JOURNAL OF ALGORITHMS 14, 331–343 (1993)

# Parallelism, Preprocessing, and Reachability: A Hybrid Algorithm for Directed Graphs*

PHILIP N. KLEIN†

*Brown University, Providence, Rhode Island 02912*

Received March 1989; accepted November 19, 1991

The problem of reachability in a directed graph has resisted attempts at efficient parallelization. Only for fairly dense graphs can we efficiently achieve significant parallel speedups, using known methods. We describe a technique allowing significant parallel speedup even for moderately sparse graphs, following a sequential preprocessing step in which a representation of the graph is created. © 1993 Academic Press, Inc.

## 1. INTRODUCTION

In this paper we demonstrate the usefulness of preprocessing a graph in order to enable reachability queries to be processed quickly in parallel. Parallel processing offers potentially vast improvements in computational performance. However, in applying parallel processing to specific problems, we encounter a serious obstacle: for some problems, even the best of known algorithms permit only a very small speedup in relation to the number of processors used. That is, while potentially we can reduce the time for solving a problem by a factor of $p$ when $p$ processors are used, in fact the speedup factor may be closer to $\sqrt{p}$ or less. Indeed, for some

*A preliminary version of this paper has appeared in the Proceedings of the AMS-IMS-SIAM Joint Summer Research Conference on Graphs and Algorithms, July 1987, published as "Graphs and Algorithms" (R. Bruce Richter, Ed.), Vol. 89 of the AMS Contemporary Mathematics series.

†Author's research supported by an ONR Graduate Fellowship, by AT&T Bell Laboratories, and by Air Force Contract AFOSR-86-0078. Additional support provided by ONR Grant N00014-88-K-0243 at Harvard University. Work done while author was at MIT and subsequently at Harvard University.



0196-6774/93 $5.00

problems, to achieve even a moderate speedup seems to require an enormous number of processors.

In this paper, we propose a way of coping with the apparent limitations of "pure" parallelism. We introduce the notion of a "hybrid" algorithm, in which a sequential preprocessing stage prepares the way for fast parallel processing of queries. We give such an algorithm for the problem of directed reachability in a graph; our new algorithm makes use of a new decomposition of a directed graph.

We define the *directed reachability problem* as follows: Let $G$ be a directed graph with $n$ nodes and $m$ arcs. For a given set $S$ of nodes of $G$, find the set $T$ of nodes reachable in $G$ from nodes of $S$. We call $S$ the set of "sources," and $T$ the set of "targets." While this problem has a simple linear-time sequential solution (time $O(m + n)$), it has so far proved difficult to parallelize efficiently.

This problem or closely related problems arise in connection with diverse other problems. In artificial intelligence, a basic operation arising in semantic network manipulation is identifying the lowest common ancestors of some nodes. In combinatorial optimization, finding an augmenting path while solving a matching or flow problem is essentially finding a path in a directed graph. In databases, handling recursive queries can be reduced to finding paths between nodes in a directed graph.

Two approaches to the directed reachability problem are known, parallel transitive closure and parallel breadth-first search. There is a parallel algorithm for computing the transitive closure of $G$ in $O(\log^2 n)$ time. However, this algorithm seems to require about $n^3$ processors for all practical purposes; while asymptotically better algorithms exist, the associated constants are huge. Moreover, even supposing the transitive closure was already computed, merely reading enough of its entries to answer a query requires $\Omega(n^2)$ operations. Hence this approach is not efficient in comparison to the $O(m + n)$ time sequential solution: very many processors are needed in relation to the speed-up achieved.

On the other hand, a parallel version of breadth-first traversal of the graph $G$, starting at the nodes of $S$, can, over a sequence of stages, identify all the reachable nodes. Initially, we mark the nodes in $S$. In each stage, we consider the set of arcs leaving newly marked nodes, and mark the nodes they enter, until the stage at which no additional nodes are marked. The number of stages needed is $\Omega(n)$ in the worst case, so the use of parallelism cannot guarantee a speedup of more than $O((n + m)/n)$. Thus for moderately sparse graphs, this method fails to be useful in the worst case.

We propose a two-part solution. First, the graph $G$ is preprocessed sequentially, and a representation of $G$ is created. The representation depends on the number $p$ of processors we intend to use. Our technique

works only for values of $p \leq (m/n)\sqrt{m}$. The preprocessing takes $O(np)$ time to build the representation. Once the representation exists, we can use $p$ processors to answer queries of the form "given sources $S$, find targets $T$" in $O((n + m)/p)$ parallel time. The approach incorporates a trade-off between speed of processing a query and compactness of the representation of $G$. The storage required by the representation is $O(n + m + n^2p/m)$, which is optimal when $p \leq m^2/n^2$. The method for handling a query is simple enough to be potentially quite practical.

