The Complexity of Parallel Search*

RICHARD M. KARP

Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California

ELI UPFAL

IBM Almaden Research Center, San Jose, California

AND

AVI WIGDERSON

Computer Science Department, Hebrew University, Jerusalem, Israel

Received May 15, 1986; revised November 17, 1987

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 analog 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. © 1988 Academic Press, Inc.

* Some of the results in this paper appear in preliminary form in [KUW85a] and [KUW85c].
† Research supported by NSF Grant DCR-8411954.
‡ Research supported by a Weizmann Fellowship.
§ Research supported by DARPA Grant N00039-83-C-1036 at the University of California at Berkeley.

0022-0000/88 $3.00
Copyright © 1988 by Academic Press, Inc.
All rights of reproduction in any form reserved.
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 \( x_1 \), 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?” can be trivially converted to: “Does there exist a \( y' \) such that \( F(x', y') \) holds?” if the length of \( y' \) is no greater than the length of the original \( y \) minus the number of bits of the solution. In such cases the decision problems present that defines the given search problem for boolean formulas is self-reducible because the formula has a satisfying truth assignment in the specified values, is easily converted to: “Does the formula is satisfiable. Most search problems In the context of sequential computation, a given search problem can be solved in a self-reducible search problem and the secondary importance, since the self-reducible search problem can be solved in parallel can.

In the context of parallel computation, the self-reducibility of decision and decision problems is far more algorithms that run in sublinear time, then the self-reducibility process, will be promising. And in fact there are many even trivial to solve in parallel, so challenging. For example, testing whether all degree larger [AIS84]. The problem of deciding an independent set of vertices is vacuously constructing a fast parallel algorithm to be challenging [KW84, Lu87]. Other associated decision problems seem bi-connectivity [VT84], pattern matching.

It is natural to ask whether there is a process. To find a satisfying instance, self-reducibility process would execute a call to a satisfiability oracle, where the parallelism. Each call would involve by making a partial truth assignment, the others are left free. But how should so as to minimize the number of calls.

We have chosen to study a general independence systems. An independent set \( E \) is a finite set of elements called the independent sets of \( S \). The element: if \( A \subset B \subseteq E \) and \( B \subseteq I \), then \( A \).
THE COMPLEXITY OF PARALLEL SEARCH

"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 often 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, Lu87]. Other examples where search problems and the associated decision problems seem to require very different approaches are connectivity [VT84], pattern matching [G84] and strong orientation [V84].

It is natural to ask whether there is a useful parallel analog 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 \( A \subseteq B \subseteq E \) 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 oracle, whose 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 can be determined whether solutions exist satisfying certain partial specifications, can be modelled within this framework. Let \( R(\cdot, \cdot) \) 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 \( \{ e_1, e_2, \ldots, e_n, \bar{e}_1, \bar{e}_2, \ldots, \bar{e}_n \} \) 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 \( \{ e_i | y_i = 1 \} \cup \{ \bar{e}_i | y_i = 0 \} \). The set \( \{ e_1, e_2, \ldots, e_n, \bar{e}_1, \bar{e}_2, \ldots, \bar{e}_n \} \) is independent if and only if there exists a solution \( y \) satisfying \( y_1 = y_2 = \cdots = y_n = 1 \) and \( y_1 = y_2 = \cdots = y_n = 0 \).

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, Lu87].

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, \( \text{rank}_S(A) \) will denote the rank of \( A \) in \( S \).

We shall be interested in parallel algorithms for the following problem: given an independence system \( S = (E, I) \) and an oracle which answers queries of the form "Is set \( A \) independent?" find a maximum independent set. The subroutine used in [KUW84] is the maximum matching correspondence.

1.2. The Model of Computation

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

The Maximal Independent Set Problem

\( S = (E, I) \) and an oracle which answers queries of the form "Is set \( A \) independent?" find a maximum independent set.

The Maximum Independent Set Problem

\( S = (E, I) \) and an oracle which answers queries of the form "Is set \( A \) independent?" find a maximum independent set in \( S \).

Associated with each parallel algorithm is an integer \( p \) called the number of processors. Each processor progresses in parallel steps; in each \( t \), \( \theta_t \) processors may query the oracle. An algorithm may be, for example, the oracle may be either an independent or a reduction oracle. For example, the oracle is the goal is to find a maximal independent set.

To find a maximum independent set in \( S \), \( S \) is partitioned into \( n \) subgraphs \( S_i \) and \( p \) processors are assigned to each subgraph. The \( i^{th} \) processor is responsible for \( S_i \) and it is given \( S_i \) as input. The \( i^{th} \) processor is \( p \) times faster than the \( (i-1)^{th} \) processor. Every processor is given an independence system \( S_i \) and a random number generator.

Given an independence system \( S \) with \( p \) processors, the algorithm corresponds to the traversal of \( S \) in parallel. A processor is considered to be busy if it is randomizing a query node, or it is waiting for a query node to be returned. If a processor reaches a query node, it is considered to be busy if it is waiting for a query node to be returned. If a processor reaches a query node, it is considered to be busy if it is waiting for a query node to be returned.
THE COMPLEXITY OF PARALLEL SEARCH

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 $\text{rank}_A(A)$?,” 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 its 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 randomized 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 \( T_{\text{prob}}^{\text{ind}}(n, p) = \min_{H} \max_{S} c(H, S) \), where \( H \) ranges over all \( (n, p) \) randomized decision trees with rank oracle, and \( S \) ranges over all independence systems with ground set \( \{ 1, 2, \ldots, n \} \).

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 \( T_{\text{det}}^{\text{ind}}(n, p) \), \( T_{\text{det}}^{\text{rank}}(n, p) \), \( T_{\text{prob}}^{\text{ind}}(n, p) \), and \( T_{\text{prob}}^{\text{rank}}(n, p) \).

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.** \( T_{\text{det}}^{\text{ind}}(n, p) = \Omega(n / \log p) \).

**Theorem 4.** \( T_{\text{det}}^{\text{rank}}(n, p) = \Omega(n / \log np) \).

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.** \( T_{\text{prob}}^{\text{ind}}(n, n^{3/2}) = O(\sqrt{n}) \).

**Theorem 7.** \( T_{\text{prob}}^{\text{ind}}(n, p) = \Omega(n / \log np)^{1/3} \).

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

**Theorem 8.** \( T_{\text{prob}}^{\text{rank}}(n, n) = O\left(\log^2 n\right) \).

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 maximum independent set problem using an independence oracle, or the maximum weight independent set problem using a rank oracle. Our algorithm with \( p = n^{1/2} \) can solve the maximum independent set problem using an independence oracle in \( O(n) \) steps using a rank oracle. Theorem 8 shows that, in the context of randomized algorithms, this is less true in the case of deterministic algorithms.

In Section 5 we consider matroids over independence systems. A matroid is an independence system \( S/B \), the maximal and non-empty independent set. In the case of a rank oracle, we consider algorithms whose inputs are matroids. The main results of this section are:

**Theorem 9.** \( T_{\text{prob}}^{\text{ind}}(n, p) = 1 \).

**Theorem 10.** \( T_{\text{prob}}^{\text{rank}}(n, p) = O\left(\log^2 n\right) \).

It turns out that the lower bound used in Section 6 for the maximum independent set problem presented as an independence oracle is still valid for the following theorem.

**Theorem 11.** \( T_{\text{prob}}^{\text{ind}}(n, n) = O\left(\log^2 n\right) \).

Finally, in Section 6, we consider the case when the ground set is a planar graph, the elements of the ground set are edges if it contains a planar graph, and the edges corresponding to the forest in the graph. The complexity of an algorithm when the ground set is a planar graph is required to be planar.
In this section, we consider the complexity of the maximum independent set problem in matroids. The main results are:

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

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.** $\text{TMAT}^{\text{prob}}_{\text{det}}(n, p) = \Omega(n \log np)^{1/3}$.

