we derive an upper bound on $Prob[E | Q_i]$. Let B be a set presented to the independence oracle at step i when the event $O_i \cap Q_i$ occurs. As noted before, we may assume that $B \subseteq A_i \cup A_{i+1} \cup ..., \cup A_{2t}$. We distinguish between two cases.

(i) $|B| \geq \frac{in}{4t^2}(2t - i + 1)\left[1 + \frac{1}{8i}\right]$. Then, by the $H-AV$ bound, $|B \cap A_i| > \frac{in}{4t^2}$ with probability at least $1 - e^{-\frac{n}{512t^3}}$. Whenever $|B \cap A_i| > \frac{in}{4t^2}$ holds the query B is local.

(ii) $|B| < \frac{in}{4t^2}(2t - i + 1)(1 + \frac{1}{8i})$. Then B can fail to be local only if, for some $k > i$, $|B \cap A_k| > (i + \frac{1}{4})\frac{n}{4t^2}$. By the $H-AV$ bound the probability that this happens for a given k is bounded above by $e^{-\frac{n}{512t^3}}$, and the probability that it happens for some k is bounded above by $2t\, e^{-\frac{n}{512t^3}}$.

Since p queries are presented to the oracle at step i, the conditional probability, given Q_i, that some query fails to be local at step i, is bounded above by $2tp\, e^{-\frac{n}{512t^3}}$. We have now shown the following two inequalities: $Prob[O_i|Q_i] \geq 1 - 2pt^2 e^{-\frac{3n}{128t^3}}$ and $Prob[E|Q_i] \leq 2pt\, e^{-\frac{n}{512t^3}}$. We can now conclude

$$prob[E|O_i \cap Q_i] \leq \frac{2pt\, e^{-\frac{n}{512t^3}}}{1 - 2pt^2 e^{-\frac{3n}{128t^3}}} < 4pt^2 e^{-\frac{n}{512t^3}}.$$

It follows that, with probability greater than $1 - 4pt^3 e^{-\frac{n}{512t^3}}$, all queries at the first t steps are local. But the verification query which the algorithm is required to present at its last step is not local, and hence, with probability $\geq 1 - 4pt^3 e^{-\frac{n}{512t^3}}$ the execution time is at least t. Choosing $t = \left[\frac{n}{2400 \log np}\right]^{\frac{1}{3}}$ we obtain

$$T_{prob}^{ind}(n,p) = \Omega\left[\left[\frac{n}{\log np}\right]^{\frac{1}{3}}\right].$$

5. A RANDOMIZED ALGORITHM FOR THE MAXIMUM INDEPENDENT SET PROBLEM

We present a randomized parallel algorithm for constructing a maximum independent set in an independence system $S = (E,I)$, using a rank oracle. Recall that, for any set $B \subseteq E$, the independence system S/B has B as its ground set, and a subset of B is independent in S/B if and only if it is independent in S. If $A \subseteq B$ then $rank_B(A)$ is defined as $\max_C |A \cap C|$, where C ranges over the maximum independent sets in S/B. A rank oracle answers queries of the form "What is $rank_B(A)$?".

In any independence system, the following randomized algorithm constructs a maximal independent set with probability 1. Let H_m, the m^{th} harmonic number, be defined by $H_m = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{m}$.

Algorithm B

```
$B \leftarrow E$; $r \leftarrow \text{rank}_E(E)$
while $|B| > r$ do
   begin
   $m \leftarrow |B|$;
   draw a sample $j$ from the probability distribution over
   $\{0,1,\ldots,m-1\}$ in which each element $i$ has probability $\frac{1}{(m-i)H_m}$;
   $A \leftarrow$ a random $j$-element subset of $B$;
   $R \leftarrow \{e \in B - A \mid \text{rank}_B(A \cup \{e\}) = \text{rank}_B(A)\}$;
   $B \leftarrow B - R$
   end
output $\leftarrow B$
```

The algorithm satisfies the invariant assertion that, at the beginning of each execution of the body of the while loop, the set B contains a maximum independent set in the independence system S. This property holds initially, when $B = E$. To see that it remains true throughout the execution of the algorithm, we show that, if B contains a maximum independent set in S then, after the next iteration, $B - R$ contains a maximum independent set in S. Let C be a maximum independent set in S which is contained in B and contains $\text{rank}_B(A)$ elements of A. Then C contains no elements of R; for if it did contain some element $e \in R$ then it would contain $\text{rank}_B(A) + 1$ elements of $A \cup \{e\}$. This would imply $\text{rank}_B(A \cup \{e\}) \geq \text{rank}_B(A) + 1$, contradicting the fact that $e \in R$.