For simplicity of presentation, we assume as our model of parallel computation the CRCW P-RAM, in which we charge constant time for concurrent reads and writes accessing a common random-access memory. However, it is easy to achieve $O(m/p)$ time when concurrent reads and writes are disallowed, as long as $p \leq (m/n \log n)\sqrt{m}$.

### 1.1. *Relation to Other Work*

Our method relies on a compressed representation of a partially ordered set (*poset*) consisting of a small number of chains. This representation has been known since at least the work of Hiraguchi [5, 7] on the *dimension* of a poset. Jagadish [6], in work done independently and concurrently with ours, has rediscovered essentially this representation and proposes its use in handling database queries. However, not every poset can be decomposed into a small number of chains, and in the worst case such a "compressed" representation is in fact no smaller than the ordinary transitive closure, even for sparse graphs.

The key to our result is our observation that every poset can be decomposed into a small number of chains and a small number of antichains. We observe, furthermore, that a poset consisting of a small number of antichains can be searched quickly in parallel. Thus it is by balancing the use of two techniques, one for chains and one for antichains, that we achieve our result.

A line of research that resembles ours in spirit but differs in the problem considered, the techniques employed, and the result obtained is that of Gambosi, Nešetřil, and Talamo [4]. They describe techniques for preprocessing a directed acyclic graph (*dag*) in order to facilitate sequentially searching for a path between two given nodes. They characterize dags by certain representability parameters and express the time required to answer a path query using their method in terms of the length of the path and the values of these parameters. For some special classes of dags, the parameters are small and the algorithm performs well; in the worst case, however, it is no better than using the transitive closure, even for sparse graphs.

## 2. Preprocessing

We assume henceforth for simplicity that the number $m$ of arcs of $G$ is at least the number $n$ of nodes, minus one. If $m$ is less than $n - 1$, the underlying graph is disconnected, and we can independently consider each connected component.

In this section, we describe our method for sequentially preprocessing the graph. The time required for preprocessing is $O(np)$, where $p$ is the number of processors to be used in answering queries.

### 2.1. *Eliminating Cycles*

First we reduce the problem to the case in which $G$ is acyclic. Given any directed graph $G$, we find the strongly connected components of $G$ and obtain a graph $\hat{G}$ from $G$ by contracting each strongly connected component to a node. The strongly connected components of $G$ can be computed in $O(n + m)$ sequential time using an algorithm of Tarjan [9].

For any node $v$ of $G$, let $\hat{v}$ denote the node of $\hat{G}$ corresponding to the strongly connected component of $G$ containing $v$. Now, given a set $S$ of sources in $G$, let $\hat{S} = \{\hat{v}: v \in S\}$. If $\hat{T}$ is the corresponding set of targets in $\hat{G}$, let $T = \{v: \hat{v} \in \hat{T}\}$. Then $T$ is the set of targets corresponding to $S$ in the original graph $G$. The correctness of this reduction follows immediately from the definition of strongly connected components.

We therefore assume henceforth that $G$ is a directed acyclic graph, or *dag*. Hence $G$ defines a partial order on its nodes, namely $v < w$ if $w$ is reachable from $v$. Let $\mathscr{P}(G)$ denote the partially ordered set $(V(G), <)$ that the dag $G$ thus defines on its nodes.

### 2.2. *The Antichain–Chain Decomposition*

We review some terminology of partially ordered sets, or *posets*. For a poset $\mathscr{P} = (V, <)$, the poset $\mathscr{P}' = (V', <')$ is a *subposet* of $\mathscr{P}$ if $V' \subseteq V$ and for every pair of nodes $v, w \in V'$, we have $v <' w$ iff $v < w$. A *chain* is a set of nodes every two of which are related by $<$. The nodes in a chain can be totally ordered: $v_1 < v_2 < \cdots < v_k$. An *antichain* is a set of nodes no two of which are related. A *chain cover* is a partition of the nodes into chains; an *antichain cover* is a partition of the nodes into antichains. Clearly, the size of any chain cover is at least the size of any antichain, because each chain can cover at most one element of the antichain. Similarly, the size of any antichain cover is at least the size of any chain. In fact, we have the following theorem.

Dilworth's Theorem [2]. *In any partially ordered set (poset), the size of the smallest chain cover equals the size of the largest antichain.*

The following theorem is well known. It appears, for example, as Proposition 8.15 on page 398 of [1]. We include a proof because we must implement the proof as part of the preprocessing.

