Are Search and Decision Problems Computationally Equivalent ?

Richard M. Karp[1] U.C. Berkeley

Eli Upfal[2] Stanford University

Avi Wigderson[3] IBM - San Jose

Abstract

From the point of view of sequential polynomial time computation, the answer to the question in the title is 'yes'. The process of *self-reducibility* is a linear time Turing (oracle) reduction from a given combinatorial search problem to an appropriately defined decision problem.

However, from the point of view of fast parallel computation, the answer is not so clear. Many of the sequential algorithms that were 'marked off' as being "inherently sequential" embed within them the self-reducibility process. Can this inherently sequential process be parallelized?

To study this problem, we define an abstract setting (namely that of an independence system) in which *one, universal* search problem captures *all* combinatorial search problems. We consider several natural decision and function oracles to which this search problem may be reduced.

On the positive side, we give efficient *probabilistic* parallel reductions to these oracles. These reductions constitute a scheme for parallelizing search problems in case the oracles for these problems are themselves efficiently computable in parallel. We give examples of problems that did not yield to parallelism before, but can be parallelized using this scheme.

On the negative side, we prove lower bounds on any *determininistic* parallel reductions to the same oracles. If p processors are used, the sequential (linear) running time cannot be enhanced by more than a factor of $O(\log p)$ and hence for any polynomial number of processors the problem remains inherently sequential. This proves that randomization can be exponentially more powerful than determinism in our model, and suggests that NC $\neq$ Random NC.

Finally, we state some intriguing conjectures and suggest new directions of research in complexity theory that arise from this work.

We acknowledge support from
[1] NSF grant MCS 8105217
[2] Weizmann Postdoctoral Fellowship and DARPA grant N00039-83-C-1036
[3] DARPA grant N00039-82-C-0235

1. Introduction

1.1 Motivation

A key question in the theory of parallel computation is which (combinatorial) problems are susceptible to efficient parallelization, or in short, does P=NC? In the study of the analogous question, P=NP?, people quickly realized that (using self-reducibility) they can dispose of search problems and consider only decision problems.

A similar line of thought for the P=NC? question *seems* to fail. Many decision problems have simple parallel algorithms, which give no clue to parallelizing the relevant search problems. For example, testing whether a graph is Eulerian can be easily done by looking in the parity of the degrees of the vertices. However, finding an Euler circuit is much harder [$AIS-84$]. Other examples are Biconnectivity [$VT-84$], Pattern Matching [$G-84$], Strong Orientation [$V-84$], and Maximal Independent Set [$KW-84$] to mention but a few. In all these cases, the seemingly inherently sequential nature of the serial algorithms caused researchers to seek (and find) completely new algorithms. The mass of new parallel algorithmic techniques that resulted has undoubtedly benefitted the field. However, it has not shed any light on whether these "inherently sequential" algorithms can be directly parallelized.

In this paper we try to "take the bull by the horns" and confront this question. To do this, we formalize an abstract setting which captures all problems of this type, and in which we can prove both upper and lower bounds. Surprisingly, our results seem to imply that efficient parallelization of such "inherently sequential" algorithms is impossible deterministically, but can be achieved with the aid of randomization.

Before describing the general setting, consider first two concrete search problems, which we use to clarify our approach and the points we shall focus on. The input to both problems is a simple undirected graph $G(V,E)$, where $V=\{1,2,...,n\}$.

1. Find a maximal independent set in G.
2. Find a maximum (cardinality) independent set in G.

Simple sequential algorithms for these problems are given below.

```
Algorithm 1
$M \leftarrow \phi$;
for $i = 1$ to $n$ do;
    if $M \cup \{i\}$ is independent in $G$ then $M \leftarrow M \cup \{i\}$;
end;
output $\leftarrow M$;
```

```
Algorithm 2
$M \leftarrow \phi$;
for $i = 1$ to $n$ do;
    if $M \cup \{i\}$ is contained in a maximum
        independent set of $G$ then $M \leftarrow M \cup \{i\}$;
end;
output $\leftarrow M$;
```

Let us observe the following.

1) The two algorithms use the input graph G in a very restricted way; they each require only a specific (Boolean) set function of the vertices. Hence they will perform equally well if an *oracle* for the appropriate function is given instead of the input. Later we shall assume that the inputs are replaced (or represented) by such oracles.

2) The computational complexity of the functions (or oracles) used in the two algorithms is strikingly different (if P≠NP); the first is in P while the second is NP-hard. However, the *relative* complexity of the search problem (computing a relation) to the appropriate decision problem (computing a function) is polynomial sequential time, as each algorithm halts after n steps given its oracle. We shall ask whether a similar statement is true if we replace "polynomial sequential time" by "fast parallel time" or "small space".