Next we show that, at any given step, the expected value of $|R|$ is $\frac{m-r}{H_m}$, where r is the size of a maximum independent set in S (and also in S/B) and $m = |B|$. For $i = 0,1,\ldots,m-1$ let

$$p_i = Prob[\text{rank}_B(A \cup \{e\}) = \text{rank}_B(A)],$$

where A is a randomly chosen i-element subset of B and e is a random element of $B - A$. Then $r = \text{rank}_B(B) = \sum_{i=0}^{m-1} p_i$. At a given iteration the probability that the algorithm chooses a set A of size i is $\frac{1}{(m-1)H_m}$, and the expected size of R, given that A is of cardinality i, is $(1-p_i)(m-i)$. Hence the expected size of R is

$$\sum_{i=0}^{m-1} \frac{1}{(m-i)H_m}(1-p_i)(m-i)$$

Since $\sum_{i=0}^{m-1} p_i = r$ it follows that the expected size of R is $\frac{m-r}{H_m}$.

We have shown that, when m elements remain, the expected number of elements eliminated at the next iteration is $\frac{m-r}{H_m}$. It now follows from the results cited in the Appendix that the expected number of iterations required by the algorithm is $O(log^2 n)$. Each iteration can be executed in constant time using n processors. This completes the proof of the following theorem.

Theorem 8. $T_{prob}^{rank}(n,n) = O(log^2 n)$.

6. THE COMPLEXITY OF FINDING A BASE IN A MATROID

A matroid is an independence system $S = (E,I)$ with the property that, for all $B \subseteq S$, all the maximal independent subsets of B have the same cardinality; in other words, for every restriction S/B of S, the maximal independent sets coincide with the maximum independent sets. Because of this property the concept of rank becomes simplified: $\text{rank}_B(A)$ assumes the same value for all sets B containing A, and it is simply equal to the maximum size of an independent subset of A. A maximum independent set in a matroid is called a base, and a minimal dependent set (i.e., a dependent set which does not properly contain any dependent set) is called a circuit.

We shall be interested in deterministic and randomized algorithms, using either an independence oracle or a rank oracle, for finding a base in a matroid. As in the previous sections, these algorithms are modeled by decision trees containing query nodes, leaves and, in the case of randomized algorithms, randomization nodes; but now these decision trees are required to correctly identify a maximal independent set only when the input is a matroid. Let $TMAT_{\text{det}}^{ind}(n,p)$, $TMAT_{prob}^{ind}(n,p)$, $TMAT_{\text{det}}^{ind}(n,p)$ and $TMAT_{prob}^{rank}(n,p)$ represent the complexities of these four matroid problems. For example, $TMAT_{\text{det}}^{in}(n,p)$ is the complexity of the best deterministic algorithm for finding a base in a matroid using an independence oracle, when the size of the ground set is n and the number of processors is p.

A striking example of the algorithmic usefulness of the additional structure present in matroids is the following easy result, which indicates that for matroids the rank oracle is too powerful to be interesting.

Theorem 9. $TMAT_{\text{det}}^{rank}(n,n) = 1$

Proof: Let $S = (E,I)$ be a matroid. The following algorithm constructs a base B.

Algorithm

```
Let the elements of the ground set $E$ be $a_1,a_2,...,a_n$;
In parallel, for $i = 1,2,...,n$, determine $\text{rank}_E(\{a_1,a_2,...,a_i\})$;
$B \leftarrow \{a_i | \text{rank}_E(\{a_1,a_2,...,a_i\}) > \text{rank}_E(\{a_1,a_2,...,a_{i-1}\})\}$
```

We prove that B is a base. It is clear from the algorithm that, for $i = 1,2,...,n$, $|B \cap \{a_1,a_2,...,a_i\}| = \text{rank}_E(\{a_1,a_2,...,a_i\})$. In particular, $|B| = \text{rank}_E(E)$; thus B has the right cardinality for a base, and it remains only to show that B is an independent set. To do so we prove by induction that, for $i = 1,2,...,n$, $B \cap \{a_1,a_2,...,a_i\}$ is independent. Assuming this is true for i, we prove it for $i+1$. In case $\text{rank}_E(\{a_1,a_2,...,a_{i+1}\}) = \text{rank}_E(\{a_1,a_2,...,a_i\})$ we have $B \cap \{a_1,a_2,...,a_{i+1}\} = B \cap \{a_1,a_2,...,a_i\}$, and, by induction hypothesis, $B \cap \{a_1,a_2,...,a_i\}$ is independent. In case $\text{rank}_E(\{a_1,a_2,...,a_{i+1}\}) > \text{rank}_E(\{a_1,a_2,...,a_i\})$ we have