DUAL OF DILWORTH'S THEOREM. *In any poset $\mathscr{P}$, the size of the smallest antichain cover equals the size of the largest chain.*

*Proof.* For each node $v$, let the *rank* $r(v)$ of $v$ be the number of nodes in the longest chain ending at $v$. Let $k$ be the maximum rank of a node in $\mathscr{P}$. Then for each $1 \le i \le k$, the set of nodes of rank $i$ is an antichain, so we have an antichain cover of size $k$. Since the size of any antichain cover is at least the size of any chain, it follows that we have found a minimum-size antichain cover and a maximum-size chain. □

An inductive characterization of rank for a poset $\mathscr{P}(G)$ is as follows: when all nodes of rank 1 through $j$ are deleted, a remaining node is rank $j+1$ if and only if it now has no incoming arcs. This characterization suggests a sequential algorithm that, given a dag $G$, finds the ranks of $\mathscr{P}(G)$ in $O(m+n)$ time.

The algorithm we give is slightly more general in that it handles any subposet $\mathscr{P}'$ of $\mathscr{P}(G)$. We represent the subposet by assigning a flag deleted$[v]$ to each node $v$. The value of deleted$[v]$ is *false* for each node $v$ in the subposet, and *true* otherwise. For notational convenience, add the node $\perp$ and the arcs $(\perp, v)$ for each $v$ in $G$, and set deleted$[\perp]$ = false. The algorithm is given in Fig. 1. It places all nodes of rank $r$ in the list $L_r$ and computes the maximum rank. The algorithm simultaneously constructs a table $f[\cdot]$ such that, for each node $v$ of rank more than one, $f[v]$ is the penultimate node of a longest chain ending at $v$.

R1 Let $L_0 := \{\perp\}$, and let $r := 0$. For each node $v$, assign to $d[v]$ the indegree of $v$.

R2 Copy $L := L_r$.

R3 While $L$ is not empty, remove a node $v$ from $L$, and consider each arc $(v, w)$ in turn. Reduce $d[w]$ by 1. If $d[w]$ thereby becomes zero, then ...

R4 Set $f[w] := \begin{cases} v & \text{if } deleted[v] = false \\ f[v] & \text{if } deleted[v] = true \end{cases}$

R5 If $deleted[w] = false$, put $w$ on the list $L_{r+1}$ of rank $r+1$ nodes. Otherwise, put $w$ on the list $L$.

R6 If $L_{r+1}$ is not empty, set $r := r+1$, and go to step 2. Otherwise, $r$ is the maximum rank.

FIG. 1. Algorithm for finding the ranks of nodes of a subposet $\mathscr{P}'$ of $\mathscr{P}(G)$.

To find a maximum-size chain, let $k$ be the maximum rank, and let $\hat{v}$ be a node on the list $L_k$. Then $\hat{v}, f[\hat{v}], f[f[\hat{v}]], \ldots, f^{k-1}[\hat{v}]$ traces backwards along a $k$-node path from a rank 1 node to $\hat{v}$.

The following corollary, which follows from either Dilworth's theorem or its dual, is the basis for our representation of dags:

COROLLARY 2.1. *For any $1 \leq s \leq n$, an $n$-element poset can be decomposed into at most $s$ chains, each of size $\lceil n/s \rceil$, and at most $\lceil n/s \rceil - 1$ antichains.*

*Proof.* If the poset has a chain of size $\lceil n/s \rceil$, remove it. Iterate this step until all remaining chains are of size less than $\lceil n/s \rceil$, so, by the dual of Dilworth's theorem, the poset remaining has an antichain cover of size less than $\lceil n/s \rceil$. Since each iteration removes $\lceil n/s \rceil$ elements, at most $s$ iterations are needed. □

DEFINITION. A decomposition of an $n$-element poset $\mathscr{P}$ into sets $\mathscr{A}, \mathscr{C}_1, \ldots, \mathscr{C}_s$ is called an *antichain–chain decomposition* if $\mathscr{C}_1, \ldots, \mathscr{C}_s$ are chains of size at most $\lceil n/s \rceil$, and $\mathscr{A}$ is the union of at most $\lceil n/s \rceil - 1$ antichains.

