Parallel Depth — First Search in Directed Graphs Imperative Multi-way Streams

Alok Aggarwal
IBM Research Division
T. J. Watson Center, Box 218
Yorktown Heights, NY 10598

Richard J. Anderson
Dept. of Computer Science
University of Washington
Seattle, WA 98195

and

Ming Y. Kao
Department of Computer Science
Indiana University
Bloomington, IN 47405

TECHNICAL REPORT N0. 264

Parallel Depth–First Search
In General Directed Graphs

By

Alok Aggarwal, Richard J. Anderson & Ming Y. Kao

September 1988

Parallel Depth-First Search in General Directed Graphs

Alok Aggarwal
IBM Research Division
T.J. Watson Center, Box 218
Yorktown Heights, NY 10598

Richard J. Anderson*
Department of Computer Science
University of Washington
Seattle, WA 98195

Ming-Yang Kao
Department of Computer Science
Indiana University
Bloomington, IN 47405

September 3, 1988

Abstract

A directed cycle separator of an n-vertex directed graph is a simple directed cycle such that when the vertices of the cycle are deleted, the resulting graph has no strongly connected component with more than $n/2$ vertices. This paper shows that the problem of finding a directed cycle separator is in randomized NC. The paper also proves that computing cycle separators and conducting depth-first search in directed graphs are deterministic NC-equivalent. These two results together yield the first RNC algorithm for depth-first search in directed graphs.

*Supported by an NSF Presidential Young Investigator Award

1 Introduction

Depth-first search is one of the most useful tools in graph theory [Tar72] [AHU74]. The *depth-first search problem* is: given a graph and a distinguished vertex, construct a tree that corresponds to performing depth-first search of the graph starting from the given vertex. In the setting of parallel computation, this problem has been studied by a number of authors [Rei85] [GB84] [Ram87] [Zha86] [Smi86] [Sha88] [HY88] [JK88] [And87] [AA88] [Kao88] [EA77] [RC78] [Tiw86] [SV85]. Reif [Rei85] shows that lexicographic depth-first search is P-complete even for general undirected graphs. Ghosh and Bhattacharjee [GB84] provide an NC algorithm for lexicographic depth-first search in acyclic directed graphs; their algorithm has an error and is corrected by Zhang [Zha86] and independently by Ramachandran [Ram87]. Smith [Smi86] gives the first NC algorithm for depth-first search in planar undirected graphs; Shannon [Sha88], He and Yesha [HY88], Ja'Ja and Kosaraju [JK88] independently reduce the processor complexity to linear. Anderson [And87] provides an RNC algorithm to find a maximal path of a general undirected graph; a maximal path is the first branch of a depth-first search tree. Aggarwal and Anderson [AA88] give an RNC depth-first search algorithm for general undirected graphs. Kao [Kao88] provides a deterministic NC algorithm for depth-first search in planar directed graphs. In this paper, we give the first RNC algorithm for depth-first search in *general directed* graphs.

Our general directed depth-first search algorithm uses a divide-and-conquer strategy similar to that used by Aggarwal and Anderson for general undirected depth-first search [AA88]. In addition to this strategy, a crucial idea used in the paper is that of *directed cycle separators*; this idea is originally introduced by Kao for planar directed depth-first search [Kao88]. At the very highest level, our algorithm finds and removes a portion of a depth-first search tree of a given directed graph; the algorithm then recurses on strongly connected components as well as certain weakly connected subgraphs of the resulting graph; to limit the depth of recursion, directed cycle separators are used to divide the given graph into small pieces. While the undirected and directed depth-first search algorithms have similar structures, directed graphs require more work. For instance, a major difference between the two algorithms arises in the construction of separators. Both of the algorithms construct separators by repeatedly joining paths until only a single path remains. However, a key idea in the undirected case is to carry out bisection by traversing and joining the longer half of a path; in the directed case, it is not possible to choose the direction of traversal. Therefore, joining directed paths requires a more sophisticated idea. Another major difference is in the application of the separator that allows the problem decomposition. In the undirected case, once a path separator is given, the decomposition of the graph is almost immediate. In the directed case, however, while the removal of a directed cycle separator makes all resulting strongly connected components reasonably small, the resulting graph may still have large weakly connected subgraphs. In order to successfully divide the graph for the recursive calls, certain weakly connected subgraphs also must be reduced, which requires a fair amount of work.

The parallel computation model used in this paper is the EREW PRAM model, i.e., no two processors are allowed to simultaneously read from or write into the same memory cell. Many of our complexity results are expressed in terms of $MM(n)$, which denotes the sequential time, currently $O(n^{2.376})$, for multiplying two $n \times n$ integer matrices in Strassen's model [AHU74] [CW87]. This paper is organized as follows. Section 2 introduces the concept of directed cycle separators and discusses a number of preliminary results. Section 3 gives the first major result of this paper, establishing a deterministic NC-equivalence between finding directed cycle separators and performing directed depth-first search. Section 4 describes an NC reduction from the problem of finding a directed cycle separator to a particular kind of a matching problem. Section 5 combines these results together, estimates time complexity and processor bounds, and discusses open problems and extensions.

2 Directed Cycle Separators

A *separator* of a graph is a subgraph whose removal disconnects the graph into small pieces. Most of the works on parallel depth-first search rely on finding some form of graph separator. The general undirected depth-first search algorithm [AA88] uses path separators. The planar undirected depth-first search algorithms [Smi86] [Sha88] [HY88] [JK88] employ undirected cycle separators. The planar directed depth-first search algorithm [Kao88] uses directed cycle separators and other kinds of separators in vertex-weighted graphs. This paper follows the directed separator definition given by Kao [Kao88]: a *separator* of an n-vertex directed graph G is a set of vertices S such that $G - S$ has no strongly connected component with more than $n/2$ vertices. A *directed path separator* is a vertex-simple directed path whose vertices form a separator; a *directed cycle separator* is a vertex-simple directed cycle whose vertices form a separator. A single vertex is considered a cycle of length zero; thus, if the removal of a vertex separates a graph, the vertex is a cycle separator. Kao [Kao88] has shown that every directed graph has a directed path separator and a directed cycle separator. Furthermore, such a path separator is computable in linear sequential time, and such a cycle separator is computable within a $\log n$ factor of the optimal linear sequential time. Here we modify his proof and obtain the optimal sequential time bound:

Observation 1 *Every directed graph has a directed cycle separator. Such a separator can be found in $O(n + e)$ sequential time for any directed graph of n vertices and e arcs.*

Proof. Because every directed graph has a directed path separator and such a path separator can be found in linear time [Kao88], it suffices to show that any directed path separator can be converted into a directed cycle separator in linear time. In the following discussion we describe such a conversion in two steps.

