INFORMATION AND COMPUTATION 95, 96–115 (1991)

Oracle Branching Programs and Logspace versus P^*

DAVID A. MIX BARRINGTON[†]

Department of Computer and Information Science,
University of Massachusetts, Amherst, Massachusetts 01003

AND

PIERRE MCKENZIE[‡]

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

We define the notion of an *oracle branching program* in order to investigate space-bounded computation. Within this new framework we examine the P-complete problem GEN which consists of determining membership in a subalgebra of a general (not necessarily associative) binary algebra (input as a multiplication table). Our work begins with the statement of a conceptually simple conjecture highlighting the combinatorics which underlie the relationship between *Logspace* and P. We show that natural subclasses of P can be expressed as natural subproblems for GEN. Finally, we prove optimal lower bounds on the size of branching programs for GEN with certain *natural* oracles.

1. INTRODUCTION

A long-standing open question in complexity theory is that of determining whether indeed $DSPACE(\log n) \neq P$ (see Hopcroft and Ullman, 1979). The following problem, denoted GEN, is P-complete (Jones and Laaser, 1977) and therefore unlikely to belong to $DSPACE((\log n)^k)$ for any k:

> *Given*: an $n \times n$ table filled with entries from $\{1, 2, ..., n\}$, which we interpret as the multiplication table of an n-element

* An extended abstract of this paper appeared under the same title in "Proceedings of the 14th Conference on Mathematical Foundations of Computer Science," pp. 370–379, Lecture Notes in Computer Science, Vol. 379, Springer-Verlag, Berlin/New York, 1989.

[†] Supported by NSF Computer and Computation Theory Grant CCR-8714714.

[‡] Supported by NSERC of Canada Grant A9979 and by Québec FCAR Grant 89-EQ-2933.

0890-5401/91 $3.00

groupoid, and a subset S of $\{1, 2, ..., n\}$ which includes element 1.
Determine: whether the subgroupoid $\langle S \rangle$, defined as the closure of S under the groupoid product, includes element n.

In this paper we consider the space complexity of GEN. We start by making the following strong conjecture (see the next section for the definition of a branching program). This conjecture is justified by the empirical observation that any branching program efficiently solving all $n \times n$ instances of GEN appears to require a "separate region" verifying the closure under the groupoid product of $T \cup \{1\}$ for each $T \subseteq \{2, 3, ..., n-1\}$:

Conjecture. For each $n > 1$, a branching program in which each node can only evaluate a binary product within an n-element groupoid, branching n ways according to the n possible outcomes, must have at least 2^{n-2} nodes to solve all $n \times n$ GEN instances with singleton starting set S.

Our conjecture implies $\text{GEN} \notin DSPACE((\log n)^k)$ for any k, and we have *not* succeeded in proving it. However, thinking of the branching programs in our conjecture as accessing a "PRODUCT oracle" (using the word *oracle* in the sense of Karp *et al.* (1988), we are led to the following questions: What are other "natural" oracles for GEN? How do such oracles relate to the PRODUCT oracles in terms of space efficiency? What can be learned about the relationship between polylog space and P by varying the available oracle? What would be an "$NSPACE(\log n)$-smart" oracle suitable for studying the relationship between $NSPACE(\log n)$ and P? In this paper we set up a framework for such investigations and merely begin a detailed study within it.

In Section 2 we define *Oracle branching programs* formally and independently from problem GEN: instead of branching at each node according to individual input bits, an oracle branching program accesses its input via an "oracle." "Oracle branching programs" thus generalize the "R-way branching programs" introduced in Borodin and Cook (1982). We contend that the flexible model which results is well-suited for testing one's intuition about the naturalness and effectiveness of various strategies for the resolution of a problem when space is restricted.

In Section 3 we exhibit GEN subproblems complete for standard subclasses of P, justifying the choice of GEN as a particularly appropriate P-complete problem to study. Bearing in mind the inclusion chain

$$NC^1 \subseteq L \subseteq NL \subseteq NC^2 \subseteq \cdots \subseteq NC \subseteq P,$$

(where L and NL denote $DSPACE(\log n)$ and $NSPACE(\log n)$, respectively, as in Cook, 1985), we describe natural L-complete and NL-complete

subproblems of GEN, and we exhibit a close relationship between circuit depth within the *NC* complexity class and "bracketing depth" (defined precisely in Section 2) of groupoid elements in a groupoid instance. We also discuss a subproblem of GEN naturally solved by a nondeterministic algorithm operating simultaneously in small space and in polynomial time, thus belonging to a nondeterministic version of the class *SC* defined in Cook (1981) (see Monien and Sudborough, 1981).

In Section 4 we study oracle branching programs for problem GEN with singleton starting set S, hereafter denoted GEN($\{1\}$). Consider for example the following *oracle*: when queried with a subset T of groupoid elements, the oracle delivers in one step the least groupoid element in $\langle T\rangle\backslash T$ (or 0 if T is already closed). Using nodes labelled with queries to this oracle it is easy to construct a branching program of size 2^{n-2} which solves all $n\times n$ GEN($\{1\}$) instances. We prove as Theorem 4.1 that this construction is optimal: a branching program whose nodes can only query such an oracle *requires* 2^{n-2} nodes to sove all $n\times n$ GEN instances. (Theorem 4.1 does not prove our main conjecture because, as we shall see and perhaps contrary to one's first intuition, the oracle in Theorem 4.1 is much weaker than the PRODUCT oracle in our conjecture.) We then turn to more powerful "natural" oracles (for example to oracles which deliver subsets of elements from $\langle T\rangle\backslash T$ in one step) and in each case we describe upper and lower bounds on the sizes of the various oracle branching programs solving GEN.

We conclude in Section 5 with a discussion of the issues raised by these results and with a number of open problems.

2. Background and Definitions

For each integer $n>0$ let $[n]$ stand for $\{1, 2, ..., n\}$. We use "$\subset$" to denote proper set inclusion. A *groupoid* is a finite set $[n]$ equipped with a binary operation denoted "$*$." Fix a groupoid G and let $T\subseteq G$. As mentioned earlier, $\langle T\rangle$ denotes the smallest groupoid containing T and closed under multiplication. We also refer to the *left closure* of T, denoted LCL(T), which is the smallest subset of G containing T and closed under multiplication on the left by elements of T. Observe that if G is not associative then LCL(T) will in general not include all of $\langle T\rangle$. We also write T^2 for $\{x * y \mid x, y\in T\}$. If $x\in\langle T\rangle$ then an expression for x in terms of elements of T can be represented as a binary tree with leaves from T, and we call the depth of this tree the *bracketing depth* of the expression. Now x can generally be expressed in many different ways as a product of elements of T, and we define the *bracketing depth of x with respect to T* to

be the minimum bracketing depth of such an expression. Given a GEN instance, the *bracketing depth of the* GEN *instance* is defined as the maximum bracketing depth of any $x \in \langle S \rangle$ with respect to S.