To preprocess a dag $G$, we let $p$ be the desired number of processors and we define the parameter $s$ by $s = (n-1)p/m$. By our assumption, stated at the beginning of Section 2, $m \geq n - 1$, and so $s \leq p$. Next, we find an antichain–chain decomposition $\mathscr{A}, \mathscr{C}_1, \ldots, \mathscr{C}_s$ of the poset $\mathscr{P}(G)$ defined by $G$. To find such a decomposition, we implement the proof of Corollary 2.1, using the procedure described above. Each iteration consists of finding the ranks of the poset corresponding to the current graph (and the table $f[\cdot]$), and then identifying a chain of size $\lceil n/s \rceil$ and deleting the nodes in this chain. After at most $s$ iterations, there is no chain of this size. Thus the decomposition can be found in $O(sm)$ sequential time. In the next subsection, we discuss further processing of the subposet $\mathscr{P}'$ of $\mathscr{P}(G)$ induced on the node-set $\mathscr{C}_1 \cup \cdots \cup \mathscr{C}_s$.

### 2.3. *Processing of the Chains: The Earliest-Entry Table*

Next we describe a table that facilitates reachability queries for a dag whose poset has a small chain cover.

DEFINITION. A *top element* of a poset $\mathscr{P}$ is an element $\top$ that is bigger than all other elements of the poset. For any poset $\mathscr{P}$ with a top element $\top$, for a chain $\mathscr{C}$ of $\mathscr{P}$, and for any node $u$ of $\mathscr{P}$, we define $u$'s

*earliest entry into* $\mathscr{C}$ to be the minimum node $x$ in $\mathscr{C} \cup \{\top\}$ reachable from $u$. The minimum is well defined because $\mathscr{C} \cup \{\top\}$ is a chain.

Given that $u$'s earliest entry into $\mathscr{C}$ is $x$, we can determine exactly which nodes $v$ in the same chain $\mathscr{C}$ are reachable from $x$; namely, $v$ is reachable if and only if $v \geq x$.

Earlier, we derived a poset $\mathscr{P}(G)$ from the dag $G$, and decomposed $\mathscr{P}(G)$ into some antichains and $s$ chains $\mathscr{C}_1, \ldots, \mathscr{C}_s$ of size at most $\lceil n/s \rceil$. Let $\mathscr{P}'$ be the subposet of $\mathscr{P}(G)$ induced by the union of the $s$ chains. That is, the elements of $\mathscr{P}'$ are the elements of the $s$ chains, and the order relation of $\mathscr{P}'$ is simply the order relation of $\mathscr{P}(G)$ restricted to the elements of $\mathscr{P}'$. For notational convenience, add a top element $\top$ to $\mathscr{P}'$. Let $n' = 1 + \sum_i |\mathscr{C}_i|$ be the number of elements of $\mathscr{P}'$.

To preprocess $\mathscr{P}'$, we compute an $n' \times s$ table $R[\cdot, \cdot]$ that is a generalization of the transitive closure of $G'$. For each element $u$ of $\mathscr{P}'$, and each $i \in \{1, \ldots, s\}$, let $R[u, i]$ be $u$'s earliest entry into $\mathscr{C}_i$. We call this table the *earliest-entry table*. It requires $O(n's)$ storage.

We also compute two $n'$-element tables: we let chain$[u]$ be the index $j$ of the chain $\mathscr{C}_j$ in which $u$ appears, and we let rank$[u]$ be the rank of $u$ in that chain. Let rank$[\top] = n'$.

The earliest-entry table $R[\cdot, \cdot]$, chain table chain$[\cdot]$, and rank table rank$[\cdot]$ can be used to determine reachability between any two nodes $u$ and $w$ of $\mathscr{P}'$. Namely, $u < w$ if and only if rank$(x) \leq$ rank$(w)$, where $x = R[u, \text{chain}(w)]$.

The above approach to representing reachability is derived directly from a theorem in the dimension theory of posets stating that a poset with largest antichain of size $\omega$ has dimension at most $\omega$. Dimension theory is concerned with the compactness of representation of a poset as the intersection of total orders. See [7] for a survey. Also, Jagadish [6] independently considered the use of a Dilworth decomposition for more efficiently computing and more compactly representing the transitive closure of a directed graph, specifically in application to handling recursive database queries.

The tables chain$[\cdot]$ and rank$[\cdot]$ can easily be computed from the chain decomposition of $\mathscr{P}'$. For each node $w$ of $G$ that does not appear in the chain decomposition, let chain$[w] = 0$. We next observe that the earliest-entry table can be computed in $O(sm)$ sequential time. In particular, for each chain $\mathscr{C}_j$, we show how in $O(m)$ time one can compute all the entries $R[v, i]$ of the table for $v \in \mathscr{C}_j$, $1 \leq i \leq s$. To do this, we avoid explicitly computing the poset $\mathscr{P}'$, and instead we work directly from the dag $G$.