3) The choice of an oracle by the algorithm is crucial. Although both problems deal with independent sets in graphs, there is no polynomial time algorithm (unless P=NP) for the second problem that uses the oracle of the first algorithm. Given a search problem, using *which* oracles can it be solved, say, in NC? (and can these oracles be computed in NC?)

4) In the two algorithms above, every decision point (oracle query) depends on the outcome of previous decisions (oracle answers). Such algorithms are often called "inherently sequential". We recognize this "inherently sequential" process as the well known "self-reducibility" process of reducing a search problem to a decision problem.

1.2 A General Setting - Independence Systems and Oracles

An *independence system* S is defined by a pair (E,I). E is a finite set of n elements, called the *ground set*. I is a family of subsets of E, called the *independent sets of* S. The independent sets are closed under containment, namely if $A \subset B \subset E$, and $B \in I$, then also $A \in I$. The sets in $2^E - I$ are called *dependent*. A set $M \subset E$ is a maximal independent set of S if M is independent, but every superset of M is dependent.

Now we are ready to state the search problem whose complexity we study.

S-search: Given an independence system $S = (E,I)$, find a maximal independent set of S.

The S-search problem is universal, in the sense that *every* combinatorial search problem can be stated in this form. More formally, let a search problem be given in terms of its input-output binary relation, say $R \subset \{0,1\}^m \times \{0,1\}^n$. Then for every input $x \in \{0,1\}^m$ one can construct an independence system S_x on $2n$ elements, whose maximal independent sets are in one-to-one correspondence with the n-bit strings y satisfying $(x,y) \in R$. The ground set of S_x will be $\{e_1, \bar{e}_1, \ldots, e_n, \bar{e}_n\}$, and for every $y = y_1 y_2 \ldots y_n$ s.t. $(x,y) \in R$, the corresponding maximal independent set M_y will contain, for $1 \le i \le n$, the element e_i or $\bar{e}_i$ iff y_i is 1 or 0, respectively.

Note: The paragraph above does not say much by itself. One can choose many other "universal" problems with enough information to encode all search problems. Also, the fact that a problem is universal does not necessarily means it is useful for understanding the problems it captures. *The results in this paper are the justification for our choice.*

The first step towards studying the complexity of the S-search problem is to explain how the input to the problem is given, and what is considered "input size". Clearly, an arbitrary independence system on n elements may require $2^{\Omega(n)}$ bits to describe it. However, as mentioned above, we shall assume the input S is (given by) an oracle (which can be thought of as a large random access input tape) to some set function $f_S: 2^E \rightarrow N$. The input size will be n, the size of the ground set E, which is also the size of an oracle query. (This is a natural choice for input size, as will be seen in a minute.)

The most natural oracle to describe an independence system is the *independence oracle*, $ind_S: 2^E \rightarrow \{0,1\}$. For every $A \subset E$, $ind_S(A) = 1$ iff $A \in I$ in S. Computing the function ind_S is the universal decision problem corresponding to our universal search problem (at least as far as sequential computation is concerned). This is seen from the self-reducibility algorithm below.

```
M ← $\phi$;
for $i = 1$ to $n$ do;
  if $ind_S(M \cup \{i\}) = 1$ then $M \leftarrow M \cup \{i\}$;
end;
output ← $M$;
```

Note: Since the running time of the self reducibility algorithm should symbolize sequential linear time, n was the natural choice for the input size. Also notice that the two algorithms in the previous subsection use an independence oracle of the appropriate independence system.

Another oracle that we shall study is the *rank oracle*, $rank_S: 2^E \rightarrow \{0,1,2,\ldots,n\}$. For every $A \subset E$, $rank_S(A) = Max\{|B| : B \subset A, B \in I\}$. In words, *rank* gives the size of the largest independent subset of a given (possibly dependent) set. (Note that $ind_S(A) = 1$ iff $rank_S(A) = |A|$).

We need some more notation before continuing. Let MI be the family of maximal independent sets in $S = (E,I)$. For $F \subset E$, the induced subsystem on F is $S(F) = (F,I(F))$, where $I(F) = \{C \in M \; M \in MI$ and $M \subset F\}$.

Now define $RANK_S$ to be the family of rank functions for all induced subsystems of S. Formally, $RANK_S = \{rank_{S(F)} \mid F \subseteq E\}$.

We shall prove probabilistic upper bounds and deterministic lower bounds on the the complexity of solving the search problem when the input is given by either an independence oracle or the family RANK of rank oracles. For brevity, other oracles for which we have similar results but no concrete applications will not be discussed here.

Before stating the results, we describe the computational model.

1.3 The model of computation

Our model will be a parallel decision tree with an input oracle. A *probabilistic decision tree of parallelism p with oracle f* is a tree with three types of nodes.

1. Randomization nodes. Every such (internal) node has some number d of branches, each taken with probability $1/d$.
2. Oracle Query nodes. Every such (internal) node is labeled with p subsets $A_1, A_2, \ldots, A_p$ of E. The branches from this node are labeled with all possible oracle answers $f(A_1), f(A_2), \ldots, f(A_p)$.
3. Leaves. Every leaf is labeled with one subset of E.