2.1. *Complexity Theory*

Throughout this paper we mostly consider subclasses of the class P of problems solvable in polynomial time, with the (likely) exceptions of the complexity classes $DSPACE((\log n)^k)$ of problems solvable in space $(\log n)^k$ on a deterministic multitape Turing machine when $k > 1$. Class L (NL) denotes the class of languages accepted in logarithmic space on a deterministic (nondeterministic) Turing machine. Class NC is defined as $\bigcup_k NC^k$, where NC^k is the set of languages accepted by a uniform family of indegree two Boolean circuits of depth $O((\log n)^k)$ and of polynomial size (Pippenger, 1979). All completeness results in this paper are under NC^1-reducibility: problem f NC^1-reduces to problem g if a uniform circuit family of logarithmic depth (hence of polynomial size) can solve f given the use of oracle gates solving g. (See Cook, 1985, for technicalities omitted from our definitions of NC^k and of NC^1-reducibility; in particular, although the exact notion of uniformity will not play a role in this paper, for definitebess assume log space uniformity of circuit families. Supplemented with Hopcroft and Ullman (1979), Cook (1985) provides an excellent overview of the known relationships between the complexity classes mentioned in this paragraph.)

The *graph accessibility problem* consists of determining whether a path joins node 1 and node m in a graph $\langle [m], E \rangle$. This problem is denoted DGAP, UGAP, or DGAP1 if the graph is a directed graph, an undirected graph, or a directed graph of outdegree 1, respectively. DGAP and DGAP1 are NL-complete and L-complete, respectively (Jones, 1975), and UGAP appears to lie somewhere in between (see Aleliunas *et al.*, 1979).

2.2. *Traditional Branching Programs*

Branching programs (abbreviated *BPs*) for the computation of Boolean functions were studied by several authors (see for example Lee (1959), Masek (1976), Borodin *et al.* (1979, 1983), Chandra *et al.* (1983), Barrington (1989). Traditionally an m-input *branching program* component[1] is a directed acyclic rooted graph, with two nodes of outdegree zero (labelled ZERO and ONE, respectively), with all other nodes of outdegree two (each labelled with some integer j, $1 \leqslant j \leqslant m$), and with the pair of outedges from any node labelled ZERO and ONE, respectively. Such a BP

[1] Most authors refer to "BP components" as "BPs". We prefer to think of a *branching program* as a family $(B_m)_{m \in \mathbb{N}}$ in which B_m is an m-input BP component.

component computes, on input $x_1 x_2, ..., x_m \in \{0, 1\}^m$, the Boolean value given by the label of the unique sink node reached by starting at the root and, once at a node labelled j, traversing edge ZERO or ONE depending on the Boolean value of input bit x_j. The Boolean function computed is often viewed as the characteristic function of a subset $L \subseteq \{0, 1\}^m$, the *language accepted* by the BP component.

As is the case in circuit complexity, an infinite subset of $\{0, 1\}^*$ requires an infinite family of BP components for its recognition (each input length being handled by a separate component), and the question of uniformity arises. In this paper we impose no uniformity condition on BPs. Note also that we use BPs to solve problems where only some of the possible input sizes (measured in bits) are of interest. In this case many of the BP components will be trivial.

Taking the *size* of a BP component to be its number of nodes, denote by $SIZE(s(m))$ the set of languages accepted by a BP whose mth component has size at most $s(m)$. The following fact, which is folklore from the sixties (for example Cobham, 1966, see Borodin *et al.*, 1983), follows from viewing the computation graph of a Turing machine on inputs of a given length as a BP component.

Fact 2.1. $DSPACE(s(m)) \subseteq \bigcup_c SIZE(c^{s(m)})$.

The following conjecture, widely believed to apply to any P-complete problem, therefore implies the nonexistence of a $(\log n)^k$ space solution to GEN for any k. (Consequences of Conjecture 2.2 thus include $NC \subset P$ since for each k $NC^k \subseteq DSPACE((\log n)^k)$ (Borodin, 1977).)

Conjecture 2.2. For each $k > 0$, $\text{GEN} \notin SIZE(2^{(\log n)^k})$.

Note that Conjecture 2.2 is apparently stronger than the statement that no $(\log n)^k$ space GEN solution exists for any k because of the uniformity issue. Indeed suppose that Conjecture 2.2 is false and (say) that a polynomial size BP exists which solves GEN. For such a BP to be useful to a space-bounded Turing machine attempting to solve GEN, it is necessary that the machine on input w of length $|w|$ be able to construct enough of the description of the $|w|$th BP component to carry out the step by step simulation of the component on w. Since no bound is imposed on the complexity of the function which maps a binary string w to an encoding of the $|w|$th BP component (indeed this function need not be recursive), this construction might not be doable in space $(\log n)^k$ for any k.

2.3. *Oracle Branching Programs*

Classical branching program nodes branch two ways according to the answer to a question of the type "what is the value of input bit i?" Nodes of an "R-way branching program" (Borodin and Cook, 1982) branch 2^k

ways according to the answer to a question involving k contiguous input bits. (Note that the branching programs involved in the conjecture of Section 1 can be thought of as R-way branching programs if we assume a natural encoding of GEN multiplication tables.) Oracle branching programs will be defined to allow nodes to "ask" more general questions.

We take $\Sigma = \{0, 1\}$, though our definitions apply to any finite alphabet. For $m \in \mathbf{N}$, Σ^m denotes the set of all strings of length m over Σ, and $(\Sigma^m)^*$ the set of all strings of length km over Σ for some $k \in \mathbf{N}$. For $w \in \Sigma^*$, w^k denotes $ww \ldots w$ (k occurrences of w).

DEFINITIONS. An *m-input oracle BP component* is a deterministic finite automaton $A = (Q, \Sigma^m, \delta, q_0, F)$ in which each final state is a sink state. The *size* of A is $|Q|$. An *oracle branching program* is a family $(A_m)_{m \in \mathbf{N}}$ of m-input oracle BP components. The *size* of the BP is a function describing for each m the size of A_m.

DEFINITIONS. The *language accepted* by oracle BP component $A = (Q, \Sigma^m, \delta, q_0, F)$ is $\mathcal{L}(A) = \{w \in \Sigma^m \mid \hat{\delta}(q_0, w^{|Q|}) \in F\}$, where $\hat{\delta}$ denotes as in Hopcroft and Ullman (1979) the natural extension of $\delta: Q \times \Sigma^m \to Q$ to the domain[2] $Q \times (\Sigma^m)^*$. The *language accepted* by $(A_m)_{m \in \mathbf{N}}$ is $\bigcup_{m \in \mathbf{N}} \mathcal{L}(A_m)$.

Any subset of Σ^* is therefore accepted by some oracle BP of size 2. The notion of an oracle BP becomes interesting only when we restrict the "power" of the transition function in each BP component. We do this by restricting the "local partitions" $\pi_{m,q}$ of Σ^m defined for each state $q \in Q_m$ of each BP component $A_m = (Q_m, \Sigma^m, \delta_m, q_0, F_m)$ as follows: $\pi_{m,q}$ is the partition induced by the equivalence relation $\equiv_{m,q}$ defined on Σ^m as $u \equiv_{m,q} v \Leftrightarrow \delta_m(q, u) = \delta_m(q, v)$.