Suppose the chain $\mathscr{C}_j$ consisted of a single node $u$. We could use, say, directed depth-first traversal of the dag $G$ rooted at $u$ to identify all the nodes $v$ of $G$ reachable from $u$. We maintain an $s$-element table $\rho[\cdot]$,

| | |
|---|---|
| 1 | Initially, unmark all nodes, and set $R[v_{k+1}, i] = \top$ for $i = 1, \ldots, s$. |
| 2 | For $\ell = k, k-1, \ldots, 1$, do |
| 3 | Initialize $\rho[i] := R[v_{\ell+1}, i]$ for $i = 1, \ldots, s$. |
| 4 | Call the recursive procedure $VISIT(v_\ell)$. |
| 5 | Copy $R[v_\ell, i] := \rho[i]$ for $i = 1, \ldots, s$. |

FIG. 2. The algorithm PROCESS-CHAIN for filling out the earliest-entry table.

initialized to $\top$. Whenever we visit a node $v$ such that chain$[v] = i \neq 0$, we compare rank$[v]$ to rank$[\rho[i]]$; if it is less, we set $\rho[i] := v$. When the traversal is finished, $\rho[\cdot]$ will be the minimum node of $\mathscr{C}_i \cup \{\top\}$ reachable from $u$.

To achieve $O(m)$ time, depth-first search marks each node as it visits it, and avoids visiting a node that has previously been marked. Thus it avoids redundant search. To process a chain $\mathscr{C}_j = (v_1 < v_2 < \cdots < v_k)$ consisting of more than one node, we imitate this idea. We start with a depth-first traversal rooted at $v_k$, and visit all nodes reachable from $v_k$, maintaining our table $\rho[\cdot]$. Next, we continue by performing a depth-first traversal rooted at $v_{k-1}$ *without first removing the marks placed on nodes by the first depth-first traversal.* Thus in the second traversal we avoid visiting nodes already visited in the first. Since $v_k$ is reachable from $v_{k-1}$, every node visited in the first traversal is reachable from $v_{k-1}$. The information we need about these nodes is conveniently summarized in the table $\rho[\cdot]$, so there is no reason to visit them again. The process continues in this fashion, considering the nodes of the chain $\mathscr{C}_j$ in reverse order.

We now give a more formal description. Let $\mathscr{C}_j$ be the chain $v_1 < \cdots < v_k$. For notational convenience, let $v_{k+1}$ denote $\top$. The algorithm for processing the chain $\mathscr{C}_j$ appears in Fig. 2. The subroutine VISIT$(v)$ is shown in Fig. 3.

The time required by the above algorithm is dominated by the time for step 4. Use of the marks to truncate the depth-first search ensures that no

| | |
|---|---|
| V1 | Mark $v$ as having been visited. |
| V2 | If $chain(v) = i \neq 0$ and $rank[v] < rank[\rho[i]]$, then set $\rho[i] := v$. |
| V3 | For each outgoing arc $(v, w)$, if $w$ is not already marked, call VISIT$(w)$. |

FIG. 3. The recursive subroutine VISIT$(v)$ for visiting reachable nodes.

arc is explored more than once during the execution of the above procedure. Hence the total time spent in step 4 is $O(m)$.

We now consider correctness of the procedure. For $i = k, k-1, \ldots, 1$, let $W_i$ be the set of nodes $w$ visited during iteration $i$ of step 4. (Let $W_{k+1} = \{\top\}$). It can be shown by a simple backwards induction on $l \leq k+1$ that the set of nodes reachable from $v_l$ is $W_l \cup W_{l+1} \cup \cdots \cup W_{k+1}$, and that for $i = 1, \ldots, s$, $R[v_l, i]$ is the minimum node $w$ in $\mathscr{C}_i \cup \{\top\}$ reachable from $v_l$.

## 3. Answering a Query

### 3.1. *Using the Earliest-Entry Table*

In this section, we give a parallel method for answering a reachability query for a poset with a small chain cover, using the earliest-entry table defined in Subsection 2.3. In particular, we consider the poset $\mathscr{P}'$ with a chain cover consisting of $s$ chains $\mathscr{C}_1, \ldots, \mathscr{C}_s$ of size at most $\lceil n/s \rceil$.

Consider a single chain $\mathscr{C}_1$ and a subset $S$ of the nodes of $\mathscr{C}_1$. The nodes of $\mathscr{C}_1$ reachable from nodes in $S$ are exactly the nodes $v \geq x$, where $x$ is the minimum node in $S$. Thus for purposes of reachability, we may as well replace the entire set $S$ with the single node $x$.

