A Fast Parallel Algorithm for the Maximal Independent Set Problem

RICHARD M. KARP AND AVI WIGDERSON

University of California at Berkeley, Berkeley, California

Abstract. A parallel algorithm is presented that accepts as input a graph G and produces a maximal independent set of vertices in G. On a P-RAM without the concurrent write or concurrent read features, the algorithm executes in $O((\log n)^4)$ time and uses $O((n/(\log n))^3)$ processors, where n is the number of vertices in G. The algorithm has several novel features that may find other applications. These include the use of balanced incomplete block designs to replace random sampling by deterministic sampling, and the use of a "dynamic pigeonhole principle" that generalizes the conventional pigeonhole principle.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Block designs, graph theory, independent sets, parallel computation

1. *Introduction*

An *independent set* in a graph is a set of vertices, no two of which are adjacent. A *maximal independent set* is an independent set that is not properly contained in any independent set. In his survey of parallel computation [6] Valiant suggested the problem of finding a maximal independent set as an example of a computationally trivial problem that appears difficult to parallelize. He discussed a sequential algorithm that has up to n stages, and concluded that "it is difficult to see how the problem can be solved in substantially fewer stages, such as $O(\sqrt{n})$ or $O(\log n)$". The problem is also mentioned by Cook in his survey of computational complexity theory [1].

We give an algorithm to solve the maximal independent set problem in $O((\log n)^4)$ time using $O(n^3/(\log n)^3)$ processors (all logarithms in this paper are to the base 2). A randomized version of our algorithm runs in $O((\log n)^3)$ expected time with $O(n^2)$ processors. Our model of computation is the weakest version of a P-RAM, in which concurrent reads or concurrent writes of the same location are disallowed.

The maximal independent set problem has the unusual property that it is specified by an input–output relation, rather than a function. This is the case because a graph may have many maximal independent sets, any one of which is acceptable. To formalize this situation, let us say that an algorithm satisfies the relation $R \subseteq \{0, 1\}^* \times \{0, 1\}^*$ if, on every input $x \in \{0, 1\}^*$, it produces an output

This research was supported by the National Science Foundation under grant MCS 81-05217 and by DARPA Grant N00039-82-C-0235.

Authors' present addresses: R. M. Karp, Computer Science Division, University of California at Berkeley, Berkeley, CA 94720; A. Wigderson, Mathematical Sciences Research Institute, 1000 Centennial Drive, Berkeley, CA 94720.

y such that $(x, y) \in R$. A relation is said to lie in the class $\widetilde{NC}$ (the analog of the standard class NC) if there exists a P-RAM algorithm that satisfies the relation and operates in $(\log n)^{O(1)}$ time using $n^{O(1)}$ processors. Our main result is that the maximal independent set problem lies in $\widetilde{NC}$.

Several problems can be placed in $\widetilde{NC}$ through NC-reductions [2] to the maximal independent set problem. These include:

(i) *The Maximal Set Packing Problem.* Given a collection of sets $\{S_1, S_2, \ldots, S_t\}$, find a maximal subcollection in which all the sets are disjoint.

(ii) *The Maximal Matching Problem.* Finding a maximal matching in a graph G is equivalent to finding a maximal independent set in the line graph of G. Lev [5] shows that the maximal matching problem for bipartite graphs is in $\widetilde{NC}$.

(iii) *The 2-Satisfiability Problem.* Given a conjunctive normal form Boolean formula F with two literals per clause, either produce a truth-value assignment satisfying F or determine that F is unsatisfiable. It was previously known [4] that the decision problem for 2-CNF formulas is in co-NSPACE($\log n$), and hence in NC, but it appears to be a new result that the problem of constructing a satisfying assignment is in $\widetilde{NC}$.

Throughout the paper we concentrate on the combinatorial arguments that make the algorithm work. Implementation details will be omitted because of their simplicity. Essentially, the algorithm uses only the ability of the model to perform an associative operation (e.g., sum, min, or) on n values in time $O(\log n)$, using n processors.

2. *Graph-Theoretic Terminology*

Let $G = (V, E)$ be an undirected graph without loops or multiple edges. For any set $S \subseteq V$, let $N(S)$, the *neighborhood of* S, be defined as $\{w \in V \mid \text{for some } u \in S, \{u, w\} \in E\}$. Then S is *independent* if $S \cap N(S) = \emptyset$; that is, no two vertices in S are adjacent. An independent set S is called a *maximal independent set* if S is not properly contained in any independent set. Equivalently, $S \subseteq V$ is a maximal independent set if $S \cap N(S) = \emptyset$ and $S \cup N(S) = V$.