DEFINITIONS. An *oracle* is a class of partitions of Σ^m for various m. A *BP with oracle O* (or an *O BP*) is an oracle BP $((Q_m, \Sigma^m, \delta_m, q_0, F_m))_{m \in \mathbf{N}}$ in which for each $m \in \mathbf{N}$ and for each $q \in Q_m$ the local partition $\pi_{m,q}$ is refined by some partition of Σ^m in O.

EXAMPLE (BIT oracle). Consider for each $m \in \mathbf{N}$ and for each $i \in [m]$ the partition $B_{m,i}$ induced on Σ^m by relating words $u_1 u_2 \cdots u_m$ and $v_1 v_2 \cdots v_m$ whenever $u_i = v_i$. Define the BIT oracle as the class of all such partitions $B_{m,i}$. BIT BPs model (nonuniform) Turing machines with input alphabet Σ and random access to their input tapes.

[2] The input w, which to the automaton is a single letter of the alphabet, is given to the automaton repeatedly, until the automaton reaches a sink state or will definitely never do so.

We now introduce several "natural" oracles which one might consider critically helpful in solving problem GEN($\{1\}$). Denote by I_n the set of all n^{n^2} groupoids with elements $[n]$. Once and for all we agree on a reasonable $O(n^2 \log n)$ length encoding of I_n over Σ such that, for each n, words of equal length encode all groupoids from I_n. Via this encoding we can view I_n itself as the input alphabet of a BP component.

DEFINITION (PRODUCT oracle). Fix $n \in \mathbf{N}$, $i \in [n]$, $j \in [n]$, and consider the partition of I_n induced by relating $g_1 \in I_n$ and $g_2 \in I_n$ whenever $i * j$ is the same in both groupoids g_1 and g_2. The PRODUCT oracle is the class of all such partitions.

Note that we have chosen to allow cycles in oracle BPs. Standard techniques to eliminate such cycles (credited to Pippenger in Borodin *et al.*, 1979) imply that to within roughly a squaring of their sizes BIT BPs, PRODUCT BPs, and traditional BPs are equivalent with respect to their ability to solve GEN ($\{1\}$) efficiently in terms of BP size. Hence each of these models provides a fair measure of Turing machine space. As we will see, this will no longer be true for the oracles defined in the remainder of this section.

It is convenient to view oracles by considering a question about the input which is answered at each node. For example, a node of a PRODUCT BP as defined above asks "what is the product $i * j$ in the input groupoid?" Assuming the usual ordering on $[n]$, we now define the WITNESS BP, whose nodes are conceptually labelled with questions of the type "what is the least new groupoid element which can be generated from a given subset $T \subseteq [n]$ of groupoid elements?"

DEFINITION (WITNESS oracle). Fix $n \in \mathbf{N}$, $T \subseteq [n]$, and consider the partition induced on I_n by relating g_1 and g_2 whenever $\langle T \rangle \backslash T$ evaluated in each groupoid either contains the same smallest element or in both cases yields the empty set. The WITNESS oracle is the set of all such partitions.

Define for any $k \in \mathbf{N}$ and $T \subseteq [n]$ the set $\min^{(k)}(T)$ to be the lexicographically least k-subset of T if $|T| > k$, and T if $|T| \leqslant k$ (for example $\{1, 3\} < \{1, n\} < \{3, 4\}$).

DEFINITION (k-WITNESS Oracle). Fix $n \in \mathbf{N}$, $T \subseteq [n]$, and consider the partition induced on I_n by relating g_1 and g_2 whenever $\min^{(k)}(\langle T \rangle \backslash T)$ evaluated in each groupoid yields the same set. The k-WITNESS oracle is the set of all such partitions.

Finally we define two more oracles which ask essentially the same questions as the last two, but do not look as hard to find a new groupoid element to return.

DEFINITION (PARTIALWITNESS and k-PARTIALWITNESS oracles). Defined as the WITNESS and k-WITNESS oracles respectively, except based on the evaluation of $T^2 \backslash T$ rather than $\langle T \rangle \backslash T$.

3. COMPLEXITY OF GEN SUBPROBLEMS

Observe that a straightforward algorithm solves GEN in polynomial time. It is known that GEN and GEN (commutative) (even with a singleton starting set) are P-complete (Jones and Laaser, 1977), whereas GEN (associative) is NL-complete (Jones *et al.*, 1976). We begin this section with a new and simple P-hardness proof for GEN, starting from the circuit value problem (Ladner, 1975). The new proof will be used in Proposition 3.2 to relate circuit depth within class NC to bracketing depth of GEN instances. Recall the P-complete problem CVP (Ladner, 1975):

> *Given*: A Boolean circuit, that is, a sequence $(\alpha_1, ..., \alpha_m)$ where each α_i is either a Boolean input, a gate AND(j, k), a gate OR(j, k), or a gate NOT(j), where in each case $j \leqslant k < i$.
> *Determine*: The Boolean value computed at the output gate α_m.

THEOREM 3.1. GEN *is P-complete under NC^1-reducibility (Jones and Laaser, 1977).*

Proof. For each i we assume that if α_i is either AND(j, k) or OR(j, k) then neither α_j nor α_k appear as input to any gate in the circuit other than α_i. (This condition can be achieved by replacing each "wire" in the original circuit by two consecutive NOT gates.) To each circuit gate α_i will correspond two groupoid elements i and $\bar{i}$, whose function is to implement double rail logic as in Goldschlager (1977): i (respectively $\bar{i}$) will belong to the subgroupoid generated by the starting set if and only if gate α_i takes on the value 1 (respectively 0). Additional groupoid elements correspond to the "wires" into NOT gates; that is, they are

$$E = \{e_{ji} \mid \alpha_i = \text{NOT}(j)\}.$$

These additional elements are part of the starting set S, which is defined as

$$S = E \cup \{i \mid \text{input gate } \alpha_i \text{ is assigned } 1\} \cup \{\bar{i} \mid \text{input gate } \alpha_i \text{ is assigned } 0\}.$$

Then gate $\alpha_i = \text{AND}(j, k)$ is simulated by defining

$$i = j * k; \qquad \bar{i} = \bar{j} * \bar{j}; \qquad \bar{i} = \bar{k} * \bar{k},$$

gate $\alpha_i = \text{OR}(j, k)$ by

$$\bar{i} = j * \bar{k}; \qquad i = j * j; \qquad i = k * k,$$

and gate $\alpha_i = \text{NOT}(j)$ by

$$i = e_{ji} * j; \qquad \bar{i} = e_{ji} * j.$$

Observe that each groupoid product defined so far is uniquely defined. To complete the construction, pick any element in the starting set S, say g, and define any remaining groupoid product as yielding g (an easy modification here would yield a commutative groupoid, proving that GEN in the commutative case remains P-complete as was noted in Jones and Laaser, 1977).

