BOUNDS FOR WIDTH TWO BRANCHING PROGRAMS

Allan Borodin*
Computer Science, University of Toronto

Danny Dolev*
Computer Science, Hebrew University, Jerusalem

Faith E. Fich*
Computer Science, University of Berkeley

Wolfgang Paul
IBM Research Laboratory
San Jose, California 95193

1. INTRODUCTION

Branching programs for the computation of Boolean functions were first studied in the Master's thesis of Masek.[7] In a rather straightforward manner they generalize the concept of a decision tree to a *decision graph.* Formally, they can be defined as acyclic labelled diagraphs with the following properties.

(i) There is exactly one source.
(ii) Every node has outdegree at most 2.
(iii) For every node v with outdegree 2, one of the edges leaving v is labelled by a Boolean variable x_i and the other edge is labelled by its complement $\bar{x}_i$.
(iv) Every sink is labelled by 0 or 1.

Let P be a branching program with edges labelled by the Boolean variables, $x_1,...,x_n$ and their complements. Given an input $\mathbf{a}=(a_1,...,a_n)\in\{0,1\}^n$, program P computes a *function value* $f_P(\mathbf{a})$ in the following way. The computation starts at the source. If the computation has reached a node v and if only one edge leaves v, then the computation proceeds via that edge. If 2 edges, with labels x_i and $\bar{x}_i$, leave v, then the computation proceeds via the edge labelled x_i if $a_i=1$, and via the edge labelled $\bar{x}_i$ otherwise. Once the computation reaches a sink, the computation ends and $f_P(\mathbf{a})$ is defined to be the label of that sink.

The nodes of P play the role of states or configurations. In particular, sinks play the role of final states or stopping configurations. We call sinks *accepting* if they are labelled 1 and *rejecting* otherwise.

The *length* of program P is the length of the longest path in P. Following Cobham,[2] *capacity* of the program is defined to be the logarithm to the base 2 of the number of nodes in P. Length and capacity are lower bounds on time and space requirements for any reasonable model of sequential computation. Clearly, any n-variable Boolean function can be computed by a branching program of length n *if* the capacity is not constrained (*e.g.*, consider a complete binary tree with 2^n leaves, one for each input). Since space lower bounds in excess of log n remain a fundamental challenge, we consider restricted branching programs in the hope of gaining insight into this problem and the closely related problem of time-space trade-offs.

We call a diagraph *levelled* if its nodes can be partitioned into levels $L_0, L_1...$ such that, for all i, an edge leaving a node in level L_i ends at a node in level L_{i+1}. The *width* of such a graph is the maximum number of nodes at any level. Every branching program can easily be transformed into a levelled program that computes the same function, has the same length, and has at most twice the capacity of the original program.[1] Therefore, if we are interested in asymptotic bounds on length and capacity, then, without loss of generality, we can assume branching programs to be levelled. In this way, the level of a node represents the time needed to reach the node starting from the source.

For any node v in a branching program P, let $I_P(v)$ be the set of inputs $\mathbf{a}$ such that the computation of P given $\mathbf{a}$ reaches v. If P is levelled, then, for each i, the system of sets $\{I_P(v): v\in L_i\}$ is a partition of the input that mirrors the knowledge (or lack of knowledge) about the input at the level L_i.

The notation #S is used to denote the cardinality of the set S. For $\mathbf{a}\in\{0,1\}^n$, let $S(\mathbf{a})\subseteq\{1,...,n\}$ be the set of indices i such that $a_i=1$. The *weight* $w(\mathbf{a})$ of $\mathbf{a}$ is $\#S(\mathbf{a})$. The n-ary functions $E^n_{h,k}$ are defined by

$$E^n_{h,k}(\mathbf{a}) = 1 \quad \text{iff} \quad h\le w(\mathbf{a})\le k\ .$$

We write E^n_k for $E^n_{k,k}$ and drop the superscript n if the number of arguments is clear from the context. $E^n_{\lceil n/2\rceil,n}$ is called the *majority function* and $E^n_{\lceil n/2\rceil}$ is called the *exactly-half function.* Masek[7] made two observations concerning the latter function.

(i) If the computation only looks at each input variable once, then, for $i\le n/2$, level L_i must contain i+1 nodes. Hence, a branching program of minimum length, n, must have capacity at least 2 $\log_2 n$+ a constant. This lower bound can be achieved by a branching program which counts the number of input variables that have value 1.

*This work was done while the author was a visitor at IBM, San Jose.

(ii) By modular counting, the capacity requirement could be reduced at the expense of increased time.

In fact, both the exactly-half function and the majority function posses algorithms which *simultaneously* achieve capacity $0(\log n)$ and length $0(n)$. However, if we severely restrict the width of the programs, we begin to observe some potentially strange behavior. This we hope gives insight into how computations become confused (and hence prolonged) if we do not allow enough states.

