SIAM J. COMPUT.
Vol. 23, No. 3, pp. 563–572, June 1994

SUBQUADRATIC SIMULATIONS OF BALANCED FORMULAE BY BRANCHING PROGRAMS [*]

JIN-YI CAI[†] AND RICHARD J. LIPTON[‡]

Abstract. This paper considers Boolean formulae and their simulations by bounded width branching programs. It is shown that every balanced Boolean formula of size s can be simulated by a constant width (width 5) branching program of length $s^{1.811\cdots}$. A lower bound for the translational cost from formulae to permutation branching programs is also presented.

Key words. branching programs, Boolean formulae, Boolean circuits, group theory

AMS subject classifications. 68Q15, 68Q25

1. Introduction. In a beautiful paper, Barrington [B] showed that the class of languages recognized by (nonuniform) NC^1 circuits (fan-in 2, depth $O(\log n)$ on n inputs) is identical to the class recognized by bounded width branching program with polynomial length. His main motivation was to resolve a conjecture that had been made by Borodin et al. [BDFP]. They had conjectured the opposite is true, namely, that bounded width computations requires exponential length to simulate NC^1; in particular, the majority function requires exponential length.

Barrington showed that any balanced Boolean formula of depth d can be recognized by a branching program of width 5 and length $(2^d)^2$. Thus, if an NC^1 function is represented by a balanced tree of depth d (so the size s of the tree is roughly 2^d), then the length of the branching program will be s^2, quadratic in the size of the tree. (For simplicity, we assume our balanced trees are fully balanced binary trees, as in Barrington's paper. Although our construction works for trees not fully balanced, the estimate of the length is more complicated.) Barrington's work has been generalized by Ben-Or and Cleve [BC] to algebraic formulae. Their simulation, which uses a construction similar to Barrington's, also has a quadratic increase in the length.

The first set of results in this paper is to improve on the upper bound established by Barrington. We show that, for width 5 branching programs, the exponent 2 in Barrington's construction can be improved to $1.811\ldots$. Then in §5 we present a lower bound on the translational cost from formulae to branching programs over any finite groups.

Our primary motivation for this work is to study branching programs and the important class NC^1 [Pi][Co]. If we can shed further light on the relationship between branching programs and the class NC^1 in the attempt to lower the exponent of the cost of simulating NC^1 by branching programs, the study will have proved to be worthwhile.

Specifically, we would like to sharpen the estimate of the translational cost from Boolean formulae to branching programs. The class of balanced Boolean formulae of polynomial size is the same as NC^1; thus, this also provides improved simulation of NC^1 by branching programs, although in general, another polynomial factor will appear [Sp], just as in Barrington's simulation. For the class of balanced Boolean formulae, one referee points out that, in fact, people have thought that the quadratic bound of Barrington was optimal, before the current paper. Also, understanding how to encode information into simple groups may help shed light

[*]Received by the editors May 7, 1990; accepted for publication (in revised form) March 3, 1993. A preliminary version of this paper appeared in the Annual Symposium on Foundations of Computer Science 1989.

[†]Department of Computer Science, Princeton University, Princeton, New Jersey 08544 (jyc@cs.princeton.edu). The work of this author was supported by National Science Foundation grants CGR-8709818 and CCR-9057986.

[‡]Department of Computer Science, Princeton University, Princeton, New Jersey 08544 (rjl@cs.princeton.edu). The work of this author was supported by Defense Advanced Research Projects Agency and Office of Naval Research grants N00014-85-C-0456 and N00014-95-K-0465.

onto why no construction seems possible in solvable groups. These questions are intimately related to constant depth circuits with modulo p gates for a fixed finite set of primes p.

Unlike Barrington's construction, our branching programs in this paper use subprograms that are not restricted to have only two output values for intermediate steps, although the final output is still Boolean. The question of this so-called "weak" versus "strong" representation, namely, whether the less restrictive weak representation (with possibly more than two output values in intermediate steps) is more powerful than strong representation (with exactly two outputs), has been of interest in other upper/lower bound proofs. The present paper provides quantitative evidence that, for branching programs, indeed the less restrictive representation is more powerful.

There already has been some extension of our work since a preliminary version [CL] of this paper appeared in FOCS '89. Cleve [Cl] has proved one of our original conjectures: that indeed the exponent can be reduced to $1+\varepsilon$ at the cost of increased (depending on ε but still constant) width for the simulating branching program. The width achieved is rather large, and the theoretical question of whether the increase is necessary is still open.

Our lower bound takes a first step toward this direction. Using a Ramsey-type argument, we show that over any finite group there must be a superlinear increase in the length in general. We establish the first superlinear lower bound for a class of branching programs that compute functions with linear formula size.

2. Preliminaries. The branching program model is a generalization of the binary decision tree model. We consider a branching program as given by a leveled directed acyclic graph. It has a "start" node. All links point from nodes of one level to the next, and all nodes of the last level are terminal nodes, labeled *accept* or *reject*. Each nonterminal node is labeled by a Boolean variable and has two links labeled *true* or *false*. A setting of the Boolean variables determines a unique path from the start node to a terminal node. The complexity measures are its width and length (see [B] for details).