We often deal with induced subgraphs of G. If $K \subseteq V$, then K is also used to denote the *subgraph induced by* K. This subgraph has vertex set K, and its edge set $E(K)$ consists of those edges from E that have both their end points in K. So, $E(K) = \{\{u, w\} \subseteq K \mid \{u, w\} \in E\}$. It is useful to denote the set $\{\{u, w\} \subseteq K \mid \{u, w\} \notin E\}$ by $\overline{E(K)}$. Together, $E(K) \cup \overline{E(K)}$ will be referred to as the *edge slots* of K. Also, for $u \in K$, $d_K(u)$ denotes the degree of vertex u in subgraph K, and, for $S \subseteq K$, $N_K(S) = N(S) \cap K$.

3. *The Sequential Algorithm*

Let G be a graph with vertex set $\{1, 2, \ldots, n\}$. The following *sequential algorithm* constructs a maximal independent set I.

```
begin
$I \leftarrow \emptyset$
for $i = 1$ to $n$ do if $i \notin N(I)$ then $I \leftarrow I \cup \{i\}$
end
```

There is no apparent way to make the sequential algorithm run in $o(n)$ time through the use of $n^{O(1)}$ processors. The intuition that this algorithm is inherently sequential is supported by the following theorem due to Cook [2].

THEOREM 1. *The problem of deciding whether vertex n lies in the independent set created by the sequential algorithm is complete in P with respect to logspace reducibility.*

Since problems that are logspace complete in P are not believed to lie in NC, it is clear that our maximal independent set algorithm cannot simply emulate the sequential algorithm.

4. *Top-Level Description of the Parallel Algorithm*

A top-level description of our algorithm is as follows:

Algorithm 1

```
begin
$I \leftarrow \emptyset$; $H \leftarrow V$;
while $H \neq \emptyset$ do
  begin
  $S \leftarrow$ an independent set in induced subgraph $H$;
  $I \leftarrow I \cup S$;
  $H \leftarrow H - (S \cup N_H(S))$
  end
end
```

Before each execution of the body of the **while** loop, the following invariant assertion holds: the sets I, $N(I)$, and H are disjoint, and together they exhaust the vertex set V. It follows that, upon termination, I is a maximal independent set in G.

Once S is chosen, I and H can easily be updated in $O(\log n)$ time. We show below that in $O((\log n)^2)$ time, an independent set S can be chosen such that $|S \cup N_H(S)| = \Omega(|H|/\log |H|)$. It follows that the number of iterations is $O((\log n)^2)$, and hence that the whole algorithm runs in $O((\log n)^4)$ time.

5. *Procedure INDFIND*

A graph with $|V|$ vertices and $|E|$ edges contains an independent set of size at least $|V| - |E|$. Our algorithm uses a procedure called *INDFIND* to find such a set. Given a set of vertices T, INDFIND (T) constructs an independent subset of T by "killing" one end-point of each edge occurring in T. In this procedure, one processor is assigned to each pair $\{u, w\} \subseteq T$. The processor assigned to $\{u, w\}$ does the following:

if $\{u, w\} \in E(T)$ **then** kill (arbitrarily) u or w.

Then a set INDFIND (T) is constructed, consisting of those vertices in T that were not killed by any processor. It is clear that INDFIND (T) is an independent set, and that INDFIND (T) can be executed in $O(\log n)$ time.

Incorporating INDFIND into Algorithm 1, we obtain the following algorithm:

Algorithm 2

```
begin
$I \leftarrow \emptyset$; $H \leftarrow V$;
while $H \neq \emptyset$ do
  begin
  $T \leftarrow$ a set of vertices in $H$;
  $S \leftarrow$ INDFIND ($T$);
  $I \leftarrow I \cup S$;
  $H \leftarrow H - (S \cup N_H(S))$;
  end
end
```

6. *A Scoring Function*

The fundamental problem in the implementation of Algorithm 2 is rapidly to construct a set T such that

$$|\,\text{INDFIND}\ (T) \cup N_H(\text{INDFIND}\ (T))\,| \tag{1}$$

is $\Omega(|H|/\log|H|)$. An upper bound on (1) is $\sum_{u\in T}(1 + d_H(u))$. This bound is tight only when T is an independent set and no two elements of T have a common neighbor. In this section we derive a useful lower bound on (1). This bound includes correction terms that account for the vertices killed by INDFIND and for the double counting of common neighbors.