Independently, Furst, Saxe and Sipser[6] were led to the study of such programs in trying to establish the relativized NP-hierarchy. It is observed that Boolean $(\wedge,\vee,\neg)$-circuits with unbounded fan-in of depth d and size s can be simulated by branching programs of width d+1 and length s^d. Furst, Saxe and Sipser establish a very nontrivial lower bound: constant depth circuits computing a parity function of n variables must be of size nonpolynomial in n. A parity function is a function of the form $x_1\oplus\ldots\oplus x_n$ or $\neg(x_1\oplus\ldots\oplus x_n)$ where $n\geq 1$ and $x_1,\ldots,x_n$ are distinct Boolean variables. A Boolean function is a parity function if and only if it changes value when one of its arguments changes value. Similar lower bounds are established for other functions (including the majority function) based on suitable reductions of parity function to these functions.

Clearly a parity function can be computed by branching programs of width 2 and length n. But what about the majority function? It has recently been shown by Chandra, Furst and Lipton[3] that the majority function cannot be computed in bounded width and linear length. We would like to show that this function or the closely related exactly-half function cannot be computed in bounded width and polynomial length. In fact, we conjecture that, in these cases, bounded width implies exponential length. Thus far we have only been able to establish much weaker results, dealing with programs of width 2. Even so, we found that width 2 branching programs offer some surprises and challenges. This is unlike the situation for depth 2 circuits which are characterized by disjunctive normal form (DNF) and conjunctive normal form (CNF).

The *formula size* of a Boolean function is the minimum number of occurrences of literals in any Boolean formula (over the basis of all binary operations) which describes the function. Although most Boolean functions of n variables have formula size $\Omega(2^n/\log n)$, the best lower bound for specific examples is $\Omega(n^2/\log n)$ due to Neciporuk.[8] Fischer, Meyer and Paterson[5] have shown that most symmetric Boolean functions, including $E_{k,n}$ for k, $n-k=\Omega(n/\log n)$) have formula size $\Omega(n\log n)$.

We will show that lower bounds for formula size directly translate into lower bounds for the length of width 2 branching programs. Precisely because width 2 branching programs constitute a more restrictive model, there is hope that better lower bounds can be more easily achieved.

2. STRICT BRANCHING PROGRAMS OF WIDTH 2 AND THEIR CHARACTERIZATION

In order to understand branching programs of width 2 (W_2-programs), it is useful to study even more restrictive models of computation. Specifically, we must indicate whether or not we allow accepting or rejecting nodes during the computation.

We call a width 2 branching program *monotone* if it has exactly one rejecting node. A *strict* width 2 branching program has exactly one accepting node and exactly one rejecting node.

In monotone programs no intermediate rejecting nodes are allowed. In strict programs neither intermediate accepting nodes nor intermediate rejecting nodes are allowed. Any W_2-program with t sinks can be decomposed into t-1 strict W_2-programs in an obvious way.

By considering DNF, it is clear that every Boolean function is computable by a monotone W_2-program. The usual counting argument establishes the existence of functions whose branching programs have length exponential in n if the width is bounded by a polynomial in n. However, we are looking for lower bounds for effectively defined functions. (A sequence of n-ary Boolean functions f_n, n=1,2,... is effectively defined if $\bigcup_n f_n^{-1}(1)\in NP$.)

It is not a priori clear, whether strict W_2-programs are powerful enough to compute every Boolean function. Let SW_2 denote the class of functions computable by strict W_2-programs. In this section we give a characterization of SW_2 and use it to show that some simple functions are not in SW_2. For instance E_1^4 is not in SW_2. It is somewhat surprising that E_1^3 and E_2^4 are in SW_2. The lower bounds which we derive later are based on the results and techniques developed here. Our characterization reveals some of the subtleties and the power of strict W_2-programs. It is not surprising that parity should play a prominent role here. We will occasionally abuse notation and identify Boolean formulas and the Boolean functions defined by the formulas.

Theorem 1. SW_2 is the smallest class $\mathscr{C}$ of Boolean functions containing the constant functions 0 and 1, and the projections $g_i(x_1,\ldots,x_n)=x_i$, for all i and n, and which has the following closure properties: if $f\in\mathscr{C}$ and a, b are literals then

R1) $f\oplus a\in\mathscr{C}$

R2) $f\wedge a\in\mathscr{C}$

R3) $f\wedge(a\oplus b)\in\mathscr{C}$

Proof: The constant functions and projections are obviously in SW_2. Let f be computed by a strict W_2-program P with accepting node u and rejecting node v. In order to compute $f\oplus a$, $f\wedge a$, and $f\wedge(a\oplus b)$ extend P by the program segments shown in Figures 1(i), 1(ii), and 1(iii), respectively. Thus SW_2 has the desired closure properties.

It remains to show $SW_2\subseteq\mathscr{C}$. Clearly $\mathscr{C}$ contains all functions that can be computed by a strict W_2-program of length 1. Now suppose $\mathscr{C}$ contains all functions that can be computed by a strict W_2-program of length n. Consider any function g computed by a strict W_2-program P of length n+1. The last two levels (*i.e.*, levels n and n+1) of P are illustrated in Figure 2, with a and b denoting (not necessarily distinct) literals or constants.