We assume our NC^1 circuits are given by balanced binary trees of depth $O(\log n)$. In this paper, we speak of "size" as the tree size, or formula size, and circuits as balanced Boolean formulae. In case the tree is not balanced, one can always balance the tree (at the cost of a polynomial blow up of the circuit size). It is also well known that any NC^1 circuit can be simulated by a Boolean formula of polynomial size. Note that the quadratic cost of Barrington's simulation is also measured in terms of the balanced binary tree size of the circuit. To compare the cost of our simulation with that of Barrington's, we just consider balanced Boolean formulae.

We use some elementary notions and results of group theory; they can be found in any standard textbook in that subject (e.g., [H]).

3. A subquadratic simulation. We first review Barrington's construction [B]. Suppose an AND gate, $f \wedge g$, is given. Let's represent the truth values of f and g (true or false) by some elements of a finite simple group in such a way that the AND gate $f \wedge g$ can be represented likewise. This can be accomplished as follows. Represent "$f =$ true" by some 5-cycle $a \in A_5$ and "$f =$ false" by the identity $1 \in A_5$, where A_n is the alternating group on n letters. Represent g by some $b \in A_5$ and 1 likewise. Call the representation α and β, respectively. Consider the commutator $[\alpha, \beta]$, namely, the representation $\alpha\beta\alpha^{-1}\beta^{-1}$, where say, $\alpha^{-1} = a^{-1}$ or 1 depending on "$f =$ true" or "$f =$ false," respectively. Because the identity 1 commutes with any element of the group, we see immediately that the commutator evaluates to 1 if $f \wedge g$ is false. The commutator construction is completed by choosing a and b carefully so that the commutator $[a, b]$ is yet another 5-cycle. The case with an OR gate is dual. It should be clear how this can be translated to a statement about width 5 branching programs.

We note that by using a commutator at every level of a Boolean formula, the overhead is quadratic, that is, for a Boolean formula of size s, the length of the simulating branching program using this construction has length s^2.

We shall first give some intuitive ideas leading toward a subquadratic simulation of a Boolean formula. Without loss of generality, we assume our Boolean formulae are given by binary trees of depth $O(\log n)$ and the AND and OR gates alternate at every level. We indicate the modifications needed when the Boolean formulae do not conform to the requirements of strict alternation.

Again, assume we are given $f \wedge g$. It is apparent that if we solely depend on commutators, we necessarily end up with a quadratic blowup. On the other hand, with any simulation (without global reconfiguration), the best one can hope for is linear (or close to) linear overhead. If we simply concatenate the branching programs representing f and g, in other words, if we multiply the respective representations instead of forming their commutator, we get the optimal linear overhead. However, over a group, this necessarily will produce multiple representations for $f \wedge g$ for (at least) one of the truth values. This cannot be continued indefinitely, as the sets of representatives for true and false must be kept disjoint. Thus, it appears that certain "cleanup" steps must be taken.

We show that by carefully combining multiplication with commutators, one can simulate a Boolean formula more efficiently, that is, with a simulation of subquadratic length.

Any balanced Boolean formula of depth d can be expressed as $u \wedge (v \vee (x \wedge y))$, $u \vee (v \vee (x \wedge y))$, $u \vee (v \wedge x \wedge y)$, or $u \vee (v \vee x \vee y)$ or their duals, where u is of depth $d-1$, v is of depth $d-2$, and x and y are of depth $d-3$, respectively.

Let $F = u \wedge (v \vee (x \wedge y))$. Let $a = (1\ 4\ 2\ 3\ 5)$, $b = (1\ 2\ 3\ 4\ 5)$, $c = (1\ 4\ 3\ 5\ 2)$, and $d = (1\ 5\ 2\ 4\ 3)$. The following relations can be easily verified:

$$[a,b] = aba^{-1}b^{-1} = (1\ 4\ 2\ 3\ 5)(1\ 2\ 3\ 4\ 5)(5\ 3\ 2\ 4\ 1)(5\ 4\ 3\ 2\ 1) = (1\ 2\ 5\ 3\ 4) = c^{-1},$$

$$[a,b^2] = ab^2a^{-1}b^{-2} = (1\ 4\ 2\ 3\ 5)(1\ 3\ 5\ 2\ 4)(5\ 3\ 2\ 4\ 1)(4\ 2\ 5\ 3\ 1) = (1\ 3\ 2),$$

$$[a,b^2][a,b]^{-1} = [a,b^2]c = (1\ 3\ 2)(1\ 4\ 3\ 5\ 2) = (1\ 5\ 2\ 4\ 3) = d,$$

$$[c,d] = (1\ 4\ 3\ 5\ 2)(1\ 5\ 2\ 4\ 3)(2\ 5\ 3\ 4\ 1)(3\ 4\ 2\ 5\ 1) = (1\ 2\ 4\ 5\ 3).$$