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

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

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 forest 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. (i) For every positive integer $d$, $\GRAPH_{\text{prob}}(n, n^{2d^2+1}) = O(n^{2d})$
(ii) $\GRAPH_{\text{det}}(n, n^{\log n + 1}) = O(\log n)$.

Theorem 13. (i) For every integer $d \geq 4$, $\GRAPH_{\text{ind}}(n, n^{3d^2+1}) = O(n^{3d})$
(ii) $\GRAPH_{\text{det}}(n, n^{\log n + 1}) = O(\log n)$.

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. $T_{\text{det}}^{\text{prob}}(n, p) = \Omega(n \log p)$.

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

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 $\{1, 2, \ldots, N\}$, 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 $\{A_1, A_2, \ldots, A_p\}$ of subsets of $\{1, 2, \ldots, N\}$, and it has $2^p$ children, corresponding to the possible outcomes of the queries “Is $A_i \cap X$ nonempty?” for $i = 1, 2, \ldots, p$. Each leaf is labelled with an element of $\{1, 2, \ldots, N\}$. The input to the algorithm is a nonempty set $X \subseteq \{1, 2, \ldots, N\}$. 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 executed 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 algorithm is one without randomization.

The complexity of the algorithm on input $X$ is denoted $\mathcal{C}(H, X)$, and $H$ when used iteratively traversing the tree. The group testing problem is defined as the ranges over $(N, p)$ probabilistic decision trees to ranges over the subsets of $\{1, 2, \ldots, N\}$, testing the solution, denoted $\mathcal{D}_{\text{det}}(N, p)$.

2.2. Complexity of the Deterministic Group Testing Problem

Theorem 1. For all $N$ and $p$, let $\mathcal{D}_{\text{det}}(N, p) = \mathcal{D}_{\text{det}}^{\text{prob}}(N, p) + 1$.

The upper bound is achieved by an algorithm that contains a set $A$ guaranteed to have $A = \{1, 2, \ldots, N\}$, and the algorithm then selects a singleton set $\{i\}$, $i = 1, 2, \ldots, p$ which is of size at most $\lceil |A|/(p+1) \rceil$ and is nonempty, for $i = 1, 2, \ldots, p$. The result is an algorithm $A_1, A_2, \ldots, A_{p+1}$ as having a nonempty intersection that is repeated for each $A_i$, such set becomes the new $A$. The resulting algorithm is at most $\lceil \log_{p+1} N \rceil$.

The lower bound is obtained by giving $\mathcal{D}_{\text{det}}(N, p)$ in [FRW84]. The starting point of the complete description of the information obtained during its execution can be specified by $\mathcal{D}_{\text{det}}(N, p)$ known to have a nonempty intersection with $\{1, 2, \ldots, N\}$ to have an empty intersection with $\{1, 2, \ldots, N\}$ called a negative set. The positive set is the only positive set, and $\mathcal{K}$ is empty. The positive sets $L$ and the negative sets $\mathcal{K}$, $A$ and it is found that $A \cap X$ is nonempty for $\{1, 2, \ldots, N\}$ on the other hand, if it is found that $A \cap X$ is nonempty by $\mathcal{K} \cup A$, and each positive set is executed simultaneously, the information that the sets by making updates corresponding to them.

The algorithm cannot terminate in this case, suppose that the algorithm could terminate in the same time as when no positive set was a set of at least $\log_{p+1} N$ since the available information is limited to all elements of $\{1, 2, \ldots, N\}$ except $X$.

The adversary strategy is based on a decision algorithm by the minimum size of
For any $d$, $TGRAPH_{prob}^{ind}(n, n^{2d+1}) = O(n^{3-d})$.

$TGRAPH_{del}^{ind}(n, n^{2d+1}) = O(n^{1-d})$.

The speed-ups are possible for graphic algorithms.

2.2. Complexity of the Deterministic Group Testing Problem

Theorem 1. For all $N$ and $p$, $\log N / \log (p+1) \leq T_{det}(N, p) \leq \log N / \log (p+1) + 1$.

The upper bound is achieved by the following natural algorithm, which maintains a set $A$ guaranteed to have a nonempty intersection with $X$. Initially $A = \{1, 2, \ldots, N\}$, and the algorithm terminates when $A$ becomes a singleton set. At each step the algorithm partitions $A$ into $p+1$ sets $A_1, A_2, \ldots, A_{p+1}$, each of which is of size at most $\lceil |A|/(p+1) \rceil$. It then tests in parallel whether $A_i \cap X$ is empty, for $i = 1, 2, \ldots, p$. The result of these tests identifies one of the sets $A_1, A_2, \ldots, A_{p+1}$ 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 $\{1, 2, \ldots, N\}$ 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 $A \cap K$ 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 $L_i \cap A$. 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 $\{1, 2, \ldots, N\}$ 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 \( \lceil t/(p + 1) \rceil \).