For convenience, we also allow the edges leaving a node of outdegree 2 to have the labels 0 and 1. The intended interpretation is that the node has outdegree 1, the edge labelled 0 is absent, and the edge labelled 1 is present (and unlabelled).

Let f be the function computed by the program obtained from P by deleting level n+1, making u accepting and v rejecting. Then $g = [f \wedge (a \oplus b)] \oplus \bar{b}$. □

The proof of Theorem 1 gives a constructive procedure for obtaining a formula for the Boolean function computed by a strict W_2-program, given the program. This enables us to relate the two complexity measures, formula size and program length.

Theorem 2: Any Boolean function that can be computed by a W_2-program of length L has formula size at most 3L.

Proof: Any W_2-program P can be uniquely decomposed into strict W_2-programs $Q_1,\ldots,Q_t$. The proof proceeds by induction on t.

If t=1 then P is strict and the result is given directly from the second part of the proof of Theorem 1. Now suppose t>1.

Let v be the sink of Q_1 which is also a sink of P and let u be the other node of P at the same level. Consider the W_2-program P' obtained from P by deleting the nodes, except for u, and edges belonging to Q_1. Let f' be the function computed by P'. Also let f_1 be the function computed by Q_1.

If v_1 is labelled by 0, let $f = f_1 \wedge f'$ and if v_1 is labelled by 1, let $f = f_1 \vee f'$. Clearly, f is the function computed by P. By the induction hypothesis, f_1 and f' have formula size at most three times the length of Q_1 and P', respectively. Therefore the formula size of f is at most three times the length of P. □

Theorem 1, the fact that $\bar{f} = f \oplus a \oplus \bar{a}$, and deMorgan's laws enable us to find more closure properties of SW_2. Specifically, if $f \in SW_2$ and a,b are literals, then $\bar{f}, f \vee a$, and $f \vee (a \oplus b)$ are in SW_2.

Notice that

$$E_1^3(x_1,x_2,x_3) = [(x_1 \wedge x_3) \vee (x_1 \oplus x_2)] \oplus x_3$$

$$E_{0,1}^3(x_1,x_2,x_3) = [(x_1 \oplus x_3) \vee (x_1 \oplus x_2)] \oplus x_3$$

$$E_2^4(x_1,x_2,x_3,x_4) = [[((x_1 \oplus x_3) \wedge (x_1 \oplus x_4)) \oplus x_3 \oplus x_4] \wedge (x_1 \oplus \bar{x}_2)] \oplus x_1 \oplus x_3$$

$$E_0^n(x_1,\ldots,x_n) = \bar{x}_1 \wedge \bar{x}_2 \wedge \ldots \wedge \bar{x}_n$$

$$E_n^n(x_1,\ldots,x_n) = x_1 \wedge x_2 \wedge \ldots \wedge x_n \text{ and}$$

$$E_{1,n-1}^n(x_1,\ldots,x_n) = (x_1 \oplus x_2) \vee (x_1 \oplus x_3) \vee \ldots \vee (x_1 \oplus x_n$$

Thus all these functions are in SW_2.

Now consider any function $g \in SW_2$. The Theorem 1, g is either a constant function or a projection or can be obtained from one of these by repeated applications of rules R1 through R3. Notice that if only rule R1 is applied, then the resulting functions are constant or parity functions. Recall that a parity function has the form $x_{i_1} \oplus \ldots \oplus x_{i_t}$ where $x_{i_1},\ldots,x_{i_t}$ are Boolean literals.

Therefore, if g is not a constant or parity function, then either

$$g = (f \wedge a) \oplus \bigoplus_{i=1}^{m} c_i \text{ or}$$

$$g = (f \wedge (a \oplus b)) \oplus \bigoplus_{i=1}^{m} c_i$$

where $m \geq 0$, $f \in SW_2$, and a, b and c_i, for $1 \leq i \leq m$, are literals.

In the first case, substituting 0 for a turns g into a constant or parity function, namely $\bigoplus_{i=1}^{m} c_i$, while in the second case, identifying a and b turns g into a constant or parity function. This observation yields the following result.

Lemma 3: Let $g \in SW_2$. Then one of the following conditions holds:

(i) g is a constant or parity function.
(ii) There is a literal a such that substituting 0 for a turns g into a constant or parity function.
(iii) There are two literals a and b such that identifying a and b turns g into a constant parity function.

We are also able to show that all functions of SW_2 can be computed by short strict W_2-programs.

Lemma 4. If $g \in SW_2$ is a Boolean function of n variables, then there is a strict W_2-program of length $O(n^2)$ that computes g.

Proof: By induction on n.

All Boolean functions of 1 variable can be computed by strict W_2-programs of length 1. If g is a constant or parity function, then g can be computed by a strict W_2-program of length 0 or n, respectively. Therefore we may assume that $n \geq 2$ and either

$$g = (f \wedge a) \oplus \bigoplus_{i=1}^{m} c_i \text{ or}$$

$$g = (f \wedge (a \oplus b)) \oplus \bigoplus_{i=1}^{m} c_i$$

where $m \geq 0$, $f \in SW_2$, and a, b and c_i, for $1 \leq i \leq m$, are literals.

