JOURNAL OF ALGORITHMS 8, 385–394 (1987)

Problems Complete for Deterministic Logarithmic Space*

STEPHEN A. COOK

Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4

AND

PIERRE MCKENZIE

Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, Succ "A", Montréal, Québec, Canada, H3C 3J7

Received March 24, 1986

We exhibit several problems complete for deterministic logarithmic space under NC^1 (i.e., log depth) reducibility. The list includes breadth-first search and depth-first search of an undirected tree, connectivity of undirected graphs known to be made up of one or more disjoint cycles, undirected graph acyclicity, and several problems related to representing and to operating with permutations of a finite set.

1. INTRODUCTION

Let FL denote the class of problems solvable by a deterministic Turing Machine in logarithmic space (see [HoU179, Co85]), and consider the class NC^1 of all problems solvable in logarithmic depth by a uniform family of Boolean circuits of bounded indegree (see [Pi79, Co85]). It is known that $NC^1 \subseteq FL$ ([FiPi74, Sc76, Bo77]), but proper inclusion has thus far eluded proof. Barrington's recent characterization of NC^1 in terms of bounded-width polynomial-size branching programs [Ba86] revived interest in the relationship between NC^1 and FL (the latter being characterized in terms of polynomial-size branching programs without width restrictions). To our knowledge only two problems have been shown in the literature complete

*Research supported by the Natural Sciences and Engineering Research Council of Canada.

0196-6774/87 $3.00

for FL under NC^1 (i.e., uniform logarithmic depth) reducibility: the "cycle-free problem" for undirected graphs and the "graph accessibility problem" for directed graphs of outdegree one ([Ho80, Jo75, HaImMa78], see [Co81]). Likely, these problems do not belong to NC^1 (otherwise $NC^1 = FL$). The purpose of this paper is to exhibit other problems complete for FL under NC^1 reducibility.

We follow [Co85] and let FL denote the class of *problems* (as opposed to just sets) solvable in deterministic log space. Here a *problem* is a multiple valued function, and the problem is solved by giving any one of its possible values for a given input. For example, the Depth-First Search problem is solved for a given input tree by listing the nodes of the tree in any order consistent with the recursive paradigm: "visit a node and then visit each of its subtrees." Similarly NC^1 is the class of problems (in the above sense) solvable in uniform log depth. Also NC^1 reducibility is defined using uniform families of log depth circuits with oracle nodes. These notions are as defined in [Co85], except for the type of uniformity, which will not be discussed here. It is clear in all cases that the circuit families presented here are log space uniform (see [Ru81]), and that criterion is sufficient for our purposes of classifying problems in FL as either complete for FL or in the subclass NC^1. Our circuit families in fact almost certainly meet the more stringent uniformity conditions introduced in [Ru81] and used in [Co85].

Consider a permutation π of the set $\Omega = \{1, \ldots, n\}$. π can be represented *pointwise* (as $\pi(1), \ldots, \pi(n)$) or as a product of *disjoint cycles*. For example, the permutation of $\{1, 2, 3, 4, 5\}$ whose pointwise representation is

$$4, 5, 1, 3, 2$$

has

$$(143)(25)$$

as its lexicographically first disjoint cycle representation. If σ is another permutation of the set Ω, we define the product $\pi\sigma$ as the permutation whose pointwise representation is $\sigma(\pi(1)), \ldots, \sigma(\pi(n))$ (hence we read products from left to right). The problem of transforming a given permutation from its disjoint cycle to its pointwise representation belongs to NC^1, but the reverse transformation appears to require logarithmic space. This motivated our investigation of the problems discussed here.

2. Construction

We describe a construction which is then used repeatedly. Let $G = (V, E)$ be an undirected graph in which the edges incident with each node are

assigned an order. We construct from G two permutations, denoted π_G and σ_G, which permute labels assigned to the edges in the graph. (We assign two labels to each edge, corresponding to each "end" of the edge.) Say $w \in V$ has degree $d(w)$; then the "edge ends incident with w" are labelled, in their assigned order, $w_1, \ldots, w_{d(w)}$ (these labels will be referred to as *the labels incident with* w). Once each edge is assigned its two labels, we define π_G as the permutation having disjoint cycle representation

$$\prod_{w \in V} (w_1 w_2 \cdots w_{d(w)}) \tag{1}$$

and we define σ_G as the permutation having disjoint cycle representation

$$\prod_{e \in E} (e_+ e_-), \tag{2}$$

where e_+ and e_- denote the two labels assigned to edge $e \in E$.