Let G be a directed graph of n vertices. For a subgraph S and a vertex v in S, let $R_{in}(v, S)$ (or $R_{out}(v, S)$) denote the set of vertices reachable *to* (or respectively, *from*) v through directed paths in S. A directed path separator $P = x_1, ..., x_p$ is called *semi-minimal* if $R_{out}(x_p, G - \{x_1, ..., x_{p-1}\})$ has more than $n/2$ vertices and $R_{in}(x_1, G - \{x_2, ..., x_p\})$ also has more than $n/2$ vertices. Given such a P, a directed cycle separator can be built in linear time as follows. There are two cases: $p = 1$ and $p > 1$. If $p = 1$, then x_1 alone forms a cycle separator. If $p > 1$, then because both $R_{in}(x_1, G - \{x_2, ..., x_p\})$ and $R_{out}(x_p, G - \{x_1, ..., x_{p-1}\})$ have more than $n/2$ vertices, the two sets share at least one common vertex. Consequently, there is vertex-simple directed path P' from x_p to x_1 such that P' is completely in the two sets. Because the two sets and P share only x_1 and x_p, P' and P form a vertex-simple directed cycle. Because P is already a separator, the cycle is a directed cycle separator. This step takes linear time because P' can be found in linear time.

To finish the proof, we show how to cut any directed path separator $Q = y_1, ..., y_q$ into a semi-minimal one in linear time as follows. The idea is that if $R_{out}(y_q, G - \{y_1, ..., y_{q-1}\})$ has no more than $n/2$ vertices, then $Q' = y_1, ..., y_{q-1}$ is still a path separator. Otherwise, let C be the strongly connected component in $G - Q'$ such that C contains more than $n/2$ vertices. Because Q is a separator, C must contain y_q. This implies that C is a subset of $R_{out}(y_q, G - \{y_1, ..., y_{q-1}\})$, which is a contradiction. We can extend the above idea: if t is the largest index such that $R_{out}(y_t, G - \{y_1, ..., y_{t-1}\})$ has more than $n/2$ vertices, then $Q'' = y_1, ..., y_t$ is still a path separator. The index t can be identified easily in linear time by using the following recurrence formula. Let $R_{out,i}$ denote $R_{out}(y_i, G - \{y_1, ..., y_{i-1}\})$ for $i = 1, \ldots, q-1$. Then $R_{out,i} = R_{out,i+1} \cup R_{out}(y_i, G - R_{out,i+1} - \{y_1, ..., y_{i-1}\})$. After t is found, we perform the same computation on Q''

at the other end by computing R_{in}. After both ends of Q are processed, we have a semi-minimal directed path separator. Since each end of Q can be cut in linear time, the whole process takes linear time. ∎

3 Using Cycle Separators to Conduct Depth-First Search

Kao [Kao88] has also shown that given a directed depth-first search forest, finding a directed path separator is in deterministic NC, and given a directed path separator, finding a directed cycle separator is also in deterministic NC. In this section, we will show that given an oracle for computing a directed cycle separator, conducting directed depth-first search is in deterministic NC. These results immediately imply the following theorem.

Theorem 2 *For general directed graphs, computing a directed path separator, computing a directed cycle separator, and conducting directed depth-first search are deterministically NC-equivalent.*

We now discuss how to use directed cycle separators to conduct directed depth-first search in parallel. Suppose that we want to perform depth-first search in an n-vertex directed graph G starting from some vertex r. Any such search will visit exactly the vertices reachable from r using directed paths; we call a graph *rooted* at a vertex if the vertex can reach all other vertices through directed paths. We assume, for the moment, that G is rooted at r and our goal is to build a directed depth-first search spanning tree rooted at r for G; we will recursively construct, in parallel, such a tree using directed cycle separators.

We first explain why the straightforward recursive approach used in the undirected case [Smi86] [AA88] does not work for directed graphs. Given a directed cycle separator, we can efficiently build a directed path separator P_r starting from r by finding a directed path from r to the cycle separator; the path and the cycle separator form an directed path separator with a certain arc on the cycle removed. The path separator P_r will be a branch of the final depth-first search tree for the directed graph G. Now let G' be the remaining graph that is not searched by P_r; in other words, $G' = G - P_r$. Suppose that we continue to search G' starting from a vertex r' which is not in P_r but is the end vertex of an arc starting from the last vertex of P_r. This time we recurse on the subgraph $G'_{r'}$ that consists of all the vertices reachable from r' using directed paths in G'. Because P_r is a separator of G, every strongly connected component of G' has at most $\lfloor n/2 \rfloor$ vertices. However, $G'_{r'}$ may contain several such strongly connected components; consequently $G'_{r'}$ may still be too large for small depth recursion. To avoid this problem, we describe below a more sophisticated subroutine that removes a set of directed paths from G such that the remaining directed graph has small rooted subgraphs; these removed paths will form a subtree in the final depth-first search tree.

A *partial* depth-first search tree in a rooted directed graph is a subtree of a depth-first search tree such that the graph and two trees are rooted at the same vertex. Let T be a partial depth-first search tree of G; let $x_1, x_2, ..., x_t$ be the vertices of T listed in the depth-first search *last-visit* order, i.e., if this partial depth-first search tree is traversed sequentially, then x_i is visited right after all its descendants are visited. For any x_i, let $y_{i,1}, ..., y_{i,k_i}$ be the vertices which are not in T but are the end vertices of the arcs starting from x_i; the order of $y_{i,1}, ..., y_{i,k_i}$ is arbitrary, and a y vertex may have several different indices if it is adjacent from several x vertices. For a directed graph D and a vertex $x \in D$, let $R(x, D)$ denote the set of vertices that can be reached from x using directed paths in D. We call a subgraph of G a *dangling subgraph*, denoted by $DSG((i,j),T)$, with respect to (i,j) and T if it is formed by the vertices in $G - T$ that are reachable from $y_{i,j}$ but not from $y_{i',j'}$ for any (i',j') such that either $i' < i$ or ($i' = i$ and $j' < j$). In other words, $DSG((i,j),T) = R(y_{i,j}, G - T) - \cup_{(i',j')} R(y_{i',j'}, G - T)$, where either $i' < i$ or ($i' = i$ and $j' < j$).

