Information Processing Letters 20 (1985) 229–234
North-Holland

12 June 1985

DEPTH-FIRST SEARCH IS INHERENTLY SEQUENTIAL

John H. REIF *

Center for Research in Computing Technology, Aiken Computation Laboratory, Division of Applied Science, Harvard University, Cambridge, MA 02138, U.S.A.

Communicated by M.A. Harrison
Received 22 November 1983
Revised 13 September 1984

This paper concerns the computational complexity of depth-first search. Suppose we are given a rooted graph G with fixed adjacency lists and vertices u, v. We wish to test if u is first visited before v in depth-first search order of G. We show that this problem, for undirected and directed graphs, is complete in deterministic polynomial time with respect to deterministic log-space reductions. This gives strong evidence that depth-first search ordering can be done neither in deterministic space $(\log n)^c$ nor in parallel time $(\log n)^c$, for any constant $c > 0$.

Keywords: Depth-first search, parallel computation, polynomial time complete

1. Introduction

Depth-first search (DFS) is one of the most versatile sequential algorithm techniques known for solving graph problems. Tarjan [25] and Hopcroft and Tarjan [15] first developed depth-first search algorithms for connected and biconnected components of undirected graphs, and strong components of directed graphs. These algorithms run in linear time on a sequential unit-cost Random Access Machine (RAM). DFS is also used in efficient sequential algorithms for planarity testing [13], bipartite matchings [12], and connectivity [8], among others.

Most recent research is now devoted to the development of parallel algorithms for graph problems. Two fundamental techniques,

(1) breadth-first search, and

(2) probabilistic search by random walks,

form the basis for many of the known efficient parallel graph algorithms. Breadth-first search is used for $O(\log n)^2$ time parallel Random Access Machine (PRAM) algorithms for connectivity, biconnectivity, and minimum spanning trees [17] and planarity testing [18] and a variety of other directed and undirected graph problems. Also, the present author [22] has used the random walk technique of Aleliunas et al. [1] for $O(\log n)$ time probabilistic PRAM algorithms for connected and unconnected components, minimum spanning trees, and a variety of other undirected graph problems. All these parallel algorithms require only a polynomial number of processors.

(U)DFS-ORDER is (informally stated) the following problem: Given a digraph (respectively, undirected graph) G with fixed adjacency lists, fixed starting vertex s, and vertices u and v, test whether u is visited before v in a depth-first search of G starting from s. Suppose DFS-ORDER or UDFS-ORDER could be done in $(\log n)^{O(1)}$ parallel time, for some model of parallel computation. From DFS-ORDER we can quickly compute in parallel the DFS numbering, the DFS spanning tree, and other useful information. Then, perhaps

* This research was supported by the National Science Foundation Grant No. MCS-82-000269 and Office of Naval Research Contract No. N00014-80-C-0647.

many of the known efficient sequential algorithms for graph problems, which use DFS, might be easily implemented in $(\log n)^{O(1)}$ parallel time.

This paper provides strong evidence that DFS cannot be significantly sped up by the use of any reasonable number of parallel processors. [1] In particular we show that given any polynomial-time sequential algorithm A with input of length n, a RAM can construct in $O(\log n)$ space DFS-ORDER and UDFS-ORDER instances each of which has a positive solution exactly when A accepts its input.

Thus, if we had a T(n) parallel time algorithm for DFS-ORDER or UDFS-ORDER, we would have a $T(n)^{O(1)}$ parallel time algorithm for simulating any polynomial time sequential computation. This seems unlikely for $T(n) = (\log n)^{O(1)}$. Hence we have strong evidence that DFS-ORDER and UDFS-ORDER have no $(\log n)^{O(1)}$ parallel time algorithms.

Our reduction from polynomial time sequential computations can also be done in $O(\log n)$ space on a deterministic Turing Machine (TM). Suppose DFS-ORDER or UDFS-ORDER can be done by a deterministic TM with space $(\log n)^{O(1)}$. Then our reduction implies that any sequential polynomial time computation can be simulated in parallel time $(\log n)^{O(1)}$. Again, this seems very unlikely.