Assume without loss of generality that every independence system on n elements has the same ground set E. A tree H is said to solve the S-search problem for size n, if for every input (system) S of size n and for every root-leaf path the tree may take on input S, the leaf is labeled with a maximal independent set of S.

For a given tree H, let $c(H,S)$, the cost of S, be the expected (with respect to the randomization nc [illegible]) number of oracle query nodes on a root-leaf path that S may take. Let $c(H,n)$ be the maximum $c(H,S)$ over all inputs S of size n. Finally, the expected time to solve a problem of size n with oracle f and parallelism p, denote $T^f_{PROB}(n,p)$, is the minimum of $c(H,n)$ over al trees H of parallelism p that solve the searc problem for size n.

A *deterministic parallel decision tree* is a special cas of the probabilistic one, in which there are n randomization nodes. The deterministic time t solve the search problem, $T^f_{DET}(n,p)$ is similarl defined.

Finally, we extend the definitions above to com putations with a family of oracles, G. The onl difference is that now a query is a pair $(g,A$ with $g \in G$ and $A \subseteq E$, when we wish $g(A)$ evalu ated. In the complexity notation, G will replac the superscript f.

The choice of a decision tree is very natural fo proving lower bounds, but should be justified a an upper bound model. In our case it is done only to avoid technical detail. Our decision tre algorithms can be implemented on a PRAM wit only a factor n more processors, and only factor of $\log n$ more time.

1.4 Main Results and Applications

The main results stated below were obtained vi other results, that we shall describe in late sections. These other results are of independen interest, and are applicable beyond the scope o this paper. We also note here that a special cas of theorem 1.3, as well as corollary 1.3 appear also in [$KUW-84$].

All logarithms in this paper are to the base 2.

Deterministic Lower Bounds

THEOREM 1.1: For every $p>1$,

$$T^{ind}_{DET}(n,p) = \Omega(\frac{n}{\log p})$$

$$T_{DET}^{RANK}(n,p) = \Omega(\frac{n}{\log np})$$

An immediate corollary to this theorem is that if we allow only polynomial number of processors, then the sequential linear running time cannot be significantly improved, as both lower bounds in the theorem become $\Omega(\frac{n}{\log n})$

The next corollary, concerning the *space* required to (sequentially) solve the search problem, is much less immediate. Here, a Turing machine will use a (write only) oracle tape to get information about the input system, and we shall bound the amount of work space used to find a maximal independent set. The self reducibility process seems to require not only sequential linear time, but also linear space. This observation is supported by:

COROLLARY 1.1: Let TM be a Turing machine with independence oracle or RANK oracles, that solves the S-search problem for size n using work-space s (a precise definition is omitted). Then $s = \Omega(\sqrt{n})$

This lower bound follows by extending results about fast parallel simulation of space bounded machines, to machines with input oracles.

Probabilistic Upper Bounds

THEOREM 1.2: $T_{PROB}^{ind}(n,n) = O(\sqrt{n})$

Unlike the upper bound in theorem 1.2, which holds for any independence system, we could prove the next one only for the systems in which all maximal independent sets have the same cardinality. We call such systems regular.

THEOREM 1.3: For regular systems, $T_{PROB}^{RANK}(n,n) = O((\log n)^2)$

These two theorems show that using randomization, one can obtain a non-trivial parallel speed-up of the self-reducibility process. Whether this speed-up translates into an actual fast algorithm for concrete search problems obviously depends on whether the appropriate oracles can efficiently computed in parallel. We now give examples of problems which had no non-trivial parallel solution before, but yield to this general method. We start with applications of theorem 1.2.

Consider the following problems, where m is polynomial in n.

1. Minimal Hitting Set.
   Input: A family $\{A_1,A_2,...A_m\}$ of subsets of a universe $U, |U| = n$.
   Problem: Find a minimal subset $V \subset U$ such that $V \cap A_i \neq \phi$.
2. Maximally positive 2-satisfiability.
   Input: A 2-CNF formula on n variables.
   Problem: Find a satisfying assignment (if one exists) whose set of positive literals is maximal.
3. Minimal Generating set.
   Input: A set $\Pi = \{\pi_1,\pi_2,...,\pi_n\}$ of permutations on m letters, generating a group G.
   Problem: Find a minimal (irredundant) subset of Π that generates G.

For problems 1 and 2, the independence oracle is easily computed in NC. For problem 3, computing independence reduces to group membership testing. The latter can be computed in Random NC when G is abelian [$CM-83$] or a 2-group [$R-83$]. Hence we have:

COROLLARY 1.2: A probabilistic PRAM with a polynomial number of processors can solve the problems 1, 2, and the aforementioned special cases of problem 3 in expected time $O(\sqrt{n}(\log n)^{O(1)})$.