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 \( W \neq \emptyset \) do
    if \( W \) contains queries whose outcomes are 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 \( A \cap \overline{K} \) is of minimum cardinality
        if \( |A \cap \overline{K}| \geq t/(p + 1) \)
            then, for every query \( B \in W \) give the answer \( B \subseteq A \) and adjoin \( B \cap \overline{K} \) to the list of positive sets
        else give the answer \( A \cap X = \emptyset \), set \( K \) to \( K \cup A \) and replace each positive set \( L \) by \( L \cap \overline{A} \).
```

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 \( t/(p + 1) \), 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 \( t/(p + 1) \);
- whenever a query is answered negatively, the number of elements deleted from any positive set is at most \( t/(p + 1) \);
- 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 \( N = (n, \rho) \). Then

(a) \( T_{\text{det}}^{\text{ind}}(n, p) \geq T_{\text{det}}(N, p) \)

(b) \( T_{\text{det}}^{\text{rank}}(n, p) \geq \frac{1}{2} T_{\text{det}}(N, np) \).

**Proof.** Let \( h \) be any bijection from the set of subsets of \( \{1, 2, ..., n\} \). This bijection provides an injection from the instances \( X \) of the group testing problem into independent sets of size \( n/2 \). The independence system \( I \) of the group testing problem has as its elements the sets which correspond to the elements of \( X \).

Let \( H \) be any \( (n, p) \) decision tree. Then, using \( H \), we construct a decision tree \( H' \) for the group testing problem with universe \( \{1, 2, ..., N\} \), which is independent of \( X \), and has complexity of the deterministic problem with \( n \)-element independence system.

The proof of (b) uses the same construction as in (a) and the instance of the group testing problem and instances of the \( (N, np) \) decision tree. We define a corresponding \((N, np)\) decision tree \( H' \) with universe \( \{1, 2, ..., N\} \), such that \( H' \) first asks \( n \) queries of the form “Is \( C \) a \( n \)-element subset of \( \{1, 2, ..., N\} \) which contain exactly \( r \) elements of \( \{1, 2, ..., n\} \) which correspond to the elements of \( X \)?” \( H' \) then executes two steps on each positive answer \( \rho(A) \) of \( H' \).
When the algorithm terminates, we have determined the minimum size of a positive set \( t \), which is \( t \). The adversary maintains a set \( A_1, A_2, \ldots, A_p \). The adversary then simultaneously executed, then the adversary maintains a set \( A_1, A_2, \ldots, A_p \). The adversary then queries are determined by the present answer \( B \subseteq X \neq \emptyset \) and the positive set \( K \subseteq A \) and replace each query with \( A \cap X \neq \emptyset \) and \( A \cap X \neq \emptyset \). The query is consistent and the negative set \( K \) correctly reflect the step, each positive set is of size at least \( t \); step, each positive set is of size at least \( t \); in 1, \ldots, \( n \). The number of elements deleted by the number of elements deleted the number of answers.

Theorem 2. Let \( N = \binom{n}{2} \). Then

\[
\begin{align*}
T_{\text{det}}(n, p) & \geq T_{\text{det}}(N, p) \\
T_{\text{det}}(n, p) & \geq \frac{1}{2} T_{\text{det}}(N, np).
\end{align*}
\]

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

Let \( H \) be any \((n, p)\) decision tree with oracle for the maximal independent set problem. Then, using the bijection \( h \), we can derive a \((N, p)\) decision tree \( H' \) for the group testing problem with universe \( \{1, 2, \ldots, N\} \). 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 \( D \cap X \) nonempty?” where \( D \) is the set of elements of \( \{1, 2, \ldots, n\} \) which correspond under the bijection \( h \) to \((n/2)\)-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 \( \{1, 2, \ldots, N\} \) and \( p \) processors is no greater than the complexity of the deterministic problem of finding a maximal independent set in an \( n \)-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 maximal independent set problem. Let \( R \) be any \((n, p)\) decision tree with oracle for the maximal independent set problem. We define a corresponding \((N, np)\) decision tree \( R' \) for the group testing problem with universe \( \{1, 2, \ldots, N\} \), 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 \( \text{rank}_p(\mathcal{A}) ? \)”, \( R' \) first asks \( n \) queries of the form “Is \( D_i \cap X \) nonempty?”, where \( D_i \) is the set of elements of \( \{1, 2, \ldots, n\} \) which correspond under the bijection \( h \) to \((n/2)\)-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, \( i = 1, 2, \ldots, r \)” and “\( D_i \cap X \) is empty, \( i = r + 1, \ldots, n \)”, it knows that, in the independence system \( S(X)/B \), the size of a maximum independent set is \( r \). At the second step of the simulation of a single step of \( R \), \( R' \) asks \( r \) queries of the form “Is \( C_i \cap X \) empty?”, where \( C_i \) is the set of elements of \( \{1, 2, \ldots, n\} \) which correspond under the bijection \( h \) to \((n/2)\)-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, \( i = 1, 2, \ldots, s \)” and “\( C_i \cap X \) is empty, \( i = s + 1, s + 2, \ldots, r \)”, then it goes on to continue the simulation of \( R \) if \( R \) has received the answer “\( \text{rank}_p(A) = s \)”. 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.** \( T_{\text{det}}^{\text{ind}}(n, p) = \Omega(n/\log p) \)

**Proof.** By Theorem 2,

\[
T_{\text{det}}^{\text{ind}}(n, p) \geq T_{\text{det}}\left(\binom{n}{n/2}, p\right)
\]

and, by Theorem 1,

\[
T_{\text{det}}\left(\binom{n}{n/2}, p\right) \geq \log_{p+1}\left(\binom{n}{n/2}\right).
\]

But

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

This completes the proof.

**Theorem 4.** \( T_{\text{det}}^{\text{rank}}(n, p) = \Omega(n/\log np) \).

**Proof.** By Theorem 2,

\[
2T_{\text{det}}^{\text{rank}}(n, p) \geq T_{\text{det}}\left(\binom{n}{n/2}, np\right)
\]

and, by Theorem 1,

\[
T_{\text{det}}\left(\binom{n}{n/2}, np\right) \geq \log_{np+1}\left(\binom{n}{n/2}\right).
\]

But

\[
\log_{np+1}\left(\binom{n}{n/2}\right) = \Omega\left(\frac{n}{\log np}\right).
\]

This completes the proof.

### 3.3. The Randomized Complexity of Group Testing

**Theorem 5.** \( \tilde{T}_{\text{prob}}(N, \log N) = \Omega(1) \)

The proof requires a probabilistic argument.

**Lemma 1.** Let \( U \) be a set with \( 2^m \) elements. Let \( a \) be an integer smaller than \( \frac{1}{2} \). If a random linear ordering be imposed on \( U \), then with probability \( \delta > \frac{1}{2} \), two different elements from \( U \) are in positions which differ by \( \leq a \).

**Proof.** The result is clearly true if the element that occurs second in the ordering is at position \( b \), then the probability that the first occurrence is at position \( b \), is \( \delta > \frac{1}{2} \), which is greater than \( \frac{1}{2} \).

**Proof of Theorem 5.** We can assume that the universe is of the form \( 2^m \); if not, let \( X \) be a set of \( m+1 \) elements, none of which lie in \( X \). We can then use \( m+1 \) processors, determines a one-to-one correspondence of \( X \) with set \( S' \), where \( S \) consists of the first \( 2^a \) elements, and so positions, and hence that there exists a \( \Omega(n/\log np) \) such that the second step, assuming the algorithm identifies that element, in at least \( 1 \) is an arbitrary one-to-one correspondence of \( a \)-tuples of zeros and ones. Then, in \( A_i \), \( i = 1, 2, \ldots, a \), where each corresponding \( a \)-tuples contain \( 1 \) in exactly one element of \( X \), the answer is “Yes” then, by \( \tilde{T}_{\text{prob}} \) the answer is “Yes” then, by \( \tilde{T}_{\text{prob}} \) the answer.

At the third step, the algorithm identifies that element. If the answer is “No,” then at least \( \tilde{T}_{\text{prob}} \) the answer is “No,” then, by \( \tilde{T}_{\text{prob}} \) the answer is “No.”