These identities have the following implications. If we represent the truth value of x and y by 1 (if it is true) and b (if it is false) and if we multiply the representations, we get 1 if $x \wedge y$ is true and b or b^2 otherwise. Now let's represent the truth value of v by 1 (if it is true) and a (if it is false) and form the commutator of the representations of v and of $x \wedge y$ and then form the product of this commutator with $[a,b]^{-1} = c = (1\ 4\ 3\ 5\ 2)$. We call this last multiplication a *shift*. As a result, we get c if $v \vee (x \wedge y)$ is true and 1 or d otherwise. Finally, if we represent u by d (if it is true) and 1 (if it is false) and form the commutator of the representations of $v \vee (x \wedge y)$ and u, we end up with a nice clean form: $[c,d] = (1\ 2\ 4\ 5\ 3)$ if $F = u \wedge (v \vee (x \wedge y))$ is true and 1 otherwise.

It should be clear that the shift step does not cost anything in the branching program length, as $[a,b] = (1\ 2\ 5\ 3\ 4)$ is a constant and can be absorbed in the previous transition step in the branching program involving a variable.

By recursively applying the construction, we can get the following estimate. Suppose x, y, v, and u respectively represent circuits of depth k, k, $k+1$, and $k+2$ and of size 2^k, 2^k, 2^{k+1}, and 2^{k+2}, and let ℓ_i be the respective lengths of the branching programs. Then we have

$$\ell_{k+3} = 2(\ell_{k+2} + 2(\ell_{k+1} + (\ell_k + \ell_k))).$$

Let $\ell_i = 2^i \ell'_i$, then

$$\ell'_{k+3} = \ell'_{k+2} + \ell'_{k+1} + \ell'_k.$$

We have the characteristic equation

$$\lambda^3 = \lambda^2 + \lambda + 1.$$

By Cardan's formula [J], we can solve this equation exactly, with the only real root

$$\lambda = \frac{1}{3}(1 + (19 + 3\sqrt{33})^{1/3} + (19 - 3\sqrt{33})^{1/3}) \approx 1.839\ldots.$$

Thus, the construction here achieves a subquadratic blowup of $O(s^{1+\log_2 \lambda}) \approx O(s^{1.879\ldots})$.

We also remark that had the function F been $u \vee (v \vee (x \wedge y))$, we would not have to carry out the shift step in the above construction. Notice that if we simply form the commutator of the representations of v and of $x \wedge y$, we get 1 if $v \vee (x \wedge y)$ is true and $[a, b] = (1\ 2\ 5\ 3\ 4)$ or $[a, b^2] = (1\ 3\ 2)$ otherwise. We represent u by 1 or $(1\ 3\ 2)^{-1}(1\ 2\ 5\ 3\ 4) = (1\ 5\ 3\ 2\ 4)$ (in case it is true or false) and form the commutator of the representations of u and of $v \vee (x \wedge y)$, and we have 1 if F is true and

$$[(1\ 5\ 3\ 2\ 4), (1\ 2\ 5\ 3\ 4)] = [(1\ 5\ 3\ 2\ 4), (1\ 3\ 2)] = (1\ 2\ 3\ 4\ 5)$$

if F is false.

Moreover, given $F = u \vee (x \wedge (y \wedge z))$ or $F' = u \vee (x \vee (y \vee z))$, we have the following construction that performs even better. (These 4 cases are exhaustive, using duality.)

Let

$$a = (1\ 3\ 4\ 2\ 5), \qquad a' = (1\ 4\ 2\ 3\ 5), \qquad b = (1\ 4\ 5\ 2\ 3),$$

$$b' = (1\ 2\ 4\ 5\ 3), \qquad c = (1\ 2\ 5\ 3\ 4), \qquad c' = (1\ 3\ 4\ 2\ 5),$$

and

$$\gamma = aa' = bb' = cc' = aba'b' = aca'c' = bcb'c' = (1\ 5\ 4\ 3\ 2),$$

$$\delta = abca'b'c' = (1\ 3\ 4\ 2\ 5).$$

If we consider $a_x b_y c_z a'_x b'_y c'_z$, where $a_x = 1$ if x is true and $a_x = a$ if x is false, similarly for the others, we have:

$$a_x b_y c_z a'_x b'_y c'_z = \begin{cases} 1 & \text{if } x = y = z = 1, \\ \delta = (1\ 3\ 4\ 2\ 5) & \text{if } x = y = z = 0, \\ \gamma = (1\ 5\ 4\ 3\ 2) & \text{otherwise.} \end{cases}$$

For $F = u \vee (x \wedge y \wedge z)$, we represent u by 1 or $\gamma^{-1}\delta = (1\ 5\ 3\ 2\ 4)$ (true or false, respectively), and we have

$$[\gamma, \gamma^{-1}\delta] = [\delta, \gamma^{-1}\delta] = (1\ 2\ 4\ 5\ 3).$$

For $F' = u \vee (x \vee y \vee z)$, we represent u by 1 or γ (true or false, respectively), and we have

$$[\gamma, \delta] = (1\ 4\ 2\ 3\ 5).$$

(This is even slightly better in complexity with a growth of $s^{1.842\cdots}$. We omit the details. For simplicity we do not consider the strict requirements on alternation and multiple fan-in in the remainder of this paper to avoid tedious case analysis. Thus, we assume our tree is balanced and alternate at each level.)