Consider the function $\bigoplus_{i=1}^{m} c_i$. If it is the constant 0 function (which is the case when m=0), then $g = f \wedge a$ or $g = f \wedge (a \oplus b)$. When $\bigoplus_{i=1}^{m} c_i$ is the constant 1 function, a program to compute g can be obtained from a program to compute $f \wedge a$ or $f \wedge (a \oplus b)$ by interchanging the labels of the two sinks. Now suppose $\bigoplus_{i=1}^{m} c_i$ is not constant. Since $c \oplus c = 0$ and $c \oplus \bar{c} = 1$, it is unnecessary to have $c_i = c_j$ or $c_i = \bar{c}_j$ for $1 \le i \ne j \le m$. In particular, this implies $m \le n$. Therefore the length of the shortest strict W_2-program that computes g exceeds the length of the shortest strict W_2-program that computes f by at most $n+2$.

Finally, we may assume, without loss of generality that neither a nor $\bar{a}$ appear in f. Otherwise, in the first case, by replacing all occurrences of a and $\bar{a}$ by 1 and 0, respectively, we could obtain a new function f' containing neither a nor $\bar{a}$ such that $f' \wedge a = f \wedge a$. Similarly, in the second case, all occurrences of a and $\bar{a}$ can be replaced by $\bar{b}$ and b, respectively.

Since f contains at most n-1 variables, it follows that there is a strict W_2-program of length $O(n^2)$ that computes g. □

Lemma 3 is useful for showing that certain functions are not in SW_2. Consider the following example. The function E_1^4 is not a constant or parity function. Let $f = E_1^4 |_{x_1=0}$. Since $f(0,0,1)=1$ and $f(0,1,1)=0$, f is not constant. Also, notice that $f(1,1,1)=0$ and, hence, f is not a parity function. Similarly, let $g = E_1^4 |_{x_1=1}$. Then $g(0,0,0)=1$ and $g(0,0,1)=g(0,1,1)=0$. If $h(x_1,x_3,x_4) = E_1^4 |_{x_1=x_2}$, then $h(0,0,0)=h(1,0,0)=0$ and $h(0,1,0)=1$. Finally, let $k(x_1,x_3,x_4) = E_1^4 |_{x_1=\bar{x}_2}$. Then $k(1,0,0)=1$ and $k(1,1,0)=k(1,1,1)=0$. Thus g, h and k are all neither constant nor parity functions. Since E_1^4 is a symmetric function, it follows from Lemma 3 that $E_1^4 \notin SW_2$.

Together with similar arguments one can show that $E_{h,k}^n \in SW_2$ if and only if one of the following conditions is true.

(i) $n \le 3$
(ii) $n=4$ and $h=k=2$
(iii) $h=k=0$
(iv) $h=k=n$
(v) $h \le 1$ and $k \ge n-1$.

Our next result shows that, in a geometric sense, the functions computed by strict W_2-programs are not too complicated. We have to introduce some notation.

A cube is a subset of $\{0,1\}^n$ of the form

$$\{x \mid x_{i_1} = a_1, \ldots, x_{i_r} = a_r\} \quad a_1, \ldots, a_r \in \{0,1\}$$

where $0 \le r \le n$. The *dimension* of the cube is defined to be $n-r$. A *striped cube* is a subset of $\{0,1\}^n$ of the form $\{x \mid x_{i_1} = a_1, \ldots, x_{i_r} = a_r \text{ and } x_{j_1} \oplus \ldots \oplus x_{j_t} = b\}$ where $0 \le r, t \le n$ and $a_1, \ldots, a_r, b \in \{0,1\}$. As above, $n-r$ is called the *dimension* of the striped cube. Let Z_n be the smallest number such that, for all n-ary functions $f \in SW_2$, the set of accepted inputs $f^{-1}(1)$, can be represented as a disjoint union of Z_n striped cubes.

Lemma 5: $Z_n \le 4 \times 2^{n/2} - 2$.

Proof: By induction on n. The theorem is clearly true for $n=1$. Consider any n-ary Boolean function $f \in SW_2$ and let Z be the smallest number such that $f^{-1}(1)$ can be represented as a disjoint union of Z striped cubes. One of the cases of Lemma 3 applies.

If f is a constant or parity function, then $Z \le 1$.

If there is a literal a such that substituting 1 for a turns f into a constant or parity function f_1, then $f = (a \wedge f_1) \vee (\bar{a} \wedge f_2)$ where f_2 is a function of $n-1$ variables. In this case $Z \le 1 + Z_{n-1}$.

Finally suppose there are two literals, a and b which, when identified, turn f into a constant or parity function. Then $f = (a \wedge b \wedge f_1) \vee (\bar{a} \wedge \bar{b} \wedge f_2) \vee (a \wedge \bar{b} \wedge f_3) \vee (\bar{a} \wedge b \wedge f_4)$ where f_1 and f_2 are constant or parity functions and f_3 and f_4 depend on at most $n-2$ variables. In this case $Z \le 2 + 2Z_{n-2}$.