Observe that a depth-first search tree of G is simply the union of the arcs in T, the set of arcs $(x_i, y_{i,j})$ for which $DSG((i,j),T)$ is non-empty, and an arbitrary depth-first search tree rooted at $y_{i,j}$ for each non-empty $DSG((i,j),T)$. Also observe that because the dangling subgraphs are disjoint, we can simultaneously compute an arbitrary depth-first search tree for each non-empty dangling subgraph. These observations together provide a natural way to recursively and concurrently extend a partial depth-first search into a complete depth-first search tree.

To achieve small depth recursion, the non-empty dangling subgraphs have to be small. Keeping this in view, we call a dangling subgraph *heavy* if it has more than $\lfloor n/2 \rfloor$ vertices; otherwise we call it *light*. Similarly, we call a partial depth-first search tree *heavy* if it has a heavy dangling subgraph; otherwise we call it *light*. If a partial depth-first search tree is light, the recursion can be readily applied to its non-empty dangling subgraphs. So we may assume that the tree is heavy. Because all dangling subgraphs are vertex-disjoint, the tree has exactly one heavy dangling subgraph; denote this subgraph by $DSG((i_o, j_o), T)$. Let H denote the rooted acyclic directed graph induced by *contracting* the strongly connected components of $DSG((i_o, j_o), T)$; further, let a vertex in H be assigned the weight equal to the number of vertices in the corresponding strongly connected component. Since $DSG((i_o, j_o), T)$ is heavy and since H is acyclic and rooted, there exists a vertex in H such that the total weight of this vertex and its descendants is greater than $\lfloor n/2 \rfloor$ but the weight of each of its children and the weights of this child's descendants sum up to at most $\lfloor n/2 \rfloor$. Call such a vertex a *splitting vertex*, and call the corresponding strongly connected component in $DSG((i_o, j_o), T)$ a *splitting component*. There may be several splitting vertices and splitting components. Now pick an *arbitrary* splitting vertex s, and denote its corresponding splitting component by G_s. Next use the given oracle to obtain a cycle separator C_s in G_s. Further build an arbitrary vertex-simple directed path P_o in $DSG((i_o, j_o), T)$ that goes from y_{i_o, j_o} to an arbitrary vertex of C_w and then traverses C_s except its last arc. P_o will be a branch of the final depth-first search tree of G; more precisely, let the new tree be $T' = T \cup \{(x_{i_o}, y_{i_o,j_o})\} \cup P_o$. Observe that T' is still a partial depth-first search tree of G. If T' has no heavy dangling subgraph, then we have achieved our goal of building a light partial depth-first search tree. So we may assume that T'' has a heavy dangling subgraph.

Lemma 3 *If T' is a heavy partial depth-first search tree, then every splitting component of T' is a strongly connected component of $G_s - P_o$, and consequently, consists of at most $|G_s|/2$ vertices.*

Proof. To locate the heavy dangling subgraph of T', observe that $DSG((x_o, y_o), T)$ is the union of $G_s \cap P_o$ and the dangling subgraphs of T' rooted at vertices that are both in $G_s - P_o$ and adjacent from $G_s \cap P_o$. This guarantees that the heavy dangling subgraph of T' is rooted at some vertex that is both in $G_s - P_o$ and adjacent from $G_s \cap P_o$. To further locate the splitting components of T', recall that from definition, G_s corresponds to a vertex s in H such that the weight of each child of s and the weights of this child's descendants sum up to at most $\lfloor n/2 \rfloor$. This implies that $G_s - P_o$ must include all splitting components of T'. Therefore, every splitting component of T' is a strongly connected components of $G_s - P_o$. Furthermore because P_o contains the cycle separator C_s of G_s, every splitting component of T' has at most $|G_s|/2$ vertices. ■

From the above lemma, it is readily seen that after $O(\log n)$ such phases of cutting up splitting components, any resulting splitting component is left with a single vertex. If we now extend the partial depth-first search tree to include the vertex of an arbitrary splitting component, we can obtain another partial depth-first search tree T' which is light so that recursion can be performed.

We now summarize the above discussion. To construct a depth-first search tree, we start from the partial depth-first search tree T that consists of only the root r. It takes $O(\log n)$ cuts of splitting components to extend T into a light partial depth-first search tree; each cut takes an oracle call to find a cycle separator in addition to $O(\log^2 n)$ time and $MM(n)$ processors for computing transitive closures of suitable graphs with at most n vertices. To build a complete depth-first search tree, we recurse on the non-empty dangling subgraphs of T; this recursion has depth $O(\log n)$. So if a directed cycle separator of an n-vertex strongly connected directed graph can be computed in $T_c(n)$ parallel time using $P_c(n)$ processors, then a depth-first search tree of any n-vertex rooted directed graph can be computed in $O(\log^2 n(T_c(n)+\log^2 n))$ time using $P_c(n)+MM(n)$ processors.

The above discussion applies only to rooted directed graphs. We now extend the discussion to any general directed graph G. Our goal is to find a depth-first search spanning forest with r being the first root. We first decompose G into rooted directed graphs as follows. Arrange the vertices of G in an arbitrary order $u_1, ..., u_n$ with $u_1 = r$. Let $D(u_i)$ be the set of vertices that can be visited by depth-first search starting from u_i but cannot be visited starting from $u_1, ..., u_{i-1}$; in other words, $D(u_i) = R(u_i, G) - \cup_{j<i} R(u_j, G)$. Clearly, each non-empty $D(u_i)$ is a directed graph rooted at u_i. Now a complete depth-first search of G starting from r can be conducted by computing an arbitrary depth-first search tree for each non-empty $D(u_i)$ with the root being u_i; these non-empty $D(u_i)$'s can be computed in $O(\log^2 n)$ time and $MM(n)$ processors. Hence, from this discussion, we obtain the following theorem:

Theorem 4 *Suppose a directed cycle separator of any n-vertex strongly connected directed graph can be computed in $T_c(n)$ time using $P_c(n)$ processors. Then a depth-first search spanning forest of any n-vertex general directed graph can be computed in $O(\log^2 n(T_c(n) + \log^2 n))$ time using $P_c(n) + MM(n)$ processors.*