4. Iterated simulation. In this section we indicate an improvement of the construction given in the last section. Again let us look at $F = u \wedge (v \vee (x \wedge y))$. The simple observation is that in the above construction when we have represented $v \vee (x \wedge y)$ by c if it is true and 1 or d if it is false, we can relax somewhat on the representation of u. More specifically, if we have $u = u_1 \vee u_2$, we may represent both u_1 and u_2 by d (true) and 1 (false) and then simply form the multiplication of the representation. This gives us the representation for u as d or d^2 when u is true and 1 otherwise. Now because all powers of d commute, we still get 1 if F is false. However, if F is true we get either

$$[c, d] = (1\,2\,4\,5\,3) \quad \text{or} \quad [c, d^2] = (2\,3\,4).$$

We note that the situation is exactly the same as what we have before in $v \vee (x \wedge y)$. More exactly, after a conjugation, we have the exact dual case as in the representation for $v \vee (x \wedge y)$ just before the shift. (We recall that a shift is free.) Specifically, the pairs a, b and c, d are conjugate to each other by the same conjugation. Let $\alpha = (1\ 4\ 2)$. Conjugation by α gives $c^\alpha = \alpha^{-1} c \alpha = (4\ 2\ 3\ 5\ 1) = a$ and $d^\alpha = (4\ 5\ 1\ 2\ 3) = b$ and thus $[c, d]^\alpha = [a, b]$ and $[c, d^2]^\alpha = [a, b^2]$. (Just like a shift, a conjugation is merely a renaming of the states of the branching program, and thus it is free. But unlike a shift, a conjugation is a simultaneous renaming of the input as well as the output states, and thus in the actual construction of the branching program, the conjugation does not even need to be carried out. We include them in the exposition only to show that inductively the construction can be carried out.)

Of course, now it is natural to iterate on this. Let h_i and h'_i be circuits of depth $k + i$ and $g_{i+1} = h_i \circ h'_i$, where $\circ = \wedge$ if i is even and $\vee$ if i is odd, and $0 \le i \le m - 2$ for some $m \ge 2$. Let f_1 be a circuit of depth $k + 1$ and $f_{i+2} = f_{i+1} \circ g_{i+1}$, where $\circ = \vee$ if i is even and $\wedge$ if i is odd. Finally, let $F_{m+1} = f_m \circ g_m$, where g_m is a circuit of depth $k + m$ and $\circ = \wedge$ if m is even and $\vee$ if m is odd. Our simulation will carry multiple values as we move up the tree. But in the last step we bring it back to the form of single-valued representation in both cases, true or false.

We have the following recurrence relation where ℓ_i denotes the length of a branching program simulating a circuit of depth i and size 2^i.

$$\ell_{k+m+1} = 2(\ell_{k+m} + 2(2\ell_{k+m-2} + \cdots + 2(2\ell_{k+1} + 2(2\ell_k + \ell_{k+1}))\ldots)),$$

or

$$\ell_{k+m+1} = 2^1 \ell_{k+m} + \sum_{i=3}^{m+1} 2^i \ell_{k+m+1-i} + 2^m \ell_{k+1}.$$

Simplifying a bit, let $\ell_i = 2^i \ell'_i$, we have

$$\ell'_{k+m+1} = \ell'_{k+m} + \sum_{i=3}^{m+1} \ell'_{k+m+1-i} + \ell'_{k+1},$$

which has a characteristic equation

$$\chi_{m+1}(\lambda) = \lambda^{m+1} - \left(\lambda^m + \sum_{i=3}^{m+1} \lambda^{m+1-i} + \lambda\right) = 0.$$

LEMMA 4.1. *Each χ_i has a unique positive root $\lambda_i, i \geq 3$. Furthermore, the set $\{\lambda_i\}$ forms a monotonic decreasing sequence with limit*

$$\lambda_* = \frac{1}{3}\left(2 + \left(\frac{25}{2} + \frac{3}{2}\sqrt{69}\right)^{1/3} + \left(\frac{25}{2} - \frac{3}{2}\sqrt{69}\right)^{1/3}\right) \approx 1.7549\ldots.$$

This implies the following theorem.

THEOREM 4.2. *Every balanced Boolean formula of size s can be simulated by a width 5 permutation branching program of length $O(s^{1+\log_2 \lambda_* + o(1)}) \approx O(s^{1.811\ldots})$. Thus, exponent of the simulation is at most 1.811 ....*

Proof of the lemma. It is easy to verify that $\chi_3(\lambda) = \lambda^3 - \lambda^2 - \lambda - 1$ has a unique real root at $\lambda_3 = \frac{1}{3}(1 + (19 + 3\sqrt{33})^{1/3} + (19 - 3\sqrt{33})^{1/3}) \approx 1.839\ldots$ by Cardan's formula.