### 4. The Randomized Complexity of Group Testing

In this section we prove lower and upper bounds.

#### 4.1. An Upper Bound

We begin by presenting a random.
3.3. The Randomized Complexity of the Group Testing Problem

**Theorem 5.** \( \tilde{T}_{\text{prob}}(n, \log N) = O(1) \).

The proof requires a probabilistic lemma.

**Lemma 1.** Let \( U \) be a set with \( 2^n \) elements and let \( X \) be a nonempty subset of \( U \). Let a random linear ordering be imposed on the set \( U \). Then, with probability greater than \( \frac{1}{2} \), 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 exactly one element of \( x \) among the first \( 2^a \) positions, is \( 2^a/(b-1) \), which is greater than \( \frac{1}{2} \). 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 \( \frac{1}{2} \). 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 \( S_k \cap X \neq \phi \)," \( k = 0, 1, \ldots, m \), where \( S_k \) consists of the first \( 2^k \) elements in the ordering. If \( x \) is the first index for which the answer is "Yes" then, by Lemma 1, \( |S_x \cap X| = 1 \) with probability greater than \( \frac{1}{2} \). At the second step, assuming that \( S_x \) 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 \( 2^m \) elements of \( S_x \) and the \( 2^m \) \( a \)-tuples of zeros and ones. Then, in parallel, the algorithm presents the queries "Is \( A_i \cap X \neq \phi \)," for \( i = 1, 2, \ldots, a \), where \( A_i \) consists of those elements of \( S_x \) whose corresponding \( a \)-tuples contain 1 in the \( i \)th position. Assuming that \( S_x \) 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 \( > \frac{1}{2} \) 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 \( T_{\text{prob}}^{\text{ind}}(n, p) \).

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.**

\[
IN \leftarrow \emptyset; \ OUT \leftarrow \emptyset; \ F \leftarrow E; \\
\text{while } F \neq \emptyset \text{ do} \\
\quad \begin{array}{l}
\quad \text{Select a random permutation } a_1, a_2, \ldots, a_F \text{ of the elements of } F; \\
\quad \text{in parallel, for } i = 1, 2, \ldots, \lfloor \sqrt{n} \rfloor, \text{ test whether } \{a_i, a_2, \ldots, a_F\} \cup IN \\
\quad \text{is independent;} \\
\quad k = \max \{j | \{a_1, a_2, \ldots, a_j\} \cup IN \text{ is independent}; \\
\quad IN \leftarrow IN \cup \{a_1, a_2, \ldots, a_k\}; \\
\quad OUT \leftarrow OUT \cup \{a \in F | \{a_1, a_2, \ldots, a_k\} \cup IN \cup \{a\} \text{ is dependent}; \\
\quad F \leftarrow E - (IN \cup OUT); \\
\end{array} \\
\text{end} \\
\text{output } \leftarrow IN;
\]

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 \( i = 0, 1, \ldots, |F| - 1 \), let \( q_i \) be the probability that an \( i \)-element set drawn at random from \( F \) is independent in \( T \). For \( j = 0, 1, \ldots, |F| - 1 \) 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 \( i = 1, 2, \ldots, |F| \),

\[
q_i = \prod_{j=0}^{i-1} p_j. \quad \text{Let } m = |F|.
\]

Now consider the behavior of the algorithm on a permutation \( a_1, a_2, \ldots, a_F \) of \( F \). For any \( s \leq \lfloor \sqrt{n} \rfloor \) such that \( \{a_1, a_2, \ldots, a_s\} \) is independent in \( T \), the algorithm will place the elements \( a_1, a_2, \ldots, a_s \) into \( IN \) and will place in \( OUT \) all elements \( a \in F - \{a_1, a_2, \ldots, a_s\} \), such that \( \{a_1, a_2, \ldots, a_s\} \cup \{a\} \) is dependent. If the permutation is chosen at random then the probability that \( \{a_1, a_2, \ldots, a_F\} \) is independent is \( q_s \), and, given that \( \{a_1, a_2, \ldots, a_s\} \) is independent, the expected number of elements \( a \in F - \{a_1, a_2, \ldots, a_s\} \) such that \( \{a_1, a_2, \ldots, a_s\} \cup \{a\} \) 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 \( p_s \), there is an \( s \) such that \( q_s(s + (1-p_s)(m-s)) \) is at least \( e^{-1} \sqrt{m} + \Theta(1) \). We distinguish two cases:

**Case 1.** For \( j = 0, 1, \ldots, |\sqrt{m}|, p_j \)

\[
q_j \geq \prod_{i=0}^{j-1} p_i \geq e^{-1} \sqrt{m} \Rightarrow q_j(s + (1-p_s)(m-s)) \geq e^{-1} \sqrt{m} + \Theta(1)
\]

and hence

\[
q_j(s + (1-p_s)(m-s)) \geq e^{-1} \sqrt{m} + \Theta(1)
\]

**Case 2.** There exists a \( j < |\sqrt{m}| \) such that \( p_j \). Then

\[
q_j(s + (1-p_s)(m-s)) \geq e^{-1} \sqrt{m} + \Theta(1)
\]

We have now shown that, at an iteration, the expected reduction in the size of \( F \) is \( m \), the expected reduction in the size of \( F \) is \( m \), and it is easy to show that the expected reduction in the size of \( F \) is \( m \). The final step is to justify this plausible conclusion.

Since each execution of the while loop requires a constant number of steps using \( n^{1/2} \) processors, we have

**Theorem 6.** \( T_{\text{rank}}^\text{prob}(n, n^{1/2}) = \Theta(\sqrt{n}) \).

**4. A Lower Bound**

The following theorem gives a lower bound on the performance of Algorithm A.

**Theorem 7.** \( T_{\text{rand}}^\text{prob}(n, p) = \Omega(n \log n) \).

**Proof.** We use the following observation about the randomized complexity of a probabilistic independence Oracle. Let \( D \) be a probability distribution on \( \{1, 2, ..., n\} \), and let \( T_{\text{rand}}^\text{prob}(n, p) \) be the expected number of steps executed by decision tree \( T_{\text{rand}}^\text{prob}(n, p) \).
set \( F \) consists of elements about which the

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

and hence

\[
q_s(n - (1 - p_s)(m - s)) \geq e^{-1}m(1 - e^{-1/\sqrt{m}})(m - s) = e^{-1} \sqrt{m} + O(1).
\]

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 \( |F| \)

dependent in the original. Let \( q_s \) be the probability that an element \( \{a_1, a_2, ..., a_r\} \) deleted from \( F \) at each step

dependent in \( T \) for \( j = 0, 1, ..., |F| - 1 \) element set \( A \) at random from \( F \) and let \( p_j \) be the conditional probability that \( A \) is dependent in \( T \). Then, for

... a_{p_j} of the elements of \( F \); whether \( \{a_1, a_2, ..., a_r\} \cup \text{IN} \)

dependent; \( \{a_1, a_2, ..., a_r\} \cup \text{IN} \) is dependent;

and \( \text{IN} \cup \{a\} \) is dependent;


\textbf{Case 1.} For \( j = 0, 1, ..., \lfloor \sqrt{m} \rfloor, \ p_j > \exp(-1/\lfloor \sqrt{m} \rfloor) \). Choose \( s = \lfloor \sqrt{m} \rfloor \). Then

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

and hence

\[
q_s(n - (1 - p_s)(m - s)) \geq e^{-1}m(1 - e^{-1/\sqrt{m}})(m - s) = e^{-1} \sqrt{m} + O(1).
\]

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 \( |F| \)

dependent in the original. Let \( q_s \) be the probability that an element \( \{a_1, a_2, ..., a_r\} \) deleted from \( F \) at each step

dependent in \( T \) for \( j = 0, 1, ..., |F| - 1 \) element set \( A \) at random from \( F \) and let \( p_j \) be the conditional probability that \( A \) is dependent in \( T \). Then, for

... a_{p_j} of the elements of \( F \); whether \( \{a_1, a_2, ..., a_r\} \cup \text{IN} \)

dependent; \( \{a_1, a_2, ..., a_r\} \cup \text{IN} \) is dependent;

and \( \text{IN} \cup \{a\} \) is dependent;