$B \cap \{a_1,a_2,...,a_{i+1}\} = (B \cap \{a_1,a_2,...,a_i\}) \cup \{a_{i+1}\}$. We also know that $B \cap \{a_1,a_2,...,a_i\}$ is a maximal independent subset of $\{a_1,a_2,...,a_i\}$ but is not a maximal independent subset of $\{a_1,a_2,...,a_{i+1}\}$; hence the only possible way to augment $B \cap \{a_1,a_2,...,a_i\}$ to a maximal independent subset of $\{a_1,a_2,...,a_{i+1}\}$ is to add the element a_{i+1}, and it follows that $(B \cap \{a_1,a_2,...,a_i\}) \cup \{a_{i+1}\}$ is independent.

We can also exploit an earlier proof to obtain the following lower bound.

Theorem 10.

$$TMAT_{prob}^{ind}\,(n,p) = \Omega\left(\left(\frac{n}{\log np}\right)^{\frac{1}{3}}\right).$$

Proof: On inspecting the proof of Theorem 7 we find that the lower bound applies even when the independence system presented as input is of the form (E,I), where E is the disjoint union of $2t$ sets $A_i, A_2,...,A_{2t}$, each of cardinality $\frac{n}{2t}$, and a set is independent if and only if its intersection with A_i has cardinality less than or equal to $\frac{in}{4t^2}$, for $i = 1,2,...,2t$. Such an independence system is a matroid; in fact, it is a special type of matroid known as a partition matroid. It follows that the lower bound holds for matroids.

The following fact about matroids will be stated without proof.

Fact 1. Let $S = (E,I)$ be a matroid. Let A be a subset of E. Let the elements of A be linearly ordered. Let D be a subset of A defined as follows: $e \in D$ if e is the largest element in some circuit contained in A. Then $A - D$ is a maximal independent subset of A.

The following theorem shows that the lower bound of Theorem 1 no longer holds in the case of matroids.

Theorem 11. $TMAT_{\det}^{ind}\,(n,n) = O(\sqrt{n})$

Proof: We present an algorithm for the construction of a base.

Algorithm

```
$IN \leftarrow OUT \leftarrow \varphi$; $F \leftarrow E$;
while $F \neq \varphi$ do
   begin
   $m \leftarrow |F|$;
   $r \leftarrow \lfloor \sqrt{m} \rfloor$;
   partition $F$ into $r$ sets $F_1, F_2, \ldots, F_r$, each of size $r$ or $r + 1$
   in parallel for $i = 1,2,\ldots,r$ test whether $IN \cup F_i$ is independent
   if at least one of the sets $IN \cup F_i$ is independent
   then let $j = \min\{i \mid IN \cup F_i$ is independent$\}$
   $IN \leftarrow IN \cup F_j$; $F \leftarrow F - F_j$
   else in parallel for $i = 1,2,\ldots,r$ do
      begin
      let the elements of $F_i$ be $a_{i1}, a_{i2}, \ldots, a_{is(i)}$, where $s(i) = |F_i|$
      in parallel for $k = 1,2,\ldots,s(i)$ test whether $IN \cup \{a_{i1}, a_{i2}, \ldots, a_{ik}\}$ is
      independent;
      let $k(i)$ be the least $k$ such that $IN \cup \{a_{i1}, a_{i2}, \ldots, a_{ik}\}$ is dependent
      [such a $k$ must exist, since $IN \cup F_i$ is dependent]
      end
   $OUT \leftarrow OUT \cup \{a_{ik(i)}, i = 1,2,\ldots,r\}$; $F \leftarrow F - \{a_{ik(i)}, i = 1,2,\ldots,r\}$
end
output $\leftarrow IN$
```

The algorithm satisfies the following invariant assertions: at the beginning of each execution of the while loop,

(i) the sets IN, OUT and F form a partition of the ground set E;

(ii) the set IN is independent;

(iii) $\text{rank}_E(IN \cup F) = \text{rank}_E(E)$

These assertions hold initially, when $IN = OUT = \varphi$ and $F = E$. To show that they remain true, we consider two cases:

Case (a). for some j, $IN \cup F_j$ is independent. Then (i) remains true after the execution of the instructions $IN \leftarrow IN \cup F_j$ and $F \leftarrow F - F_j$; (ii) remains true because $IN \cup F_j$ is independent; and (iii) remains true because $IN \cup F$ does not change.