Furthermore, χ_3 has a unique local maximum at $-\frac{1}{3}$ of $\chi_3(-\frac{1}{3}) = -\frac{22}{27}$ and a unique local minimum at 1 of $\chi_3(1) = -2$; thus, $\chi_3(\lambda) < 0$ for $\lambda < \lambda_3$ and monotonic increasing for $\lambda > 1$.

Let

$$\lambda_* = \frac{1}{3}\left(2 + \left(\frac{25}{2} + \frac{3}{2}\sqrt{69}\right)^{1/3} + \left(\frac{25}{2} - \frac{3}{2}\sqrt{69}\right)^{1/3}\right) \approx 1.7549\ldots$$

be the unique real root of $\chi_*(\lambda) = (\lambda - 1)^2\lambda - 1$ (Cardan's formulas). Inductively, we assume that χ_m has a unique positive real root λ_m, $\lambda_* < \lambda_m < \cdots < \lambda_3$, and, moreover, χ_m is monotonic increasing for $\lambda > \lambda_*$, and $\chi_m(\lambda) < 0$ for $0 < \lambda < \lambda_m$.

Let

$$\Delta_m(\lambda) = \chi_{m+1}(\lambda) - \chi_m(\lambda) = \lambda^{m-2}((\lambda - 1)^2\lambda - 1) = \lambda^{m-2}\chi_*(\lambda). \tag{1}$$

It follows, by the definition of λ_*, that $\Delta_m(\lambda) > 0$ for $\lambda > \lambda_*$ and $\Delta_m(\lambda) < 0$ for $0 < \lambda < \lambda_*$. Furthermore, it can be verified directly that $\Delta'_m(\lambda) > 0$ for $\lambda > \lambda_*$.

Thus, $\chi_{m+1}(\lambda) = \chi_m(\lambda) + \Delta_m(\lambda) < 0$ for $0 < \lambda \leq \lambda_*$ and monotonic increasing for $\lambda > \lambda_*$; therefore, $\chi_{m+1}(\lambda)$ has a unique positive real root λ_{m+1}, $\lambda_* < \lambda_{m+1} < \lambda_m < \cdots < \lambda_3$. Moreover, $\chi_{m+1}(\lambda) < 0$ for $0 < \lambda < \lambda_{m+1}$. The induction is completed.

It follows that $\lim \lambda_m$ exists. Let $\lambda^* = \lim \lambda_m$. We claim $\lambda^* = \lambda_*$. Suppose not; thus, $\lambda_* < \lambda^*$. By (1), $\chi_m(\lambda_*)$ is some negative constant $-c$ independent of m.

It is easy to show, by directly taking and estimating the derivative on χ_{m+1}, that for all $\lambda > \lambda_*$,

$$\chi'_{m+1}(\lambda) \geq \lambda^m + m\chi_m(\lambda) \geq \lambda_*^m - mc,$$

which tends to infinity as $m \to \infty$.

However, by the intermediate value theorem, for all m, there exists $\tilde{\lambda}_m \geq \lambda_*$, such that the value of $\chi'_m(\lambda)$ at $\tilde{\lambda}_m$ is bounded above:

$$\chi'_m(\tilde{\lambda}_m) = \frac{\chi_m(\lambda_m) - \chi_m(\lambda_*)}{\lambda_m - \lambda_*} \leq \frac{c}{\lambda^* - \lambda_*}.$$

This contradiction shows that $\lim \lambda_m = \lambda_*$. □

5. A superlinear lower bound. We turn our attention to lower bounds in this section. In particular, we ask what is the minimum length of a branching program for functions with linear formula size. In this section we present a lower bound of $\Omega(n \log\log n)$ for the translational cost from NC^1 circuits to permutation branching programs over any finite group. This bound is the first known lower bound for the translational cost.

Several super linear lower bounds are known for branching programs. Chandra, Furst, and Lipton [CFL] showed that the function $\sum_i x_i = n/2$ requires $\Omega(nw(n))$ in the length of any bounded-width branching program, where $w(n)$ is the inverse function of the van der Waerden numbers. Pudlák [Pu] proved an $\Omega(n \log\log n/\log\log\log n)$ lower bound for threshold functions. A lower bound of $\Omega(n \log n)$ for symmetric Boolean functions was achieved by Ajtal et al [Aj]. However, it is not known whether any lower bound applies to functions with linear circuit size. See also [B2] and [BT] for lower bounds for branching programs over specific groups such as S_3 and some solvable groups. Our bound is the first over an arbitrary finite group.

Our lower bound applies only to permutation branching programs and not to (unrestricted) branching programs in general. In fact, we establish the $\Omega(n \log\log n)$ lower bound for the AND function $\bigwedge_{i+1}^n x_i$, which has a trivial width two branching program of length n. Our proof is Ramseyian; a similar method has been used in [AM].

THEOREM 5.1. *Any permutation branching program computing the function* AND $\bigwedge_{i=1}^n x_i$ *requires length* $\Omega(n \log\log n)$.