4 Constructing a Directed Cycle Separator

A *directed multi-path separator* is a set of vertex-disjoint vertex-simple directed paths whose vertices form a separator. For obtaining a directed cycle separator of any n-vertex directed graph, in Subsection 4.1 we describe a routine REDUCE that given a directed multi-path separator Ω of more than $10\lceil \log n \rceil + 20$ paths, reduces the number of paths in Ω by at least one half so that the resulting set is still a directed multi-path separator. Initially Ω is the directed multi-path separator that consists of any $\lceil n/2 \rceil$ vertices in G, each vertex being a directed path of length 0. Since each call to REDUCE decreases the size of Ω to at most one half of its original size, $O(\log n)$ calls are sufficient to reduce the size of Ω to at most $10\lceil \log n \rceil + 20$. Once Ω has at most $10\lceil \log n \rceil + 20$ paths, we will merge the paths, one by one, into a single directed path separator, and then convert this path separator into a directed cycle separator; these last two steps are described in Subsection 4.2.

In the following discussion, the *lower segment* of a directed path $P = x_1, ..., x_k$ refers to the directed path $x_1, ..., x_{\lfloor k/2 \rfloor}$; the *upper segment* of P refers to the directed path $x_{\lfloor k/2 \rfloor + 1}, ..., x_k$. For a set Ω of m directed paths, we use $|\Omega| = m$ to denote the number of paths in Ω. In unambiguous cases, we often refer to a directed path when, in fact, we mean the vertices of that directed path. For example, $G - P$ denotes the induced subgraph where all the vertices contained in P are removed, and $G - \Omega$ denotes the induced subgraph where all the vertices contained in the paths of Ω are removed.

4.1 Reducing the Number of Paths While Maintaining the Separator Property

As highlighted in the Introduction, the basic idea used in the routine REDUCE is similar to that used by Aggarwal and Anderson [AA88] for general undirected depth-first search. The undirected depth-first search algorithm constructs a path separator by repeatedly joining paths until only a single path remains. A key idea in the undirected case is to carry out bisection by traversing and joining the longer half of a path. This bisection idea is not applicable in the directed case because it is not possible to choose the direction of traversal in directed paths. One of the new ideas for the directed case is that given two paths L and S, we find a directed path P that goes from L to the *lower* segment of S so that we can always traverse the longer half of S. To make this idea work, the directed REDUCE uses several other new ideas; we describe these ideas in the following discussion.

REDUCE takes as input a multi-path separator Ω of m paths with $m > 10\lceil\log n\rceil + 20$. REDUCE will operate for $O(\log^2 n)$ phases to reduce the size of Ω by at least one half so that the resulting set still forms a directed multi-path separator. At the beginning of REDUCE, Ω is partitioned into two sets Δ and Γ. The set Γ is further partitioned into two sets, the set of *active* paths and that of *inactive* paths; denote these two subsets by Γ_a and Γ_{in} respectively. In each phase of REDUCE, these five sets is modified as follows. For notational brevity, let $\alpha = \lfloor\frac{m}{2\lceil\log n\rceil+4}\rfloor$. At the very beginning of the routine, $|\Gamma| = \alpha$, $\Gamma_a = \Gamma$, $|\Gamma_{in}| = 0$, and $|\Delta| = m - \alpha$; at the very end of the routine, $|\Gamma_a| \leq \alpha$, $|\Gamma_{in}| \leq (\lceil\log n\rceil + 1)\cdot\alpha$, and $|\Delta| = 0$. To achieve this reduction, in each phase we find a set of paths, Π, that joins paths in Ω in a suitable manner. (See Figure 1.) More precisely, let Δ_u and Δ_l denote the sets of, respectively, upper segments and lower segments of the paths in Δ. Π is a maximal set of vertex-disjoint vertex-simple directed paths that go from Γ_a to Δ_l such

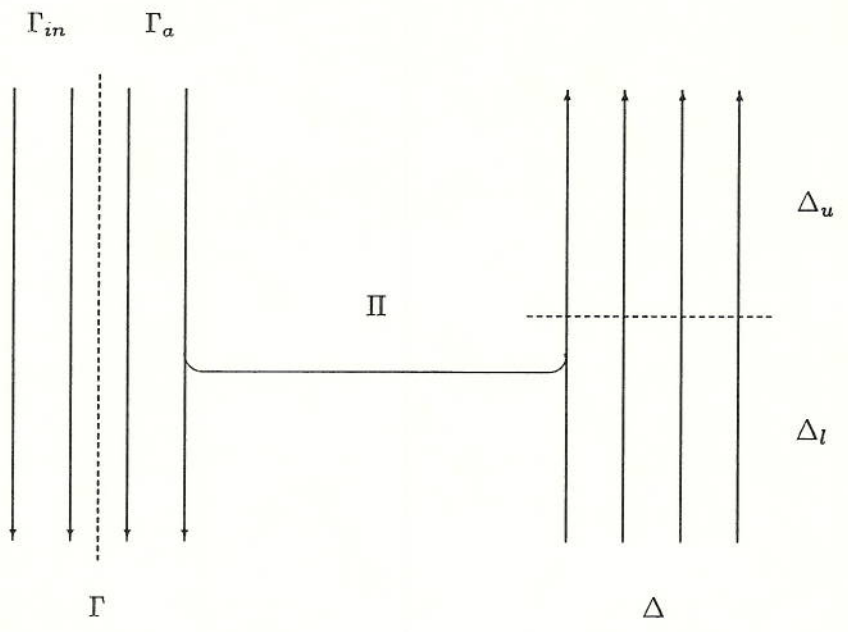

Figure 1: Π joins Γ and Δ.