The above construction yields a groupoid whose subgroupoid $\langle S \rangle$ contains m if and only if output gate α_m computes 1. ∎

Our next proposition makes precise the correspondence between bracketing depth of a GEN instance and depth of a polynomial size Boolean circuit solving GEN. Note that GEN (associative) instances have bracketing depths at most $\log n$.

PROPOSITION 3.2. GEN *instances with bracketing depth* $(\log n)^k$ *are hard for* NC^k *and can be done in* NC^{k+1}.

Proof. Feeding an NC^k circuit through the groupoid construction described in the proof of Theorem 3.1 produces a groupoid with bracketing depth in $O((\log n)^k)$, which proves the hardness claim. The NC^{k+1} upper bound follows from cascading $(\log n)^k$ stages of logarithmic depth each computing

$$\{i * j \mid i \text{ and } j \text{ were obtained at some previous stage}\}$$

and eliminating duplicates, with S the elements available at stage 0. ∎

Now define problem GEN($r(n)$-rows) for a function $r(n) \leqslant n$ to be the subcase of GEN in which at most $r(n)$ rows of an $n \times n$ multiplication table contain elements other than 1.

LEMMA 3.3. *Let* T *be the set of (indices of) nontrivial rows in the multiplication table of some groupoid* G *and suppose* $1 \in T$. *Then* $\langle T \rangle = \text{LCL}(T)$.

Proof. Trivially $\text{LCL}(T) \subseteq \langle T \rangle$. We prove the reverse inclusion by induction on the bracketing depth of $x \in \langle T \rangle$ with respect to T. In the induction step we must have $x = y * z$ for y and z of depths smaller than that of x, thus for y and z in $\text{LCL}(T)$ by the induction hypothesis. If y is in T then x is in $\text{LCL}(T)$. If y is not in T then the yth row is trivial and $y * z = 1$, so that $x = 1$ and x is in T. ∎

THEOREM 3.4. *$NSPACE(\max\{\log n, r(n)\}) \cap DSPACE(\max\{(\log n)^2, r(n)\})$ contains the problem* GEN *($r(n)$-rows).*

Proof. The following nondeterministic algorithm determines membership of n in $\langle S\rangle$:

$T \leftarrow \{x \in S \mid x \text{ is the index of a nontrivial row}\}$
for $i \leftarrow 1$ **to** $r(n)$
  guess a nontrivial row j
  if $j \in \mathrm{LCL}(T)$ **then** $T \leftarrow T \cup \{j\}$
if $n \in \mathrm{LCL}(T)$ **then accept**.

This algorithm never accepts incorrectly, and Lemma 3.3 together with an induction imply that the **for** loop is able to compute

$$\{x \in \langle S\rangle \mid x \text{ is the index of a nontrivial row}\}.$$

Another application of Lemma 3.3 justifies the last instruction. As to the space claims, without loss of generality we can assume that the nontrivial rows are the first $r(n)$ rows since the index of each nontrivial row can be computed in $\log n$ deterministic space. Hence a bit map of length $r(n)$ can represent T. Testing membership in $\mathrm{LCL}(T)$ nondeterministically requires remembering only one element and thus uses at most $\log n$ space (and thus at most $(\log n)^2$ space deterministically by Savitch, 1970). ∎

Note that unless $r(n)$ is in $O(\log n)$ the deterministic version of the above algorithm does not run in polynomial time, although of course a different algorithm solves even the general problem in polynomial time. An interesting aspect of problem GEN($(\log n)^k$-rows) is that the above nondeterministic algorithm solves the problem *simultaneously* in space $O((\log n)^k)$ and in polynomial time. The class of problems solvable nondeterministically in space $f(n)$ and polynomial time was introduced in Monien and Sudborough (1981) and denoted $NTISP(\mathrm{poly}, f(n))$. Interestingly, known members of $NTISP(\mathrm{poly}, (\log n)^k)$ were obtained by imposing natural "bandwith constraints" on various *NP*-complete problems; here we are led to the same class by restricting a problem in *P*.

COROLLARY 3.5. *Problems* GEN(2-*rows*) *and* GEN($\log n$-*rows*) *are NL-complete.*

Proof. By Theorem 3.4 it suffices to prove that GEN(2-rows) is *NL*-hard. Consider a directed graph $\langle [n], E\rangle$ of outdegree two which includes edge (1, 2). Then a directed path exists from node 1 to node n if and only if element n belongs to the subgroupoid $\langle \{1, 2\}\rangle$ of the groupoid with

elements $[n]$ defined as follows: for each $j \in [n]$, with outgoing edges (j, k) and (j, l), set $1 * j = k$ and $2 * j = l$. ∎

THEOREM 3.6. *Problem* GEN(1-*row*) *is L-complete.*

Proof. Starting from DGAP1, a reduction almost identical to that in Corollary 3.5 proves that GEN(1-row) is L-hard. To see that GEN(1-row) $\in L$, let i be the unique nontrivial row. Then, because $j * k = 1$ whenever $j \neq i$, the following holds: $n \in \langle S \rangle$ if and only if either $n \in S$, or $i \in S$ and $n \in \{j * k \mid j \in \mathrm{LCL}(\{i\}), k \in S\}$. Now $\mathrm{LCL}(T)$ can be computed in L for any singleton T. ∎

An easy reduction from GEN proves:

PROPOSITION 3.7. GEN($\{1\}$) *is P-complete.*

The following is also clear:

PROPOSITION 3.8. GEN($\{1\}$ *and associative*) $\in L$.

Hence unlike GEN, GEN (associative) apparently becomes easier when the starting set S is $\{1\}$. We suspect GEN($\{1\}$ and associative) to be L-hard but cannot yet prove this. By contrast the proof of Theorem 3.6 implies:

PROPOSITION 3.9. GEN($\{1\}$ *and* 1-*row*) *is L-complete.*

This last problem remains L-complete even if we further insist that all elements of $[n]$ appear in the only non-trivial row. This is seen by a reduction from the L-complete problem (Cook and McKenzie, 1987) of determining whether points 1 and n belong to the same cycle of a permutation π of $[n]$ which is input as a sequence $\pi(1), \pi(2), \ldots, \pi(n)$. Finally,

PROPOSITION 3.10. GEN($\{1\}$ *and* 2-*rows*) *is NL-complete.*

4. ORACLE BRANCHING PROGRAMS SOLVING GEN

Fix n and say that a BP component is *valid* if it solves all $n \times n$ GEN($\{1\}$) instances.

THEOREM 4.1. *The size of a valid WITNESS* BP *component is at least* 2^{n-2}.