The most important applications we have are for theorem 1.3. In [$KUW-84$] we show how to compute a class of rank functions for matchingrelated problems in Random NC. These results yield

COROLLARY 1.3: The following search problems can be solved in Random NC: finding a perfect matching, finding a maximum edge-weight matching when weights are given in unary encoding, finding a maximum node-weight matching in any node-weighted graph, and finding a maximum flow in a network when the capacities are given in unary encoding.

2. Group Testing and Lower Bounds

In this section we describe and analyze a search problem in a different setting, in which it is easier to prove lower bounds. Relating this setting to independence systems we can translate these results into lower bounds for S-search.

2.1 The X-search Problem

The setting is a universe of U of N boolean variables $\{x_1, x_2, ..., x_N\}$. An instance is any truth assignment to these variables, or equivalently, a subset $X \subset U$ of "TRUE" variables. The search problem is

X-search: Given X, find an element of X (if one exists)

As before, X is not given explicitly, but by an oracle. The *or oracle*, $or_X : 2^U \to \{0,1\}$ is defined by $or_X(V) = 1$ iff $V \cap X \neq \phi$.

This setting was used in the past to capture several seemingly unrelated problems. One is the so called "group testing" problem $[H-84]$, in which one is given a set of coins, some legal and some counterfeit, and is asked to identify one (or all) counterfeit coins using different types of scales (which amount to different oracles). Stockmeyer $[S-83]$ considers the complexity of approximating the size of X using the *or* and weaker oracles, as a structured model to stud the complexity of problems in #P. In $[FRW-8$ this setting comes up naturally in considering th relative power of different conflict resolutio schemes in concurrent-write models of parall computation.

As the *X-search* problem is interesting in itsel we consider also upper bounds on it. As befor we use a p-way decision tree as our model, wit queries to the or oracle and leaves labeled wit a variable from U. Denote the probabilistic an deterministic complexities of the *X-search* prob lem for universe of size N by $\tilde{T}^{or}_{PROB}(N,p)$ an $\tilde{T}^{or}_{DET}(N,p)$, respectively. We give tight bound on these quantities below.

THEOREM 2.1: For every p

$$\frac{\log N}{\log(p+1)} \le \tilde{T}^{or}_{DET}(N,p) \le \frac{\log N}{\log(p+1)} + 1$$

THEOREM 2.2: $\tilde{T}^{or}_{PROB}(N, \log N) = O(1)$

Proof of THEOREM 2.1:

The upper bound in the theorem is a simpl $(p+1)$-way search, which is an extension o binary search for $p=1$. The lower bound is generalization of an adversary argument tha appears in $[FRW-84]$, which we describe below.

The adversary will exhibit a subset of inputs that follow a long path down the tree. This will be done by successively restricting a set of inputs J_t that arrives to a particular tree node v of depth t, by appropriately answering the queries at v. Every input in the restricted input set, J_{t+1}, will arrive at the child u of v defined by these answers.

The set J_t will have a simple structure; it will contain all satisfying assignments to a simple conjunctive normal form formula g_t on the variables in U. The set of satisfying assignments of a formula g will be denoted by $J(g)$.

A CNF formula g is called *positive* if every negated variable appears in a singleton clause. So, $g = (N,P)$ where N is a set of clauses of the form $(\bar{x}_j)$, and P is a set of clauses of the form $(x_{j1} \cup x_{j2} \cup \ldots \cup x_{jl})$. An important parameter of a positive formula is its "size", $s(g)$, defined by $s(g) = \min\{|C| : C \in P\}$ when P is nonempty, which will always be the case.

Note that if $J_t = J(g_t)$ is the set of inputs that reaches a depth t node v in the tree, and if $s(g_t) > 1$, then v cannot be a leaf (why?). We shall start with $g_0 = (x_1 \cup x_2 \cup \ldots \cup x_N)$ at the root of the tree (so $s(g_0) = N$). Our adversary will ensure that for every t, $s(g_t) \geq N(p+1)^{-t}$, which will complete the proof. This is shown in the next lemma. Let $r_t = N(p+1)^{-t}$.

LEMMA 2.1: Let g_t be a positive formula with $s(g_t) \geq r_t > 1$. Let $J_t = J(g_t)$ be the set of inputs that reach a node v at depth t in the tree. Then there exists a child u of v, and a positive formula g_{t+1}, such that all inputs in $J_{t+1} = J(g_{t+1})$ reach the node u, and $s(g_{t+1}) \geq r_{t+1}$.