Let a permutation branching program over a finite group G be given that computes the logical AND function of n Boolean variables $x_1, x_2, \ldots, x_n$.

We assume the branching program has the following normal form

$$BP(x_1, x_2, \ldots, x_n) = g_1(x_{i_1})g_2(x_{i_2})\ldots g_L(x_{i_L}).$$

(See [B2].) Thus, for each step, the transition depends on one Boolean variable x. Furthermore, if x is true, then the transition is identity $g_k(1) = 1$, and if x is false, then the transition is some element of the group G, $g_k(0) = g_k \in G$. Without loss of generality, every permutation branching program can be brought to this form without any increase of length.

We prove a lower bound on the length L of the branching program. Let n_k denote the number of variables that occur exactly k times in the branching program, then $\sum_{k\ge 1} n_k = n$ and $\sum_{k\ge 1} kn_k = L$. If $\sum_{k\ge(\log_3\log_3 n)/2} n_k \ge n/2$, then we are done: $L \ge (n \log\log n)/4$. We assume $\sum_{k<(\log_3\log_3 n)/2} n_k \ge n/2$, and thus there exists $k < (\log_3\log_3 n)/2$, $n_k \ge n/\log_3\log_3 n$. Set all other variables to true, we get a branching program on n_k variables, each variable appears exactly k times, where $k < (\log_3\log_3 n)/2$. Denote $N = n_k$.

Let ℓ_{ij} be the location of the jth appearance of the ith variable x_i. By renaming variables if necessary, we assume that the first appearances are in order, that is,

$$\ell_{11} < \ell_{21} < \cdots < \ell_{N1}.$$

Consider the second appearances of these variables. We select a subset of the variables of cardinality $\ge N^{1/3}$, such that the second appearances of these variables are nicely correlated to the first appearances of the same variables. More precisely, we show that there exists a subsequence $i_1 < i_2 < \cdots < i_m$ of $1, 2, \ldots, N$, where $m \ge N^{1/3}$, such that one of the following three alternatives hold:

(1) $\ell_{i_1 2} > \ell_{i_2 2} > \cdots > \ell_{i_m 2}$, or

(2) $\ell_{i_1 2} < \ell_{i_2 2} < \cdots < \ell_{i_m 2}$ and $\ell_{i_1 2} > \ell_{i_m 1}$, or

(3) $\ell_{i_1 1} < \ell_{i_1 2} < \ell_{i_2 1} < \ell_{i_2 2} < \cdots < \ell_{i_m 1} < \ell_{i_m 2}$.

For any sequence of integers, it is well known that we can first obtain either a monotonic decreasing subsequence of length $m' \ge N^{1/3}$ or a monotonic increasing subsequence of length

$m' \geq N^{2/3}$ of the sequence $\ell_{12}, \ell_{22}, \ldots, \ell_{N2}$. If the subsequence is monotonic decreasing, then the first altenative holds.

Suppose it is monotonic increasing. By setting all the other variables to true and by renaming the variables, we may assume the subsequence is $\ell_{12} < \ell_{22} < \cdots < \ell_{m'2}$. For $1 \leq i \leq m'$, let $p(i) = \max\{p \mid \ell_{i2} > \ell_{p1}\}$, that is, the relative place of ℓ_{i2} in the first sequence. Clearly, $p(i) \geq i$ for all i, and $1 \leq p(1) \leq \cdots \leq p(m') = m'$. Now we ask the key question: Is there an i, $1 \leq i \leq m'$, such that $p(i) - i \geq \sqrt{m'}$? If so, then we choose the subset as those with indices between i and $p(i)$:

$$\ell_{i2} < \ell_{i+12} < \cdots < \ell_{p(i)2},$$

and $\ell_{i2} > \ell_{p(i)1}$. Thus, the second alternative holds.

Now suppose the answer is no, that is, for all i, $p(i) - i < \sqrt{m'}$. Then we are going to select our subsequence greedily as follows: Let $q(i) = p(i) + 1$, and

$$i_j = q^{(j-1)}(1), \qquad 1 \leq j \leq \sqrt{m'},$$

where $f^{(k)}$ denotes the kth iterate of a function f. That all i_j, $1 \leq j \leq \sqrt{m'}$, are no greater than m', and thus well defined, is a consequence of our hypothesis for all i, $p(i) - i < \sqrt{m'}$. This implies the third alternative and completes the proof of our claim.

Now one can iterate this process. Suppose s iterations are done, and $N' \geq N^{1/3^s}$ variables remain. Inductively, every successive sequence (consisting of the rth occurrence, $1 < r \leq s$, of the remaining variables) is related to its previous sequences similarly as in that between the first and the second sequence just shown. In fact, we can group together those successive sequences related as in alternative 3, namely

$$\ell_{i_1 r} < \ell_{i_1 r+1} < \cdots < \ell_{i_1 r'} < \ell_{i_2 r} < \ell_{i_2 r+1} < \cdots < \ell_{i_2 r'} < \cdots < \ell_{i_m r} < \ell_{i_m r+1} < \cdots < \ell_{i_m r'},$$

or its reverse,

$$\ell_{i_1 r} > \ell_{i_1 r+1} > \cdots > \ell_{i_1 r'} > \ell_{i_2 r} > \ell_{i_2 r+1} > \cdots > \ell_{i_2 r'} > \cdots > \ell_{i_m r} > \ell_{i_m r+1} > \cdots > \ell_{i_m r'}.$$

Call any maximal such internal $\{j | r \leq j \leq r'\}$ a block.