Since f was arbitrary, we have $Z_n \le \max\{1, 1+Z_{n-1}, 2+2Z_{n-2}\}$. □

Now consider the Boolean function

$$f(x_1, \ldots x_n) = (x_1 \oplus x_2) \wedge (x_3 \oplus x_4) \wedge \ldots \wedge (x_{n-1} \oplus x_n).$$

From Theorem 1 it is easy to see that $f \in SW_2$. Each accepted input contains exactly n/2 variables with value 1. Therefore if $S \subseteq f^{-1}(1)$ is a striped cube, then $\#S \le 2$. In particular, this implies that $Z_n \ge 2^{n/2-1}$.

3. LOWER BOUNDS FOR MONOTONE W_2-PROGRAMS

Lemma 5 can be used to obtain lower bounds on the length of W_2-programs and monotone W_2-programs.

Let $\mathscr{S}$ be a system of subsets of $\{0,1\}^n$. For example, $\mathscr{S}$ might be the system of cubes or the system of striped cubes. An $\mathscr{S}$-program is a sequence

$$(S_1, a_1)(S_2, a_2), \ldots, (S_m, a_m)$$

where $S_i \in \mathscr{S}$ and $a_i \in \{0,1\}$ for all i. The *length* of the program is m. This program computes an n-ary function f in the following way: Let $\mathbf{b} \in \{0,1\}^n$. If $\mathbf{b} \notin \bigcup S_i$ then $f(\mathbf{b})=0$. If $\mathbf{b} \in S_i \setminus \bigcup_{j \le i} S_j$, then $f(\mathbf{b}) = a_i$. For any Boolean function f, the $\mathscr{S}$-complexity $C_{\mathscr{S}}(f)$ of f is defined to be the length of the shortest $\mathscr{S}$-program that computes f.

By Lemma 8, for any function g computable by a strict W_2-program, $g^{-1}(1)$ can be represented as a disjoint union of at most $4 \times 2^{n/2} - 2$ striped cubes. Therefore, if $\mathscr{S}$ is the system of all striped cubes, then $C_{\mathscr{S}}(f)/(4 \times 2^{n/2} - 2)$ is a lower bound for the number of strict W_2-programs comprising any W_2-program that computes f and, hence, the length of any W_2-program that computes f.

If $\mathscr{S}$ is the system of all cubes, then $2(3/2)^{n-1} \le C_{\mathscr{S}}(x_1 \oplus \cdots \oplus x_n) \le 5^{n/3}$ [0].

We call an $\mathscr{S}$-program $(S_1, a_1), \ldots, (S_m, a_m)$ *monotone* if $a_i = 1$ for all i. For any Boolean function f, the monotone $\mathscr{S}$-complexity $MC_{\mathscr{S}}(f)$ is defined as the length of the shortest monotone $\mathscr{P}$-program that computes f. By Lemma 5, $MC_{\mathscr{P}}(f)/(4\times 2^{n/2}-2)$ is a lower bound for the length of any monotone W_2-program that computes f.

Theorem 6: Every monotone W_2-program that computes $E^n_{k,n}$ has length at least $\binom{n}{k}/((4\times 2^{n/2}-2)n)$.

Proof: Let $S=\{x \mid x_{i_1}=a_1,\ldots,x_{i_r}=a_r$, and $x_{j_1}\oplus\cdots\oplus x_{j_t}=b\}$ be any striped cube occurring in a monotone $\mathscr{P}$-program for $E^n_{k,n}$. It is easily seen that at least $k-1$ of the a_i's are 1. Otherwise the $\mathscr{P}$-program would accept an input in which fewer than k variables have value 1. Hence the number of inputs x, such that $x\in S$ and $w(x)=k$ is at most $n-r\leq n$. Thus $MC_{\mathscr{P}}(E^n_{k,n})\geq\binom{n}{k}/n$. □

Let $\mathscr{D}=\{S_1,\ldots,S_m\}$ be a system of sets. $\mathscr{D}$ is called a Δ-system if $S_i\cap S_j=\bigcap_{h=1}^{m}S_h$ for all $i\neq j$. Stated alternatively, any element in $\bigcup_{h=1}^{m}S_h$ is either contained in every set or is contained in exactly one set.

Erdös and Rado[3] showed that for all natural numbers p and k; if $\mathscr{S}$ is a system of more than $F(k,p)=k+k^k(p-1)^{k+1}$ sets each of cardinality at most k, then $\mathscr{S}$ contains a subsystem of p sets which is a Δ-system. We will use this fact in order to derive lower bounds for the length of monotone W_2-programs that compute the functions E^n_k.

Theorem 7: Let P be a monotone W_2-program that computes E^n_k. Then length of P is at least $n\binom{n}{k}/F(k,4)$.

Proof: Suppose that the input $a\in\{0,1\}^n$ is accepted by P at some accepting node v. Among the strict W_2-programs comprising P, let Q be the one which contains v. Recall that any W_2-program can be uniquely decomposed into strict W_2-programs.