Let $K \subseteq H$ be a set of vertices. With every vertex $u \in K$, we associate a *profit* $\text{prof}_K(u) = 1 + d_K(u)$. With every edge slot $\{u, w\} \subseteq K$, we associate a *cost*

$$\text{cost}_K(\{u, w\}) = \begin{cases} \text{kill}_K(\{u, w\}) & \text{if } \{u, w\} \in E(K), \\ \text{double}_K(\{u, w\}) & \text{if } \{u, w\} \in \overline{E(K)}, \end{cases}$$

where

$$\text{kill}_K(\{u, w\}) = 1 + \max\{d_K(u), d_K(w)\},$$

and

$$\text{double}_K(\{u, w\}) = |N_K(u) \cap N_K(w)|.$$

For every set $T \subseteq K$, define,

$$\text{Score}_K(T) = \sum_{u\in T} \text{prof}_K(u) - \sum_{\{u,w\}\subseteq T} \text{cost}_K(\{u, w\}).$$

LEMMA 1. *For every T, K with $T \subseteq K \subseteq H$,*

$$\textit{Score}_K(T) \le |\,\textit{INDFIND}(T) \cup N_H(\textit{INDFIND}(T))\,|.$$

PROOF. Let $S = \text{INDFIND}(T)$. Note that the first two terms in the Inclusion–Exclusion formula give a lower bound on $|N_K(S)|$:

$$|N_K(S)| \ge \sum_{u\in S} |N_K(u)| - \sum_{\{u,w\}\subseteq S} |N_K(u) \cap N_K(w)|.$$

$$\begin{aligned}
&|\,\text{INDFIND}(T) \cup N_H(\text{INDFIND}(T))\,| \\
&\quad = S \cup N_H(S)\,| \\
&\quad = |S| + |N_H(S)| \ge |S| + |N_K(S)| \\
&\quad \ge |S| + \sum_{u\in S} d_K(u) - \sum_{\{u,w\}\subseteq S} |N_K(u) \cap N_K(w)| \\
&\quad = \sum_{u\in S} 1 + d_K(u) - \sum_{\{u,w\}\subseteq S} |N_K(u) \cap N_K(w)| \\
&\quad \ge \sum_{u\in S} 1 + d_K(u) - \sum_{\{u,w\}\in\overline{E(T)}} |N_K(u) \cap N_K(w)| \\
&\quad = \sum_{u\in T} 1 + d_K(u) - \sum_{u\in T-S} 1 + d_K(u) - \sum_{\{u,w\}\in\overline{E(T)}} \text{double}_K(\{u, w\}) \\
&\quad \ge \sum_{u\in T} \text{prof}_K(u) - \sum_{\{u,w\}\in E(T)} 1 + \max\{d_K(u), d_K(w)\} \\
&\qquad - \sum_{\{u,w\}\in\overline{E(T)}} \text{double}_K(\{u, w\}) \\
&\quad = \sum_{u\in T} \text{prof}_K(u) - \sum_{\{u,w\}\subseteq T} \text{cost}_K(\{u, w\}) = \text{Score}_K(T).
\end{aligned}$$

□

7. *A Rating Function*

We need to show the existence of a set $T \subseteq K$, say of cardinality $|T| = t$, such that $\text{Score}_K(T)$ is large. It is clearly sufficient to prove, that for some $R \subseteq K$, the average of $\text{Score}_K(T)$ over all t-subsets of R is large. It turns out that this average has a simple expression as a function of t and the average profits and costs of vertices and edge slots (respectively) in R. We proceed to define this function, and prove that its value is indeed the required average.

For any $R \subseteq K$, let

$$\overline{\text{prof}_K}(R) = \frac{1}{|R|}\sum_{u\in R}\text{prof}_K(u), \qquad \overline{\text{cost}_K}(R) = \frac{1}{\binom{|R|}{2}}\sum_{\{u,w\}\subseteq R}\text{cost}_K(\{u, w\}).$$

Then $\overline{\text{prof}_K}(R)$ is the average profit of a vertex in R and $\overline{\text{cost}_K}(R)$ is the average cost of an edge slot in R. Let the positive integer t be fixed. Define the function

$$\text{Rating}_K(R) = t\,\overline{\text{prof}_K}(R) - \binom{t}{2}\overline{\text{cost}_K}(R).$$

Note that if $|R| = t$, then $\text{Score}_K(R) = \text{Rating}_K(R)$.

LEMMA 2. *The average of $\text{Score}_K(T)$, as T ranges over all t-subsets of R, is $\text{Rating}_K(R)$. (Hence, for some $T \subseteq R$, $|T| = t$, $\text{Score}_K(T) \geq \text{Rating}_K(R)$.)*

PROOF

$$\frac{1}{\binom{|R|}{t}}\sum_{\substack{T\subseteq R\\|T|=t}}\text{Score}_K(T) = \frac{1}{\binom{|R|}{t}}\left(\sum_{\substack{T\subseteq R\\|T|=t}}\sum_{u\in T}\text{prof}_K(u) - \sum_{\substack{T\subseteq R\\|T|=t}}\sum_{\{u,w\}\subseteq T}\text{cost}_K(\{u, w\})\right)$$

$$= \frac{1}{\binom{|R|}{t}}\left(\sum_{u\in R}\sum_{\substack{T\subseteq R\\|T|=t\\u\in T}}\text{prof}_K(u) - \sum_{\{u,w\}\subseteq R}\sum_{\substack{T\subseteq R\\|T|=t\\\{u,w\}\subseteq T}}\text{cost}_K(\{u, w\})\right)$$