More generally, suppose $S$ is a subset of the nodes of chains $\mathscr{C}_1, \ldots, \mathscr{C}_s$. Let $x_i$ be the minimum node in $S \cap \mathscr{C}_i$ for $i = 1, \ldots, s$. We may as well replace the set $S$ with the $s$-element set $\{x_1, \ldots, x_s\}$. Since each chain $\mathscr{C}_i$ has size at most $\lceil n/s \rceil$, the node $x_i$ can easily be found in $O(n/s)$ time by scanning up the chain $\mathscr{C}_i$ in order until a node of $S$ is encountered. Hence the set $\{x_1, \ldots, x_s\}$ can be found in $O(n/s)$ time using $p \geq s$ processors.

Now for each chain $\mathscr{C}_j$, the minimum node $y_j$ reachable from any of the nodes $x_1, \ldots, x_s$ is just the minimum of the earliest entries of $x_1, \ldots, x_s$ into $\mathscr{C}_j$. That is, using the earliest-entry table, we have $y_j = \min\{R[x_i, j]: i = 1, \ldots, s\}$. We can find the minimum in each chain $\mathscr{C}_j$ in a two-step process. First we mark all the earliest entries of nodes $x_i$ into chains $\mathscr{C}_j$. Since there are $s^2$ pairs $(x_i, \mathscr{C}_j)$, this step requires a total of $O(s^2)$ operations. We evenly divide the set of such pairs among the $p$ processors and let each processor be responsible for $\lceil s^2/p \rceil$ marking operations.

In the second step, we scan up each chain $\mathscr{C}_j$ until a marked node is encountered; the first marked node encountered in $\mathscr{C}_j$ is designated $y_j$. Since each chain has $O(n/s)$ elements, scanning a single chain takes $O(n/s)$ time. In order that the scans all take place in parallel, we first assign each chain to a processor; since $s \leq p$, enough processors are available. In summary, the set $\{y_1, \ldots, y_s\}$ can be found in $O(s^2/p + n/s)$ time using $p$ processors.

Finally, to find the set $T_j$ of all reachable nodes in the chain $\mathscr{C}_j$, we let $T_j$ be the set of nodes in the chain that have rank at least that of $y_j$. These nodes can be marked by yet another scan up the chain (starting at $y_j$) taking $O(n/s)$ time. The set $T = T_1 \cup \cdots \cup T_s$ of all reachable nodes can be determined in $O(n/s)$ time using $p$ processors.

The total time for the above procedure is $O(s^2/p + n/s)$ using $p$ processors. We now consider correctness. Suppose $v \in \mathscr{C}_i$ is a node of $S$ and a node $w \in \mathscr{C}_j$ is reachable from $v$. By definition of $x_i$, $x_i \leq v$. Since $v \leq w$, certainly $x_i \leq w$, so $x_i$'s earliest entry $R[x_i, j]$ into $\mathscr{C}_j$ is $\leq w$. Hence the minimum earliest entry $y_j$ is $\leq w$, so $w \in T_j$. This argument shows that $T$ contains all nodes reachable from nodes in $S$.

### 3.2. *Solving the Directed Reachability Problem on a Union of Antichains*

In Subsection 2.2 we showed that the nodes of a dag $G$ could be decomposed into a set $\mathscr{A}$ and a poset $\mathscr{P}'$ such that $\mathscr{A}$ consists of at most $n/s$ antichains, and the poset $\mathscr{P}'$ consists of at most $s$ chains. In Subsection 3.1, we described a parallel solution to the directed reachability problem for $\mathscr{P}'$. In this section, we consider the same problem as applied to the subgraph $\hat{G}$ of $G$ induced by $\mathscr{A}$.

Observe that a directed path cannot contain two nodes in a single antichain. Since $\hat{G}$ consists of at most $n/s$ antichains, any directed path in $\hat{G}$ contains at most $n/s$ nodes. Recall the method of "parallel breadth-first search" sketched in the introduction: start by marking the nodes in $S$; in each stage, consider the arcs leaving newly marked nodes and mark the nodes they enter. If there is a path in $\hat{G}$ from $x$ to $y$, the path contains at most $n/s$ nodes, so $n/s$ stages of parallel breadth-first search are sufficient to identify all reachable nodes.

Let $m_i$ be the number of arcs considered in stage $i$, for $i = 1, \ldots, n/s$. Each arc is considered at most once (immediately after the node it leaves has been marked), so $\Sigma_i m_i$ is no more than $m$, the number of arcs in $G$. The number of operations needed to carry out stage $i$ is $O(m_i)$. With $p$ processors available, therefore, stage $i$ can be carried out in $O(\lceil m_i/p \rceil)$ time. Summing over all stages yields $O(m/p + n/s)$ parallel time to solve the directed reachability problem on $\hat{G}$.