Case (b). for all i, $IN \cup F_i$ is dependent. Then (i) remains true after the execution of the instructions $OUT \leftarrow OUT \cup \{a_{ik(i)}, i = 1,2,...,r\}$; $F \leftarrow F - \{a_{ik(i)}, i = 1,2,...,r\}$; (ii) remains true because IN does not change; and (iii) remains true by applying Fact 1, with $A = IN \cup F$, and with the linear ordering such that all elements of IN precede all elements of F and, for each i, the elements of F_i occur in the order $a_{i1},a_{i2},...,a_{is(i)}$. It follows that, upon completion of the algorithm, the set IN is a base.

Since each iteration of the while loop entails the execution of n queries in parallel, the algorithm requires n processors. Also, at each iteration which begins with $|F| = m$, the number of elements deleted from F is at least $\lfloor \sqrt{m} \rfloor$. It follows that the total number of iterations is $O(\sqrt{n})$. This completes the proof.

7. THE COMPLEXITY OF FINDING A BASE IN A GRAPHIC MATROID

A matroid with ground set E is graphic if there exists a graph G with edge set E such that, for all $A \subseteq E$, A is independent if and only if the set of edges of A determines a forest (subgraph without cycles) in G. The graph G is said to realize the graphic matroid S. We shall consider decision trees with an independence oracle in which the input is restricted to graphic matroids. Note that these decision trees acquire information only by queries to the independence oracle; the adjacency structure of the graph G is not directly available to them. Let the complexity measures corresponding to randomized and deterministic decision trees be denoted $TGRAPH^{ind}_{\det}(n,p)$ and $TGRAPH^{ind}_{prob}(n,p)$, respectively.

In order to study these complexity measures we shall require some further concepts about matroids. Let $S = (E,I)$ be a matroid and let B be a subset of E. We have already defined the matroid S/B; its ground set is B, and a subset of B is independent in S/B if and only if it is independent in S. The matroid S/B is called the restriction of S to B. Let D be an independent set in S. Then the matroid $S[D]$, the contraction of S on $E - D$, has $E - D$ as its ground set, and a set $A \in E - D$ is independent in $S[E - D]$ if and only if $A \cup D$ is independent in S. It is immediate from these definitions that an independence oracle for the original matroid S can be used to determine whether sets in S/B or $S[D]$ are independent. Define the girth of a matroid as the minimum number of elements in a circuit.

If $S = (E,I)$ is a graphic matroid then the matroids S/B and $S[D]$ are also graphic. If G is a graph realizing graphic matroid S then the graph induced by the edges in B realizes the matroid S/B and the graph obtained by contracting the edges in D (i.e., identifying the two end points of each such edge) realizes the matroid $S[D]$. The girth of a graphic matroid is the length of the shortest cycle in any graph realizing S.

Lemma 2. Let d and n be positive integers. Let $S = (E,I)$ be a graphic matroid of girth $\geq 2d + 1$ with n elements in its ground set. Let $s = \lfloor n^{1-\frac{2}{d}} \rfloor$. Then at least half of the s-element subsets of the ground set E are independent.

Proof: We may assume that S contains at least one circuit, since otherwise the result holds trivially. Let G be a connected graph realizing the graphic matroid S. Since G has n edges and at least one cycle, it has at most n vertices. Every cycle of G is of length at least $2d + 1$. It follows that, between any two given vertices of G, there is at most one simple path of length $\leq d$; for if two such paths existed then there would exist a cycle of length $\leq 2d$. Since the number of pairs of vertices is at most $\binom{n}{2}$, G contains at most $\binom{n}{2}$ simple paths of length $\leq d$. An s-element subset of E contains a cycle only if it contains all the edges of some simple path of length d. The fraction of s-element subsets of the n-element ground set containing a given d-element set is

$$\frac{\binom{n-d}{s-d}}{\binom{n}{s}} = \frac{s(s-1)\dots(s-d+1)}{n(n-1)\dots(n-d+1)} < \left(\frac{s-d}{n-d}\right)^d$$

Hence the fraction of s-element subsets of the ground set containing some simple path of length d is at most $\binom{n}{2}$. But, for the given choice of s, $\left(\frac{s-d}{n-d}\right)^d < \frac{1}{n^2}$, and thus $\binom{n}{2}\left(\frac{s-d}{n-d}\right)^d < \frac{1}{2}$.