Proof: Let $A_1, A_2, \ldots, A_p \subset U$ be the set of oracle queries at node v. Our adversary will answer the queries according to the following algorithm. Let $g_t = (N_t, P_t)$.

```
$N_{t+1} \leftarrow N_t$;  $P_{t+1} \leftarrow P_t$;
for $i = 1$ to $p$ do;
   ($A_i, A_{i+1}, \ldots, A_p$ are still unanswered.Assume
   $A_i$ is the smallest unanswered query).
   if $|A_i| \geq r_{t+1}$ then do;
      answer$or(A_i) = or(A_{i+1}) = \ldots = or(A^p) = 1$;
      $P_{t+1} \leftarrow P_{t+1} \cup A_i \cup A_{i+1} \cup \ldots \cup A_p$;
      halt;
   end;
   else do; ( $|A_i| < r_{t+1}$ )
      answer $or(A_i) = 0$;
      $N_{t+1} \leftarrow N_{t+1} \cup \{(\bar{x}_j) \mid x_j \in A_i\}$;
      for every $C \in P_{t+1}$ set $C \leftarrow C - A_i$;
      for every $j, i < j \leq p$ set $A_j \leftarrow A_j - A_i$;
   end;
   $g_{t+1} = (N_{t+1}, P_{t+1})$;
end;
```

The loop is performed at most p times, in each of which the size of every clause is decreased by at most r_{t+1}. Furthermore, at the last iteration, all clauses added to P_{t+1} have size at least r_{t+1}.

Proof of THEOREM 2.2: Assume that $|X| \geq 1$ (this can be verified in a single query). This algorithm has two parts, which are encompassed in the following two lemmas. The first lemma shows that a special case of the X-search problem, when $|X| = 1$, can be solved deterministically in one step. The second shows how to probabilistically reduce the general case to the special one in expected constant time.

Assume wlog that $N = 2^m$ for some integer m (otherwise add dummy false variables to U).

LEMMA 2.2: If $|X| = 1$, then $\tilde{T}^{or}_{DET}(N, \log N) = 1$.

Proof: Let the variables in U be indexed by distinct m-bit vectors $b_1 b_2 \ldots b_m$. The set of *or* queries of the oracle will be $A_1, A_2, \ldots A_m$, defined by $A_i = \{x \in U \mid$ the i'th bit in the index of x is $1\}$. It is immediate that if the answers to the queries are $a_1, a_2, \ldots a_m$, resp., then the index of the unique positive variable in U is $a_1 a_2 \ldots a_m$.

LEMMA 2.3: Assume that $2^{m-k-1} \leq |X| \leq 2^{m-k}$ for some $k \in \{0, 1, \ldots, m-1\}$. Uniformly choose a random subset V of U with $|V| = 2^k$. Then $\Pr[|V \cap X| = 1] \geq (2e)^{-1}$.

Proof: $\Pr[|V \cap X| = 1] = |V|(\frac{|X|}{|U|})(1 - \frac{|X|}{|U|})^{|V|-1}$
$\geq 2^k 2^{-k-1}(1 - 2^{-k})^{2k} \geq (2e)^{-1}$.

To complete the proof of the theorem, as we don't know $|X|$ in advance, we simply try all possible values for k in lemma 2.3. More precisely, the first set of queries in the algorithm is $V_0, V_1, \ldots V_{m-1}$ where V_k is a random subset of U of cardinality 2^k. Then, for the smallest k such that $or(V_k) = 1$, the algorithm of lemma 2.2 is applied on the universe V_k. As $|V_k \cap X|$ may be different than one, the output of the algorithm is verified (by a single query), and in case it is false, we restart the whole algorithm again. It is clear that the algorithm solves the X-search problem in an expected constant number of steps, and it uses only $m = \log N$ queries per step.

2.2 The Relation between S-search and X-search

We shall now establish the relationship between the S-search and the X-search problems. Theorem 1.1 will follow as a corollary of theorem 2.1 above and the following theorem.

THEOREM 2.3: Let $N = \binom{n}{\frac{n}{2}}$. Then

$$T_{DET}^{ind}(n,p) \geq \tilde{T}_{DET}^{or}(N,p)$$

$$T_{DET}^{RANK}(n,p) \geq \tilde{T}_{DET}^{or}(N,np)$$

Proof : First we deal with the first part of the theorem. Consider the family of independence systems of size n all of whose maximal independent sets have cardinality $n/2$. Associate with each $(n/2)$-subset M of E a boolean variable x_M, and let U be the set of these variables. Note that $|U| = N$. A natural 1-1 correspondence between instances of S-search from this family and instances of X-search from U is given when for every system S, the corresponding X_S contains exactly the variables associated with maximal independent sets of S.

We now show that every p-way tree H for these S-search instances can be transformed into a p-way tree H' for X-search instances from U. Simply replace every independence oracle query $A \subset E$ in H by the *or* oracle query $V_A \subset U$, where $V_A = \{x_M \in U | A \subset M\}$ in H'. Similarly, replace every leaf label $M \subset E$ in H by the variable x_M in H'. It is straightforward that the 1-1 correspondence between instances extends to the computations of the trees H and H', which completes the proof for the first part.