\textbf{Case 1.} For \( j = 0, 1, ..., \lfloor \sqrt{m} \rfloor, \ p_j > \exp(-1/\lfloor \sqrt{m} \rfloor) \). Choose \( s = \lfloor \sqrt{m} \rfloor \). Then

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

and hence

\[
q_s(n - (1 - p_s)(m - s)) \geq e^{-1}m(1 - e^{-1/\sqrt{m}})(m - s) = e^{-1} \sqrt{m} + O(1).
\]

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 \( |F| \)

dependent in the original. Let \( q_s \) be the probability that an element \( \{a_1, a_2, ..., a_r\} \) deleted from \( F \) at each step

dependent in \( T \) for \( j = 0, 1, ..., |F| - 1 \) element set \( A \) at random from \( F \) and let \( p_j \) be the conditional probability that \( A \) is dependent in \( T \). Then, for

... a_{p_j} of the elements of \( F \); whether \( \{a_1, a_2, ..., a_r\} \cup \text{IN} \)

dependent; \( \{a_1, a_2, ..., a_r\} \cup \text{IN} \) is dependent;

and \( \text{IN} \cup \{a\} \) is dependent;


\textbf{Case 1.} For \( j = 0, 1, ..., \lfloor \sqrt{m} \rfloor, \ p_j > \exp(-1/\lfloor \sqrt{m} \rfloor) \). Choose \( s = \lfloor \sqrt{m} \rfloor \). Then

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

and hence

\[
q_s(n - (1 - p_s)(m - s)) \geq e^{-1}m(1 - e^{-1/\sqrt{m}})(m - s) = e^{-1} \sqrt{m} + O(1).
\]

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 \( |F| \)

dependent in the original. Let \( q_s \) be the probability that an element \( \{a_1, a_2, ..., a_r\} \) deleted from \( F \) at each step

dependent in \( T \) for \( j = 0, 1, ..., |F| - 1 \) element set \( A \) at random from \( F \) and let \( p_j \) be the conditional probability that \( A \) is dependent in \( T \). Then, for

... a_{p_j} of the elements of \( F \); whether \( \{a_1, a_2, ..., a_r\} \cup \text{IN} \)

dependent; \( \{a_1, a_2, ..., a_r\} \cup \text{IN} \) is dependent;

and \( \text{IN} \cup \{a\} \) is dependent;


\textbf{Case 1.} For \( j = 0, 1, ..., \lfloor \sqrt{m} \rfloor, \ p_j > \exp(-1/\lfloor \sqrt{m} \rfloor) \). Choose \( s = \lfloor \sqrt{m} \rfloor \). Then

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

and hence

\[
q_s(n - (1 - p_s)(m - s)) \geq e^{-1}m(1 - e^{-1/\sqrt{m}})(m - s) = e^{-1} \sqrt{m} + O(1).
\]
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 
\[
(n \log np)^{1/3}.
\]
The distribution \( D \) is concentrated on independence systems of the following form: the \( n \)-element ground set \( E \) is partitioned into \( 2t \) sets \( A_1, A_2, ..., A_{2t} \), each of size \( n/2t \), where \( t \) is a parameter to be specified later. A set \( B \) is independent if and only if, for all \( i, j = 1, 2, ..., 2t \), \(|B \cap A_i| \leq \sqrt{n}/4t^2 \). 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 \( A_i \), 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 \( O_1, O_2, ..., O_i \) and \( Q_1, Q_2, ..., Q_i \), 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 \( A_1, A_2, ..., A_{i-1} \). 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 \( A_1, A_2, ..., A_{i-1} \) is determined before the \( j \)th step, we may assume that, if \( B \) is an oracle query presented at the \( j \)th step, then \( B \subseteq A_j \cup A_{j+1} \cup ... \cup A_{2t} \). 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 \( A_j \cup A_{j+1} \cup ... \cup A_{2t} \). 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 with respect to a given independence system \((A_1, ..., A_{2t})\) if either

(i) \(|B \cap A_i| > jn/4t^2\) (so that \( B \) is a dependent set) or
(ii) \( B \) is independent and, for every \( i > j \), \(|B \cap A_i| \leq (j+i)(2t-i+1)\)
\((n/4t^2)\).

Note that, by definition, 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 for integers with \( M \leq N \) and \( m \leq N \). Let
\[
\sum_{k=m}^{M} \binom{M}{k} \left( \frac{k}{N} \right)^m \left( 1 - \frac{k}{N} \right)^{N-m} = \frac{\sum_{k=m}^{M} \binom{M}{k} \left( \frac{k}{N} \right)^m \left( 1 - \frac{k}{N} \right)^{N-m}}{\binom{N}{m}}
\]
and
\[
\sum_{k=m}^{M} \binom{M}{k} \left( \frac{k}{N} \right)^m \left( 1 - \frac{k}{N} \right)^{N-m} = \frac{\sum_{k=m}^{M} \binom{M}{k} \left( \frac{k}{N} \right)^m \left( 1 - \frac{k}{N} \right)^{N-m}}{\binom{N}{m}}
\]

Analogous bounds are derived for
\[
\sum_{k=m}^{M} \binom{M}{k} \left( \frac{k}{N} \right)^m \left( 1 - \frac{k}{N} \right)^{N-m} = \frac{\sum_{k=m}^{M} \binom{M}{k} \left( \frac{k}{N} \right)^m \left( 1 - \frac{k}{N} \right)^{N-m}}{\binom{N}{m}}
\]

We prove an \( \Omega(t) \) lower bound on \( \Pr[\Omega_j] \) by showing that if a random input is local with high probability all the queries asked are local. Since the "verification query" step is not local the algorithm must fail.

Let \( E \) denote the event "at least one query is local" and let \( L \) denote the event "all the queries are local."

\[
\Pr[E_j | O_j \cap Q_j \cap L_j] = \frac{\Pr[E_j]}{\Pr[L_j]} \frac{\Pr[L_j]}{\Pr[L_j]\Pr[O_j \cap Q_j \cap L_j]}
\]

We shall derive a lower bound on the dependence of \( B \) on the independence oracle in some steps; for the first \( i - 1 \) steps are local, thus, \( B \) is independent.

(i) \(|B \cap A_i| > jn/4t^2\) (so that \( B \) is a dependent set) or
The dependence of \( B \) is implied by \( |B \cap A_i| > jn/4t^2 \).
Thus the probability that \( B \) is dependent is
(ii) \( B \) is independent and, for every \( i > j \), \(|B \cap A_i| \leq (j+i)(2t-i+1)\)
\((n/4t^2)\).

Note that, by definition, 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
and show that, for all \((n, p)\) determined steps executed by \(H\) on instances of permutation \(f(n, p)\) which grows as fast as \(n!\) on independence systems of the \(2t\) sets \(A_1, A_2, \ldots, A_{2t}\), specified later. A set \(B\) is independent \(\frac{n}{4t^2}\). All independence systems of distribution \(D\).

A second number of steps required for a dependent set in an independence bound may we assume that that free of charge in the course of the run consists of the identity of the set \(A_i\), required to work correctly only for the identity in the distribution \(D\). Thus, we a generic \((n, p)\) decision tree, in which a get only the answers to the \(p\) queries of the identity of the set \(A_i\), and in correct on instances with nonzero computation path in \(H\) defined the two where \(O_j\) is the event defined by the event \(A_{i-1}\), and \(\mathcal{Q}_i\) is the event \(A_1, A_2, \ldots, A_{i-1}\). The probability the computation is its condition

Assumed before the \(j\)th step, we may be presented at the \(j\)th step, that the algorithm terminates by \(B\) independent?,” where \(B\) is a \(\omega(A_{2t})\). Since the algorithm is in a to determined a maximal independent that the algorithm must have its execution time by at most one

with respect to a given independence set or

\[|B \cap \bigcup_{k=1}^{2t} A_k| \leq \frac{1}{2} \cdot (2t - 8 + 1)\]

query \(B\) at step \(j\) is determined by

the proof are based on the following

bounds on the tail of the hypergeometric distribution. Let \(n, M,\) and \(N\) be positive integers with \(M \leq N\) and \(n \leq N\). Let \(p = M/N\). Then, for \(0 < \beta < 1\)

\[\sum_{k=0}^{|I(t \beta^{m} p)} \binom{n}{k} \frac{(N-M)}{(n-k)} \leq \exp\left(-\frac{\beta^3 np}{3}\right)\]

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.

We prove an \(\Omega(t)\) lower bound on the expected execution time of the algorithm by showing that if a random input is drawn from the input distribution \(D\) than with high probability all the queries asked during the first \(t\) steps of the algorithm are local. Since the “verification query” that the algorithm is required to ask in its last step is not local the algorithm must run for \(\Omega(t)\) steps.