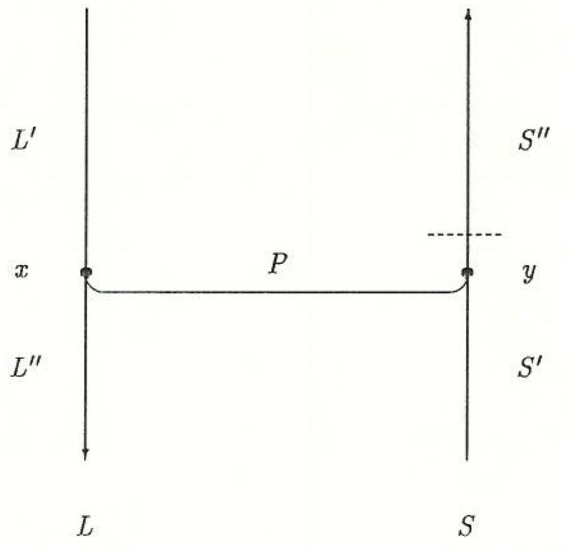

Figure 2: Rearrange two paths.

that for each directed path P in Π, the first vertex belongs to a path in Γ_a, the last vertex belongs to a path in Δ_l, and the interior vertices are taken from $G - \Omega$. Each directed path in $\Gamma_a \cup \Delta_l$ contains an end vertex of at most one path of Π; of course, there may be directed paths in $\Gamma_a \cup \Delta_l$ that do not contain any end vertices of the paths in Π. Below we describe the basic step in using Π to rearrange paths in Ω. (See Figure 2.) Suppose that a directed path $P \in \Pi$ joins directed paths $L \in \Gamma_a$ and $S \in \Delta$, and that P has end vertices x and y where $L = L'xL''$ and $S = S'yS''$. In each phase of REDUCE, L is replaced by $L'PS''$, and S is replaced by S'. The path L'' is either added to Γ_{in} or is discarded from Ω; the conditions under which L'' is discarded from Ω and those under which L'' is added to Γ_{in} will be discussed later. Irrespective of whether L'' is discarded from Ω or added to Γ_{in}, note that the directed path S has been reduced to half its original length because S' is only a subpath of the lower segment of the original S. Furthermore, the paths in Γ_a and those in Δ do not increase in number; in fact, the number of paths in Γ_a and Δ may have decreased if some paths of Π have been joined to the lowest end vertices in Δ_l. However, the number of paths in Γ_{in} may increase, which may, in turn, lead to an increase in the total number of paths in $\Gamma = \Gamma_a \cup \Gamma_{in}$. If the size of Δ does not decrease, then the increase in $|\Gamma|$ can increase the number of paths in $\Omega = \Gamma \cup \Delta$. We have mentioned above that Π is a maximal set, and in the following discussion, we will specify another property of Π that will help in eventually reducing the number of directed paths in Ω rather than increasing it.

We introduce two kinds of notations to describe a more detailed picture of how Π is used to reduce the cardinality of Ω (see Figure 3): (1) $\Gamma_{a,new}$ denotes the set of all paths of the kind $L'PS''$. Γ_a^* denotes the set of all paths of the kind L''. Δ_{new} denotes the set of all paths of the kind S'. (2) $\hat{\Gamma}_a$ denotes the set of all paths in Γ_a that are not connected by any paths in Π to any paths in Δ_l. $\hat{\Delta}$ denotes the set of all paths in Δ whose lower segments are not connected by any paths in

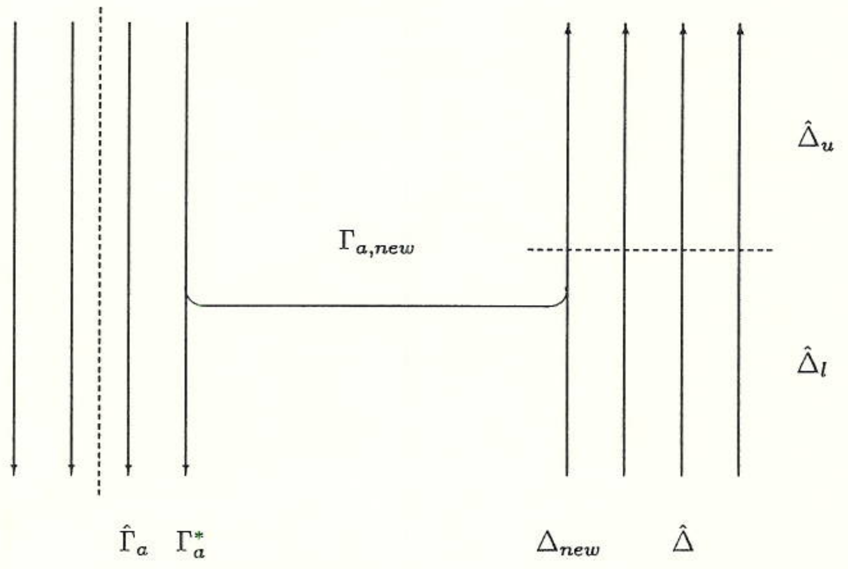

Figure 3: Notations for the subroutine Reduce.

Π to any paths in Γ_a. Also $\hat{\Delta}_l$ and $\hat{\Delta}_u$ denote the sets of, respectively, lower segments and upper segments of paths in $\hat{\Delta}$. We can now specify the additional property for Π: Π *is such that there are no directed paths from* $\hat{\Gamma}_a \cup \Gamma_a^*$ *to* $\hat{\Delta}_l$ *using vertices in* $G - \Omega - \Pi$. Clearly Π is a maximal set and we call it a *maximal joining set from* Γ *to* Δ. Later in this section we will discuss how to compute such Π; here we continue to explain how Π is used to reduce the cardinality of Ω. Let $scc(D)$ denote the size of the largest strongly connected component in any directed graph D. The properties of Π and the fact that $\Pi \cup \Omega$ is a separator of G imply $\Gamma_a^* \cup \hat{\Gamma}_a$ and $\hat{\Delta}_l$ cannot have vertices in the same strongly connected component of $(G - \Omega - \Pi) \cup (\Gamma_a^* \cup \hat{\Gamma}_a) \cup \hat{\Delta}_l$. This in turn implies either $scc((G - \Omega - \Pi) \cup (\Gamma_a^* \cup \hat{\Gamma}_a)) \leq \lfloor n/2 \rfloor$, or $scc((G - \Omega - \Pi) \cup \hat{\Delta}_l) \leq \lfloor n/2 \rfloor$, or both. Based on these bounds, we have two cases for updating the sets Δ, Γ_a, and Γ_{in}; the new version of these sets will be used in the next phase of Reduce.

Case 1: If $scc((G - \Omega - \Pi) \cup \hat{\Delta}_l) \leq \lfloor n/2 \rfloor$, we add Π to Ω but discard $\hat{\Delta}_l$ from Ω, and observe that the new Ω is still a directed multi-path separator. Moreover, we perform the following replacements:

- $\Delta \leftarrow \Delta_{new} \cup \hat{\Delta}_u$,
- $\Gamma_a \leftarrow \Gamma_{a,new} \cup \hat{\Gamma}_a$, and
- $\Gamma_{in} \leftarrow \Gamma_{in} \cup \Gamma_a^*$.