For the second part we use the same correspondence between instances of the two problems. It remains to show the correspondence between queries. Every query is now of the form $(rank_{S(F)}, A)$ where $A \subset F \subset E$. We show how this query can be "simulated" by $|A| + 1$ *or* oracle queries $V_A^0, V_A^1, \ldots V_A^{|A|}$ defined as follows. $V_A^k = \{x_M : |M \cap A| \geq k \text{ and } M \subset F\}$. It follows from the definitions that $rank_{S(F)}(A) = \max\{k : or(V_A^k) = 1\}$, which completes this part of the proof.

3. Probabilistic Recurrence Relations and Upper Bounds

For convenience, we state again our main upper bounds.

THEOREM 1.2: $T_{PROB}^{ind}(n,n) = O(\sqrt{n})$

THEOREM 1.3: For regular independence systems, $T_{PROB}^{RANK}(n,n) = O((\log n)^2)$

A similar result to theorem 1.3, for the special case of the matching problem, appears in [$KUW - 84$]. In subsection 3.1 we prove theorems 1.2 and 1.3. From the analyses of the two probabilistic algorithms that give the upper bounds above we extract a general problem, and treat it in subsection 3.2.

3.1 Algorithms

Proof of THEOREM 1.2:

Below is an algorithm for S-search when S is given by an independence oracle. The algorithm will maintain three sets, IN, OUT and F, such that before every stage of the algorithm, IN will contain elements that will appear in the final maximal independent set, OUT will contain elements that will not appear in the final maximal independent set, and F will contain elements about which the algorithm is still undecided. Note that $IN \cup OUT \cup F = E$.

Algorithm

```
$IN \leftarrow \phi$ /; $OUT \leftarrow \phi$ /; $F \leftarrow E$;
while $|F|>0$ do;
    for each $i = 1,2,...,|F|$ uniformly choose a
        random $i$-subset $A_i$ of $F$;
    $R_i \leftarrow \{a \in F - A_i \mid ind_S(IN \cup \{a\}) = 1\}$;
    $r_i \leftarrow |R_i|$;
    $k \leftarrow i$ that maximizes
        $\{i + r_i \mid ind_S(IN \cup A_i) = 1\}$;
    $IN \leftarrow IN \cup A_k$;
    $OUT \leftarrow OUT \cup R_k$;
    $F \leftarrow F - A_k - R_k$;
end;
output $\leftarrow IN$;
```

To analyze the expected number of iterations, we study the expected decrease of the size of F in one iteration of the while loop. First observe that what happens in each iteration depends only on the independence subsystem $S' = (F,J)$ where $J = \{C \subseteq F \mid IN \cup C \in I\}$.

We need some notation. Let $|F|=m$, and let A_i be defined as in the algorithm, and a a random element in $F - A_i$. Set q_i=Pr[$ind_{S'}(A_i) = 1$], and p_i=Pr[$ind_{S'}(A_i \cup \{a\}) = 1 \mid ind_{S'}(A_i) = 1$]. We obviously have $q_l = \prod_{j=1}^{l-1} p_j$.

Let R_i, r_i and k be as in the algorithm. Then after one iteration $|F| = m$ decreases by $k + r_k$. We now show that $\mathrm{E}[k + r_k] \geq \frac{\sqrt{m}}{e}$.

Let l be the smallest integer such that $p_l < 1 - \frac{1}{\sqrt{m}}$. We distinguish two cases:

Case 1. $l > \sqrt{m}$. Then $\Pr[k + r_k > \sqrt{m}] \geq q_{\sqrt{m}} = \prod_{j=0}^{\sqrt{m}-1} p_j \geq (1 - \frac{1}{\sqrt{m}})^{\sqrt{m}} > e^{-1}$. Hence, $\mathrm{E}[k + r_k] \geq \mathrm{E}[k] \geq \frac{\sqrt{m}}{e}$.

Case 2. $l \leq \sqrt{m}$. By a similar argument, $q_l > e^{-1}$. From the definition of l, $\mathrm{E}[l + r_l] \geq \sqrt{m}$. Therefore, $\mathrm{E}[k + r_k] \geq e^{-1}\mathrm{E}[l + r_l] \geq \frac{\sqrt{m}}{e}$.

It follows from the above that if before an iteration of the while loop $|F| = m$, then after this iteration, $\mathrm{E}[|F|] \leq m - e^{-1}\sqrt{m}$. Deriving theorem 1.2 directly from this inequality is a simple matter. However, we recognize here a very general problem that is described in the next subsection.

Proof of THEOREM 1.3: We give a probabilistic algorithm that uses the rank oracles to find a maximal independent set. The algorithm will maintain one set, F, that initially will be E, the ground set. In every iteration of the loop in the algorithm we shall remove "redundant" elements from F, until F will be a maximal independent set. This is in contrast to the previous algorithm, in which the maximal independent set was constructed gradually.