Let \(E_i\) denote the event “at least one of the queries in the \(i\)th step is not local,” and let \(L\), denote the event “all the queries in the first \(t - 1\) steps are local.” Then

\[\text{Prob}[E_i | O_i \cap Q_i \cap L_i] = \frac{\text{Prob}[E_i \cap O_i | Q_i \cap L_i]}{\text{Prob}[O_i | Q_i \cap L_i]} \leq \frac{\text{Prob}[E_i | Q_i \cap L_i]}{\text{Prob}[O_i | Q_i \cap L_i]}\]

We shall derive a lower bound on \(\text{Prob}[O_i | Q_i \cap L_i]\). Let \(B\) be a query presented to the independence oracle in some step \(j < i\). By the condition \(L_i\), the queries at the first \(t - 1\) steps are local, thus \(B\) is a local query. There are two cases to consider:

(i) \(|B \cap A_i| \leq m/4t^2\). 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, \(|B \cap \bigcup_{k=1}^{2t} A_k| \leq (j + 1)(2t - i + 1)(m/4t^2)\). The probability that \(B\) fails to be independent given \(Q_i\) is bounded above by the probability that a set of \((i - 1/2)(2t - i + 1)(m/4t^2)\) elements drawn at random from \(A_1 \cup A_2 \cup \ldots \cup A_{2t}\) has an intersection of size at least \(m/4t^2\) 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 \((i - 1/2)(m/4t^2)\) is greater than or equal to \(m/4t^2\). Using the
H-AV bound we find that this probability is bounded above by 
\[ \exp\left(-\frac{3}{2}(n/4t^2)\cdot \left(\frac{3}{2}/t\right)^2\right). \]
As \( i \) ranges from 1 to \( t \) this quantity remains bounded above by \( e^{-3n/64r^2} \).

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 \( k \), \( |B \cap A_k| > kn/4t^2 \). But, since \( B \) is a local query, we have showed that the probability of this event is bounded above by \( e^{-3n/64r^2} \). Hence, the conditional probability, given \( Q_r \), that some query during the first \( i-1 \) steps has an outcome different from the "normal" outcome indicated by the event \( O_r \) is bounded above by \( 2pt^2e^{-3n/64r^2} \), and it follows that \( \text{Prob}[O_r \mid Q_r \cap L_i] \geq 1 - 2pt^2e^{-3n/64r^2} \).

Next we derive an upper bound on \( \text{Prob}[E_i \mid Q_r \cap L_i] \). Let \( B \) be a set presented to the independence oracle at step \( i \) when the event \( Q_r \cap L_i \) occurs. As noted before, we may assume that \( B \subseteq A_1 \cup A_2 \cup \cdots \cup A_{2r} \). We distinguish between two cases:

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

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

Since \( p \) queries are presented to the oracle at step \( i \), the conditional probability, given \( Q_r \), that some query fails to be local at step \( i \), is bounded above by \( 2pt e^{-n/512r^3} \). We have now shown the following two inequalities:

\[ \text{Prob}[O_r \mid Q_r \cap L_i] \geq 1 - 2pt^2e^{-3n/64r^2} \] and \( \text{Prob}[E_i \mid Q_r \cap L_i] \leq 2pt e^{-n/512r^3} \). We can now conclude

\[ \text{Prob}[L_{i+1}] \geq 1 - \sum_{i=1}^{t} \text{Prob}[E_i \mid O_i \cap Q_r \cap L_i] \geq 1 - t \frac{2pt e^{-n/512r^3}}{1 - 2pt^2e^{-3n/64r^2}}. \]

It follows that, with probability greater than \( \text{Prob}[L_{i+1}] \) 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 that probability the execution time is at least \( t \). Choosing \( t = (n/400 \log np)^{1/3} \) we obtain

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

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 independent subset of \( B \) is independent in \( S/B \) if and only if \( \text{rank}_B(A) \) is defined as \( \max \{ A \cap C \mid C \subseteq E \} \), the maximum independent set in \( S/B \). A rank oracle answers

In any independence system, the maximum independent set problem can be defined by \( H_m = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{m} \).

ALGORITHM B.

\[ B \leftarrow E; \quad r \leftarrow \text{rank}_E(B); \]
while \( |B| > r \) do
begin
\( m \leftarrow |B|; \)
\begin{center}
\begin{tabular}{|c|}
\hline
\end{tabular}
\end{center}
draw a sample \( j \) from the probability distribution \( \{0, 1, \ldots, m-1\} \) in which each \( A \) is a random set of size \( S \), \( A \leftarrow \text{random set of size } S \); \( R \leftarrow \{ e \in B \mid \text{rank}_B(A \cup \{ e \}) \geq \text{rank}_B(B) \}; \)
\( B \leftarrow B - R; \)
end
output \( B; \)

The algorithm satisfies the invariant that it contains a maximum independent set in \( S \). It contains a maximum independent set in \( S \) if its local search set \( R \) contains a maximum independent set in \( S \) which is contained in \( B \) and contains no elements of \( R \); for it contains \( \text{rank}_B(A) + 1 \) elements of \( A \) and \( \text{rank}_B(A) + 1 \), contradicting the fact that \( A \) is independent.

Next we show that, at any given point in the algorithm, the size of \( B \) is \( m = |B| \). For \( i = 0, 1, \ldots, m-1 \) let

\[ p_i = \text{Prob}\left[ \text{rank}_B(B) = i + 1 \right], \]

where \( B \) is a randomly chosen set of size \( i \) from \( B - A \). Then \( r = \text{rank}_B(B) = \sum_{i=0}^{m-1} p_i \).

The algorithm chooses a set \( A \) of size \( i \) if \( \sum_{i=0}^{m-1} p_i = i \).

Since \( \sum_{i=0}^{m-1} p_i = r \) it follows that the
probability is bounded above by
1 2 3

is response that it is independent could be.
But, since B is a local query, we have a bound on it by 
bounded above by \( e^{-3n^{5/4}i^3} \). Hence, the 
query during the first \( i - 1 \) steps has an 
error indicated by the event \( Q_i \), is bounded 
\( \Pr[Q_i \cap L_i] \geq 1 - 2pt^2 e^{-3n^{5/4}i^3} \).
Let B be a set presented to 
the event \( Q_i \cap L_i \). As noted before, we 
identify between two cases:

- Then, by the H-AV bound, 
  \( e^{-n^{5/2}i^3} \). Whenever \( |B \cap A| > \frac{m}{4i^2} \)
  then B can fail to be local only if, for 
  the H-AV bound the probability that this 
  is \( e^{-n^{5/2}i^3} \), and the probability that it 
  is \( e^{-n^{5/2}i^3} \).

- At step \( i \), the conditional probability, 
  which at step \( i \), is bounded above by 
  the following two inequalities:
  \( \Pr[E_i, V \cap L_i] \geq 1 - \frac{2pt e^{-n^{5/2}i^3}}{1 - 2pt^2 e^{-3n^{5/2}i^3}} \).

Prob\([L_{i+1}] \) all queries at the first \( i \)
then the algorithm is required to present 
that probability the execution time is at
in
\[
\left( \frac{n}{\log np} \right)^{1/3}.
\]

THE COMPLEXITY OF PARALLEL SEARCH

243

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 
\( \text{rank}_B(A) \) is defined as max \( B \cap A \subseteq C \), where \( C \) ranges over the maximum independent 
set in \( S/B \). A rank oracle answers queries of the form "What is \( \text{rank}_B(A) ? \)."

In any independence system, the following randomized algorithm constructs a 
maximum independent set with probability 1. Let \( H_m \), the mth harmonic number, 
be defined by \( H_m = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{m} \).

Algorithm B.

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

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 rank \( B \) elements of \( A \). Then \( C \) 
contains no elements of \( R \); for if it did contains some element \( e \in R \) then it would 
contain rank \( B \) elements of \( A \cup \{e\} \). This would imply rank \( B \cup \{e\} \) > 
rank \( B \) + 1, contradicting the fact that \( e \in R \).

Next we show that, at any given step, the expected value of \( |R| \) is \( (m - r)/H_m \), 
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
\[
p_i = \Pr[\text{rank}_B(A \cup \{e\}) > \text{rank}_B(A)].
\]

