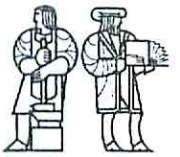
LABORATORY FOR
COMPUTER SCIENCE



MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-293

WIDTH-3 PERMUTATION BRANCHING PROGRAMS

David A. Barrington

December 1985

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Width-3 Permutation Branching Programs

David A. Barrington
Department of Mathematics
and Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

5 December 1985

Abstract:

We consider a restricted class of width-3 branching programs where each column of nodes depends on a single variable, and the 0-edges and the 1-edges out of each column form a permutation. In this model, parity and the mod-3 function are easy to calculate, but the and-function is hard. We show that any function of n inputs can be calculated in length $O(2^n)$, and that the and-function in particular requires length $O(2^n)$ if the branching program has one accept node and one reject node.

Key words and Phrases:

Branching programs, parallel complexity, lower bounds.

This work was supported by National Science Foundation grant MCS-8304769 and U.S. Air Force grant AFOSR-82-0326, and by an NSF Graduate Fellowship.

1. Introduction

We define a branching program of width w (a w-BP) to be a sequence of instructions $\langle j_i, f_i, g_i \rangle$ for $1 \leq i \leq l$, where x_{j_i} is one of n input variables, f_i and g_i are functions from $[w]$ to $[w]$ (throughout, $[w]$ is the set $\{0, \ldots, w-1\}$), and l is the length. Given a setting of the input variables, the instruction $\langle j_i, f_i, g_i \rangle$ yields the function f_i if x_{j_i} is on and g_i if x_{j_i} is off. A branching program B on input setting x yields the composition of the functions yielded by each of its instructions – we call this composition $B(\mathrm{x})$. The justification for the details of this model will be discussed in Section 3.

B is a permutation branching program (w-PBP) if each f_i and g_i is a permutation of $[w]$ – in general we will use Greek letters for permutations. A k-PBP weakly recognizes a set $A \subseteq [2]^n$ if for some permutation σ of $[k]$, $\mathrm{x} \in A \leftrightarrow B(\mathrm{x}) = \sigma$. It strongly recognizes A if in addition for some $\tau \neq \sigma$, $\mathrm{x} \notin A \leftrightarrow B(\mathrm{x}) = \tau$.

We show that any set $A \subseteq [2]^n$ can be strongly recognized by a 3-PBP of length $O(n2^n)$, and that length $O(2^n)$ is required to strongly recognize the singleton set $\{1^n\}$,i.e., to strongly calculate the and-function of n variables. To do this we define in Section 4 a class of circuits composed of mod-2 and mod-3 gates and show that these circuits and 3-PBP's are interconvertible, with size blowup constant one way and $O(n)$ the other. We then show in Section 5 that these circuits are in one-one correspondence with functions from $[2^n]$ to $[3]$. The circuit corresponding to the (strong) and-function can be given explicitly and has size $\Theta(2^n)$.

This lower bound result is in some sense a dual of the result in [FSS], that parity is difficult to calculate by a circuit having the and-function as a primitive. It may be possible to extend these techniques to more powerful classes of branching programs – we discuss this briefly in Section 6.

2. Background and Motivation

Branching programs were defined by Lee [Le59] as an alternative to Boolean circuits in the description of switching problems – he called them 'binary decision programs'. They were later studied in the Master's thesis of Masek [Ma76] under the name of 'decision graphs'.

Borodin, Dolev, Fich and Paul [BDFP83] raised the question of the power of bounded-width branching programs. They noted that the class BWBP contains AC^0 (languages recognized by unbounded fan-in, constant-depth, polynomial-size Boolean circuits) as well as the parity function (shown to be outside AC^0 in [FSS81]). They conjectured that the majority function was not in $BWBP$, in fact that for bounded width it requires exponential length.

This conjecture was shown to be false in [Ba86], but only after a considerable body of work appeared to support it. Chandra, Furst, and Lipton [CFL83] and Pudlak [Pu84] showed linear and superlinear length lower bounds respectively for arbitrary constant width. In [BDFP83] the idea was to work with width-2 and get exponential bounds. They succeeded for a restricted class of BP's , and Yao [Ya83] followed with a superpolynomial lower bound for general width-2. Shearer [Sh85] proved an exponential lower bound for the mod-3 function.

The earlier papers define width-k branching programs to be more powerful than our k-BP's. They allow accepting or rejecting nodes in the middle of the program, and allow nodes on the same column to access different input variables. Our model can simulate this additional power at the cost of increasing the width. If the intermediate nodes are all accepting or all rejecting, we can add one additional row of nodes to extend paths from these nodes to the end. If both accepting and rejecting nodes occur in the middle, we can use two additional rows. Finally, by doubling the width and multiplying the length by k we can get a program with all the nodes on one column accessing the same variable. So while the two models differ as to the definition of width-2, for example, they define the same class of polysize bounded-width branching programs.