Proof. For each of the 2^{n-2} sets $T \subseteq [n-1]$ which contain element 1 we will exhibit two groupoids $g_+(T)$ and $g_-(T)$ with the properties

(subscripts "+" and "−" distinguish between the two groupoids under consideration)

1. $n \in \langle 1 \rangle_+$
2. $n \notin \langle 1 \rangle_-$
3. $(\forall Q \subseteq [n])[(T \neq Q) \Rightarrow (\min_+(\langle Q \rangle \backslash Q) = \min_-(\langle Q \rangle \backslash Q))]$,

where we take the min of an empty set to be zero. In words these conditions state that $g_+(T)$ must travel to a final state, that $g_-(T)$ must not travel to a final state, and yet that the only WITNESS oracle query which can tell $g_+(T)$ apart from $g_-(T)$ is T. Necessarily all 2^{n-2} distinct queries T must therefore appear in a valid BP component.

Fixing $T = \{1, x_2, x_3, ..., x_i\} \subseteq [n-1]$, where the relative order of the elements in T is immaterial, we now describe groupoids $g_+(T)$ and $g_-(T)$. All entries other than those in the first row of the respective multiplication tables are set to 1. Letting $\{y_{i+1}, y_{i+2}, ..., y_{n-1}, n\} = [n] \backslash T$ and permuting columns of the multiplication tables for ease of presentation, we define the first row of $g_+(T)$ as

	1	x_2	x_3	$\cdots$	x_{i-1}	x_i	y_{i+1}	y_{i+2}	$\cdots$	n
1	x_2	x_3	x_4	$\cdots$	x_i	n	n	n	$\cdots$	n

and the first row of $g_-(T)$ as

	1	x_2	x_3	$\cdots$	x_{i-1}	x_i	y_{i+1}	y_{i+2}	$\cdots$	n
1	x_2	x_3	x_4	$\cdots$	x_i	x_i	n	n	$\cdots$	n

Groupoids $g_+(T)$ and $g_-(T)$ clearly satisfy conditions 1 and 2 above. To verify condition 3, pick a nonempty set $Q \subseteq [n]$ other than T. Observe that by definition of both groupoids $1 \in \langle Q \rangle$ and thus $T \cup Q \subseteq \langle Q \rangle \subseteq T \cup Q \cup \{n\}$.

Case 1. $T \backslash Q \neq \emptyset$. Then $\min_+(\langle Q \rangle \backslash Q) = \min_-(\langle Q \rangle \backslash Q) = \min(T \backslash Q)$.

Case 2. $T \subset Q$. Then $\min_+(\langle Q \rangle \backslash Q) = \min_-(\langle Q \rangle \backslash Q) = y$, where $y = 0$ if $n \in Q$ and $y = n$ otherwise (since any element outside T generates n by construction in both groupoids). ∎

We note that it is not difficult to construct a valid WITNESS BP component using 2^{n-2} nodes, showing that Theorem 4.1 is optimal.

A more realistic oracle than WITNESS is PARTIALWITNESS, which

for a query set Q returns the least element in $Q^2 \backslash Q$. Such an oracle can be constructed out of a polynomial number of PRODUCT nodes, and one's intuition might be that PARTIALWITNESS and PRODUCT oracles are equally helpful in constructing size-efficient valid BPs. The next theorem shatters this intuition.

THEOREM 4.2. *The size of a valid PARTIALWITNESS* BP *component is at least* 2^{n-2}.

Proof. The argument is almost identical to that in the proof of Theorem 4.1. Fix $T = \{1, x_2, x_3, ..., x_i\}$, this time choosing the labels so that $1 < x_2 < x_3 < \cdots < x_i$, and construct $g_+(T)$ and $g_-(T)$ exactly as before. To verify that no PARTIALWITNESS oracle query Q can distinguish $g_+(T)$ from $g_-(T)$ unless $Q = T$, simply observe that for each such set $Q \subseteq [n]$ the minimal element of $\langle Q \rangle \backslash Q$ is also in $Q^2 \backslash Q$ for both groupoids. ∎

Again here Theorem 4.2 is easily seen to be optimal.

In defining WITNESS and PARTIALWITNESS BPs we chose to have an oracle query Q return the *least* element among the set of possible answers. This may seem unfairly biased against the BP because n is only returned when no other new element is available. Consider then modifying the WITNESS and PARTIALWITNESS oracles to consistently return the *largest* element among the set of possible answers. Although WITNESS oracles modified in this way become so powerful as to solve GEN with the single oracle query $\{1\}$, an exponential size lower bound still holds in the case of PARTIALWITNESS oracles.

LEMMA 4.3. *Fix any total order* $\pi_1 \prec \pi_2 \prec \cdots \prec \pi_n$ *on* $[n]$. *In a valid* BP *component with PARTIALWITNESS oracle modified to consistently choose its least element according to the new order, each query* T *containing* 1 *but not* n *and satisfying*

$$|\{x \in T : \mathbf{max}([n] \backslash T) \prec x\}| \leqslant 2 \tag{1}$$

must appear, where **max** *is taken with respect to* $\prec$.

Proof. It is easy to show that in a valid BP the query $T = \{1\}$ must appear. For the rest we proceed as in the proof of Theorem 4.1. Pick $T = \{x_1, x_2, x_3, ..., x_i\}$, $|T| \geqslant 2$, with $1 \in T$ but $n \notin T$ satisfying condition (1), whence we can write $x_1 \prec x_2 \prec \cdots \prec x_{i-2} \prec m$ for $m = \mathbf{max}([n] \backslash T)$. Writing $\{y_{i+1}, y_{i+2}, ..., y_{n-1}, m\} = [n] \backslash T$, we now construct $g_+(T)$ and $g_-(T)$. All entries other than those in row x_{i-1} of the respective multiplication tables are set to x_1 except $1 * 1 = x_{i-1}$ if $1 \neq x_{i-1}$. Again

permuting columns of the multiplication tables, row x_{i-1} of $g_+(T)$ is defined as

	x_1	x_2	$\cdots$	x_{i-1}	x_i	y_{i+1}	y_{i+2}	$\cdots$	y_{n-1}	m
x_{i-1}	x_2	x_3	$\cdots$	x_i	m	m	m	$\cdots$	m	n

and row x_{i-1} of $g_-(T)$ as

	x_1	x_2	$\cdots$	x_{i-1}	x_i	y_{i+1}	y_{i+2}	$\cdots$	y_{n-1}	m
x_{i-1}	x_2	x_3	$\cdots$	x_i	x_i	m	m	$\cdots$	m	n

Groupoids $g_+(T)$ and $g_-(T)$ satisfy $n \in \langle 1 \rangle_+$ and $n \notin \langle 1 \rangle_-$. We must now verify that no query Q other than T itself can distinguish $g_+(T)$ from $g_-(T)$. Pick and $Q \subseteq [n]$, $Q \neq T$.

Case 1. $x_{i-1} \notin Q$ or $x_i \notin Q$. Then computing Q^2 does not involve the only table entry which distinguishes $g_+(T)$ from $g_-(T)$.