We focus on the last block, say starting from $t + 1$ to s. By renaming the variables, we assume inductively $x_1, x_2, \ldots, x_{N'}$ are remaining, and

$$\ell_{1,t+1} < \ell_{1,t+2} < \cdots < \ell_{1s} < \ell_{2,t+1} < \ell_{2,t+2} \cdots < \ell_{2s} < \cdots < \ell_{N',t+1} < \ell_{N',t+2} < \cdots < \ell_{N's},$$

and, if $t > 0$, the tth and $t + 1$st sequence are related as in alternative (1) or (2). (The other alternative of reversing all $<$ to $>$ in the above ℓ-sequence is symmetric.)

We will select a subsequence of $\ell_{1s+1}, \ldots, \ell_{N's+1}$, indexed by $i_1 < i_2 < \cdots < i_m$, where $m \geq N'^{1/3}$, such that one of the following three alternatives is true:

(1) $\ell_{i_1 s+1} > \ell_{i_2 s+1} > \cdots > \ell_{i_m s+1}$, or

(2) $\ell_{i_1 s+1} < \ell_{i_2 s+1} < \cdots < \ell_{i_m s+1}$ and $\ell_{i_1 s+1} > \ell_{i_m s}$, or

(3) $\ell_{i_1 t+1} < \ell_{i_1 t+2} < \cdots < \ell_{i_1 s} < \ell_{i_1 s+1} < \ell_{i_2 t+1} < \ell_{i_2 t+2} < \cdots < \ell_{i_2 s} < \ell_{i_2 s+1} < \cdots < \ell_{i_m t+1} < \ell_{i_m t+2} < \cdots < \ell_{i_m s} < \ell_{i_m s+1}$.

The proof is identical to the base case, and we will not repeat it here.

Because $k < (\log_3 \log_3 n)/2 < \log_3 \log_3 N$, the process can be iterated k times. We end up with k sequences each of which has $n' \geq N^{1/3^k}$ variables, and the branching program computes the AND function of these variables (all others are set to true). Moreover, all adjacent blocks of sequences are related in one of three ways as above. Clearly, any block can be collapsed to just one sequence. After the collapse (and renaming the variables) we have $k' \leq k$ sequences, and the branching program looks like

$$S_1 S_2 \dots S_{k'},$$

where each S_j is either

$$g_{1j}(x_1) g_{2j}(x_2) \dots g_{n'j}(x_{n'}),$$

or

$$g_{n'j}(x_{n'}) g_{n'-1j}(x_{n'-1}) \dots g_{1j}(x_1).$$

Consider the map F from $1 \leq i \leq n'$ to $G^{k'}$:

$$i \mapsto \langle s_1(i), s_2(i), \dots, s_{k'}(i) \rangle,$$

where

$$s_j(i) = g_{1j} g_{2j} \dots g_{ij},$$

in the first case of S_j, or

$$s_j(i) = g_{ij} g_{i-1,j} \dots g_{1j},$$

in the second case of S_j. (Recall that $g_{\sigma j} = g_{\sigma j}(0) \in G$.)

Because $k < (\log_3 \log_3 n)/2$ and $N \geq \frac{n}{\log_3 \log_3 n}$, it follows easily that[1] $n' \geq N^{1/3^k} > |G|^{k'}$; thus, $F(i) = F(i')$ for some $i < i'$ by the pigeonhole principle. Therefore, for all j, $g_{i+1\,j} \cdots g_{i'j} = 1 \in G$, or $g_{i'j} \cdots g_{i+1\,j} = 1 \in G$, which ever the case may be. This implies that the original branching program evaluates to 1 when all variables (after renaming) between x_{i+1} and $x_{i'}$ are set to false and others set to true. Hence it does not compute the AND function. This completes the proof of our lower bound.

The method we used here to prove our $\Omega(n \log \log n)$ lower bound has been used by Barrington and Straubing to obtain several other lower bounds [BS].

6. Open problems. There are many unanswered questions raised here. We mentioned in the beginning of the paper that Cleve [Cl] has proved the following theorem, which was the first conjecture in the preliminary version of this paper.

THEOREM 6.1 (Cleve). *For any $\epsilon > 0$, an NC^1 circuit of size s can be simulated by a width $2^{2^{O(1/\epsilon)}}$ (permutation) branching program of length $O(s^{1+\epsilon})$.*

In view of this, it is natural to define *Barrington's constants* $\mathcal{B}_k$, for each width k, that is, $\mathcal{B}_k$ is the infimum of $\mathcal{B}$ such that any NC^1 circuit of size s can be simulated by a width k (permutation) branching program of length $O(s^{\mathcal{B}})$. We conjecture that these Barrington's constants are greater than one (and hence nontrivial).