Theorem 12. For every positive integer d, $TGRAPH_{prob}^{ind}(n, n^{2d+1}) = O(n^{\frac{2}{d}})$

Proof: We prove the theorem by presenting an appropriate randomized algorithm. Let the number of processors p be n^{2d+1}. Then the algorithm is as follows.

Algorithm C

```
  $IN \leftarrow OUT \leftarrow \varphi$; $F \leftarrow E$
  while $F \neq \varphi$ do
    begin
    [the sets $IN$, $OUT$ and $F$ form a partition of $E$]
    let $M$ be the graphic matroid $S[IN]/F$;
    [the ground set of $M$ is $F$; if graph $G$ realizes $S$ then a
    graph realizing $M$ is obtained by contracting the edges in $IN$
    and deleting the edges in $OUT$; an independence oracle for
    the original matroid $S$ suffices for independence tests in
    $M$]
    linearly order the set $F$ (any linear ordering will do);
    in parallel, for each set $A \subseteq F$ such that $|A| \leq 2d$, test
    whether $A$ is independent in $M$;
    let $K = \{e \in F|$ for some circuit $C$ in $M$ of size $\leq$
    $2d$, $e$ is the largest element of $C$ in the linear
    ordering$\}$
    [the independence tests performed at the previous step
    provide enough information to determine $K$]
    $OUT \leftarrow OUT \cup K$; $F \leftarrow F - K$;
    let $N$ be the graphic matroid $S[IN]/F$;
    $m \leftarrow |F|$;
    in parallel, choose $n^{2d+1}$ random subsets of $N$, each of
    cardinality $\lfloor m^{1-\frac{2}{d}} \rfloor$;
    if at least one of these sets is independent in $N$ then
    choose any such independent set $A$;
    $IN \leftarrow IN \cup A$; $F \leftarrow F - A$
    end
```

At the beginning of each iteration of the while loop the ground set E is partitioned into three sets, IN, F and OUT, such that IN is independent and $IN \cup F$ contains a base. The base that is eventually found will contain IN and will be a subset of $IN \cup F$. To enforce this we contract on $S - IN$ to force the set of elements IN into any base that is found, and then restrict to F to ensure that no elements of OUT are included; this is done by working in the matroid $M = S[IN]/F$. If B is a base in M then $IN \cup F$ is a base in the original matroid S. The iteration begins by deleting elements from the ground set of M to produce a new matroid N which has two crucial properties: first, that the ground set of N contains a base of M; and second, that the girth of N is at least $2d+1$. This is achieved by linearly ordering the elements of F, and then deleting the largest element in each circuit of length $\leq 2d$ in M. Fact 1 ensures that there is at least one base which contains none of the deleted elements. This process of deleting elements to increase the girth while preserving a base works for any matroid.

The next part of the algorithm exploits the special properties of graphic matroids. Because the graphic matroid N has m elements in its ground set and girth at least $2d + 1$, Lemma 2 ensures that a randomly chosen $\lfloor m^{1-\frac{2}{d}}\rfloor$-element subset of the ground set of N has a better than 50% chance of being independent. When n^{2d+1} such subsets are chosen independently, the chance that all of them fail to be independent is less than $2^{-n^{2d+1}}$, which is minuscule. Thus, with very high probability, an independent set of size $\lfloor m^{1-\frac{2}{d}}\rfloor$ will be found at each iteration. Whenever such a set is found it is inserted into IN and deleted from F, so the size of F is reduced by $\lfloor m^{1-\frac{2}{d}}\rfloor$. It follows that, with high probability, the set F will become empty after $O(n^{\frac{2}{d}})$ iterations. The number of processors required to execute one iteration of the while loop in constant time is $O(n^{2d+1})$.

We have established the correctness of the algorithm, determined the number of processors it requires and analyzed its running time. This completes the proof of Theorem 12.

Theorem 13.