If the length of Q is less than n, then some variable x_i would not be tested during the computation of Q on input a. Let a' be the input obtained from a by changing the value of its ith component. Then $a'\in I_Q(v)$. Recall that $I_Q(v)$ is the set of inputs which cause the computation of Q to reach vertex v. Also note that $w(a')\neq k$ and, therefore a' is not accepted by P. Since P is monotone, the computation of P on input a' must reach the source of Q. It will continue from there to v, thereby accepting a'. Hence the length of Q is at least n.

Next we show that $\#I_P(v)\leq F(k,4)$. Suppose, to the contrary, that $\#I_P(v)>F(k,4)$. Let $\mathscr{S}=\{S(a)\mid a\in I_p(v)\}$. Then $\mathscr{S}$ contains a Δ-system $\mathscr{D}=\{D_1,D_2,D_3,D_4\}$. Let $G=\bigcap_{i=1}^{4}D_i$ and $H=\bigcup_{i=1}^{4}D_i$.

A new strict W_2-program Q' can be obtained from Q by the following modifcations. For all $j\in G$, delete all edges labelled $\bar{x}_j$ and delete all occurrences of the label x_j. This corresponds to fixing the value of the variable x_j to be 1. For all $j\notin H$, delete all edges labelled x_j and delete all occurrences of the label $\bar{x}_j$. This corresponds to fixing the value of the variable x_j to be 0. For $i=1,2,3,4$, choose a new variable y_i. Then, for each $j\notin D_i\setminus G$, replace the labels x_j and $\bar{x}_j$ by y_i and $\bar{y}_i$, respectively.

Let $b=(b_1,\ldots,b_4)\in\{0,1\}^4$. Then $f_{Q'}(b)=f_Q(c)$ where $c_j=1$ if $j\in G$, $c_j=0$ if $j\notin H$, and $c_j=b_i$ if $j\in D_i\setminus G$. Notice that $w(c)=|G|+\sum_{i=1}^{4}b_i(|D_i|-|G|)$. Since $|D_1|=|D_2|=|D_3|=|D_4|=k>|G|$, it follows that $w(c)=k$ if and only if $b_i=1$ for exactly one value of i. In this case $S(c)=D_i\in\mathscr{S}$. Thus $E^4_1(b)=1$ implies $c\in I_P(v)$. If $E^4_1(b)=0$, then P does not accept c and the computation of P on input c does not reach the accepting node v. However, since P is monotone, the computation does reach the source of Q. It follows that $f_{Q'}=E^4_1$. This contradicts the fact that Q is a strict W_2-program.

Therefore P must be comprised of at least $\binom{n}{k}/F(k,4)$ strict W_2-programs, each of length at least n. □

A similar argument can be used to show that the length of any program which computes $E^n_{h,k}$ is at least $\binom{n}{k}(n-k+h)/F(k,4)$ for $0\leq h\leq k\leq n$.

4. A LOWER BOUND FOR W_2-PROGRAMS

Theorem 8: Every W_2-program P that computes $E^n_{\lceil n/2\rceil,n}$ has length $\Omega(n^2/\log n)$.

Proof: Decompose P into strict W_2-programs $Q_1,Q_2,\ldots$ such that, for all ℓ, the nodes in Q_ℓ are closer to the source of P than the nodes in $Q_{\ell+1}$. For $\ell=1,2,\ldots$ let v_ℓ be a sink of P which is also a sink of Q_ℓ.

Consider the border region $B=\{x\mid\lceil n/2\rceil-2\leq w(x)\leq\lceil n/2\rceil+1\}\subset\{0,1\}^n$ and, for $\ell=1,2,\ldots$, let $\tau_\ell=\sum_{d\leq\ell}\#(I_P(v_d)\cap B))$. We want to find a recurrence relation for the numbers τ_ℓ.

By Lemma 5, $I_{Q_\ell}(v_\ell)$ can be represented as a disjoint union of striped cubes $S_i,\ldots,S_m$ where $m\leq 4\times 2^{n/2}-2$. Consider any such striped cube

$$S=\{x\mid x_{i_1}=a_1,\ldots,x_{i_r}=a_r \text{ and } x_{j_1}\oplus\cdots\oplus x_{j_t}=b\}.$$

We can assume that $\{i_1,\ldots,i_r\}\cap\{j_i,\ldots,j_t\}=\emptyset$ and $t\neq 1$. Let $\sigma_d(S)=(I_P(v_d)\cap B\cap S)$.

First suppose that v_ℓ is an accepting node of P. When $\ell=1$, no inputs have yet been rejected. Therefore, at least $\lceil n/2\rceil-1$ of the a_i's are 1 and $\sigma_1(S)=\#(S\cap B)\leq n^2$. Hence $\tau_1=\sum_{h=1}^{m}\sigma_1(S_h)\leq n^2(4\times 2^{n/2}-2)$.

More generally, if at least $\lceil n/2\rceil-2$ of the a_i's are 1, we have $\sigma_\ell(S)\leq\#(S\cap B)\leq n^3$. Now consider the case when fewer than $\lceil n/2\rceil-2$ of the a_i's are 1. Let $x\in I_P(v_\ell)\cap B\cap S$.