[1]Let $c = |G|$, a constant. Because $k < (\log_3 \log_3 n)/2$, $3^k < \sqrt{\log_3 n}$. As $N \geq \frac{n}{\log_3 \log_3 n}$,

$$\begin{aligned} N \cdot \log_3 \log_3 n &\geq n \\ &\geq c^{\frac{\sqrt{\log_3 n}\, \log_3 \log_3 n}{2}} + \log_c \log_3 \log_3 n \\ &> c^{\frac{\log_3 \log_3 n}{2} \cdot 3^k} \cdot \log_3 \log_3 n. \end{aligned}$$

Thus, $N^{1/3^k} > c^{\frac{\log_3 \log_3 n}{2}} > c^k \geq c^{k'}$, for large n.

CONJECTURE 6.2. *Width k (permutation) branching programs simulating any NC^1 circuit of size s requires length $\Omega(s^{1+\epsilon_k})$, for some $\epsilon_k > 0$.*

Our lower bound in §5 can be viewed as the first step toward settling this conjecture. If this conjecture is true, one may further inquire the exact order of growth of these *Barrington's constants $\mathcal{B}_k$*. The best known bound for $\mathcal{B}_5$ is 1.811 .... It is not clear what to expect in general. A tight simulation of circuits by branching programs could offer the possibility of proving lower bounds for circuits size, whereas a reasonable width could be valuable in hardware design pertaining reconfigurable chips.

Acknowledgments. We thank Sandeep Bhatt, Walter Feit, Mike Fischer, Merrick Furst, Roger Howe, Herb Scarf, and George Seligman for helpful conversations. We thank the two anonymous referees for many comments.

REFERENCES

[Aj] M. AJTAL, L. BABAI, P. HAJNAL, J. KOMLÓS, P. PUDLÁK, V. RÖDL, E. SZEMERÉDI, AND G. TURÁN, *Two lower bounds for branching programs*, in Proc. 18th ACM STOC, Berkeley, CA, 1986, pp. 30–38.

[AM] N. ALON AND W. MAASS, *Meanders and their applications in lower bounds arguments*, JCSS, 38 (1988), pp. 118–129.

[B] D. BARRINGTON, *Bounded-width polynomial-size branching programs recognizes exactly those languages* in NC^1, JCSS, 38 (1990) pp. 150–324.

[B2] ———, *Width-3 permutation branching programs*, Tech. memorandum TM-291, MIT Laboratory for Computer Science, Cambridge, MA, 1985.

[BS] D. BARRINGTON AND H. STRAUBING, *Superlinear lower bounds for bounded-width branching programs*, in Proc. 6th Structure in Complexity Theory Conference, IEEE Computer Society Press, Alamitos, CA, 1991, pp. 305–313.

[BT] D. BARRINGTON AND D. THÉRIEN, *Non-uniform automata over groups*, Lecture Notes in Comp. Sci., Springer-Verlag, 267, 1987, pp. 163–173.

[BC] M. BEN-OR AND R. CLEVE, *Computing algebraic formulas using a constant number of registers*, in Proc. 20th ACM STOC, Chicago, Illinois, 1988, pp. 254–257.

[BDFP] A. BORODIN, D. DOLEV, F. E. FICH, AND W. PAUL, *Bounds for width-2 branching programs*, SIAM J. Comput., 15 (1986), pp. 549–560.

[CL] J. CAI AND R. LIPTON, *Subquadratic simulations of circuits by branching programs*, in Proc. 30th IEEE FOCS IEEE Computer Society Press, Alamitos, CA, 1989, pp. 568–573.

[CFL] A. CHANDRA, M. FURST, AND R. J. LIPTON, *Multiparty protocols*, in Proc. 15th ACM STOC, the ACM Inc., 11 West 42nd Street, New York, 1983, pp. 94–99.

[Cl] R. CLEVE, *Towards optimal simulations of formulas by bounded-width programs*, in Proc. STOC, 22 (1990), Baltimore, MD, pp. 271–277.

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

[H] M. HALL, *The Theory of Groups*, MacMillan, New York, 1959.

[J] N. JACOBSON, *Basic Algebra*, Vol 1, W. H. Freeman and Company, New York, 1985.

[Pi] N. PIPPENGER, *On simultaneous resource bounds (preliminary version)*, in Proc. 20th IEEE FOCS, IEEE Computer Society Press, Alamitos, CA, 1979, pp. 307–311.

[Pu] P. PUDLÁK, *A lower bound on complexity of branching programs*, 11th MFCS, Lecture Notes in Comput. Sci., Springer-Verlag, vol. 176, pp. 480–489.

[Sp] P. SPIRA, *On time-hardware complexity tradeoffs for Boolean functions*, in Proc. 4th Hawaii Symposium on System Sciences, North Hollywood, CA, Western Periodicals Co., 1971, pp. 525–527.