One might object that the above analysis has omitted the time to assign arcs to processors in a way that balances the work. In fact, we can carry out such load-balancing during the preprocessing. Using the notion of *ranks* outlined in Subsection 2.2, we can choose an assignment of arcs to processors that will work regardless of the choice of $S$. I describe this assignment below. The key to the assignment is the following property of

partitioning into ranks: if a rank $i$ node is reachable from another node, it is reachable via a path of length at most $i$.

Let $M_i$ be the set of arcs leaving nodes of rank $i$, for $i = 1, \ldots, n/s - 1$. During preprocessing, we evenly divide up the arcs of $M_i$ among the $p$ processors, for each $i$. Now to answer a query, we first mark the nodes of $S$ and then carry out a modified version of breadth-first search in which, in stage $i$, we consider the set $M_i$ of arcs. Each processor considers one by one the arcs of $M_i$ assigned to it; for each such arc, the processor marks the arc's head if the arc's tail is already marked. It is easy to verify that, after stage $i$, the set of marked nodes consists of nodes of $S$, together with all nodes of rank $\leq i$ reachable from nodes of $S$.

### 3.3. *Combining the Techniques*

In this subsection, we observe that the techniques used in Subsections 3.1 and 3.2 may be combined to yield a solution for the dag $G$. Suppose we have found an antichain–chain decomposition $A, \mathscr{C}_1, \ldots, \mathscr{C}_s$ of $G$ as described in Subsection 2.2, we have preprocessed the poset $\mathscr{P}'$ induced by the chains, forming the earliest-entry table, and we have preprocessed the graph $\hat{G}$ induced on the antichains $\mathscr{A}$, choosing an assignment of arcs to processors.

Suppose we are given a set $S$ of sources to process. We write $S = S^a \cup S^c$, where $S^a$ consists of "antichain" nodes, *i.e.*, nodes in $\mathscr{A}$, and $S^c$ consists of "chain nodes," nodes appearing in some $\mathscr{C}_i$. The algorithm consists of an "antichain" step, a "chain" step, and a final "antichain" step, with transitional steps in between:

*Step* 1 (antichain step). Apply the technique of Subsection 3.2 to find the set $T_1$ of nodes reachable from $S^a$ via paths entirely in $\hat{G}$.

*Step* 2 (transitional step). Consider the set of arcs from antichain nodes to chain nodes, marking each chain node that is the head of an arc whose tail is an antichain node in $T_1$. Let $S'$ be the set of chain nodes thus marked.

*Step* 3 (chain step). Apply the technique of Subsection 3.1 to find the set $T_2$ of nodes reachable from $S^c \cup S'$ (i.e., $T_2 = \{v \in \mathscr{P}' : u < v$ for some $u \in S^c \cup S'\}$).

*Step* 4 (transitional step). Consider the set of arcs from chain nodes to antichain nodes, marking each antichain node that is the head of an arc whose tail is a chain node in $T_2$. Let $S''$ be the set of antichain nodes thus marked.

*Step* 5 (antichain step). Apply the technique of Subsection 3.2 to $S''$, yielding a set $T_3$ of antichain nodes. We return the set $T = T_1 \cup T_2 \cup T_3$.

We now consider the correctness of the above procedure for computing the set $T$ of targets from the given set $S$ of sources. It is easy to see that every node in $T$ is in fact reachable from a node in $S$. To see that every reachable node $w$ is in $T$, let $v_1 v_2 \ldots v_k = w$ be any directed path in $G$, where $v_1$ belongs to $S$. If this path consists entirely of nodes of $\mathscr{A}$, then $w$ is in $T_1$. Suppose therefore that the path contains at least one node not in $\mathscr{A}$; let $v_i$ be the first such node, and let $v_j$ be the last. If $i = 1$ then $v_i \in S^c$. If $i > 1$ then $v_{i-1} \in T_1$ after step 1, so $v_i \in S'$ after step 2. In either case $v_i \in S^c \cup S'$. Since $v_i < v_j$ in the poset $\mathscr{P}'$ induced on the chain nodes, we have that $v_j \in T_2$ after step 3. If $v_j = w$, we are done; otherwise, step 4 ensures that $v_{j+1} \in S''$, and hence step 5 ensures that $w \in T_3$.