EXAMPLE. Let $G = (V, E)$ with $V = \{a, b, c, d, f\}$ and $E = \{ab, bc, bf, cd\}$. A possible result of the construction is $\pi_G = (a_1)(b_1 b_2 b_3)(c_1 c_2)(d_1)(f_1)$ and $\sigma_G = (a_1 b_1)(b_2 c_1)(b_3 f_1)(c_2 d_1)$.

PROPOSITION 1. *Let $G = (V, E)$ be an acyclic undirected graph with no isolated node and let permutations π_G and σ_G be constructed as above. Then the number of connected components in G is equal to the number of cycles in the disjoint cycle representation of the product $\pi_G \sigma_G$.*

Proof. No product of the permutations π_G and σ_G can mix labels pertaining to distinct connected components of G, so there are at least as many cycles in the permutation as components of G. The reverse inequality is proved by showing that the given permutation corresponds to a depth-first traversal of each component of the graph. More precisely, let H be a connected component of G and let γ be a cycle in $\pi_G \sigma_G$ which contains a label w_i incident with some node w in H. Suppose that $\pi_G(w_i) = w_j$ and $\sigma_G(w_j) = v_k$, where (w, v) is an edge in G whose edge end v_k is incident with v. Let H_v^w be the subtree of H rooted at v consisting of the v-component of the result of removing edge (w, v) from H. Then there is a smallest positive integer b such that $(\pi_G \sigma_G)^b(w_i) = w_j$, and H_v^w consists precisely of those nodes u such that $(\pi_G \sigma_G)^c(w_i) = u_l$ for some integer c with $1 \le c < b$ and some edge end u_l incident with u. This fact is proved for all edges (w, v) by induction on the size of H_v^w. The fact implies that γ "hits" all nodes in H. □

Observe that, given an undirected graph G, both the pointwise and the disjoint cycle representations of the permutations π_G and σ_G can be constructed in NC^1. Moreover, the pointwise representation of the product

$\pi_G\sigma_G$ (or of the product of any fixed number of permutations for that matter) can be computed in NC^1.

3. Completeness for *FL*

THEOREM 2. *The following problems are complete for FL under* NC^1 *reducibility (whenever relevant, "points" are elements of* $\Omega = \{1, \ldots, n\}$ *and permutations are permutations of* Ω*; we assume any reasonable encoding of the inputs over the binary alphabet (see* [*Mc*84])):

DCA (Disjoint Cycle Accessibility)
Given: permutation presented pointwise and points a and b.
Problem: determine whether a and b belong to the same cycle in the disjoint cycle representation of the permutation.
Remark: remains complete when the given permutation has precisely two disjoint cycles.

DCR (Disjoint Cycle Representation)
Given: permutation presented pointwise.
Problem: compute the disjoint cycle representation of the permutation.
Remark: remains complete when the given permutation consists of a single cycle.

SCP (Single Cycle Permutation)
Given: permutation presented pointwise.
Problem: determine whether the permutation consists of a single cycle.

DFA (Directed Forest Accessibility)
Given: acyclic directed graph of outdegree zero or one, nodes u and v.
Problem: determine whether there is a directed path from u to v.
Remark: remains complete when the given graph has exactly two weakly connected components.

UFA (Undirected Forest Accessibility)
Given: acyclic undirected graph, nodes u and v.
Problem: determine whether there is a path between u and v.
Remark: remains complete when the given graph has exactly two connected components.

CFP (Cycle-Free Problem) [Co81]
Given: undirected graph.
Problem: determine whether the graph is acyclic.
Remark: remains complete when the given graph contains at most one cycle.

DFS (Depth-First Search)
Given: rooted undirected tree.
Problem: compute a depth-first traversal of the tree starting at the

root (i.e., list the nodes of the tree in any order consistent with the recursive paradigm: "visit a node and then visit each of its subtrees").

BFS (Breadth-First Search)
Given: rooted undirected tree.
Problem: compute a breadth-first search of the tree starting at the root (i.e., list the nodes in some order such that nodes farther from the root come after nearer nodes).

PP2 (Permutation Product)
Given: permutations g and h represented as disjoint cycles.
Problem: compute the disjoint cycle representation of gh.
Remark: belongs to NC^1 when only the pointwise representation of gh is required.