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 \( 1/(m - i) H_m \), 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 \( (m - r)/H_m \).
We have shown that, when \( m \) elements remain, the expected number of elements eliminated at the next iteration is \( (m - r) / H_m \). 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.** \( T_{\text{rank}}(n, n) = O(\log^2 n) \).

### 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: \( \text{rank}_f(A) \) 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 \( \text{TMAT}_{\text{ind}}(n, p) \), \( \text{TMAT}_{\text{prob}}(n, p) \), \( \text{TMAT}_{\text{det}}^{\text{rank}}(n, p) \), and \( \text{TMAT}_{\text{prob}}^{\text{rank}}(n, p) \) represents the complexities of these four matroid problems. For example, \( \text{TMAT}_{\text{det}}^{\text{rank}}(n, p) \) 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.** \( \text{TMAT}_{\text{det}}^{\text{rank}}(n, p) = 1 \).

**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 \( a_1, a_2, ..., a_n \);

1. In parallel, for \( i = 1, 2, ..., n \), determine \( \text{rank}_f(\{a_1, a_2, ..., a_i\}) \);
2. \( B \leftarrow \{a_i \mid \text{rank}_f(\{a_1, a_2, ..., a_i\}) > \text{rank}_f(\{a_1, a_2, ..., a_{i-1}\})\} \).

We prove that \( B \) is a base. It is clear from the algorithm that, for \( i = 1, 2, ..., n \),  

\( |B \cap \{a_1, a_2, ..., a_i\}| = \text{rank}_f(\{a_1, a_2, ..., a_i\}) \)

has the right cardinality for a base, and \( B \) has the right core for a base, and \( B \) is the only base of \( E \) with this property.

In case \( \text{rank}_f(\{a_1, a_2, ..., a_{i+1}\}) > \text{rank}_f(\{a_1, a_2, ..., a_i\}) \), and, by induction on \( i \), \( \{a_1, a_2, ..., a_i\} \) is a maximal independent subset of \( \{a_1, a_2, ..., a_{i+1}\} \), hence the only maximal independent subset of \( \{a_1, a_2, ..., a_i\} \) that follows that \( B \cap \{a_1, a_2, ..., a_i\} \) contains a maximal independent subset of \( \{a_1, a_2, ..., a_i\} \), and it is equal to \( \{a_1, a_2, ..., a_i\} \) if and only if it is a base.

We can also exploit an earlier proof.

**Theorem 10.** \( \text{TMAT}_{\text{prob}}^{\text{ind}}(n, p) = O(1) \).

**Proof.** On inspecting the proof of \( \text{TMAT}_{\text{prob}}^{\text{ind}}(n, p) \), it applies even when the independence oracle is replaced by a set oracle, where \( E \) is the disjoint union of \( 2 \) sets, and a set is independent if and only if its size is \( i \), equal to \( \lceil in/4r \rceil \), for \( i = 1, 2, ..., 2r \). Such a set oracle is a special type of matroid known as a lower bound holds for matroids.

The following fact about matroids is true for all matroids [W76].

**Fact 1.** Let \( S = (E, I) \) be a matroid. \( A \) be linearly ordered. Let \( D \) be a set oracle, where \( E \) is the disjoint union of \( 2 \) sets, and a set is independent if and only if its size is \( i \), equal to \( \lceil in/4r \rceil \), for \( i = 1, 2, ..., 2r \). Such a set oracle is a special type of matroid known as a lower bound holds for matroids.

The following theorem shows that this is also true for matroids.

**Theorem 11.** \( \text{TMAT}_{\text{det}}^{\text{ind}}(n, n) = O(1) \).

**Proof.** We present an algorithm.
THE COMPLEXITY OF PARALLEL SEARCH

245

WIGDERSON

remain, the expected number of elements to be determined, is now from the results cited in the previous section.

A BASE IN A MATROID

For $A = (E, I)$ with the property that, for all $B$ such that $B$ have the same cardinality; in other words, independent sets coincide with this property, the concept of rank is defined. Every set is the rank of an independent subset of $A$. A set $B$ is a base, and a minimal dependent (proper) subset of $B$ is to be considered a base in a matroid. As in the case of decision trees containing queries on algorithms, randomization nodes; but in the present case, we now identify a maximal independent set.

$\text{TMAT}\text{i}\text{nd}_{\text{det}}(n, p), \text{TMAT}\text{i}\text{nd}_{\text{prob}}(n, p)$, and $\text{TMAT}\text{i}\text{nd}_{\text{det}}(n, n)$ refer to the complexity of the best deterministic algorithm using an independence oracle. The number of processors is $p$.

Usefulness of the additional structure is that it indicates that for matroids, the following algorithm constructs a base $B$.

$$B \cap \{a_1, a_2, ..., a_i\} = \text{rank}_E(\{a_1, a_2, ..., a_i\}).$$

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.** $\text{TMAT}\text{i}\text{nd}_{\text{prob}}(n, p) = \Omega((n/\log n)p^{1/4})$.
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 $j = \min \{i | IN \cup F_i$ 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 $a_{t_1}, a_{t_2}, \ldots, a_{t_{|F_i|}}$, where $s(i) = |F_i|;$

in parallel for $k = 1, 2, \ldots, s(i)$ test whether $IN \cup \{a_{t_1}, a_{t_2}, \ldots, a_{t_k}\}$ is independent;

let $k(i)$ be the least $k$ such that $IN \cup \{a_{t_1}, a_{t_2}, \ldots, a_{t_k}\}$ is independent; [such a $k$ must exist, since $IN \cup F_i$ is dependent]

end

$OUT \leftarrow OUT \cup \{a_{t_{k(i)}}, i = 1, 2, \ldots, r\}; F \leftarrow F - \{a_{t_{k(i)}}, i = 1, 2, \ldots, r\};$

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) $\text{rank}_E(IN \cup F) = \text{rank}_E(E)$.\footnote{Since $G$ has $n$ edges and at least one is of length at least $2d + 1$. If $s \leq 2d$, otherwise we are in the case $s \geq 2d + 1$ of $G$, there is at most one simple path. Then there would exist a cycle of length at most $\sqrt{m}$. $S \cup T$ contains at most $(\sqrt{m})^2$ edges. $E$ contains a cycle only if it contains $d$. The number of $s$-element subsets of a $d$-element set is }

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

Case (a). For some $i$, $IN \cup F_i$ is independent. Then (i) remains true after the execution of the instructions $IN \rightarrow IN \cup F_i,$ and $F \rightarrow F - F_i$; (ii) remains true because $IN \cup F_i$ 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 $OUT \leftarrow OUT \cup \{a_{t_{k(i)}}, i = 1, 2, \ldots, r\};$ (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 $a_{t_1}, a_{t_2}, \ldots, a_{t_{|F_i|}}.$ 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 $\text{TGRAPH}_w(n,p)$ and $\text{TGRAPH}_\text{det}(n,p)$, 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 $s = \lceil \frac{1}{2} n^{1 - \frac{2}{d}} \rceil$. 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 $2d + 1$. If $s \leq 2d$ then all $s$-element subsets of $E$ are independent. Otherwise, we are in the case $s \geq 2d + 1$. In this case, 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 number of $s$-element subsets of the $n$-element ground set containing a given $d$-element set is
\[
\binom{n-d}{s-d} \leq \frac{s(s-1) \cdots (s-d+1)}{n(n-1) \cdots (n-d+1)} \leq \left( \frac{s-d}{n-d} \right)^d \leq \left( \frac{s}{n} \right)^d.
\]

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

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

(ii) \(\text{TGRAPH}^{\text{ind}}_{\text{prob}}(n, n^{2d+1} + \log n) = O(\log n)\).

**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.**

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

At the beginning of each iteration, \(\text{IN} \cup \text{OUT}\) is partitioned into three sets, \(\text{IN}, \text{OUT}, \) and \(F\), and \(F\) contains a base. The base that is eventually deleted from \(\text{IN} \cup \text{OUT}\) is included; this is done by what is called a base in \(M\) then \(\text{IN} \cup \text{OUT}\) is a base in \(G\). Deleting elements from the ground set \(\text{IN} \cup \text{OUT}\) has two crucial properties: first, that the second, that the girth of \(N\) is at least \(\delta d\) and not less than \(\delta d\) in \(M\). Fact 1 ensures that there are \(\delta d\) elements of \(F\), and then deleting elements of \(\text{IN} \cup \text{OUT}\) preserves a base works for any \(\delta d\).