$$= \frac{\binom{|R|-1}{t-1}}{\binom{|R|}{t}}\sum_{u\in R}\text{prof}_K(u) - \frac{\binom{|R|-2}{t-2}}{\binom{|R|}{t}}\sum_{\{u,w\}\subseteq R}\text{cost}_K(\{u, w\})$$

$$= \frac{t}{|R|}\sum_{u\in R}\text{prof}_K(u) - \frac{\binom{t}{2}}{\binom{|R|}{2}}\sum_{\{u,w\}\subseteq R}\text{cost}_K(\{u, w\})$$

$$= t\,\overline{\text{prof}_K}(R) - \binom{t}{2}\overline{\text{cost}_K}(\{u, w\}) = \text{Rating}_K(R). \qquad \square$$

The next lemma implies that if K contains many vertices of nearly maximum degree, then K contains subsets with a high rating.

LEMMA 3. *Let Δ be the maximum degree of a vertex in K. Call a vertex $u \in K$* heavy *if $d_K(u) \geq \Delta/2$. Let M be a set of heavy vertices in K, with $|M| = m$. Let t be a positive integer less than m, and let $\epsilon = t\Delta/m$. Then $\text{Rating}_K(M) \geq ((\epsilon/2) - 2\epsilon^2)\, m$. (Note that this lower bound is maximized if $\epsilon = 1/8$ and $t = m/8\Delta$. For this choice of ϵ and t the lower bound is independent of Δ.)*

PROOF

$$\begin{aligned}
\text{Rating}_K(M) &= t\,\overline{\text{prof}_K(M)} - \binom{t}{2}\overline{\text{cost}_K(M)} \\
&= \frac{t}{m}\sum_{u\in M}\text{prof}_K(u) - \frac{\binom{t}{2}}{\binom{m}{2}}\sum_{\{u,w\}\subseteq M}\text{cost}_K(\{u, w\}) \\
&= \frac{t}{m}\sum_{u\in M}\text{prof}_K(u) \\
&\quad - \frac{\binom{t}{2}}{\binom{m}{2}}\left(\sum_{\{u,w\}\in E(M)}\text{kill}_K(\{u, w\}) + \sum_{\{u,w\}\in \overline{E(M)}}\text{double}_K(\{u, w\})\right). \qquad (2)
\end{aligned}$$

We now bound each of the three summations in (2) separately.

$$\sum_{u\in M}\text{prof}_K(u) = \sum_{u\in M} 1 + d_K(u) \geq \sum_{u\in M}\frac{\Delta}{2} = \frac{m\Delta}{2}.$$

$$\begin{aligned}
\sum_{\{u,w\}\in E(M)}\text{kill}_K(\{u, w\}) &= \sum_{\{u,w\}\in E(M)} 1 + \max\{d_K(u), d_K(w)\} \\
&\leq \sum_{\{u,w\}\in E(M)} 1 + \Delta = |E(M)|(1 + \Delta) \leq \frac{m\Delta}{2}(1 + \Delta).
\end{aligned}$$

$$\begin{aligned}
\sum_{\{u,w\}\in \overline{E(M)}}\text{double}_K(\{u, w\}) &= \sum_{\{u,w\}\in \overline{E(M)}} |N_K(u) \cap N_K(w)| \\
&\leq \sum_{\{u,w\}\subseteq M} |N_K(u) \cap N_K(w)| = \sum_{z\in K}\binom{|N_K(z)\cap M|}{2}.
\end{aligned}$$

Note that

(i) $|N_K(z) \cap M| \leq d_K(z) \leq \Delta$, and
(ii) $\sum_{z\in K} |N_K(z) \cap M| \leq m\Delta$,

so by convexity

$$\sum_{z\in K}\binom{|N_{K_K}(z) \cap M|}{2} \leq m\binom{\Delta}{2} = \frac{m\Delta}{2}(\Delta - 1).$$

Plugging these bounds back into (2), we get

$$\begin{aligned}
\text{Rating}_K(M) &\geq \frac{t}{m}\frac{m\Delta}{2} - \frac{t(t-1)}{m(m-1)}\left[\frac{m\Delta}{2}(\Delta + 1) + \frac{m\Delta}{2}(\Delta - 1)\right] \\
&\geq \frac{t\Delta}{2} - \frac{t^2\Delta^2}{m-1} = \frac{\epsilon}{2}m - \epsilon^2\frac{m^2}{m-1} \geq \left(\frac{\epsilon}{2} - 2\epsilon^2\right)m. \qquad \square
\end{aligned}$$

8. *Procedure HEAVYFIND*

At a general step within Algorithm 2 we are given a subgraph H and are required to find a set T such that $|\text{INDFIND}(T) \cup N_H(\text{INDFIND}(T))|$ is large. Lemma 1 tells us that, for every $K \subseteq H$, $\text{Score}_K(T)$ is a lower bound on this quantity. Lemmas 2 and 3 say that, if K contains many heavy vertices, then, for some $T \subseteq K$, $\text{Score}_K(T)$ will be large. Thus, the remaining task for our algorithm is twofold: first, to find a subgraph K of H with a large number of heavy vertices and, second, to find a set T within K such that $\mathrm{Score}_K(T)$ is large.