PP (Permutation Product)
Given: permutations $g_1, \ldots, g_r$ represented as disjoint cycles.
Problem: compute the disjoint cycle representation of the product $g_1 \cdots g_r$.
Remark: remains complete when only the pointwise representation of the product is required.

PPOW (Permutation Powering)
Given: permutation g presented pointwise and integer m in binary notation.
Problem: compute the pointwise representation of g^m.
Remark: remains complete when m is in unary or when the problem is to compute the image of a given point under g^m; belongs to NC^1 when g is presented as disjoint cycles.

Proof. First we show that DFA is NC^1-hard for FL. Then we exhibit a number of NC^1 reductions (" $\leq$ " stands for "NC^1-reduces to"):

$$\text{DFA} \leq \text{UFA} \leq \text{DCA} \leq \text{SCP},$$
$$\text{DFA} \leq \text{DFS} \leq \text{DCR},$$
$$\text{DFA} \leq \text{BFS},$$
$$\text{UFA} \leq \text{CFP} \leq \text{PP2},$$
$$\text{UFA} \leq \text{PPOW},$$

and

$$\text{UFA} \leq \text{PP}.$$

This will show that each of the problems in question is NC^1-hard for FL. To show that the problems are in FL, it suffices to show that the end points of the reduction chains (namely SCP, DCR, BFS, PP, and PPOW) are each in FL. (To be accurate, our reduction DCA $\leq$ SCP only applies to the special case DCA with two cycles, but it is clear that DCA in general is in FL.) It is straightforward to show that SCP, DCR, and PP are in FL. To

show that BFS (Breadth-First Search) is in FL, for each i enumerate all nodes at level i using a depth-first search. As for PPOW, [Mc84] (and also [McCo85]), contains a proof that Permutation Powering belongs to NC^1 provided the permutation is represented as disjoint cycles; but then DCR is in FL.

DFA *is* NC^1*-Hard for* FL

It suffices to show hardness for $L = \text{DSPACE}(\log n)$ [Co85]. Jones [Jo75] proved that the graph accessibility problem for directed graphs of outdegree one is hard for L by observing that the set of configurations of a Turing machine with a fixed input w forms the vertices of such a graph G, and the (unique) accepting configuration is accessible from the initial configuration iff the Turing machine accepts the input w. In the present proof we need only add that the graph G can be made acyclic using the time stamp idea used in the proof that the cycle free problem is hard for FL [Co81]. The idea is to associate a time stamp with each node, and insist that an edge always joins a configuration at time t to a configuration at time $t + 1$. If T is an upper bound on the computation time of the Turing machine with input w, then we let the node v in the DFA instance be the accepting configuration with time stamp T, and u will be the initial configuration with time stamp 0. In order to make sure that the graph has precisely two weakly connected components, we create a "sink node" s, and create an edge from every node of outdegree 0 except v to s. The edge leading out from v should be removed, if it exists. Thus s and v are the only two nodes of outdegree 0 and represent the two components.

DFA $\leq$ UFA

First we remove the only edge out of node v (if any) in the DFA instance, on the grounds that no path from u to v can use this edge. The outdegree of v is now zero, so viewing each edge in the resulting digraph as undirected yields an undirected forest with the following property: u and v belong to the same tree if and only if a directed path existed from u to v in the original digraph. Note that if in the original directed forest v has outdegree 0 and there are precisely two weakly connected components (as was the case in the proof that DFA is hard for FL), then the resulting undirected forest has precisely two components.

UFA $\leq$ DCA

Let G be the undirected graph instance of UFA (assuming that neither u nor v is isolated). First we construct the permutations π_G and σ_G corresponding to G (see construction). Then (in NC^1) we compute the pointwise

representation of the product $\pi_G\sigma_G$. By the proof of Proposition 1, the labels u_1 and v_1 occur on the same cycle of this permutation if and only if u and v belong to the same component of G. Note that by Proposition 1, the permutation has precisely two cycles if G has precisely two components.

DCA (*with two cycles*) $\leq$ SCP

Assume that the DCA instance has exactly two disjoint cycles. Modify the permutation by sending a to b and the predecessor of b to the successor of a. The resulting permutation has a single cycle iff a and b are on different cycles in the original permutation.