This new definition of width-k is motivated by an analogy to finite automata. Define a non-uniform DFA (NUDFA) to be a finite-state machine with a two-way read-only input tape and a one-way program tape. The latter can contain instructions to move right or left on the input or to change state according to any given function of the current state and the visible input character. With this definition, a k-state NUDFA can be simulated by one of our k-BP's and vice versa. The complexity (length of the BP, program length time for the NUDFA) is the same up to a multiplicative factor of $O(n)$. So k-BP's are a non-uniform analogue of k-state DFA's, just as polysize BP's and polysize Boolean circuits are analogues of logspace and polytime Turing machines respectively. Viewed from this perspective, it seems more natural to charge in additional states for the extra power in the branching programs of [BDFP83].

A first analysis similar to that in [BDFP83] leads us to look at 3-BP's and 3-PBP's in particular. A depth-d unbounded fan-in Boolean circuit of size s can be simulated by a $d+1$-BP of length s^d. This means that 3-BP's, by simulating depth-2 circuits, can compute any Boolean function in length $2^{O(n)}$. By contrast, 2-BP's are very weak – the small class of functions they can compute are all obtained with length $O(n^2)$. (This class is very like the class SW_2 in [BDFP83].) In between is the class of 3-PBP's, raising two natural questions. Can 3-PBP's compute all Boolean functions given unlimited length? If so, are they significantly weaker than general 3-BP's? The answers, both affirmative, are our two main results.

The problem of weak versus strong recognition of sets arises as soon as the BP can give three or more possible answers rather than two. We will view 3-PBP's

as computing functions from $[2]^n$ into [3], because without loss of generality (for purposes of strong computation) we can concatenate two copies of the same 3-PBP and guarantee that the output is a even permutation. In the original models, sink nodes could accept or reject arbitrarily, leading to a definition of recognition between our 'weak' and 'strong', but much more like weak recognition. Is strong computation, then, a reasonable model? The results of [Ba86] suggest that it is, as a language can be strongly recognized by a family of 5-PBP's of polynomial size iff it can be weakly recognized by any branching program family of constant width and polynomial size.

3. 3-2-Parity Circuits

A 2-parity gate outputs 1 iff the number of its 0-1 inputs which are 1 is odd. A 3-parity gate outputs 0, 1, or 2 depending on the number mod 3 of its 0-1 inputs which are 1. A 3-2-parity circuit consists of a single 3-parity gate whose inputs are constant 1-gates or 2-parity gates of inputs. Its size is defined to be the fan-in of the 3-parity gate. We may assume without loss of generality that the inputs are not negated and that there are only zero, one, or two of each of the 2^n types of gates. (There is one type for each nonempty subset of [n] and the one type of constant gate.)

Lemma 1: A 3-2-parity circuit of size s may be simulated by a 3-PBP of length $O(ns)$.

Lemma 2: A 3-PBP with $B(0^n) = e$ may be put in a normal form with $g = e$ for all gates. (Here e is the identity permutation.)

Lemma 3: A 3-PBP in normal form of length l can be simulated by a 3-2-parity circuit of size $O(l)$.

The proofs of Lemmas 1 and 2 are fairly straightforward. We briefly sketch the proof of Lemma 3. View the permutation group S_3 as being generated by two elements x and r with $x^2 = 0$, $r^3 = 0$, and $xr = r^2x$. A normal-form 3-PBP can then be given as a sequence $< a_i, b_i, c_i >_{i \leq l}$, where the i'th instruction outputs $x^{b_i} r^{c_i}$ iff variable number a_i is on. Recall that without loss of generality, the product of all these terms is even (a power of r). We will use our single 3-parity gate to total up the r's contributed by each instruction of the 3-PBP. If a_i is on, this number depends only on c_i and on the 2-parity of the number of x's contributed by instructions before the i'th gate. This last can be computed by a 2-parity gate. An additional argument is needed to show that a constant number of constant and 2-parity gates can be designed which will output a given 2-parity function of the inputs if a_i is on and id otherwise.

To summarize, we have:

Theorem: The minimum 3-2-parity circuit size and minimum 3-PBP length needed to compute a function differ by at most a factor of $O(n)$.

4. The Circuit-Function Mapping and the Main Result

Our upper and lower bounds for 3-PBP's follow from the above theorem and the following:

Theorem: Every Boolean function may be strongly calculated by a 3-2-parity circuit of size $O(2^n)$. The and-function cannot be strongly calculated by such a circuit with size less than $\frac{3}{2}2^n$.