For the first task we use a *dynamic pigeonhole principle.* Suppose we have pigeonholes $A_1, A_2, \ldots, A_a$ that collectively contain b pigeons. After the contents of pigeonhole A_i is inspected, A_i and the pigeons it contains disappear, and the remaining pigeons redistribute themselves among the remaining pigeonholes. The dynamic pigeonhole principle asserts that, if the pigeonholes are inspected one-by-one, then one of them will contain at least b/a pigeons at the time of its inspection. The principle is easily proved from the ordinary pigeonhole principle.

In our application the pigeons are the vertices in H and the degree of a vertex determines the pigeonhole. Initially vertex u is placed in pigeonhole i if $2^{i-1} - 1 \le d_H(u) < 2^i - 1$. The number of pigeonholes is $\lceil \log |H| \rceil$. Thus, in this case, $b = |H|$ and $a = \lceil \log |H| \rceil$.

At each step the pigeonhole corresponding to the highest range of degrees is inspected. If it contains at least b/a pigeons (i.e., vertices), then the process halts. Otherwise, the pigeons in this pigeonhole are released (i.e., the vertices with degrees in the highest range are deleted). The remaining pigeons then redistribute themselves (i.e., the deletion of these vertices causes the degrees of some of the remaining vertices to be reduced), and the step is repeated. If K denotes the vertices remaining when the process terminates, then all the vertices in the last pigeonhole inspected are heavy in K; hence K contains at least $|H|/\lceil \log |H| \rceil$ heavy vertices.

A more precise description of this process is the following:

```
Procedure HEAVYFIND (H)
$K \leftarrow H$; $i \leftarrow \lceil \log |H| \rceil$; success $\leftarrow$ FALSE;
while success = FALSE do
  begin
  $i \leftarrow i - 1$;
  if $|\{u \mid d_K(u) \ge 2^i - 1\}| \ge |H|/\lceil \log |H| \rceil$
  then success $\leftarrow$ TRUE
  else $K \leftarrow \{u \mid d_K(u) < 2^i - 1\}$
  end
end
```

Procedure HEAVYFIND (H) produces a subgraph K with at least $|H|/\lceil \log |H| \rceil$ heavy vertices. It requires $O(\log |H|)$ executions of the body of the **while** loop, and each of these executions can be performed in $O(\log |H|)$ time using $|H|^2$ processors.

9. *Procedure SCOREFIND*

Having found a set K with many heavy vertices, the maximal independent set algorithm proceeds to find a set T within K such that $\mathrm{Score}_K(T)$ is large. Lemma 3 tells us that, if M is a set of heavy vertices within K and t is a suitably chosen positive integer, then the average value of $\mathrm{Score}_K(T)$, as T ranges over all t-element subsets of M, is $\Omega(|M|)$. Our task is to find a specific set $T \subseteq M$ of cardinality t with at least an average score.

One natural approach is repeatedly to choose random t-element subsets of M until an acceptable one is found. A randomized algorithm along these lines can indeed be given. This algorithm runs in time $O(\log^3 n)$ and requires $O(n^2)$ processors. This can be seen informally as follows. Vertices are eliminated at an average rate of $\Omega(n/\log n)$ vertices per iteration. Hence, with high probability, the process terminates after $O(\log^2 n)$ iterations, each of which requires $O(\log n)$ parallel time and $O(n^2)$ processors. Since similar considerations will occur in bounding the resources required by the deterministic algorithm, we omit details.

Our next goal is to show that the set T can be found efficiently in parallel without randomization. The main idea behind our deterministic approach is the use of balanced incomplete block designs.

A *Balanced Incomplete Block Design* [3] with parameters v, b, k, r, λ is a family of subsets $B_1, B_2, \ldots, B_b$ of a finite set B such that:

(1) $|B| = v$.
(2) For all i, $|B_i| = k$.
(3) For every $x \in B$, the number of sets B_i containing x is r.
(4) For every two distinct elements $x, y \in B$, the number of sets B_i containing both x and y is λ.

The sets B_i are called the *blocks* of the design. Note that the parameters are not independent, namely, the relations $bk = vr$ and $b\binom{k}{2} = \lambda\binom{v}{2}$ hold.

Example 1. $B = \{1, 2, 3, 4, 5, 6, 7\}$ and the blocks are $\{2, 4, 6\}$, $\{1, 4, 5\}$, $\{3, 4, 7\}$, $\{1, 2, 3\}$, $\{2, 5, 7\}$, $\{1, 6, 7\}$, $\{3, 5, 6\}$. In this case $v = b = 7$, $k = r = 3$, $\lambda = 1$.