Case 2. $\{x_{i-1}, x_i\} \subseteq Q$ and $T \backslash Q \neq \varnothing$. Let x be the minimal element of $T \backslash Q$ under $\prec$. We have $x \prec m$ since $x \in T \backslash \{x_{i-1}, x_i\}$. Then $\mathbf{min}_+(Q^2 \backslash Q) = \mathbf{min}_-(Q^2 \backslash Q) = y$, where $y = x$ if $m \notin Q$ and $y = \mathbf{min}(x, n)$ otherwise.

Case 3. $T \subset Q$. Then $\mathbf{min}_+(Q^2 \backslash Q) = \mathbf{min}_-(Q^2 \backslash Q) = y$, where $y = 0$ if $m \in Q$ and $n \in Q$, $y = n$ if $m \in Q$ and $n \notin Q$, $y = m$ if $m \notin Q$. ∎

THEOREM 4.4. *Fix any total order* $\pi_1 \prec \pi_2 \prec \cdots \prec \pi_n$ *on* $[n]$. *The size of a valid* BP *component with PARTIALWITNESS oracle modified to consistently choose its least element according to the new order is at least* 2^{n-2} *if* $\pi_{n-3} \prec n$ *and at least* $(\frac{3}{4})2^{n-2}$ *otherwise.*

Proof. Using Lemma 4.3 it suffices to count the number of distinct candidates satisfying condition (1). (A *candidate* is a set T with $1 \in T$ and $n \notin T$.) If $\pi_{n-3} \prec n$ then all 2^{n-2} candidates satisfy the condition since at most two elements of $[n]$, and hence of any candidate T, are larger than $n \in [n] \backslash T$. Condition (1) can now fail only if all three of the elements $\{\pi_{n-2}, \pi_{n-1}, \pi_n\}$ are in T. Normally this will not be the case for $\frac{7}{8}$ of the T's but there is the possibility that 1 is in this set and then only $\frac{3}{4}$ of the T's omit one of the other two. ∎

Our best upper bound for the size of a valid BP component with the oracle of Theorem 4.4, regardless of the ordering $\prec$, is the obvious 2^{n-2}. Except when element n happens to be the largest, the second largest, or the third largest element under the ordering, our upper and lower bounds therefore do not quite match.

Now recall the k-WITNESS BP, whose nodes can be thought of as branching $\sum_{i=0}^{k}\binom{n}{i}$ ways according to the set $\min^{(k)}(\langle T\rangle\backslash T)$ as defined in Section 2. Let us write $f(k,m)=\sum_{i=0}^{\lfloor m/k\rfloor}\binom{m}{ik}$ for the number of subsets of $[m]$ whose cardinalities are a multiple of k. Note that $f(k,m)$ is approximately $2^m/k$ for m much larger than k. For any fixed k, it is easy to construct a valid k-WITNESS BP component of size $f(k,n-2)$.

THEOREM 4.5. *The size of a valid k-WITNESS* BP *component is at least* $2^{n-2}/n^{k-1}$.

Proof. It suffices to oberve that, upon having constructed $g_+(T)$ and $g_-(T)$ exactly as in the proof of Theorem 4.1, any query set Q for which $|T\backslash Q|\geq k$ or $Q\backslash T\neq\varnothing$ is such that $\min_+^{(k)}(\langle Q\rangle\backslash Q)=\min_-^{(k)}(\langle Q\rangle\backslash Q)$. Hence although several queries Q are capable of telling $g_+(T)$ apart from $g_-(T)$, any single query Q can only take care of those sets T for which $Q\subseteq T$ and $|T\backslash Q|\leq k-1$. Since there are at most n^{k-1} such sets T for any Q, the result follows. ∎

Consider finally the more realistic k-PARTIALWITNESS oracle, which "computes" $\min^{(k)}(Q^2\backslash Q)$.

THEOREM 4.6. *The following holds for* $k=1,2,3$: *a valid k-PARTIALWITNESS* BP *component has size at least* $2^{n-2}/n^{k-1}$.

Proof. For $k=1$ this is the content of Theorem 4.2. Consider $k=2$. For any $T=\{1,x_2,x_3,\ldots,x_i\}\subseteq[n-1]$ with $1<x_2<x_3<\cdots<x_i$ and $\{y_{i+1},y_{i+2},\ldots,y_{n-1},n\}=[n]\backslash T$, define the nontrivial rows of $g_+(T)$ and of $g_-(T)$ to be

	1	x_2	x_3	x_4	$\cdots$	x_{i-2}	x_{i-1}	x_i	y_{i+1}	$\cdots$	n
1	x_2	x_3	x_4	x_5	$\cdots$	x_{i-1}	x_i	*	n	$\cdots$	n
x_i	x_3	x_4	x_5	x_6	$\cdots$	x_i	x_i	x_i	n	$\cdots$	n

where * stands for n in the definition of $g_+(T)$ and for x_i in that of $g_-(T)$. Then a 2-PARTIALWITNESS oracle query Q can tell $g_+(T)$ apart from $g_-(T)$ only if $\{1,x_i\}\subseteq Q\subseteq T$. Moreover, such a query Q fails whenever $|T\backslash Q|>1$ since in that case $\min_+^{(2)}(Q^2\backslash Q)=\min^{(2)}(T\backslash Q)=\min_-^{(2)}(Q^2\backslash Q)$ by construction. This means that a query Q can succeed for at most n distinct sets T, and the bound for $k=2$ follows.

Now let $k=3$. For any $T=\{1,x_2,x_3,\ldots,x_i\}\subseteq[n-1]$ with $1<x_2<x_3<\cdots<x_i$ and $\{y_{i+1},y_{i+2},\ldots,y_{n-1},n\}=[n]\backslash T$, define the nontrivial rows of $g_+(T)$ and of $g_-(T)$ to be

	1	x_2	x_3	x_4	$\cdots$	x_{i-3}	x_{i-2}	x_{i-1}	x_i	y_{i+1}	$\cdots$	n
1	x_2	x_3	x_4	x_5	$\cdots$	x_{i-2}	x_{i-1}	x_i	$*$	n	$\cdots$	n
x_i	x_3	x_4	x_5	x_6	$\cdots$	x_{i-1}	x_i	x_i	x_4	n	$\cdots$	n
x_2	x_4	x_5	x_6	x_7	$\cdots$	x_i	x_i	x_i	x_i	n	$\cdots$	n

where as before $*$ stands for n in the definition of $g_+(T)$ and for x_i in that of $g_-(T)$. (Note that for $|T| \leqslant 4$ these definitions of $g_+(T)$ and of $g_-(T)$ still apply with the understanding that $x_j = x_{|T|}$ for $j > |T|$.) We claim that a query Q can only tell $g_+(T)$ apart from $g_-(T)$ if $\{1, x_i\} \subseteq Q \subseteq T$ and $|T \backslash Q| \leqslant 2$. Since the first condition is clear, pick Q such that $\{1, x_i\} \subseteq Q \subseteq T$ and $|T \backslash Q| > 2$. If $x_2 \in Q$ then it is easily seen that $\min_+^{(3)}(Q^2 \backslash Q) = \min^{(3)}(T \backslash Q) = \min_-^{(3)}(Q^2 \backslash Q)$. So assume $x_2 \notin Q$. Then $x_2 \in Q^2 \backslash Q$, which in $g_+(T)$ and $g_-(T)$ also includes the second and third smallest elements of $T \backslash Q$ since the definition of $x_i * x_i$ as x_4 takes care of the critical cases in which $\{x_3, x_4\} \cap Q = \varnothing$: this means once again that Q cannot tell $g_+(T)$ apart from $g_-(T)$, proving our claim and implying our lower bound in the case $k = 3$. ∎