DFA $\leq$ DFS

Starting from the DFA instance, we define a rooted undirected tree as follows. As in the reduction DFA $\leq$ UFA, we first remove the edge (if any) out of node v and we view each edge in the resulting digraph as undirected. Then we add a node r (for "root"), and, for each node w such that $w = v$ or w had outdegree zero in the original digraph, we join r and w with an undirected edge. This yields an undirected tree with the property that node u belongs to the subtree of r rooted at v if and only if a path existed from u to v in the original digraph.

To complete the reduction to DFS, consider any depth-first traversal of the tree starting at r: u belongs to the subtree of r rooted at v if and only if u is visited after v and no immediate son of r is visited after v but before u (neglecting the trivial case in which u has outdegree zero in the DFA instance). The reader can verify that constructing the tree and checking that no immediate son of r appears between v and u in the depth-first list of the nodes (computed by a DFS oracle) can be performed in NC^1.

DFA $\leq$ BFS

Starting from the DFA instance construct an undirected tree exactly as in the reduction above, except let the tree be rooted at u instead of r. Then a path from u to v exists in the original digraph iff in the tree constructed the distance from u to v is less than the distance from u to r, and hence iff v is enumerated before r by the BFS oracle applied to the tree.

DFS $\leq$ DCR

Applying our construction to the single tree instance G of DFS, we compute in NC^1 the pointwise representation of the permutation product $\pi_G\sigma_G$. By Proposition 1 this product must consist of a single cycle. Now in

proving Proposition 1 we showed that the following procedure, applied to the disjoint cycle representation of $\pi_G\sigma_G$ (obtained by a DCR oracle), yields a depth-first traversal of the tree: Starting at any label incident with the root, list the nodes w in the order in which some edge end incident to w first appears in the cycle.

UFA $\leq$ CFP

Let G be the acyclic undirected graph instance of UFA and assume no edge between u and v. If a path between u and v exists in G, then the new graph obtained by adding edge (u, v) to G contains exactly one cycle; otherwise the new graph remains acyclic.

CFP $\leq$ PP2

Remark. This reduction is included mainly to show that CFP is in *FL*.

Let G be the undirected graph instance of CFP and consider a cycle γ in the disjoint cycle representation of the permutation $\pi_G\sigma_G$, obtained in NC^1 with the help of a PP2 oracle gate (see construction). Then γ describes a closed walk in G in the obvious way; the walk starts at an arbitrary node u with some edge end u_i occurring in γ. Expanding on an argument in [Ho80], we make the

Claim. G is acyclic if and only if for each cycle γ in $\pi_G\sigma_G$ and each edge (u, v) traversed in the direction u to v in the walk described by γ, the walk does not again visit node u until the edge is visited in the reverse direction (from v to u).

The *only if* follows from the proof of Proposition 1. Conversely, suppose that $v_1, v_2, \ldots, v_k$ are nodes forming a cycle in G. Then the walk described by some cycle γ of $\pi_G\sigma_G$ traverses the edge (v_1, v_2) in the direction v_1 to v_2. The condition of the *Claim* and the definition of $\pi_G\sigma_G$ imply that each edge incident with node v_2 is traversed first in the outward direction and then in the reverse direction, until eventually the edge (v_2, v_3) is traversed in the indicated direction (v_2 to v_3). Furthermore, this traversal from v_2 to v_3 occurs before any return visit to v_1, because such return visit must be via edge (v_2, v_1) by the *Claim* condition. Applying the same reasoning to node v_3 instead of v_2, it follows that the walk traverses the edge (v_3, v_4) before returning either to v_2 or v_1. Continuing this reasoning, we see that the walk eventually traverses edge (v_k, v_1) before returning to node v_1, in violation of the condition of the *Claim*.

Note that the conditions of the *Claim* can be checked in NC^1 given the disjoint cycle representation of $\pi_G\sigma_G$ and the input graph G.

UFA $\leq$ PPOW

Recall the proof that UFA $\leq$ DCA, up to the point where it remains to check, given the pointwise representation of $\pi_G\sigma_G$, whether two distinguished labels u_1 and v_1 belong to the same cycle of $\pi_G\sigma_G$. Checking can be done using parallel PPOW oracle gates with inputs $(\pi_G\sigma_G, i)$, for $i = 1, 2, \ldots, n-1$, where n is the total number of labels involved: indeed u_1 and v_1 belong to the same cycle of $\pi_G\sigma_G$ iff $(\pi_G\sigma_G)^i$ maps u_1 and v_1 for at least one such value i. Observe that $i < n$ can be produced in unary or in binary notation, and that only the image of u_1 under the ith power of $\pi_G\sigma_G$ is really required.