Example 2. B is the set of all nonempty subsets of a nonempty set X. The blocks are also in one-to-one correspondence with the nonempty subsets of X. For each such $A \subseteq X$, $A \neq \emptyset$, the corresponding block B_A is defined as $\{C \subseteq X \mid C \neq \emptyset, |A \cap C| \text{ is even}\}$. Then $v = b = 2^{|X|} - 1$, $k = r = 2^{|X|-1} - 1$, and $\lambda = 2^{|X|-2} - 1$.

Example 3. B is a finite v-set, and $B_1, B_2, \ldots, B_b$ are all the k-subsets of B. Then $b = \binom{v}{k}$, $r = \binom{v-1}{k-1}$, $\lambda = \binom{v-2}{k-2}$.

In Lemma 2 we proved that the average of $\text{Score}_K(T)$ over all t-subsets of the set R (the design of Example 3) is $\text{Rating}_K(R)$. A close look at the proof shows that it depended only on the fact that every element in R appears in the same number of t-subsets, and that every pair of distinct elements in R appear in the same number of t-subsets. Therefore, it immediately follows, from the definition of a block design, that the same lemma will hold if the average is taken over the blocks of any design.

LEMMA 4. *Let $B \subseteq K$, and let $B_1, B_2, \ldots, B_b$ be a balanced incomplete block design over B, with $|B_i| = t$. Then $1/b \sum_{i=1}^{b} \text{Score}_K(B_i) = \text{Rating}_K(B)$.*

The natural way to use Lemma 4 in our algorithm is to take $B = M$, a set of heavy vertices, compute in parallel $\text{Score}_K(B_i)$ for all blocks B_i, and take T to be the block that achieves the highest score.

To be able to carry out this procedure, three conditions must be satisfied:

(i) Since m and t may be arbitrary, we need a class of designs with $v = m$, $k = t$ for all integers m, t, such that $m \geq t$.
(ii) Since the number of processors in our model is limited, the number of blocks in the design must be bounded by a polynomial in v.
(iii) Since time in the model is limited, the blocks of the design must be computable in time polylog in v.

The design of Example 3 satisfies conditions (i) and (iii). However, the number of blocks will in general be superpolynomial in v. Our aim is to use the class of designs given by Example 2. Elementary combinatorial arguments show that these are indeed balanced incomplete block designs with the given parameters. Condition (ii) is clearly satisfied. For condition (iii), the following describes how to compute the blocks in time $O(\log v)$ using v^2 processors. For each ordered pair (A, C) of nonempty subsets of X, a processor tests whether $|A \cap C|$ is even, and if so, records that $C \in B_A$.

The problem is, however, that this class of designs always have parameters $v = 2^l - 1$, $k = 2^{l-1} - 1$, and therefore violate condition (i). Our final observation is that a stronger version of Lemma 4, which is given below, eliminates the need for condition (i). Together with the leeway in choosing m and t, guaranteed by Lemma 3, it gives rise to a homing-in strategy to find a subset T with a high score, using only designs defined in Example 2.

LEMMA 5. *Let $B \subseteq K$, and let $B_1, B_2, \ldots, B_b$ be any balanced incomplete block design over B. Then $1/b \sum_{i=1}^{b} \mathrm{Rating}_K(B_i) = \mathrm{Rating}_K(B)$. Hence, for some i, $\mathrm{Rating}_K(B_i) \geq \mathrm{Rating}_K(B)$. In particular, if $|B_i| = t$, then $\mathrm{Score}_K(B_i) = \mathrm{Rating}_K(B_i) \geq \mathrm{Rating}_K(B)$.*

PROOF. Let the design have parameters v, b, k, r, λ.

$$\frac{1}{b}\sum_{i=1}^{b} \mathrm{Rating}_K(B_i) = \frac{1}{b}\left[\sum_{i=1}^{b} t\,\overline{\mathrm{prof}_K(B_i)} - \sum_{i=1}^{b}\binom{t}{2}\overline{\mathrm{cost}_K(B_i)}\right]$$

$$= \frac{1}{b}\left[\sum_{i=1}^{b}\frac{t}{k}\sum_{u\in B_i}\mathrm{prof}_K(u) - \sum_{i=1}^{b}\frac{\binom{t}{2}}{\binom{k}{2}}\sum_{\{u,w\}\subseteq B_i}\mathrm{cost}_K(\{u,w\})\right]$$

$$= \frac{1}{b}\left[\frac{t}{k}\sum_{u\in B}\sum_{\{B_i|\,u\in B_i\}}\mathrm{prof}_K(u) - \frac{\binom{t}{2}}{\binom{k}{2}}\sum_{\{u,w\}\subseteq B}\sum_{\{B_i|\,\{u,w\}\subseteq B_i\}}\mathrm{cost}_K(\{u,w\})\right]$$