(i) For every integer $d \geq 4$, $TGRAPH_{\mathrm{det}}^{ind}(n, n^{2d+1}) = O(n^{\frac{3}{d}})$

(ii) $TGRAPH_{\mathrm{det}}^{ind}(n, n^{4\lceil \log_2 n\rceil+1}) = O(\log n)$

Proof: The proof will be based on the analysis of a deterministic algorithm which is similar to Algorithm C, but in which the randomized process used in Algorithm C to search for an independent set A of suitable size in matroid N is replaced by a deterministic process based on the properties of polynomials over finite fields. We shall show that, when matroid N has m elements in its ground set, this construction is guaranteed to produce an independent set of size $\Omega(m^{1-\frac{3}{d}})$ in matroid N, and from this it will follow that the algorithm terminates within

$O(n^{\frac{3}{d}})$ iterations. The algorithm is as follows.

Algorithm D

```
$IN \leftarrow OUT \leftarrow \varphi$; $F \leftarrow E$;
while $|F| \geq 16$ do
    begin
    let $M$ be the graphic matroid $S[IN]/F$;
    linearly order the set $F$;
    in parallel, for each set $A \subseteq F$ such that $|A| \leq 2d$, test
    whether $A$ is independent in $M$;
    let $K = \{e \in F \mid$ for some circuit $C$ in $M$ of size $\leq$
    $2d$, $e$ is the largest element of $C$ in the linear
    ordering$\}$
    $OUT \leftarrow OUT \cup K$; $F \leftarrow F - K$;
    let $N$ be the graphic matroid $S[IN]/F$;
    $m \leftarrow |F|$;
    if $m < 16$ then go to exit
    $q \leftarrow$ the least power of $2 \geq m$;
    let $\alpha$ be a one-to-one function from $F$ into $GF[q]$, the
    finite field with $q$ elements;
    let $T$ be a subset of $GF[q]$ of cardinality $\lfloor q\ m^{-\frac{3}{d}} \rfloor$;
    in parallel, for each polynomial $f$ of degree $d - 1$ over
    $GF[q]$ do
    begin $A(f) = \{e \in F | f(\alpha(e)) \in T\}$; test whether $A[f]$ is independent in $N$ end;
    let $A$ be any independent set of maximum cardinality among
    the independent sets of the form $A(f)$;
    $IN \leftarrow IN \cup A$; $F \leftarrow F - A$
    end
    $[|F| < 16]$
    exit: using any decision tree of constant depth, find a
    maximal independent set $A$ in $F$;
    output $\leftarrow IN \cup A$.
```

As in Algorithm C, the following will hold at the beginning of each iteration of the while loop: the sets IN, F and OUT form a partition of E, and there is a base which contains IN and is contained in F. At each iteration, elements are deleted from M to produce a matroid N with girth $\geq 2d + 1$, where $d = 2\lceil \log_2 n \rceil$; the number of elements in the ground set of N is denoted m.

We will show that there exists a polynomial f of degree $d - 1$ over $GF(q)$ such that the set $A(f)$ is independent and has cardinality $\geq \frac{1}{2}(|T| - 1)$. For all d, this quantity is $\Omega(m^{1-\frac{3}{d}})$ and is at least 1 when $m \geq 16$.

Recall that if N is a graphic matroid with m elements and girth $\geq 2d + 1$ then a set A is independent provided it does not contain any of at most $\binom{m}{2}$ d-element sets. These sets correspond to simple paths of length d in a graph G that realizes N; we refer to them as the forbidden sets.

Let B be one of the forbidden sets. We will prove that, among the q^d sets $A(f)$, exactly $\lfloor q\, m^{-\frac{3}{d}} \rfloor^d$ contain B. Consider $\alpha(B)$, the subset of $GF[q]$ which is the image under α of the forbidden set B. Since a polynomial of degree $d - 1$ over $GF[q]$ is determined by its values at any d points, the q^d polynomials of degree $d - 1$ take on different d-tuples of values at the points in $\alpha(B)$, and since the number of possible d-tuples is equal to the number of polynomials, each d-tuple of values is assumed by exactly one polynomial. But $A(f)$ contains B if and only if it assumes a value in T at every point in $\alpha(B)$. The number of d-tuples of values from T is $|T|^d = \lfloor q\, m^{-\frac{3}{d}} \rfloor^d$, and hence the number of sets $A(f)$ containing the forbidden set B is $\lfloor q\, m^{-\frac{3}{d}} \rfloor^d$. Since there are at most $\binom{m}{2}$ forbidden sets the number of sets $A(f)$ which contain some forbidden set, and hence are not independent, is at most

$$\binom{m}{2} \lfloor q\, m^{-\frac{3}{d}} \rfloor^d < \binom{m}{2} q^d\, m^{-3} < \frac{q^d}{2}\, m^{-1}.$$