Since $w(x)=\lceil n/2\rceil$ or $\lceil n/2\rceil+1$, there exist at least three indices $q\notin\{i_1,\ldots,i_r\}$ such that $x_q=1$. Among these indices, at least two, say q_1 and q_2, must both be elements of $\{j_i,\ldots,j_t\}$ or both be elements of the complement of this set. In either case, let x' be obtained from x by changing both x_{q_1} and x_{q_2} to 0. Then $x'\in S$. However, $x'\notin I_P(v_\ell)$ because v_ℓ is an accepting node and $w(x')=\lceil n/2\rceil-2$ or $\lceil n/2\rceil-1$. It follows that x' was rejected previously and thus $x' \in \bigcup_{d<\ell} (I_P(v_d) \cap B \cap S)$. On the other hand every such x' can be obtained in this way from at most $\binom{n}{2}$ vectors x. Therefore

$$\sigma_\ell(S) \leq \max \left\{ n^3, \binom{n}{2} \# \bigcup_{d<\ell} (I_P(v_d) \cap B \cap S) \right\} .$$

Since the cubes $S_1,\ldots,S_m$ are disjoint,

$$\tau_\ell = \tau_{\ell-1} + \sum_{h=1}^{m} \sigma_\ell(S_h)$$

$$\leq \tau_{\ell-1} + \sum_{h=1}^{m} \left[n^3 + \binom{n}{2} \sum_{d<\ell} \#(I_P(v_d) \cap B \cap S_h) \right]$$

$$= \tau_{\ell-1} + n^3 m + \binom{n}{2} \sum_{d<\ell} \sum_{h=1}^{m} \#(I_P(v_d) \cap B \cap S_h)$$

$$\leq \tau_{\ell-1} + n^3(4 \times 2^{n/2} - 2) + \binom{n}{2} \sum_{d<\ell} \#(I_P(v_d) \cap B)$$

$$= n^3(4 \times 2^{n/2} - 2) + \left(\binom{n}{2} + 1 \right) \tau_{\ell-1}$$

$$< n^3 2^{n/2+2} + \frac{n^2}{2} \tau_{\ell-1} .$$

Similarly, if v_ℓ is a rejecting node, then the same inequalities concerning τ_ℓ can be derived. Thus

$$\tau_\ell < 2^{n/2-\ell+4}\, n^{2\ell} .$$

Let $L=(5n/16-4)/(2 \log n-1)$. Then $\tau_\ell < 2^{13n/16}$ for all $\ell \leq L$. Therefore $\underline{P}$ must be composed of more than L strict W_2-programs.

Let $\lambda = \min\{\text{length}(Q_d) \mid 1 \leq d \leq L\}$. If $\lambda \geq n/8$ the theorem is proven. Thus, assume $\lambda < n/8$.

Consider Q_ℓ where $1 \leq \ell \leq L$ and length $(Q_\ell) = \lambda$. On any path from the source of Q_ℓ to v_ℓ at most λ variables are examined. Therefore $I_{Q_\ell}(v_\ell)$ can be represented as a union of striped cubes of dimension at least $n-\lambda$ (this follows directly from the proof of Lemma 5). Consider any such striped cube

$$S = \{x \mid x_{i_1} = a_r,\ldots,x_{i_r} = a_r \text{ and } x_{j_1} \oplus \ldots \oplus x_{j_t} = b\} .$$

Assume v_ℓ is an accepting node of P. Let α be the number of a_i's that are 1 and let β be the the number of a_i's that are 0. We want to estimate from below, the number of vectors in S that have weight $\lceil n/2 \rceil - 1$ or $\lceil n/2 \rceil - 2$. We first consider

$$C = \{x \mid x_{i_1} = a_1,\ldots,x_{i_r} = a_r \text{ , and}$$

$$w(x) = \lceil n/2 \rceil - 1 \text{ or } \lceil n/2 \rceil - 2\}.$$

Then,

$$\#C = \binom{n-r}{\lceil \frac{n}{2} \rceil - 1 - \alpha} + \binom{n-r}{\lceil \frac{n}{2} \rceil - 2 - \alpha}$$

$$= \binom{n+1-r}{\lceil \frac{n}{2} \rceil - 1 - \alpha}$$

$$= \binom{n+1-\alpha-\beta}{\lceil n/2 \rceil + (1-\alpha-\beta)/2 - (3+\alpha-\beta)/2}$$

$$= \binom{n+1-\alpha-\beta}{\lceil (n+1-\alpha-\beta)/2 \rceil - (c+\alpha-\beta)/2}$$

where $c \in \{2,3,4\}$. This is minimized if $\beta=0$ and $\alpha=r$. Hence

$$\#C \geq \binom{n+1-r}{\lceil n/2 \rceil - 1 - r} \geq \binom{n+1-\lambda}{\lceil n/2 \rceil - 1 - \lambda}$$