Let H_m denote the mth harmonic number.

Algorithm

```
$F \leftarrow E$; $r \leftarrow rank_S(E)$;
while $|F| > r$ do;
```

```
    $m \leftarrow |F|$;
    pick a random $i \in \{0,1,\dots,m-1\}$ with prob. $\frac{1}{(m-i)H_m}$;
    pick uniformly a random $i$-subset $A$ of $F$;

$R \leftarrow \{e \in F - A \mid rank_{S(F)}(A \cup \{e\}) = rank_{S(F)}(A)\}$
    $F \leftarrow F - R$;
end;
output $\leftarrow F$;
```

For the correctness of the algorithm, it is sufficient to prove that if at the start of a while loop iteration F contains a maximal independent set of the input system S, then so does $F - R$ at the end of this iteration, which means that the elements in R are really redundant. This is shown in lemma 3.1.

The bound on the number of queries per step is obvious. As for the running time, we will show in lemma 3.2 that in every iteration the expected size of R is a large fraction of $|F| - r$, and it is here that we use the *regularity* of the system, as we know that the algorithm will halt when $|F| = r$. Then, as in the previous algorithm, the bound on the expected total running time will follow from the general *results* in the next subsection.

LEMMA 3.1: Consider a specific iteration of the loop. If initially F contains a maximal independent set of S, then so does $F - R$.

Proof: Consider any subset A of F. By definition (see page 3), $rank_{S(F)}(A) = \max\{|A \cap M| : M$ is a maximal ind. set in $S(F)\}$. By the assumption in the lemma, $S(F)$ is not empty. Take any maximal independent set M that achieves this maximum. By definition of $S(F)$, M is also a maximal independent set in S. It is sufficient to prove that $M \in F - R$. But notice that for any $e \in M - A$, $rank_{S(F)}(A \cup \{e\}) \geq |(A \cup \{e\}) \cap M| = |A \cap M| + 1 \geq rank_{S(F)}(A) + 1 > rank_{S(F)}(A)$, so these elements will never belong to R.

LEMMA 3.2: Let $|F| = m$ in the start of a particular loop iteration. Then $E[|R|] = \frac{m-r}{H_m}$.

Proof: Let A_i be a uniformly chosen random i-subset of F, and e_i a uniformly chosen element in $F - A_i$. For $i = 0,1,\dots,m-1$ define $p_i = \Pr[rank_{S(F)}(A_i \cup \{e_i\}) > rank_{S(F)}(A_i)]$. Clearly, $E[rank_{S(F)}(A_i)] = \sum_{j=0}^{i-1} p_j$. In particular, for $i = m$ we have $r = rank_{S(F)}(F) = \sum_{j=0}^{m-1} p_j$. Using this fact and the distribution defined in the algorithm we have $E[|R|] = \sum_{j=0}^{m-1} \frac{1}{(m-j)H_m}(1-p_j)(m-j) = \frac{1}{H_m}\sum_{j=0}^{m-1}(1-p_j) = \frac{m-r}{H_m}$.

3.2 Solving Probabilistic Recurrence Relations

These recurrence relations come up in the analysis of any algorithm that works in stages. Given that after every stage some natural parameter of the problem decreases by a random amount, about which we know only the expectation, we must find as much as possible about the distribution of the number of stages the algorithm requires. This is the situation not only in the analysis of probabilistic algorithms, but also in the probabilistic analysis of deterministic algorithms. The literature is full of specific analyses of specific algorithms, which are special cases of the results below. We now state precisely the problem and our results.

Let $\{X_i\}, i = 1,2,\dots$ be a family of random variables s.t. X_i assumes values from $\{0,1,2,\dots,i\}$. Consider the following process:

```
input $m \leftarrow n$
$t \leftarrow 0$;
while $m > 0$ do;
    $m \leftarrow m - X_m$;
    $t \leftarrow t + 1$;
```

end;
output $T \leftarrow t$;

What can be said about the distribution of T, given n, and the information that $E[X_i] \geq g(i)$, where $g: R^+ \to R^+$ is a monotone non-decreasing function.

THEOREM 3.1:

$$E[T] \leq \int_1^n \frac{dx}{g(x)}, \text{ and}$$

$$\Pr[T > (a+1) \int_1^n \frac{dx}{g(x)}] < e^{-a} \text{ for every } a > 0.$$

This theorem is best possible in the sense that there are distributions of the random variables X_i, with the given expectations, for which the upper bounds are tight. The proof supplies these distributions. The second part of the theorem is somewhat surprising, as we get information about the tail of the distribution of T, although we are given only the expectation of the X_i. This is very useful, as we can extend results on expectation to results with probability tending to 1, simply by letting a be any divergent function of n.

4. Discussion

The main issue of this paper is the relative complexity of solving search problems (computing relations) to solving decision problems (recognizing languages, computing functions). This issue becomes nontrivial for "weak" models of computation, such as small depth-circuits and small-space Turing machines. Our paper seems to be the first to address this issue, which we feel is fundamental in complexity theory, and we hope to see continuing research in this area.