This paper is organized as follows: Section 2 defines the DFS-ORDER and UDFS-ORDER problems. Section 3 reviews the relevant complexity theory. Section 4 proves the DFS-ORDER is polynomial time complete. Section 5 concludes our paper.

2. Depth-first search

This section describes exactly the DFS algorithm given by Tarjan [25] and Hopcroft and Tarjan [15]. Let $G = (V, E)$ be a graph with *vertex set* $V = \{1, \ldots, n\}$ and *edge set* E. If G is an *undirected graph* (respectively, *directed*), then E consists of unordered (respectively, ordered) pairs of distinct vertices. For each vertex $v \in V$, we assume a fixed *adjacency list* ADJ(v) of vertices linked by an edge to v. The *root* of G is vertex 1.

DFS begins at the root. On first visiting a vertex v, the search proceeds to the first unvisited vertex appearing in the ordered list ADJ(v), if such a vertex exists. Otherwise, the search is *exhausted at* v, so it proceeds to the last unexhausted vertex visited before v. If no unexhausted previously visited vertex exists, then the search proceeds to the minimum vertex not visited thus far. The search terminates when all vertices have been visited. The *DFS tree edges* are the edges visited which lead to previously unvisited vertices.

To mark and number vertices, we use an integer array *visit*, which is initially 0 at each index, Also, we require an integer counter i, initially 0.

```
begin
  i ← 0
  for each v = 1 to n do visit(v) ← 0
  for each v = 1 to n do DFS(v)
end
```

The recursive procedure DFS numbers the vertices as follows:

```
procedure DFS(v)
  begin
    if visit(v) = 0 then
      begin
        local u
        i ← i + 1
        visit(v) ← i
        for each u ∈ ADJ(v) in given order
          do DFS(u)
      end
  end
```

The sequential RAM time for execution of DFS is $O(|V| + |E|)$, and a parallel execution can decrease this time to at best linear in $|V|$ by parallel examination of the adjacency lists. However, no further speed-up, using less than an exponential

[1] Theoretically, some speed-up is always possible. Dymond and Tompa [7] show that any $T(n) \geqslant n$ time multi-tape Turing machine can be simulated by an $O(\sqrt{T(n)})$ time PRAM, and Reif [23] shows that any $T(n) \geqslant n$ time sequential RAM or probabilistic RAM can be simulated by an $O(\sqrt{T(n) \log T(n)})$ time PRAM. However, both these results require an exponential number of processors.

number of processors, seems possible.

(U)DFS-ORDER is the following problem: Given a directed (respectively, undirected) rooted graph $G = (V, E)$ as represented above, and vertices $u, v \in V$, is u visited before v in DFS?

3. Computational complexity definitions

Let L and L′ be languages over a finite alphabet Σ. We say $L \leqslant_{\log} L'$ (L′ is *log-space reducible* to L) if there exists a function f such that

(a) for each $\omega \in \Sigma^*$, $\omega \in L'$ iff $f(\omega) \in L$,

(b) f is computable in log space by a deterministic TM.

Let P be the class of languages accepted in deterministic polynomial time by TMs. L is P-*complete* if $L \in P$ and, for each $L' \in P$, L′ is log-space reducible to L. It is known (see [16]) that log-space reducibility is a transitive relation. Thus, L is P-complete if L′ is P-complete and L′ is log-space reducible to L.

For the purposes of this paper, we define a *boolean circuit* to be a sequence $B = (B_0, \ldots, B_n)$ where each B_i is either *true, false* or an expression $op(B_{i_1}, B_{i_2})$ where i_1, $i_2 < i$ and op is a binary boolean operation. Let $value(B_i) = B_i$ if B_i is a truth value *true* or *false*, and if $B_i = (B_{i_1}, B_{i_2})$, then let $value(B_i) = op(value(B_{i_1}), value(B_{i_2}))$. Let $value(B) = value(B_n)$.