$$\geq \binom{7n/8}{3n/8} / n$$

$$= (7n/8)^{7n/8} / ((3n/8)^{3n/8} (n/2)^{n/2} 0(n^{3/2}))$$

$$= 2^{n((\log 7 - 3)7/8 + (3 - \log 3)3/8 + 1/2} / 0(n^{3/2})$$

$$> n^2 2^{n(-14/80+39/80+1/2)} = n^2 2^{13n/16} .$$

Next we show that any vector $x \in C$ can be obtained from some vector x' in $C \cap S$ by changing at most 2 bits of x'. From any $x' \in C \cap S$, we can obtain fewer than n^2 vectors x in this way. Therefore

$$n^2 \#(S \cap C) \geq \#C .$$

Let $x \in C \setminus S$. We construct x'. We first assume that $w(x) = \lceil n/2 \rceil - 2$. If $x_q = 0$ for some $q \in \{j_1,\ldots,j_t\}$, set $x_q = 1$. If $x_{j_1} = \ldots = x_{j_t} = 1$ find a component $q \notin \{a_1,\ldots,a_r\}$ such that $x_q = 0$; this is possible since $\lambda \leq n/8$. Set $x_{j_1} = 0$ and $x_q = 1$. The case $w(x) = \lceil n/2 \rceil - 1$ is handled similarly.

Recall that v_ℓ is assumed to be an accepting node of P. Hence all elements in $S \cap C$ must have been rejected previously. Thus

$$2^{13n/16} < \#(S \cap C) \leq \tau_{\ell-1} < 2^{13n/16} .$$

If v_ℓ is a rejecting node of P the same inequality is derived using analogous arguments. □

5. CONCLUSIONS AND OPEN PROBLEMS

Obviously, we are just beginning to understand the limited power of bounded width branching programs. The few examples given (both positive and negative) all concern "counting". In some cases (*e.g.*, E_2^4) we observe nontrivial ways of counting.

Our nonpolynomial lower bounds hold only for monotone W_2-programs. Many interesting problems remain open, including the following.

(i) Prove nonpolynomial lower bounds on the length of W_2-programs.
(ii) Prove lower bounds on the length of $\mathscr{S}$-programs, where $\mathscr{S}$ is the system of cubes or striped cubes.

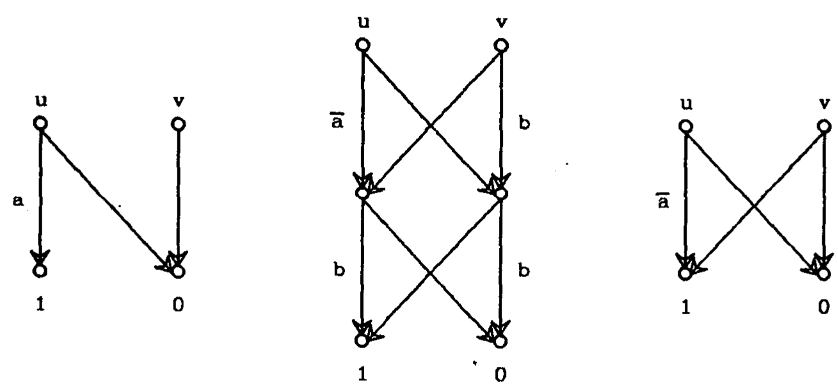
Of course, we need not restrict ourselves to counting functions. However, counting is a basic component in many computationally nontrivial problems, and eventually we should be able to understand the extent to which bounded width programs can count.

ACKNOWLEDGMENTS

The authors would like to thank Michael Fischer for helpful discussions.

REFERENCES

0. B. R. Plumstead and J. B. Plumstead, *Bounds for Cube Coloring*, Preprint, Berkeley, 1982.
1. A. Borodin, M. Fischer, D. Kirkpatrick, N. Lynch and M. Tompa, *A Time-Space Tradeoff for Sorting on Non-oblivious Machines*, Proceedings of 20th Annual Symposium on Foundations of Computer Science, 1979, pp. 319-327.
2. A. Cobham, The Recognition Problem for the Set of Perfect Squares, *Research Paper RC-1704*, IBM Watson Research Center, Yorktown Heights, New York, April 1966.
3. A Chandra, M. Furst and R. Lipton, private communication.
4. P. Erdös and R. Rado, *Intersection Theorems for Systems of Sets*, J. London Math Society, 35, 1960, pp. 85-90.
5. Michael J. Fischer, Albert R. Meyer and Michael S. Paterson, $\Omega(n \log n)$ Lower Bounds on Length of Boolean Formulas, *Siam J. Comput*, **11**, 3 (1982), pp. 416-427.
6. M. Furst, J. Saxe and M. Sipser, *Parity, Circuits and the Polynomial-Time Hierarchy*, Proceedings of 22nd Annual Symposium on Foundations of Computer Science, 1981, pp. 260-270.
7. W. Masek, *A Fast Algorithm for the String Editing Problem and Decision Graph Complexity*, M. Sc. Thesis, MIT, May 1976.
8. E. I. Neciporuk, A. *A Boolean Function, Soviet Math Dokl.*, **2**, 4 (1966), pp. 999-1000.



(i) (ii) (iii)

Figure 1

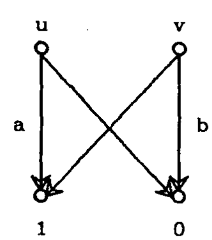


Figure 2