$$= \frac{1}{b}\cdot\frac{t}{k}r\sum_{u\in B}\mathrm{prof}_K(u) - \frac{1}{b}\frac{\binom{t}{2}}{\binom{k}{2}}\lambda\sum_{\{u,w\}\subseteq B}\mathrm{cost}_K(\{u,w\})$$

$$= \frac{t}{v}\sum_{u\in B}\mathrm{prof}_K(u) - \frac{\binom{t}{2}}{\binom{v}{2}}\sum_{\{u,w\}\subseteq B}\mathrm{cost}_K(\{u,w\})$$

$$= t\,\overline{\mathrm{prof}_K(B)} - \binom{t}{2}\overline{\mathrm{cost}_K(B)} = \mathrm{Rating}_K(B). \qquad \square$$

The following procedure selects T.

```
Procedure SCOREFIND (K)
begin
$l \leftarrow \max\{l' / l'$ integer and $2^{l'} - 1 \in [1, (|H|/\lceil \log |H| \rceil)]\}$;
$m \leftarrow 2^l - 1$
$M \leftarrow$ an arbitrary set of $m$ heavy vertices in $K$;
for every $u \in M$, compute $\mathrm{prof}_K(u)$;
for every $\{u, w\} \subseteq M$ compute $\mathrm{cost}_K(\{u, w\})$;
$\Delta \leftarrow$ maximum degree of a vertex in $K$;
$s \leftarrow \max\{s' | s'$ integer and $2^{s'} - 1 \in [1, \lceil m/16\Delta \rceil]\}$;
$t \leftarrow 2^s - 1$;
$U_l \leftarrow M$;
for $j = l$ down to $s + 1$ do
```

```
  begin
  construct a block design with set of elements
    $U_j$ and parameters $v = b = 2^j - 1$,
    $r = k = 2^{j-1} - 1, \lambda = 2^{j-2} - 1$;
  for each block $R$ compute Rating$_K(R)$;
  $U_{j-1} \leftarrow$ the block for which Rating$_K(\cdot)$ is largest
  end
$T \leftarrow U_s$
end
```

LEMMA 6. *Let T be the set produced by SCOREFIND(K). Then Score$_K(T) \geq (1/256)(|H|/\log|H|)$.*

PROOF. By Lemma 5,

$$\begin{aligned}\text{Score}_K(T) &= \text{Rating}_K(T)\\ &= \text{Rating}_K(U_s) \geq \text{Rating}_K(U_{s+1}) \geq \cdots \geq \text{Rating}_K(U_l) = \text{Rating}_K(M).\end{aligned}$$

By Lemma 3,

$$\text{Rating}_K(M) \geq \left(\frac{\epsilon}{2} - 2\epsilon^2\right)m,$$

where

$$m \geq \frac{|H|}{2\log|H|} \quad \text{and} \quad \epsilon = \frac{t\Delta}{m} \Rightarrow \frac{1}{32} \leq \epsilon \leq \frac{1}{16},$$

so

$$\text{Rating}_K(M) \geq \left(\frac{1}{64} - \frac{1}{128}\right) \cdot \frac{|H|}{2\log|H|} = \frac{1}{256}\frac{|H|}{\log|H|}. \qquad \square$$

The construction of the block design on the vertices in U_j requires the construction of a bijection between these vertices and the nonempty subsets of a 2^j-set. This is easily done by numbering the vertices from $[1, 2^j - 1]$ (using sorting, say), and considering the binary representation of each number as the characteristic vector of a subset of the 2^j-set.

The dominant requirements for processors in SCOREFIND occur during the sorting, the parallel computation of cost$_K(\{u, v\})$ for all $\{u, v\}$ and of Rating$_K(R)$ for all blocks R. In each case $O(m^3) = O(n^3/(\log n)^3)$ processors suffice to perform the computation in $O(\log n)$ time. The running time is dominated by the $O(\log n)$ executions of the body of the **for** loop, each of which requires $O(\log n)$ time.

10. *The Overall Algorithm*

The following is a complete statement of our algorithm for constructing a maximal independent set.

Maximal Independent Set Algorithm

```
begin
$I \leftarrow \emptyset$; $H \leftarrow V$;
while $H \neq \emptyset$ do
  begin
  $K \leftarrow$ HEAVYFIND($H$);
  $T \leftarrow$ SCOREFIND($K$);
  $S \leftarrow$ INDFIND($T$);
  $I \leftarrow I \cup S$;
  $H \leftarrow H - (S \cup N_H(S))$
  end
end
```

The number of executions of the **while** loop is $O((\log n)^2)$ and each such execution requires $O((\log n)^2)$ time using $O(n^3/(\log n)^3)$ processors. The overall execution time is $O((\log n)^4)$.