We shall use an averaging argument to show that there exists an independent set $A(f)$ of cardinality $\frac{1}{2}|T| - 1 = \Omega(m^{1-\frac{3}{d}})$. We sum the cardinalities of all the sets $A(f)$, as f ranges over all polynomials of degree $d - 1$, and then subtract off an upper bound on the sum of the cardinalities of those sets $A(f)$ which are dependent. This leads to a lower bound on the average size of the independent sets $A(f)$.

$$\sum_f |A(f)| = \sum_{e \in F} \sum_{y \in T} |\{f | f(\alpha(e)) = y\}$$
$$= \sum_{e \in F} \sum_{y \in T} q^{d-1} = m\, q^{d-1}\, |T|.$$

$$\sum_{\{f | A(f) \text{ is dependent}\}} |A(f)| = m |\{f | A(f) \text{ is dependent}\}|$$
$$< m(\frac{q^d}{2} m^{-1}) = \frac{q^d}{2}.$$

Hence

$$\sum_{\{f | A(f) \text{ is independent}\}} \geq m\, q^{d-1}\, |T| - \frac{q^d}{2.}$$

Since the number of independent sets $A(f)$ is at most q^d, the average size of an independent set $A(f)$ is at least

$$\frac{m\, q^{d-1}\, |T| - \frac{q^d}{2}}{q^d} = \frac{m}{q} |T| - \frac{1}{2} \geq \frac{1}{2} (|T| - 1)$$

and it follows that there exists an independent set $A(f)$ of cardinality $\geq \frac{1}{2} |T| - 1 = \Omega\, (m^{1 - \frac{3}{d}})$.

It follows that the number of iterations of the while loop is $O(n^{\frac{3}{d}})$. With n^{2d+1} processors it is possible to execute each such iteration in constant time. This completes the proof of (i).

In order to prove (ii) we modify Algorithm D by inserting the statement $d \leftarrow 2 \lceil \log_2 n \rceil$ at the beginning of the algorithm, and changing the statement "let T be a subset of $GF[q]$ of cardinality $\lfloor q\, m^{-\frac{3}{d}} \rfloor$"; to "let T be a subset of $GF[q]$ of cardinality $\frac{q}{2}$". We shall prove that, in an iteration where $|F| = m$, an independent set A of size at least $\frac{m}{4}$ is found.

The number of sets $A(f)$ that contain a given forbidden set B is $|T|^d = (\frac{q}{2})^d$, and the number containing some one of the $\binom{m}{2}$ forbidden sets is $\leq \binom{m}{2} \left(\frac{q}{2}\right)^d$, which is less than $\frac{q^d}{2}$. This establishes that more than half of the sets $A(f)$ are independent.

Next we examine how the cardinalities of the sets $A(f)$ are distributed. Since $m \geq 16$, $d \geq 8$. Let C be any c-element subset of F, where $2 \leq c \leq d$. We use the basic fact that, over any c-tuple of points in $GF[q]$, where $c \leq d$, each c-tuple of values from $GF[q]$ is assumed by the same number of polynomials of degree $d - 1$. Thus, it follows that, for every set $D \subseteq C, A(f) \cap C = D$ for exactly a fraction 2^{-c} of the polynomials f. Hence the expected value of $|A(f) \cap C|$ as f ranges over all polynomials is $\frac{c}{2}$. If we restrict attention to those sets $A(f)$ which are independent then, since the independent sets comprise more than half the total number of such sets, the average value of $|A(f) \cap C|$ is at least its average value over that half of the set of degree-$(d - 1)$ polynomials for which $|A(f) \cap C|$ is least; and that average value is at least $\frac{c}{4}$. Hence, when we fix C and average $|A(f) \cap C|$ over the independent sets $A(f)$, the average value of $|A(f) \cap C|$ is at least $\frac{|C|}{4}$. It follows that the average value of $|A(f)|$ over all independent sets $A(f)$ is at least $\frac{m}{4}$, and hence there exists an independent set $A(f)$ of cardinality at least $\frac{m}{4}$.

It follows that the number of iterations of the while loop is $O(logn)$. Since $n^{4\lceil \log_2 n \rceil + 1}$ processors suffice to execute an iteration in constant time, the proof is complete.