Case 2: If $scc((G - \Omega - \Pi) \cup (\Gamma_a^* \cup \hat{\Gamma}_a)) \leq \lfloor n/2 \rfloor$, we add Π to Ω but discard $\Gamma_a^* \cup \hat{\Gamma}_a$, and notice that the new Ω is still a directed multi-path separator. The number of paths in Γ_a may drop below α, which is the size of Γ_a at the very beginning of REDUCE. To restore this size, we take enough paths from $\Delta_{new} \cup \hat{\Delta}$ and add them to Γ_a until Γ_a is restored to its original cardinality or $\Delta_{new} \cup \hat{\Delta}$ is exhausted. Let Δ_a denote the set of paths taken from $\Delta_{new} \cup \hat{\Delta}$. Now we employ the following replacements:

- $\Delta \leftarrow \Delta_{new} \cup \hat{\Delta} - \Delta_a$,
- $\Gamma_a = \Gamma_{a,new} \cup \Delta_a$, and
- Γ_{in} remains unchanged.

(See Figure 4 for a brief summary of the routine REDUCE.)

Lemma 5 *If we have a directed multi-path separator of m paths with $m > 10\lceil \log n \rceil + 20$, then after executing $O(\log^2 n)$ phases of REDUCE, we can obtain a directed multi-path separator with at most $\lfloor \frac{m}{2} \rfloor$ paths.*

Proof. First of all, observe that Ω remains a directed multi-path separator after the replacements in each phase. Consequently, at the end of the routine, Ω is still a directed multi-path separator.

```
routine REDUCE($\Omega$):
    Partition $\Omega$ into $\Delta$ and $\Gamma$, and further partition $\Gamma$ into $\Gamma_a$ and $\Gamma_{in}$;
    while $\Delta$ is not empty do
        begin
            Find a maximal joining set $\Pi$ from $\Gamma$ to $\Delta$;
            If $scc((G - \Omega - \Pi) \cup \hat{\Delta}_l) \leq \lfloor n/2 \rfloor$
                then
                    begin
                        $\Delta \leftarrow \Delta_{new} \cup \hat{\Delta}_u$;
                        $\Gamma_a \leftarrow \Gamma_{a,new} \cup \hat{\Gamma}_a$;
                        $\Gamma_{in} \leftarrow \Gamma_{in} \cup \Gamma_a^*$
                    end
                else {comment: $scc((G - \Omega - \Pi) \cup (\Gamma_a^* \cup \hat{\Gamma}_a)) \leq \lfloor n/2 \rfloor$}
                    begin
                        $\Delta \leftarrow \Delta_{new} \cup \hat{\Delta} - \Delta_a$;
                        $\Gamma_a = \Gamma_{a,new} \cup \Delta_a$;
                        $\Gamma_{in}$ remains unchanged
                    end
        end
```

Figure 4: A brief summary of REDUCE

To reduce the size of Ω, REDUCE is executed until Δ becomes empty. In the following discussion, we will first discuss the situations under which Δ gets exhausted. We will then estimate $|\Gamma_a|$, $|\Gamma_{in}|$, and $|\Omega| = |\Gamma_a| + |\Gamma_{in}| + |\Delta|$ as at the end of REDUCE.

We divide case 2 given above into two cases: case 2a: $|\Gamma_a^*| \geq \frac{1}{2} \cdot \alpha$, and case 2b: $|\Gamma_a^*| < \frac{1}{2} \cdot \alpha$. Also, let t_1, t_{2a}, and t_{2b}, respectively, denote the numbers of phases in which cases 1, 2a, and 2b occur. In all three cases, $|\Delta|$ never increases and may sometimes decrease. This decrease happens when a path of Δ is taken to replenish Γ_a in cases 2a and 2b, or when a path of Δ is cut sufficiently many times in cases 1, 2a, and 2b. If a path is cut, it is cut by at least one half because it is replaced either by a subpath of its lower segment or by its upper segment. There are two situations in which Δ may become empty. One situation is that the paths of Δ are primarily cut off. Because a path has at most n vertices, it can allow at most $\lceil \log n \rceil + 1$ cuts; after these cuts, the path becomes empty. In case 1, all paths in Δ are cut; in case 2a, at least $\frac{1}{2} \cdot \alpha$ paths are cut; in case 2b, the number is insignificant for our analysis. Because originally $|\Delta| = m - \alpha$, after $t_1 + t_{2a} + t_{2b}$ phases, Δ is left with a reserve of at most C_1 cuts where $C_1 = (m-\alpha)\cdot(\lceil \log n \rceil + 1) - t_1 \cdot (m - \alpha) - t_{2a} \cdot (\frac{1}{2} \cdot \alpha)$. Some of the paths in Δ may be taken away to replenish Γ_a in cases 2a and 2b; the effect of this happening is canceled out in the estimate. Now, if C_1 is negative or zero, then Δ must be empty. C_1 can be negative or zero if $t_1 = \lceil \log n \rceil + 1$ (referred as condition 1) or if $t_1 < \lceil \log n \rceil + 1$ but $t_{2a} \geq 5 \cdot (\lceil \log n \rceil + 2) \cdot (\lceil \log n \rceil + 1)$ (referred as condition 2a); these conditions are derived under the assumption that $m > 10\lceil \log n \rceil + 20$. The other situation is that the paths of Δ are primarily taken away to replenish Γ_a in cases 2a and 2b. Because $|\Gamma_a| = \alpha$ before the replacements in each phase, at least $\frac{1}{2} \cdot \alpha$ paths are taken away in case 2b; in case 2a, the number is insignificant. After $t_1 + t_{2a} + t_{2b}$ phases, $|\Delta|$ is at most $C_2 = (m - \alpha) - t_{2b} \cdot (\frac{1}{2} \cdot \alpha)$. So if C_2 is negative or zero, then Δ must be empty. C_2 can be negative or zero if $t_1 < \lceil \log n \rceil + 1$ but $t_{2b} \geq 5 \cdot (\lceil \log n \rceil + 2)$ (referred as condition 2b).