Our lower bound strategy in this section has been to define pairs of groupoids which differ in a single entry. We cannot directly extend this strategy to the case of 4-PARTIALWITNESS oracles. Suppose for example that $i * j = k$ in groupoid g_+ and $i * j = l$ in groupoid g_-. A 4-PARTIALWITNESS query $\{i, j\}$ will correctly return the set $\{i * i, i * j, j * i, j * j\} \backslash \{i, j\}$. It is difficult to imagine how these sets could be the same in groupoids which have significantly different computation properties.

5. Discussion and Open Problems

It is tempting to view the lower bounds of Section 4 as evidence supporting our conjecture of Section 1. These lower bounds reflect instead the weakness of the oracles studied. Indeed we have seen in Section 3 that GEN(1-row) $\in L$ and yet the proofs of Theorems 4.1 and 4.2 show that WITNESS and PARTIALWITNESS BPs solving GEN($\{1\}$ and 1-row) require 2^{n-2} nodes. This suggests that the obvious resolution method for GEN in which no attention is paid to the origin of the new element generated at each step is indeed a poor strategy in terms of space usage. In this connection it is interesting to scrutinize the way in which PRODUCT oracles manage to bypass this strategy when solving GEN($\{1\}$ and 1-row) in polynomial size and GEN($\{1\}$ and 2-rows) in size $n^{\log n}$ (which is possible by Savitch's theorem (Savitch, 1970)). Naturally we expect to be

unable to mimic these shortcuts in solving GEN($\{1\}$) because of the overwhelming amount of information which would in effect need storing within the BP "topology".

Our investigation of GEN subproblems has uncovered interesting questions concerning the effect of imposing the algebraic condition of associativity together with other restrictions. Given a group and a set of generators, the Cayley graph of the group is a graph with one vertex per group element and, for each generator g and each element x, an edge labelled g from element x to element xg. The Cayley graph of a group is easily constructed from the group's multiplication table, so that GEN(groups) NC^1-reduces to UGAP by including only the Cayley graph edges corresponding to elements in the starting set S and by asking whether a path joins the group identity and element n. Does GEN(groups) belong to L? We doubt that this is the case: we believe rather that GEN(groups) is complete for the NC^1-closure of UGAP, though we do not yet see how to apply the techniques in Cook and McKenzie (1987) to prove that GEN(groups) is even L-hard. A further restriction is GEN(cyclic groups): this problem is clearly in L, but is it in NC^1? Another interesting algebraic problem is GEN(commutative and associative): how does it relate to UGAP or to L? Developing the appropriate hardness proofs for these problems will yield further insight into the expanding connections between algebra and low level complexity classes (Barrington, 1989; Barrington and Thérien, 1988; McKenzie and Thérien, 1989).

The restricted version of GEN with only $(\log n)^k$ nontrivial rows further led us to the class $NTISP(\text{poly}, (\log n)^k)$ (Monien and Sudborough, 1981) of problems solvable by nondeterministic machines with simultaneous bounds of polynomial time and $O((\log n)^k)$ space. By analogy with Cook's class SC^k (Cook, 1981) let us call this class NSC^k. NSC^1 is of course just NL and is thus within P, but we know little about even NSC^2. It it equal to $NSPACE((\log n)^2)$? (This would imply $NSPACE((\log n^2) \subseteq NP$.) Is it within P? Is is closed under complement? In this last case the recent proof by Immerman (1988) and Szelepcsényi (1987) that nondeterministic space classes are closed under complement does not appear to apply, because the nondeterministic algorithm used there to solve a problem in co-$NSPACE(f(n))$ uses time $2^{f(n)}$.

A plethora of open questions concerns oracle branching programs for GEN. Theorem 4.4 and the discussion preceding it reveal crucial differences between the WITNESS and PARTIALWITNESS oracles. The WITNESS oracle's ability to choose its output from $\langle Q \rangle$ is a two-sided coin: a bad choice function hinders progress (relative to the PARTIALWITNESS oracle which has fewer bad answers to choose from) and a good choice function solves GEN trivially. PARTIALWITNESS oracles on the other hand are hardly dependent on the choice function provided that this func-

tion returns the smallest element (in $Q^2\backslash Q$) according to a total order fixed in advance. PARTIALWITNESS oracles are also realistic in that they are easily simulated by the PRODUCT oracle. Observe that both types of oracle (as well as PRODUCT and BIT oracles, though somewhat artificially) are in effect prescribed by a family of functions $eval_n: I_n \times \mathcal{Q}_n \to 2^{[n]}$ and $choice_n: 2^{[n]} \to 2^{[n]}$. Fixing n and query $Q \in \mathcal{Q}_n$, a partition is induced on I_n by relating g_1 and g_2 whenever $choice_n(eval_n(g_1, Q)) = choice_n(eval_n(g_2, Q))$, and formally the oracle is the class of all such partitions. The interplay between meaningful *eval* and *choice* functions affects the behavior of resulting oracles dramatically and deserves further study.

Theorem 4.6 suggests taking a closer look at 4-PARTIALWITNESS oracles. Are these equivalent to the PRODUCT oracles? Will lower bound techniques applicable to these also apply to PRODUCT oracle BPs for GEN?

Future research on oracle BPs for GEN should also involve oracles whose power is intermediate between that of the PRODUCT and that of the WITNESS or PARTIALWITNESS oracles. In particular, upper bounds for BPs with oracles as "powerful" as L, NL, NSC^k, or NC^k should be investigated. The results in Section 3 suggest defining such oracles as having the ability to extract all the information from one complete row of a multiplication table ("L-smart oracle"), from two complete rows ("NL-smart oracle"), from a $(\log n)^k$ size subset of rows ("NSC^k-smart oracles"), or from all expressions of bracketing depth at most $(\log n)^k$ in terms of the elements in a query set Q ("NC^k-smart oracles").

All oracles discussed in this paper (including the fundamental BIT and PRODUCT oracles) are *nonadaptive* in the sense that the answer to oracle query Q does not depend on the "history" of the "algorithm". Consider instead an adaptive PARTIALWITNESS oracle which returns the least element in $Q^2\backslash Q$ *among those which the algorithm has not yet seen*: can we prove an exponential size lower bound for a valid BP component in that case? Note that under this new definition one can construct a size $O(n^2)$ PARTIALWITNESS oracle BP component correctly solving all GEN instances used in proving Theorem 4.1. Of course in an algorithm using adaptive oracles the number of nodes no longer yields a fair measure of memory usage and thus will no longer be related to Turing machine space.