11. *An Application-2-Satisfiability*

We show that the following problem is in $\widetilde{NC}$: given a conjunctive normal form Boolean formula $F(x_1, x_2, \ldots, x_n)$ with two literals per clause, either find a truth-value assignment satisfying F or determine that none exists.

Each clause in F is of the form $a \cup b$ where a and b are literals; that is, they lie in the set $\{x_1, x_2, \ldots, x_n\} \cup \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}$. Such a clause is logically equivalent to either of the following two implications: $\bar{a} \Rightarrow b$, $\bar{b} \Rightarrow a$.

The first step in our algorithm is to construct an *implication digraph* G. The vertex set of G is $\{x_1, x_2, \ldots, x_n\} \cup \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}$. For each clause $a \cup b$, the directed edges $(\bar{a}, b)$ and $(\bar{b}, a)$ are edges of G.

The second step is to construct G^*, the transitive closure of G. This can be done in $O((\log n)^2)$ time with n^3 processors. The new edges added to G represent implications that follow from the original set. They also correspond to the new clauses that can be derived from the original set by repeated application of the resolution rule: from $a \cup b$ and $\bar{a} \cup c$ infer $b \cup c$.

The third step is to construct from G^* an undirected *conflict graph* C. The vertex set of C is $\{x_1, x_2, \ldots, x_n\} \cup \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}$. For each directed edge (a, b) in G^*, C contains the edge $\{a, \bar{b}\}$; in addition, C contains the edges $\{x_i, \bar{x}_i\}$, $i = 1, 2, \ldots, n$. Each edge of C indicates a conflict between two literals that cannot both be true in any satisfying assignment.

It is left to the reader to verify that, when F is satisfiable, the maximal independent sets of vertices in C are in one-to-one correspondence with the truth-value assignments that satisfy F. Each maximal independent set contains exactly n literals, and F can be satisfied by making these literals true. On the other hand, if F is not satisfiable, then the size of every independent set is less than n. It follows that constructing a satisfying assignment is reducible to the maximal independent set problem, and thus the former problem lies in $\widetilde{NC}$.

12. *Discussion and Open Problems*

In view of the growing importance of parallel computation, it is essential to understand the characteristics that make a problem amenable to parallelism and discover useful strategies for the construction of parallel algorithms. Of special interest are problems for which the exploitation of parallelism requires the invention of entirely new algorithms, rather than the conversion of known sequential algorithms to parallel form.

The main contribution of our paper is in introducing combinatorial design theory as an algorithmic technique. This theory originated more than a century ago, and is still a major research area among combinatorialists. It has applications in statistics, coding theory, and agriculture. The original motivation for studying combinatorial designs was to replace random sampling by deterministic sampling; this is exactly the way our algorithm makes use of the designs. We believe that combinatorial designs will find many applications in the design of efficient deterministic algorithms, and particularly in parallel algorithms, where they seem to fit so naturally.

A word of caution about our algorithm. If it is ever to be programmed, we strongly recommend using the randomized version—it would save programming time, running time, and processors.

There are many other problems that are not known to lie in $\widetilde{NC}$ although they are solvable sequentially by simple polynomial-time algorithms. Examples include the construction of a maximal simple path in a graph (Mayr), the construction of a minimal hitting set for a family of sets, and, in general, finding locally optimal solutions to various combinatorial optimization problems endowed with a neighborhood structure. Also of interest is fast parallel construction of combinatorial objects whose existence is guaranteed by famous theorems in graph theory. Examples of such objects are an edge coloring with $\Delta + 1$ colors in a graph with maximum degree of Δ (Vizing's Theorem) and a vertex coloring with Δ colors in a graph that is neither complete nor an odd cycle (Brooks' Theorem). The sequential algorithms for these problems seem to offer no obvious parallelization, and, therefore, it seems that new techniques will be needed for their solution.

REFERENCES

1. COOK, S. A. An overview of computational complexity. *Commun. ACM 26*, 6 (1983), 400–408.
2. COOK, S. A. The classification of problems which have fast parallel algorithms. Tech. Rep. No. 164/83, Dept. of Comput. Sci., Univ. of Toronto, Toronto, Ont., Canada, 1983.
3. HALL, M. *Combinatorial Theory.* Blaisdell, Waltham, Mass., 1967.
4. JONES, N. D., LIEN, Y. E., AND LAASER, W. T. New problems complete for nondeterministic log space. *Math. Syst. Theory 10* (1976), 1–17.
5. LEV, G. Size bounds and parallel algorithms for networks. Rep. CST-8-80, Dept. of Comput. Sci., Univ. of Edinburgh, Edinburgh, Scotland, 1980.
6. VALIANT, L. G. Parallel computation. In *Proceedings of the 7th IBM Symposium on Mathematical Foundations of Computer Science.* IBM, New York, 1982.

RECEIVED JULY 1983; REVISED APRIL 1985; ACCEPTED JUNE 1985