The *circuit value problem* is: Given a boolean circuit B, test if value(B) = *true*. Ladner [20] has shown that the following theorem holds.

Theorem 3.1. *The circuit value problem is* P-*complete.*

It is easy to show that the following position holds.

Proposition 3.2. *The circuit value problem remains* P-*complete if the circuits are restricted to only the boolean operations*: $B_0 = \underline{true}$ *and*

$$B_i = \neg(B_{i_1} \vee B_{i_2}) \quad \textit{for } i = 1, \ldots, n.$$

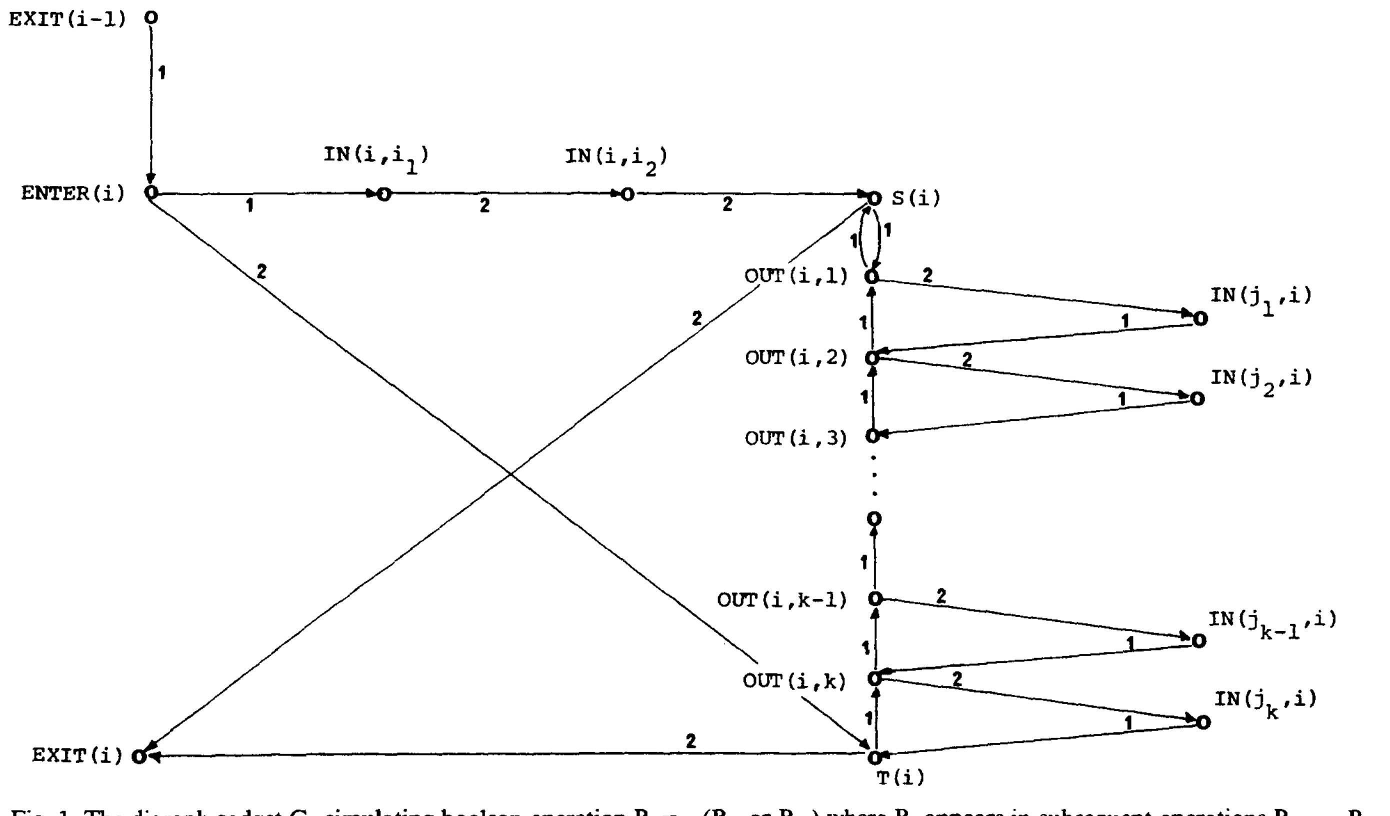

Fig. 1. The digraph gadget G_i simulating boolean operation $B_i = \neg(B_{i_1}$ or $B_{i_2})$ where B_i appears in subsequent operations $B_{j_1}, \ldots, B_{j_k}$.

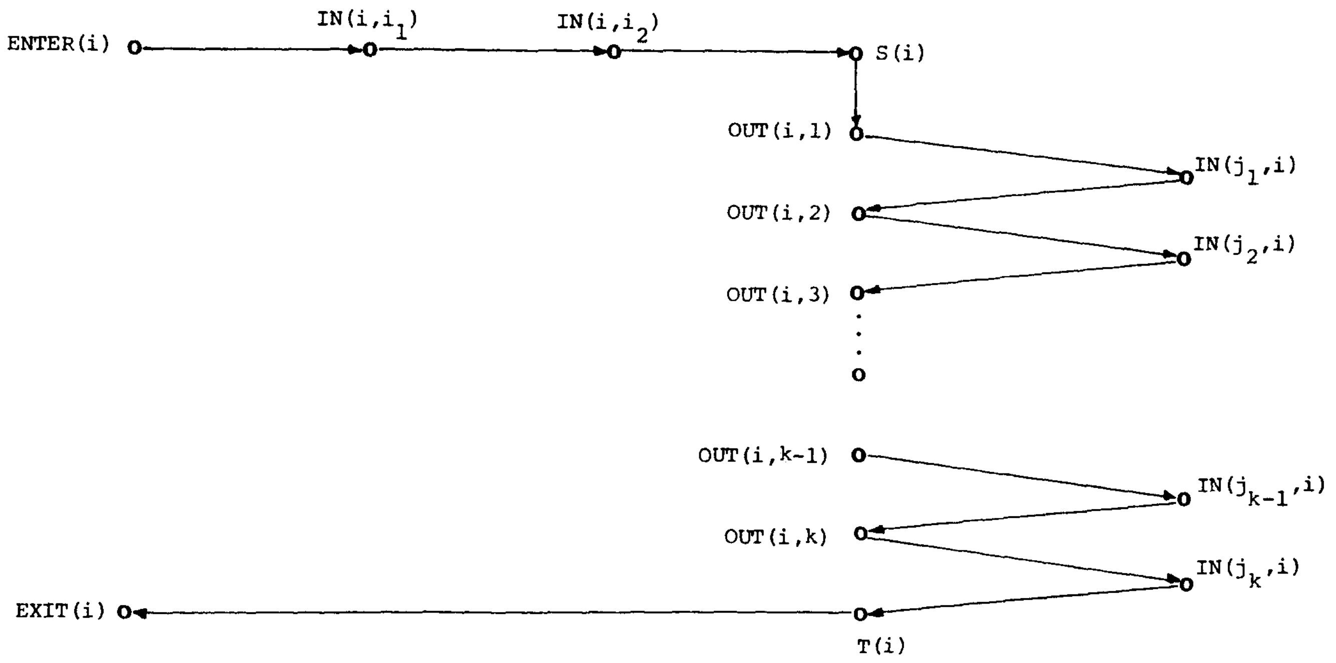

Fig. 2. The depth-first search tree edges in digraph gadget G_i in the case ENTER(i) is visited before both IN(i, i_1) and IN(i, i_2).