In yet a different vein nondeterminism could be allowed. Consider letting a node with oracle query Q pick its output nondeterministically from the arising sets $\langle Q\rangle\backslash Q$. For such nondeterministic WITNESS and PARTIALWITNESS oracles the lower bounds in Theorems 4.1 and 4.2 still apply. Another possible modification is to assume "friendly" oracles which pick their outputs in the interest of the shortest possible computation of $\langle\{1\}\rangle$ for each groupoid: can we prove lower bounds for generalized WITNESS and PARTIALWITNESS oracles of this type?

All investigations of oracle BPs in this paper stopped short of considering restrictions imposed by the underlying graph of a BP component. Although valuable intuition was distilled from our results, it is clear that no such simple-minded analysis will prove $\mathrm{GEN} \notin DSPACE((\log n)^k)$. It would be desirable to develop lower bound techniques gradually taking care of more and more such "topological" restrictions. One possible starting point might be to reexamine the pebbling arguments so pervasive to former work (for example, Cook, 1974, Kozen, 1977) on relating space and time complexities.

We were led to the tantalizing conjecture in Section 1 by studying the P-complete problem which in our opinion provides the most transparent view of the fundamental combinatorics underlying the reltionship between polylogarithmic space and P. We venture the claim that this relationship will not be elucidated without a resolution of our main conjecture (of course partial results are possible: for example any nonpolynomial size lower bound in our conjecture implies $L \subset P$). In the hope of attracting the attention of combinatoricists we conclude with a challenge to the reader: for some $n > 1$, work out the exact number $s(n) \in O(n^2 2^n)$ of PRODUCT BP nodes needed to solve all $n \times n$ $\mathrm{GEN}(\{1\})$ instances in the obvious way, and then exhibit a PRODUCT BP component solving these $n \times n$ instances using only $s(n) - 1$ nodes.

ACKNOWLEDGMENTS

We thank François Lemieux for identifying an omission in an earlier version of the proof of Theorem 3.1 and Eric Allendeer for pointing out Monien and Sudborough (1981) to us. We also thank the anonymous referee for a very careful reading of our manuscript.

RECEIVED October 27, 1989; FINAL MANUSCRIPT RECEIVED March 20, 1990

REFERENCES

ALELIUNAS, R., KARP, R., LIPTON, R., LOVASZ, L., AND RACKOFF, C. (1979), Random walks, universal traversal sequences, and the complexity of maze problems, *in* "Proceedings of the 20th IEEE Symposium on the Foundations of Computer Science," pp. 218–233.

BARRINGTON, D. A. (1989), Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1, *J. Comput. System Sci.* **38**, 150–164.

BARRINGTON, D. A. M. AND THÉRIEN, D. (1988), Finite monoids and the fine structure of NC^1, *J. Assoc. Comput. Mach.* **35** (4), 941–952.

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

BORODIN, A. AND COOK, S. (1982), A time-space tradeoff for sorting on a general sequential model of computation, *SIAM J. Comput.* **11** (2), 287–297.

BORODIN, A., DOLEV, D., FICH, F., AND PAUL, W. (1983), Bounds for width-two branching

programs, *in* "Proceedings of the 15th ACM Symposium on the Theory of Computing," pp. 87–93.

BORODIN, A., FISHER, M. J., KIRKPATRICK, D. G., LYNCH, N. A., AND TOMPA, M. (1979), A time-space tradeoff for sorting on non-oblivious machines, *in* "Proceeding of the 20th IEEE Symposium on the Foundations of Computer Science," pp. 319–327.

CHANDRA, A., FURST, M., AND LIPTON, R. (1983), Multi-party protocols, *in* "Proceedings of the 15th ACM Symposium on the Theory of Computing," pp. 94–99.

COBHAM, A. (1966), "The Recognition Problem for the Set of Perfect Squares, "Research Paper RC-1704, IBM Watson Research Center, Yorktown Heights, New York.

COOK, S. A. (1974), An observation on time-storage trade-off, *J. Comput. System Sci.* **9**, 308–316.

COOK, S. A. (1981), Towards a complexity theory of synchronous parallel computation, *Enseign. math. (2)* **27**, 1–2.

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

COOK, S. A. AND MCKENZIE, P. (1987), Problems complete for deterministic logarithmic space, *J. Algorithms* **8**, 385–394.

GOLDSCHLAGER, L. M. (1977), The monotone and planar circuit value problems are log space complete for *P*, *SIGACT News* **9** (2), 25–29.

HOPCROFT, J. E. AND ULLMAN, J. D. (1974), "Introduction to Automata Theory, Languages, and Computation," Addison–Wesley, Reading, MA.

IMMERMAN, N. (1988), Nondeterministic space is closed under complement, *SIAM J. Comput.* **17** (5), 935–938.

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

JONES, N. D. AND LAASER, W. T. (1977), Complete problems for deterministic polynomial time, *Theoret. Comput. Sci.* **3**, 105–117.

JONES, N. D., LIEN, E., AND LAASER, W. T. (1976), New problems complete for nondeterministic log space, *Math. Systems Theory* **10**, 1–17.

KARP, R., UPFAL, E., AND WIGDERSON, A. (1988), The complexity of parallel search, *J. Comput. System. Sci.* **36**, 225–253.

KOZEN, D. (1977), Lower bounds for natural proof systems, *in* "Proceedings of the 18th ACM Symposium on the Theory of Computing," pp. 254–266.

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

LEE, C. Y. (1959), Representation of switching functions by binary decision programs, *Bell Systems Techn. J.* **38**, 985–999.

MASEK, W. (1976), "A Fast Algorithm for the String Editing Problem and Decision Graph Complexity," M. Sc. Thesis, M.I.T.

MCKENZIE, P. AND THÉRIEN, D. (1989), Automata theory meets circuit complexity, *in* "Proceedings of the 16th International Colloquium on Automata, Languages and Programming," pp. 589–602, Lecture Notes in Computer Science, Vol. 372, Springer-Verlag, Berlin/New York.

MONIEN, B. AND SUDBOROUGH, H. (1981), Bandwith constrained *NP*-complete problems, *in* "Proceedings of the 13th ACM Symposium on the Theory of Computing, pp. 207–217.

PIPPENGER, N. (1979), On simultaneous resource bounds, *in* "Proceedings of the 20th IEEE Symposium on the Foundations of Computer Science," pp. 307–311.

SAVITCH, W. J. (1970), Relationships between nondeterministic and deterministic tape complexities, *J. Comput. System. Sci.* **4**, 177–192.

SZELEPCSÉNYI, R. (1987), The method of forcing for nondeterministic automata, *Bull. European Assoc. Theoret. Comput. Sci.*, 96–100.

Printed in Belgium