UFA $\leq$ PP (*with product required only in pointwise representation*)

The argument is the same as that in UFA $\leq$ PPOW, with the added observation that in NC^1, $(\pi_G\sigma_G)^i$ for $i < n$ can be written as $\pi_G\sigma_G\pi_G\sigma_G \cdots \pi_G\sigma_G$ and fed into a PP oracle gate, thus allowing computation of the image of u_1 under $(\pi_G\sigma_G)^i$. (Recall from the construction that disjoint cycle representations of π_G and of σ_G were each available in NC^1.)

This completes the proof of Theorem 2. □

It is interesting to note that with respect to NC^1 reducibility, from the pointwise representation of a permutation it is as difficult to determine whether two points belong to the same disjoint cycle as it is to order the points within a cycle (DCA and DCR (with a single cycle) are each NC^1-complete for FL).

An interesting open question is whether undirected graph connectivity (UCONN) is in FL. By Aleliunas *et al.* [AKLLR79] this problem is in nonuniform log-space and in a restricted form of probabilistic log space. Further, UCONN restricted to cycle free graphs is clearly in NC^1, since such a graph is connected iff it has one more vertex than edges. On the other hand UCONN in general is at least hard for FL. To see this, note that a permutation presented pointwise is also a presentation of an undirected graph whose vertices are the points permuted and whose edges indicate the predecessor and successor relation. Hence SCP $\leq$ UCONN. This proves

THEOREM 3. Undirected graph connectivity is NC^1-hard for FL. When the given graph is known to be a disjoint union of cycles, the connectivity problem is NC^1-complete for FL.

ACKNOWLEDGMENT

We thank Mike Luby for help in showing the completeness of DCR.

REFERENCES

[AKLLR79] R. ALELIUNAS, R. KARP, R. LIPTON, L. LOVASZ, AND C. RACKOFF, Random walks, universal traversal sequences, and the complexity of maze problems, *in* "Proc. 20th IEEE FOCS," pp. 218–223, (1979).

[Ba86] D. A. BARRINGTON, Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1, *in* "Proc. 18th ACM STOC," pp. 1–5, 1986.

[Bo77] A. BORODIN, On relating time and space to size and depth, *SIAM J. Comput.* **6**, (1977), 733–744.

[Co81] S. A. COOK, Towards a Complexity Theory of Synchronous Parallel Computation, *in* "*L'enseignement mathématique*, Série II," Vol. XXVII, fasc. 1–2, 1981.

[Co85] S. A. COOK, A taxonomy of problems with fast parallel algorithms, *Inform. and Control* **64** (1985), 2–22.

[FiPi74] M. FISHER AND N. PIPPENGER, "*M. J. Fisher Lecture Notes on Network Complexity*," Universitat Frankfurt, 1974.

[HaImMa78] J. HARTMANIS, N. IMMERMAN, AND S. MAHANEY, One-way log tape reductions, *in* "Proc. 19th IEEE FOCS," pp. 65–71, (1978).

[Ho80] J. W. HONG, On some space complexity problems about the set of assignments satisfying a Boolean formula, *in* "Proc. 12th ACM STOC," pp. 310–317, 1980.

[HoUl79] J. E. HOPCROFT AND J. D. ULLMAN, "*Introduction to Automata Theory, Languages, and Computation*," Addison–Wesley, Reading, MA, 1979.

[Jo75] N. D. JONES, Space-bounded reducibility among combinatorial problems, *J. Comput. System Sci.* **11** (1975), 68–85.

[Mc84] P. MCKENZIE, "*Parallel Complexity and Permutation Groups*," Doctoral thesis, Department of Computer Science, Tech. Rep. No. 173/84, Univ. of Toronto, 1984.

[McCo85] P. MCKENZIE AND S. A. COOK, "*The Parallel Complexity of Abelian Permutation Group Problems*," Tech. Rep. No. 181/85, Department of Computer Science, Univ. of Toronto, April 1985.

[Pi79] N. PIPPENGER, On simultaneous resource bounds, *in* "Proc. 20th IEEE FOCS," pp. 307–311, 1979.

[Ru81] W. L. RUZZO, On uniform circuit complexity, *J. Comput. System Sci.* **22** (1981) 365–383.

[Sc76] C. P. SCHNORR, The network complexity and the Turing machine complexity of finite functions, *Acta Informa.* **7** (1976) 95–107.