4. DFS-order is P-complete

Let $B = (B_0, \ldots, B_n)$ be a boolean circuit. We can assume that $B_0 = true$, and, for each $i = 1, \ldots, n$, $B_i = \neg(B_{i_1} \vee B_{i_2})$. Consider some $i > 0$, where $B_i = \neg(B_{i_1} \vee B_{i_2})$. Let $j_1, \ldots, j_k > i$ be the indices of the subsequent boolean operations $B_{j_1}, \ldots, B_{j_k}$ where B_i appears. We construct in this case a digraph gadget G_i with vertices $V_i = \{$ENTER(i), IN(i, i_1), IN(i, i_2), S(i), OUT(i, 1), ..., OUT(i, u), T(i), EXIT(i)$\}$ and edges as diagrammed in Fig. 1. For each vertex v of G_i, the numbers labeling edges departing v give the order in which these edges appear on the adjacency list of G_i.

Suppose we perform DFS on G_i from vertex ENTER(i), assuming no vertex in G_i has been previously visited. Then, Fig. 2 gives the resulting DFS tree edges. In the case, however, that IN(i, i_1) or IN(i, i_2) has been previously visited, but no other vertices of G_i have been visited, then DFS from vertex ENTER(i) results in the DFS tree edges given in Fig. 3. In either case, all vertices of G_i are visited except possibly IN(i, i_2), and if $i < n$, the DFS proceeds to vertex ENTER(i + 1).

Note that Fig. 1 also gives an edge (EXIT(i − 1), ENTER(i)) between G_{i-1} and G_i, and 2k edges between G_i and $G_{j_1}, \ldots, G_{j_k}$.

Let G be the graph consisting of the union of all these graphs $G_1, \ldots, G_n$ and their connecting edges, as described above. Let the root of G be EXIT(0). We now show the following.

Lemma 4.1. S(n) *is visited before* T(n) *in* G *iff* *value*(B) = *true*.

Proof. Fix an i, $1 \leqslant i \leqslant n$. Suppose we have just visited vertex ENTER(i) for the first time. We assume as our induction hypothesis that for each i′, $1 \leqslant i' \leqslant i$,

(i) if value($B_{i'}$) = *true*, then the DFS tree edges in $G_{i'}$ are as described in Fig. 2, otherwise

(ii) the DFS tree edges of $G_{i'}$ are as described in Fig. 3.

As a consequence, the DFS tree so far constructed consists of a single path of DFS tree edges starting at EXIT(0) and ending at ENTER(i). Furthermore, the only vertices visited before ENTER(i) during DFS have at least one index less than i.

Suppose $B_i = \neg(B_{i_1} \vee B_{i_2})$. The value($B_{i_1}$) = value($B_{i_2}$) = *false*. By the induction hypothesis