Now we estimate $|\Gamma_a|$, $|\Gamma_{in}|$, $|\Gamma| = |\Gamma_a| + |\Gamma_{in}|$, and $|\Omega| = |\Gamma| + |\Delta|$. Originally $|\Gamma_a| = \alpha$. $|\Gamma_a|$ does not change in case 1 and does not increase in case 2a and 2b. So $|\Gamma_a| \leq \alpha$ throughout the execution of REDUCE. Because $|\Gamma_{in}|$ may increase by at most $|\Gamma_a| \leq \alpha$ in case 1 and does not change in cases 2a and 2b, after $t_1 + t_{2a} + t_{2b}$ phases, $|\Gamma_{in}| \leq t_1 \cdot \alpha$ and consequently, $|\Gamma| \leq \alpha + t_1 \cdot \alpha$. Furthermore, $|\Gamma| \leq \lfloor \frac{m}{2} \rfloor$ if any one of the conditions 1, 2a, and 2b is true. REDUCE can achieve at least one of these three conditions within $(\lceil \log n \rceil + 1) + \{5 \cdot (\lceil \log n \rceil + 2) \cdot (\lceil \log n \rceil + 1)\} + \{5 \cdot (\lceil \log n \rceil + 2)\}$ phases. Hence, within $O(\log^2 n)$ phases, $|\Omega| = |\Gamma| \leq \lfloor \frac{m}{2} \rfloor$. ∎

To complete the description of REDUCE, we explain how to compute Π as follows. Recall that Π has to be a maximal joining set from Γ to Δ. Here we give a stronger property for Π. For the previously described path P in the set Π, we assign a cost equal to the length of the cut-off segment of L, namely, the number of vertices in L''. We call Π a *minimum-cost maximum-cardinality joining set* if Π has the maximum number of paths from Γ_a to Δ_l, and if Π also minimizes the total cost under the maximum-cardinality constraint. Below we prove that if Π is a minimum-cost maximum-cardinality joining set from Γ to Δ, then Π is a maximal joining set from Γ to Δ: Because Π is of the maximum-cardinality, there can be no directed path from $\hat{\Gamma}_a$ to $\hat{\Delta}_l$ using vertices in $G - \Omega - \Pi$; if there were such a path, this path could be added to Π and the original Π would not be of the maximum cardinality. Because Π is also of the minimum total cost, there can be no directed path from Γ_a^* to $\hat{\Delta}_l$ using vertices in $G - \Omega - \Pi$; if there were such a path, then a certain path of Π could be replaced by this smaller cost path while Π maintained the same maximum cardinality.

In view of the above discussion, to find a maximal joining set Π from Γ to Δ, we only need to find a minimum-cost maximum-cardinality joining set from Γ to Δ. Aggarwal and Anderson [AA88] have shown how to reduce an undirected version of the problem of finding a minimum-cost maximum-cardinality joining set to that of finding a minimum-weight perfect matching in a bipartite graph. Essentially the same reduction can be used to solve our problem of finding Π. The reader is referred to their paper for details; here we simply state the result as follows: Let $P_{mm}(n)$ and $T_{mm}(n)$ denote the number of processors and the parallel time to compute a minimum-weight perfect matching of any n-vertex bipartite graph that has an integer weight of at most n on each of its arcs. Then a maximal joining set Π from Γ to Δ can be found in exactly the same complexity. We summarize the discussion of this subsection in the following theorem.

Theorem 6 *Let $P_{mm}(n)$ and $T_{mm}(n)$ denote the number of processors and the parallel time to compute a minimum-weight perfect matching of any n-vertex bipartite graph that has an integer weight of at most n on each of its arcs. Then the routine REDUCE can be used to obtain a directed multi-path separator of at most $10\lceil \log n \rceil + 20$ paths in $O(\log^3 n \cdot (T_{mm}(n) + \log^2 n))$ time using $P_{mm}(n) + MM(n)$ processors.*

Proof. Initially we have the directed multi-path separator Ω that consists of any $\lceil n/2 \rceil$ vertices in G, each vertex being a directed path of length 0. From Lemma 5, if $|\Omega| > 10\lceil \log n \rceil + 20$, then a call to REDUCE cuts $|\Omega|$ by at least one half. Therefore, $O(\log n)$ calls to REDUCE are sufficient to obtain a directed multi-path separator of the desired size. Each call has $O(\log^2 n)$ phases. Each phase does two major computations: (1) computing a maximal joining set Π, and (2) computing the strongly connected components of an n-vertex directed graph in $O(\log^2 n)$ time using $MM(n)$ processors. So the total complexity is $O(\log^3 n \cdot (T_{mm}(n) + \log^2 n))$ parallel time and $P_{mm}(n) + MM(n)$ processors. ∎

4.2 Constructing a Cycle Separator from a Small Set of Separating paths

Given a directed multi-path separator Ω with at most $10\lceil \log n \rceil + 20$ paths, we explain below how to construct a directed path separator and convert this path separator into a directed cycle separator. The idea is to repeatedly join two paths of Ω into a new path while the other paths stay untouched and, together with the new path, remain a directed multi-path separator. To join two paths, we first recall that Kao [Kao88] has given an NC algorithm to convert any directed path separator into a directed cycle separator; in fact, the proof of Observation 1 can also be used to do the conversion in parallel. The idea of joining two directed paths into one is almost the same as converting a directed path separator into a directed cycle separator. The only difference is that in the path-to-cycle conversion, we merge the two ends of the given directed path separator while in the path-to-path conversion, we merge the ends of two paths, one end from each path. More precisely, let $\Omega = \{P_1, ..., P_k\}$ with $k \leq 10\lceil \log n \rceil + 20$; also let $\Omega' = \{P_3, ..., P_k\}$. Below, we describe how to merge P_1 and P_2 into a single path P' so that P' and Ω' form a multi-path separator; we will process P_1 and P_2 in essentially the same way as the proof of Observation 1. Let $P_1 = x_1, ..., x_{p_1}$. Find the largest index t such that $R_{out}(x_t, G - \{x_1, ..., x_{t-1}\} - P_2 - \Omega')$ has more than $n/2$ vertices. If t does not exist, then let $P' = P_2$ and observe that P' and Ω' form a multi-path separator. If t exists, then let $P_1' = x_1, ..., x_t$ and notice that P_1', P_2, and Ω' form a multi-path separator. Now let $P_2 = y_1, ..., y_{p_2}$. Find the smallest index s such that $R_{in}(y_s, g - \{y_{s+1}, ..., y_{p_2}\} - P_1' - \Omega')$ has more than $n/2$ vertices. If s does not exist, then let $P' = P_1'$ and observe that P' and Ω' form a multi-path separator. If s exists, then let $P_2' = y_s, ..., y_{p_2}$, and observe that P_1', P_2', and Ω' form a multi-path separator. Now find a vertex-simple directed path Q from x_t to y_s with internal vertices belonging to $G - P_1' - P_2' - \Omega'$. Let P' be the path formed by P_1', Q, and P_2'. It is readily seen that P' and Ω' form a multi-path separator.