We have chosen independence systems and oracles as a general setting in which this general problem can be studied, for arbitrary models of computation. The natural way in which independence systems capture the problem, and the results we have, seem to justify this choice.

Input oracles (as in $[HK-81]$) make a useful structured model for two reasons. Firstly, we are able to prove lower bounds which capture at least our intuition about the gap between deterministic and probabilistic parallel computation. Secondly, the choice of an oracle requires a deep understanding of what is the essential information from the input that an algorithm uses to solve a particular problem.

The independence and RANK oracles cannot significantly enhance the sequential running time of the self-reducibility algorithm. How about other oracles? We vaguely state here the following conjecture. For deterministic parallel computation with a polynomial number of processors, no significant speed-up of sequential time is possible, *regardless* of the oracle used, as long it gives "intrinsic" information. ("Intrinsic" roughly means independent of the order of elements in E, so, for example, the oracle cannot have some maximal independent set as the answer to some query.)

The relationship between computing relations and functions when space is restricted, is almost completely open, except for our result in corollary 1.1. It is particularly interesting whether randomization or even nondeterminism help to get good upper bounds. Such bounds are *not* precluded by corollary 1.4, since *the analogues* of Savitch $[Sa-70]$ and Borodin-Cook-Pippenger $[BCP-83]$ theorems about converting nondeterministic and probabilistic small space computations into deterministic ones *are false* for machines with input oracles!

To conclude, the theory of computation has concentrated on function computation, namely on computations in which the output is uniquely defined by the input. As in practice (e.g. search problems) this may be more than we need, we hope that this paper will initiate a study of the computation of relations.

References

[*AIS* – 84] B. Awerbuch, A. Israeli and Y. Siloach, "Finding Euler Circuits in Logarithmic Parallel Time", *Proc. 16th STOC*, 1984, pp. 249-257.

[*BCP* – 83] A. Borodin, S. Cook and N. Pippenger, "Parallel Computation for Well-Endowed Rings and Space Bounded Probabilistic Machines", *Information and Control* 58, nos. 1-3, 1983, pp.113-136.

[*C* – 83] S.Cook, "The Classification of Problems which have Fast Parallel Algorithms", Technical Report no. 164/83, Department of Computer Science, University of Toronto, 1983.

[*CM* – 84] S. Cook and P. McKenzie, "The Parallel Complexity of the Abelian Permutation Group Membership Problem", *Proc. 24th FOCS*, pp. 154-161.

[*FRW* – 84] F. Fich, P. Ragde and A. Wigderson, "Relations between Concurrent-Write Models of Parallel Computation", *Proc 3rd PODC* 1984, pp. 179-189.

[*G* – 84] Z. Galil, "Optimal Parallel Algorithms for String Matching", *Proc. 16th STOC*, 1984, pp. 240-248.

[*H* – 84] F. K. Hwang, "Three Versions of a Group Testing Game", *SIAM J. Alg. Disc. Methods*, 5 no. 2, 1984, pp. 145-153.

[*HK* – 81] D. Hausmann and B. Korte, "On Algorithmic versus Axiomatic Definitions of Matroids", *Mathematical Programming Study*, 14, 1981, pp. 98-111.

[*KUW* – 84] R. M. Karp, E. Upfal and A. Wigderson, "Constructing a Perfect Matching is in Random NC", *Proc. 17th STOC*, 1985.

[*KW* – 84] R. M. Karp and A. Wigderson, "A fast Parallel Algorithm for the Maximal Independent Set Problem", *Proc. 16th STOC*, 1984, pp. 266-272.

[*R* – 83] J. Reif, Parallel Algorithms for Graph Isomorphism", Technical Report no. 14-83, Aiken Computation Lab, Harvard University, 1983.

[*S* – 83] L. Stockmeyer, "The Complexity of Approximate Counting", *Proc. 15th STOC*, 1983, pp. 118-126.

[*Sa* – 70] W. J. Savitch, "Relationships between Nondeterministic and Deterministic Space Complexities", *J. Comput. System Sci.*, 4, 1970, pp.177-192.

[*TV* – 84] R. E. Tarjan and U. Vishkin, "Finding Biconnected components and Computing Tree Functions in Logarithmic Parallel Time", *Proc. 25th FOCS*, 1984, pp. 12-20.

[*Va* – 75] L.G. Valiant, "Parallelism in Comparison Problems", *SIAM J. Comput.* 4, no. 3, 1975, pp. 348-355.

[*Vi* – 84] U. Vishkin, "An Efficient Parallel Strong Orientation", Technical Report no. 109, Computer Science Department, New York University, 1984.

[*W* – 76] D. J. A. Welsh, *Matroid Theory*, Academic Press, 1976.