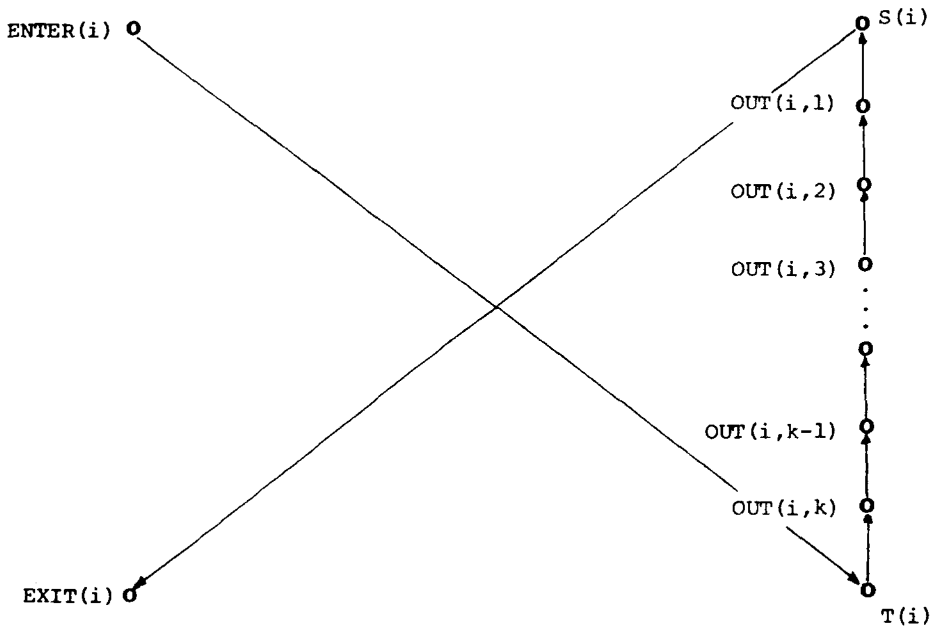

Fig. 3. The depth-first search tree edges in digraph gadget G_i in the case ENTER(i) is visited before IN(i, i_1) but after IN(i, i_2).

IN(i, i_1) and IN(i, i_2) have not been previously visited so the DFS tree edges are as described in Fig. 2.

On the other hand, if value(B_{i_1}) $\vee$ value(B_{i_2}) = *true*, then again by the induction hypothesis, either IN(i, i_1) or IN(i, i_2) has been previously visited so the DFS tree edges are as described in Fig. 3. In the case where $B_i = \neg(B_{i_1} \wedge B_{i_2})$ we can similarly establish (i) and (ii) from the induction hypothesis. □

The construction of digraph G for circuit B can obviously be done by a deterministic TM in O(log n) space (and in fact in O(log n) time by a PRAM). Lemma 4.1 and Proposition 3.2 imply the following theorem.

Theorem 4.2. DFS-*order is* P-*complete.*

Let $G' = (V, E')$ be the undirected graph derived from $G = (V, E)$ by substituting an undirected edge $\{u, v\}$ for each edge (u, v) or $(v, u) \in E$. For each vertex $v \in V$, the edges which in G depart v are ordered in the adjacency list of G′ just as in G, and the edges which in G enter v are ordered arbitrarily in the adjacency list of G′, but higher (i.e., later) than any edges which departed v.

Again, the root of G′ is EXIT(0). It is easy to verify that the DFS spanning tree of G′ is identical to that of G. Hence, we have the following lemma.

Lemma 4.3. S(n) *is visited before* T(n) *in* G′ *iff value*(B) = *true*.

This implies the following theorem.

Theorem 4.4. UDFS-*order is* P-*complete*.

5. Conclusion

There is a growing number of problems shown to be P-complete, including path systems [3], max flow [11], unit resolution [19]. Proofs of P-completeness are useful both to the theory and practice of the design of parallel algorithms. Their use is analogous to the use of NP-completeness proofs.

Suppose one attempts to design a parallel algorithm for a given problem with polynomial number of processors and significant speed-up over known sequential algorithms. If unsuccessful, an alternative line of attack is to show that the problem is deterministic polynomial time complete. Thus, one can at least state that a large number of competent researchers has also tackled this problem and achieved no significant parallel speed-up with a subexponential number of processes.

In fact, our proof of the polynomial time completeness of DFS-ORDER and UDFS-ORDER followed only after unsuccessful attempts to develop a fast parallel algorithm for these problems.

Miklail and Kosaraju [21] posed as an open problem to compute a DFS tree in less than linear time, given a subexponential number of processors. The problem remains open if the DFS search is not constrained to follow edges in the order they appear in the graph's adjacency lists.

References

[1] R. Aleliunas, R.M. Karp, R.H. Lipton, L. Lovasz and C. Rackoff, Random walks, universal traversal sequences, and complexity of maze problems, Proc. 20th Annual Symp. on Foundations of Computer Science (1979) 218–223.

[2] F. Chin, J. Lam and I. Chen, Optimal parallel algorithms for the connected components problem, Foundations of Computer Science (FOCS), 1981.