After repeating the above process $k-1$ times, we can obtain a directed path separator. We then convert this path separator into a cycle separator. The complexity for merging two paths or converting a path into a cycle is $O(\log^2 n)$ time and $MM(n)$ processors because the only major computation in the merge is to find transitive closure of an appropriate directed graph. Moreover, because Ω has $O(\log n)$ paths, the total complexity of merging Ω into a directed cycle separator is $O(\log^3 n)$ time and $MM(n)$ processors. This discussion and Theorem 6 immediately yield the following theorem.

Theorem 7 *Let $P_{mm}(n)$ and $T_{mm}(n)$ denote the processor and time complexities for computing a minimum-weight perfect matching of any n-vertex bipartite graph with an integer weight of most n on each arc. Then a directed cycle separator of any general n-vertex directed graph can be found in $O(\log^3 n(T_{mm}(n) + \log^2 n))$ time using $P_{mm}(n) + MM(n)$ processors.*

5 Discussions

Using Theorems 4 and 7, we can now state the other main results of this paper:

Theorem 8 *Let $T_{mm}(n)$ and $P_{mm}(n)$ denote the parallel time and the number of processors to compute a minimum-weight perfect matching of any n-vertex bipartite graph with an integer weight of at most n on each arc. Furthermore, let $MM(n)$ denote the sequential time complexity of multiplying two $n \times n$ integer matrices in Strassen's model. Then a depth-first search forest of any general n-vertex directed graph can be computed in $O(\log^5 n(T_{mm}(n) + \log^2 n))$ time using $P_{mm}(n) + MM(n)$ processors.*

Mulmuley, Vazirani, and Vazirani [MVV87] provide an RNC minimum-weight perfect matching algorithm such that $T_{mm}(n) = O(\log^2 n)$ and $P_{mm}(n) = n \cdot MM(n)$. Consequently, the above theorem has the following implication:

Theorem 9 *For any general n-vertex directed graph, a depth-first search forest can be constructed probabilistically in $O(\log^7 n)$ time using $n \cdot MM(n)$ processors.*

Finally, we conclude the paper with two challenging open problems. One problem is to devise a deterministic NC algorithm for general directed depth-first search. For the time being, we can only show that general directed depth-first search can be conducted deterministically in $O(\log^{11} n \cdot \sqrt{n})$ time using $O(n^3)$ processors; the result is obtained from modifying an undirected maximal joining set algorithm by Goldberg, Plotkin, and Vaidya [GPV88]. The other problem is to find a more efficient RNC algorithm for general directed depth-first search; because depth-first search is extremely useful in graph theory, an RNC algorithm with almost linear processor-time product will have significant impacts.

References

[AA88] A. Aggarwal and R.J. Anderson. A random NC algorithm for depth first search. *Combinatorica*, 8:1–12, 1988.

[AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[And87] R.J. Anderson. A parallel algorithm for the maximal path problem. *Combinatorica*, 7:315–326, 1987.

[CW87] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *the Proceedings of the* 19th *Annual ACM Symposium on Theory of Computing*, pages 1–6, 1987.

[EA77] D. Eckstein and D. Alton. Parallel graph processing using depth-first search. In *the Proceedings of the Conference on Theoretical Computer Science at the University of Waterloo*, pages 21–29, 1977.

[GB84] R.K. Ghosh and G.P. Bhattacharjee. A parallel search algorithm for directed acyclic graphs. *BIT*, 24:134–150, 1984.

[GPV88] A. Goldberg, S. Plotkin, and P. Vaidya. Sublinear-time parallel algorithms for matching and related problems, 1988. To appear in the Proceedings of the 29^{th} Annual IEEE Symposium on Foundations of Computer Science.

[HY88] X. He and Y. Yesha. A nearly optimal parallel algorithm for constructing depth first spanning trees in planar graphs. *SIAM Journal on Computing*, 17(3):486–491, June 1988.

[JK88] J. Ja'Ja and S. Kosaraju. Parallel algorithms for planar graphs and related problems. *IEEE Transactions on Circuits and Systems*, March 1988.

[Kao88] M.Y. Kao. All graphs have cycle separators and planar directed depth-first search is in DNC. In *the Proceedings of the 3rd Aegean Workshop on Computing, Corfu, Greece*, pages 53–63, 1988. Springer-Verlag Lecture Notes in Computer Science 319.

[MVV87] K. Mulmuley, U.V. Vazirani, and U.V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–114, 1987.

[Ram87] V. Ramachandran. Fast algorithms for reducible flow graphs. In *the Proceedings of the 1987 Princeton Workshop on Algorithm, Architecture, and Technology Issues for Models of Concurrent Computation, Princeton University*, 1987.

[RC78] E. Reghbati and D. Corneil. Parallel computations in graph theory. *SIAM Journal on Computing*, 7(2):230–237, 1978.

[Rei85] J.H. Reif. Depth-first search is inherently sequential. *Information Processing Letters*, 20:229–234, June 1985.

[Sha88] G.E. Shannon. A linear-processor algorithm for depth-first search in planar graphs, 1988. To appear in Information Processing Letters.

[Smi86] J.R. Smith. Parallel algorithms for depth first searchs I. planar graphs. *SIAM Journal of Computing*, 15(3):814–830, August 1986.

[SV85] C.A. Schevon and J.S. Vitter. A parallel algorithm for recognizing unordered depth-first search. Technical Report 21, Department of Computer Science, Brown University, 1985.

[Tar72] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, June 1972.

[Tiw86] P. Tiwari. An efficient parallel algorithm for shifting the root of a depth first spanning tree. *Journal of Algorithms*, 7(1):105–119, March 1986.

[Zha86] Y. Zhang, 1986. Manuscript.