APPENDIX

SOLVING PROBABILISTIC RECURRENCE RELATIONS

Several of the randomized algorithms studied in this paper start with a set F of cardinality n and repeat an iteration step which attempts to reduce the cardinality of F; the process continues until F becomes empty. The amount by which m, the cardinality of F, is reduced at each iteration is a random variable $X(m)$. The analysis of each of these algorithms was in two parts: first, a lower bound on the expectation of $X(m)$ was derived, and then an upper bound on the expected number of iterations was stated. In this section we state a theorem which was used, in each of these cases, to pass from a lower bound on the expectation of $X(m)$ to a statement about the distribution of the number of iterations.

The theorem is concerned with the following iteration process, which captures the structure of each of our randomized algorithms.

```
$m \leftarrow n$; $t \leftarrow 0$;
while $m > 0$ do
   begin $m \leftarrow m - X(m)$; $t \leftarrow t + 1$ end
$T \leftarrow t$
```

Here $X(m)$ is a random variable ranging over $\{0,1,...,m\}$, and T is a random variable which represents the number of iterations executed.

What can we say about the distribution of T, given that $E[X(m)] \geq g(m)$, where g is a monotone nondecreasing function from the positive reals to the positive reals?

Theorem 14. $E[T] \leq \int_1^n \frac{dx}{g(x)}$, and for every $a > 0$, $Prob[T > (a+1)\int_1^n \frac{dx}{g(x)}] < e^{-a}$.

This theorem is best possible in the sense that there are distributions of the random variables $X(m)$ for which the upper bounds are tight. The proof of the theorem is contained in a forthcoming paper by the present authors. We believe that the theorem will find many applications to the analysis of randomized algorithms and to the probabilistic analysis of deterministic algorithms.

REFERENCES

[AV79] D. Angluin and L. G. Valiant, "Fast Probabilistic Algorithm for Hamiltonian Circuits and Matchings", *J. Comp. Syst. Sci., 12,* 6 (1979) 155-193.

[AIS84] B. Awerbuch, A. Israeli and Y. Siloach, "Finding Euler Circuits in Logarithmic Parallel Time", *Proc. 16th STOC,* (1984) 249-257.

[C79] V. Chvátal, "The Tail of the Hypergeometric Distribution", *Discrete Math, 25,* (1979) 285-287.

[FRW84] F. Fich, P. Ragde and A. Wigderson, "Relations between Concurrent-Write Models of Parallel Computation", *Proc. 3rd PODC,* (1984) 179-189.

[G84] Z. Galil, "Optimal Parallel Algorithms for String Matching", *Proc. 16th STOC,* (1984) 240-248.

[H84] F. K. Hwang, "Three Versions of a Group Testing Game", *SIAM J. Alg. Disc. Methods, 5,* no. 2, (1984) 145-153.

[HK81] D. Hausmann and B. Korte, "On Algorithmic versus Axiomatic Definitions of Matroids", *Mathematical Programming Study, 14,* (1981) 98-111.

[H63] W. Hoeffding, "Probability Inequalities for Sums of Bounded Random Variables", *J. Am. Stat. Assoc., 58,* (1963) 13-30.

[KUW85a] R. M. Karp, E. Upfal and A. Wigderson, "Are Search and Decision Problems Computationally Equivalent?", *Proc. 17th STOC,* (1985) 464-475.

[KUW85b] R. M. Karp, E. Upfal and A. Wigderson, "Constructing a Perfect Matching is in Random NC", *Proc. 17th STOC,* (1985) 22-32.

[KUW85c] R. M. Karp, E. Upfal and A. Wigderson, "The Complexity of Parallel Computation on Matroids", *Proc. 26th FOCS,* (1985) 541-550.

[KW84] R. M. Karp and A. Wigderson, "A Fast Parallel Algorithm for the Maximal Independent Set Problem", *Proc. 16th STOC,* (1984) 266-272.

[L76] E. L. Lawler, *Combinatorial Optimization, Networks and Matroids,* Holt, Rienhart and Winston, (1976).

[L85] M. Luby, "A Simple Parallel Algorithm for the Maximum Independent Set Problem", *Proc. 17th STOC,* (1985) 1-10.

[TV84] R. E. Tarjan and U. Vishkin, "Finding Biconnected Components and Computing Tree Functions in Logarithmic Parallel Time", *Proc. 25th FOCS,* (1984) 12-20.

[V84] U. Vishkin, "An Efficient Parallel Strong Orientation", Technical Report No. 109, Computer Science Department, New York University, (1984).

[W76] D. J. A. Welsh, *Matroid Theory,* Academic Press, (1976).

[Y77] A. C. Yao, "A Probabilistic Computation: Towards a Unified Measure of Complexity", *Proc. 18th FOCS,* (1977) 222-227.