[3] S.A. Cook, An observation on time-storage trade off, J. Comput. System Sci. 9 (3) (1974) 308–316.

[4] S.A. Cook, Towards a complexity theory of synchronous parallel computation, Presented at: Internat. Symp. über Logik und Algorithmik zu Ehren von Professor Hort Specker, Zürich, Switzerland, 1980.

[5] D. Dobkin, R.J. Lipton and S. Reiss, Linear programming is log-space hard for P, Inform. Process. Lett. 9 (2) (1979) 96–97.

[6] P.W. Dymond, Speed-up of multi-take Turing machines by synchronous parallel machines, Tech. Rept., Dept. of EE and Computer Science, Univ. of California, San Diego, CA.

[7] P.W. Dymond and M. Tompa, Speed-ups of deterministic machines by synchronous parallel machines, Proc. 15th Symp. on Theory of Computing, Boston, MA (1983) 336–346.

[8] S. Even and R.E. Tarjan, Network flow and testing graph connectivity, J. SIAM Comput. 4 (4) (1975) 507–512.

[9] S. Fortune and J.C. Wyllie, Parallelism in random access machines, in: Proc. 10th ACM Symp. on Theory of Computation (1978) 114–118.

[10] L. Goldschlager, A unified approach to models of synchronous parallel machines, in: Proc. 10th Annual ACM Symp. on the Theory of Computing, San Diego, CA (1978) 89–94.

[11] L.M. Goldschlager, R.A. Shaw and J. Staples, The maximum flow problem is log-space complete for P, Theoret. Comput. Sci. 21 (1982) 105–111.

[12] J.E. Hopcroft and R.M. Karp, An $n^{5/2}$ algorithm for maximum matching in bipartite graphs, J. SIAM Comput. 2 (1973) 225–231.

[13] J.E. Hopcroft and R.E. Tarjan, Efficient planarity testing, J. ACM 21 (1974) 549–568.

[14] J.E. Hopcroft and R.E. Tarjan, Dividing a graph into triconnected components, SIAM J. Comput. 2 (3) (1973).

[15] J.E. Hopcroft and R.E. Tarjan, Efficient algorithms for graph manipulation, Comm. ACM 16 (6) (1973) 372–378.

[16] J.E. Hopcroft and J.D. Ullman, Introduction to Automata Theory, Languages and Computation (Addison-Wesley, Reading, MA, 1979).

[17] J. Ja'Ja', Graph connectivity problems on parallel computers, Tech. Rept. GS-78-05, Dept. of Computer Science, Penn. State Univ., PA, 1978.

[18] J. Ja'Ja and J. Simon, Parallel algorithms in graph theory: Planarity testing, SIAM J. Comput. 11 (2) (1982) 372–378.

[19] N.D. Jones and W.T. Laaser, Complete problems for deterministic polynomial time, Theoret. Comput. Sci. 3 (1) (1976) 105–117.

[20] R.E. Ladner, The circuit value problem is log-space complete for P, SIGACT News 7 (1) (1975) 18–20.

[21] A.J. Miklail and S.R. Kosaraju, Graph problems on a mesh-connected processor array, Proc. 14th Annual ACM Symp. on Theory of Computing, San Francisco, CA (1982) 345–353.

[22] J.H. Reif, Symmetric complementation, J. ACM 31 (2) (1984) 401–421.

[23] J.H. Reif, On the power of probabilistic choice in synchronous parallel computations, SIAM J. Comput. 13 (1) (1984) 46–55.

[24] C. Savage and J. Ja'Ja', Fast, efficient parallel algorithms for some graph problems, SIAM J. Comput. 10 (4) (1981) 682–691.

[25] R.E. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Comput. 1 (2) (1972) 146–160.

[26] J.C. Wyllie, The complexity of parallel computations, Ph.D. Thesis and Tech. Rept. 79-387, Dept. of Computer Science, Cornell University, 1979.