The next part of the algorithm exploits these properties. Because the graphic matroid \(N\) has girth \(2d +1\), Lemma 2 ensures that a random set of \(\delta d\) elements of \(N\) has a probability greater than \(1/2d\) that such subsets are chosen independently. The independent set \(F\) is independent of size \(\delta d\). Whether \(A\) is independent 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 \(K = \{e \in F \mid \text{for some circuit } C \text{ in } M \text{ of size } \leq 2d, e \text{ is the largest element of } C \text{ in the linear ordering} \}; \) [the independence tests performed at the previous step provide enough information to determine \(K\)].

The proof will be based on a technique that is similar to Algorithm C, but which is implemented in Algorithm A by searching for an independent set using a deterministic process. Theorem 13(i) will be replaced by a deterministic process that ensures one independent set is never deleted in a \(d\)-wise independent random variable. This technique was first introduced in randomized algorithms to determine if a property was present or not. We use essentially the same technique here all the details for completeness.

We shall show that, when matroid \(M\) is guaranteed to produce a...
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 \( \lfloor m^{1-2d} \rfloor \)-element subset of the ground set of \( N \) has a probability greater than \( \frac{1}{2} \) 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 \( \lfloor m^{1-2d} \rfloor \) will be found at each iteration. Whenever such sets are found one of them is inserted into \( IN \) and deleted from \( F \), so the size of \( F \) is reduced by \( \lfloor m^{1-2d} \rfloor \). It follows that, with high probability, the set \( F \) will become empty after \( O(n^{2d}) \) 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 \( d \geq 4 \), \( TGRAPH_{\text{ind}}(n, n^{2d+1}) = O(n^{3-d}) \).

(ii) \( TGRAPH_{\text{det}}(n, n^{3-d} \log^{n+1} n) = O(\log n) \).

**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. The crucial observation in converting Algorithm C to a deterministic one is that the analysis of Algorithm C requires only \( d \)-wise independent random variables. Once this fact is established, we can use techniques that were first introduced in [KW84] and [Lu87], for converting randomized algorithms to deterministic ones exploiting the \( d \)-wise independence property. We use essentially the same combinatorial construction as in [Lu87]. We present here all the details for completeness.

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 \( \Omega(m^{1-3/d}) \) in matroid
$N$, and from this it will follow that the algorithm terminates within $O(n^{d/2})$ iterations. The algorithm is as follows.

**Algorithm D.**

$IN \leftarrow OUT \leftarrow \varnothing; F \leftarrow E;$

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 $K = \{e \in F\}$ 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 $x$ 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 $\lceil q^{m - 3d} \rceil$;

in parallel, for each polynomial $f$ of degree $d - 1/2|GF(q)|$ do

begin

$A(f) = \{e \in F \mid f(x(e)) \in T \} ;$ 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 a decision tree of constant depth, find a maximal independent set $A$ in $F$;

output $\leftarrow IN \cup A ;$

end

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 $\geq 2d + 1$; 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 $\Omega(m^{1/3})$ and is at least 1 when $m \geq 16$.

Recall that if $N$ is a graphic matroid, a set $A$ is independent provided it does not contain a circuit. These sets correspond to simple paths in the graph, which refer to them as the forbidden sets.

Let $B$ be one of the forbidden sets, and let $m$ be the number of circuits of exactly $\lceil q^{m^{3/3d}} \rceil$ contain $B$. Consider the map $z$ over the forbidden sets $B$. The image under $z$ of the forbidden sets $B$ is determined by its values $\lceil q^{m^{3/3d}} \rceil$ of possible circuits of $B$. If the number of circuits of $B$ is equal to the number of possible $d$-tuples of value in $T$ at every point in $z(B)$. Then $\lceil q^{m^{3/3d}} \rceil$, and hence the number of forbidden sets, is $\lceil q^{m^{3/3d}} \rceil$.

Since there are at most $\binom{m}{2} \lceil q^{m^{3/3d}} \rceil$ forbidden sets, we obtain a lower bound on $\sum \lceil q^{m^{3/3d}} \rceil$.

We shall use an averaging argument to lower bound $\sum (|A(f)| - 1) = N(m) \Omega(1)$ for all $f$, as $A(f)$ varies over all polynomials of degree $d - 1$. This leads to a lower bound on the number of independent sets $A(f)$.

Hence

$$\sum |A(f)| = \sum \lceil q^{m^{3/3d}} \rceil$$

Since the number of independent sets $A(f)$ is at least

$$mq^{d - 1/|T|}$$

we have

$$mq^{d - 1/|T|} \geq \frac{1}{2}(|T| - 1)$$

for all $d$. This concludes the proof.
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 $\binom{m}{d}$ $d$-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 $\lceil qm^{\frac{d-1}{2}} \rceil$ contain $B$. Consider $z(B)$, the subset of $GF(q)$ which is the image under $\chi$ 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 $z(B)$, 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 $z(B)$. The number of $d$-tuples of values from $T$ is $\lceil |T|^d \rceil = \lceil qm^{\frac{d-1}{2}} \rceil$, and hence the number of sets $A(f)$ containing the forbidden set $B$ is $\lceil qm^{\frac{d-1}{2}} \rceil$. Since there are at most $\binom{m}{d}$ forbidden sets the number of sets $A(f)$ which contains some forbidden set, and hence are not independent, is at most

$$\binom{m}{2} qm^{\frac{d-1}{2}} < \binom{m}{2} q^d m^{-\frac{d}{2}} < \frac{q^d}{2} m^{-1}.$$  

We shall use an averaging argument to show that there exists an independent set $A(f)$ of cardinality $\frac{1}{2}|T| - 1 = \Omega(m^{\frac{d-1}{2}}).$ 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 $A(f)$.

$$\sum_{f} |A(f)| = \sum_{f} \sum_{z(e) = y} |\{ f(f(z(e)) = y) \}|$$

$$= \sum_{f} \sum_{z(e) = y} q^{d-1} = mq^{d-1} |T|. $$

$$\sum_{|A(f)| \leq m |\{ f | A(f) \text{ is dependent} \}} |A(f)| \leq m |\{ f | A(f) \text{ is dependent} \}|$$

$$< m \left( \frac{q^d}{2} m^{\frac{d}{2}} \right) = \frac{q^d}{2}.$$  

Hence

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

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

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

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

In order to prove (ii) we observe that if \( d = \log n \) then \( m^{1 - 3/4d} \) is a fixed fraction of \( m \). Thus, the algorithm terminates in \( O(\log n) \) iterations.

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.

\[
m \leftarrow n; \quad t \leftarrow 0; \\
\text{while } m > 0 \text{ do} \\
\quad \text{begin } m \leftarrow m - X(m); \quad t \leftarrow t + 1 \text{ end} \\
T \leftarrow t;
\]

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.** \( E[T] \leq \int_0^1 dx/g(x) \), and for every \( a > 0 \), \( \text{Prob}[T > (a + 1)\int_0^1 dx/g(x)] < e^{-a} \).

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 the probabilistic analysis of deterministic algorithms.
in the while loop is $O(n^{1/d})$. With $n^{2d-1}$ such iteration in constant time. This implies $\log n$ then $m^{1-3/d}$ is a fixed fraction of the iterations.

THE COMPLEXITY OF PARALLEL SEARCH

REFERENCES