Each 3-2-parity circuit may be characterized by the function from $[2]^n$ to $[3]$ given by $\{n_A : A \subseteq [n]\}$, where $n_\emptyset$ in the number of constant gates and n_A for each $A \neq \emptyset$ is the number of gates computing the function $\bigoplus_{x \in A} x$. The output function is also from $[2]^n$ to $[3]$, given by $\{C(B) : B \subseteq [n]\}$, where $C(B)$ is the output with exactly those variables $x \in B$ turned on. The mapping from circuits to functions is linear over the field $GF(3)$, given by the matrix $M = \{m_{AB} : A \subseteq [n], B \subseteq [n]\}$ where $m_{AB} = 1$ if $A = \emptyset$ and $m_{AB} = |A \cap B| \bmod 2$ otherwise.

It is easier to look at an equivalent matrix M' where $m'_{AB} = 2 - (|A \cap B| \bmod 2)$. M' is derivable from M by Gaussian operations over $GF(3)$ and thus has nonzero determinant iff M does. M' is nonsingular – to see this one may explicitly calculate that its square is $\pm I$ or note that it is defined by the induction:

$$M'_{i+1} = \begin{bmatrix} I & I \\ I & -I \end{bmatrix} \begin{bmatrix} M'_i & 0 \\ 0 & M'_i \end{bmatrix} \tag{1}$$

So the mapping from circuits to functions is an isomorphism of the vector space $GF(3)^{2^n}$. Each function has a unique circuit calculating it of size $O(2^n)$. We can thus find the circuit complexity of any specific function, such as the strong calculation of the and-function, if we can find a general form for its inverse image under this mapping. In the case of the and-function ($C(B) = 1$ if $B = [n]$; $C(B) = 0$ otherwise) this inverse image is given by $n_A = 2^n \cdot (2 - (|A| \bmod 3)$, and so the circuit has size $\frac{3}{2} \cdot 2^n$.

5. Prospects for Extensions

5.1. Weak Computation

Though the and-function has a unique circuit strongly computing it, there are

2^{2^n-1} circuits weakly computing it. One of these circuits has size $\Theta(2^{n/2})$ – we conjecture that this is optimal for sufficiently large n.

5.2. Other Functions

It would be interesting to find a general form for the inverse image of other functions such as the majority or exactly-half functions of [BDFP83].

5.3. General 3-BP's

A 3-BP can be divided into sections which are 3-PBP's. Each 3-PBP divides $[2]^n$ into three subsets, and each connection between 3-PBP's essentially merges two of these subsets. Further exploration might lead to a general understanding of 3-BP length as a complexity measure.

5.4. Larger Width

The method used to transform a 3-PBP into a circuit of parity gates depends on the solvability of the permutation group S_3. To be precise, define G-PBP's where the permutations in each instruction come from a particular permutation group G. Then, if G is a solvable group, the method of Lemma 3 can be used to construct a circuit of constant depth and polynomial size equivalent to any polynomial length G-PBP. However, this circuit contains 'mod g' gates which output 1 iff the sum mod g of the 0-1 inputs is i, where i is fixed for the gate and g is the order of the group. This construction is given explicitly in [Ba86].

The group S_4 is solvable but S_k is not for $k \geq 5$. We make the conjecture (strengthening one in [FSS81]) that a constant-depth polynomial-size circuit family with mod-k gates of bounded k cannot calculate the majority function, and hence that no G-PBP of polynomial length with solvable G can do so. (In [Ba86] the converse of this is proved – G-PBP's can do majority, along with the rest of NC^1, if G is not solvable). This may bear on the general question (raised in [FSS81]) of the power of bounded-depth circuits with parity gates – how big a subset of NC^1 can they recognize?

6. References

[Ba86] D. A. Barrington, 'Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1', submitted to 1986 ACM STOC.

[BDFP83] A. Borodin, D. Dolev, F. E. Fich, and W. Paul, 'Bounds for width two branching programs', Proc. 15th ACM STOC, 1983, 87-93.

[CFL83] A. K. Chandra, M. L. Furst, and R. J. Lipton, 'Multiparty protocols', Proc. 15th ACM STOC, 1983, 94-99.

[FSS81] M. Furst, J. B. Saxe, and M. Sipser, 'Parity, circuits, and the polynomial time heirarchy', Proc. 22nd IEEE FOCS, 1981, 260-270.

[Le59] C. Y. Lee, 'Representation of switching functions by binary decision programs', Bell System Technical Journal 38 (1959) 985-999.

[Ma76] W. Masek, 'A fast algorithm for the string editing problem and decision graph complexity', M. Sc. Thesis, MIT, May 1976.

[Pu84] P. Pudlak, 'A lower bound on complexity of branching programs', Proc. Conference on the Mathematical Foundations of Computer Science, 1984, 480-489.

[Sh85] J. B. Shearer, personal communication, 1985.

[Ya83] A. C. Yao, 'Lower bounds by probabilistic arguments', Proc. 24th IEEE FOCS, 1983, 420-428.