Next we consider the complexity of the procedure. Steps 1 and 5 take $O(m/p + n/s)$ time using $p$ processors. Each of steps 2 and 4 is essentially a single stage of parallel breadth-first search and hence takes $O(m/p)$ time. Step 3 takes $O(n/s + s^2/p)$ time using $p$ processors. (Recall that $p \geq s$.) Hence the procedure takes $O(m/p + n/s + s^2/p)$ time using $p$ processors.

In order that the procedure take time $O(m/p)$, we need to choose $s$ so that $n/s = O(m/p)$ and $s^2 \leq m$. We can achieve this when $p \leq m^{1.5}/n$ by choosing $s = (n-1)p/m$. With this valid of $s$, the storage needed for the results of preprocessing is $O(m + ns)$, or equivalently $O(m + n^2 p/m)$, which is optimal when $p \leq (m/n)^2$.

We evidently have a greater range of choice for our speedup $p$ when the graph is denser. This is not surprising; both of the algorithms described in the Introduction, computing transitive closure and parallel breadth-first search, make better use of parallelism when the input graph is very dense. In contrast to these two algorithms, however, our algorithm can achieve optimal speedup of $O(\sqrt{n})$ no matter how sparse the graphs (although for very sparse graphs, the storage is not optimal). Moreover, for even moderately sparse graphs, say $m \approx n^{1.3}$, we can achieve optimal speedup by a factor of $\approx n^{.95}$—and optimal storage as well if the speedup is no more than $n^{.6}$.

## 4. Concluding Remarks

We have presented a method for representing a directed graph in a way that permits fast parallel answers to reachability queries. Preprocessing is proposed as a way of coping with our current inability to efficiently solve the general, unpreprocessed problem in parallel.

The method can be extended to handle reachability in a graph slightly different from the original, preprocessed graph $G$. That is, given a set $S$ of

sources, a set $E$ of arcs to be added, and a set $F$ of arcs to be removed, we can in $O(m/p + |E| + |F|)$ parallel time find the set $T$ of nodes reachable from nodes of $S$ in the graph $G \cup E - F$.

This extension to handling a slightly modified graph suggests the following open question: can updates to the graph representation be efficiently carried out in parallel? That is, can the representation be dynamically modified?

We have presented a hybrid algorithm for the directed reachability problem. Hybrid algorithms may exist for other problems as well. One candidate is 0–1 node-weighted matching: Given a graph $G$ and a subset $S$ of its nodes, find a matching in $G$ that saturates as many of the nodes of $S$ as possible. We observe that using the Gallai–Edmonds Structure Theorem, one can reduce the problem to the special case in which $G$ is bipartite and has positive surplus [8].

## ACKNOWLEDGMENTS

Thanks to David Shmoys for advising me during this research. Thanks to Ramesh Patil for suggesting the problem. Thanks to Mike Saks for noting that Corollary 2.1, which was originally proved using Dilworth's theorem, could be proved using the dual of Dilworth's theorem, thereby simplifying the preprocessing. Thanks to Bruce Maggs for a helpful discussion.

## REFERENCES

1. M. AIGNER, "Combinatorial Theory," Springer-Verlag, New York, 1979.
2. R. P. DILWORTH, A decomposition theorem for partially ordered sets, *Ann. of Math.* **51** (1950), 161–166.
3. L. R. FORD, JR. AND D. R. FULKERSON, "Flows in Networks," Princeton Univ. Press, Princeton, NJ, 1962.
4. G. GAMBOSI, J. NEŠETŘIL, AND M. TALAMO, Posets, Boolean representations, and quick path searching, *in* "Proceedings, ICALP 87" (T. Ottman, Ed.), Lecture Notes in Computer Science, Vol. 267, pp. 404–424, Springer-Verlag, New York, 1987.
5. HIRAGUCHI, On the dimension of orders, *Sci. Rep. Kanazawa Univ.* **4** (1955), 1–20.
6. H. V. JAGADISH, A compressed transitive closure technique for efficient fixed-point query processing, manuscript, AT&T Bell Labs, Murray Hill, NJ, 1987.
7. D. KELLY AND W. T. TROTTER, Dimension theory for posets, *in* "Ordered Sets: Proceedings of the NATO Advanced Study Institute held at Banff, Canada, 1981" (I. Rival, Ed.), pp. 171–211, Reidel, Boston, 1982.
8. L. LOVÁSZ AND M. D. PLUMMER, "Matching Theory," *Akad.* Kiadó, Budapest, 1986.
9. R. TARJAN, Depth-first search and linear graph algorithms, *SIAM J. Comput.* **1**, No. 2 (1972), 146–160.