národní
úložiště
šedé
literatury

A Large Lower Bound for 1-branching Programs

Savický, Petr
1996

INSTITUTE OF COMPUTER SCIENCE

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

A large lower bound for 1-branching programs

P. Savický, S. Žák

Technical report No. 669

April 15, 1996

Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Prague 8, Czech Republic
phone: (+422) 66414244 fax: (+422) 8585789
e-mail: {stan,savicky}@uivt.cas.cz

INSTITUTE OF COMPUTER SCIENCE

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

A large lower bound for 1-branching programs

P. Savický, S. Žák

Technical report No. 669
April 15, 1996

Abstract

Branching programs (b. p.'s) or decision diagrams are a general graph-based model of sequential computation. B.p.'s of polynomial size are a nonuniform counterpart of LOG. Lower bounds for different kinds of restricted b. p.'s are intensively investigated. An important restriction are so called 1–b. p.'s, where each computation reads each variable at most once. There is a series of lower bounds for 1–b. p.'s. The largest known lower bound was $2^{n/10}$, see [11]. In the present paper, a lower bound of $2^{n-o(n)}$ is given.

Keywords
branching programs, lower bounds

1 Introduction

A branching program (b. p.) is a computation model for representing the Boolean functions. The input of a branching program is a vector consisting of the values of n Boolean variables. The branching program itself is a directed acyclic graph with one source. The out-degree of each node is at most 2. Every branching node, i.e. a node of out-degree 2, is labeled by an input variable and one of its out-going edges is labeled by 0, the other one by 1. The sinks (out-degree 0) are labeled by 0 and 1. A branching program determines a Boolean function as follows. The computation starts at the source. If a node of out-degree 1 is reached, the computation follows the unique edge leaving the node. In each branching node the variable assigned to the node is tested and the out-going edge labeled by the actual value of the variable is chosen. Finaly, a sink is reached. Its label determines the value of the function for the given input. By the size of a branching program we mean the number of its nodes.

The branching programs are a model of the configuration space of Turing machines where each node corresponds to a configuration. Thus the polynomial size b. p.'s represent a nonuniform variant of LOG. Hence, a superpolynomial lower bound on b. p.'s for a Boolean function computable within polynomial time would imply $P \neq LOG$.

In order to investigate the computing power of branching programs, restricted models were suggested. From the beginning of 80's among these models the central role is played by read-once branching programs (1–b. p.), where the restriction is such that during each computation on any input each variable is tested at most once. The 1–b. p.'s are a point of departure for three directions of the research.

The first direction of the research appeared immediately after the first lower bounds $2^{c\sqrt{n}}$ in [12], [14]. This is the natural question of lower bounds for read k times b. p.'s or, shortly, for k–b. p.s for $k > 1$. Here, a k–b. p. is such a branching program in which every computation path tests each variable at most k times. The Boolean function of so-called half-cliques [14] requires exponential size in 1–b. p.'s on one hand and a polynomial size in 2–b. p.'s on the other hand. This suggests the question whether the hierarchy according to k is a proper hierarchy for all k. For b. p., where some kind of restriction of repeated tests of the same variable is applied only to the valid computation paths, the only known results are the superpolynomial lower bounds for $(1, +k)$–b. p.'s which are something between 1–b. p.'s and 2–b. p.'s – only a limited number (k) of variables may be tested more than once [5], [8] and [15].

In the second half of 80's a special type of 1–b. p.'s - ordered binary decision diagrams (OBDD) - became important as a data structure for representing the Boolean functions in some CAD applications, e.g. design or verification of Boolean circuits, as a review paper see e.g. [13].

At the beginning of 90's syntactic k–b. p.'s were introduced, where the restriction of k allowed tests is taken not only over all computations but over all paths in the b. p. in question. Note that each 1–b. p. is a syntactic 1–b. p., while for $k \geq 2$ this is not true in general. For syntactic read k times b. p.'s, exponential lower bounds are known, see [2], [4], [6]. Also a hierarchy according to k for syntactic $(1, +k)$–b. p.'s was proved [10], [9].

In the present paper the authors follow the way of enlarging of lower bounds for 1–b. p.'s. In [3] there is a bound $2^{c\sqrt{n}}$, in [1] a lower bound $2^{n/c}$ for a large c and in [11] a lower bound $2^{n/10}$. In [7], a bound $2^{c\sqrt{n}}$ is proved for multiplication.

Our lower bound is $2^{n-3\,n^{2/3}(\ln n)^{1/3}}$ and it is proved by elementary means.

2 The result

We shall consider Boolean functions of n variables. A partial input is an element of $\{0,1,*\}^n$. As usual, the positions containing 0 or 1 mean that the corresponding variable is fixed to the specified value, while a $*$ means that the variable remains free.

Let f be a function of n variables. Let u be a partial input. By $f|_u$ we mean the subfunction of f obtained from f by setting x_i to u_i if $u_i \in \{0,1\}$. The function $f|_u$ is considered a function of n variables, although it may depend on x_i only if $u_i = *$. Hence, for any partial inputs u and v, $f|_u$ and $f|_v$ have formally the same set of variables.

Definition 2.1 *Let f be a Boolean function. We say that f is k–separable, if for every two different partial inputs u, v with at most k fixed positions we have $f|_u \neq f|_v$.*

Theorem 2.2 *Any 1–b. p. computing a k–separable function is a tree up to level k.*

Proof: Consider two different partial computations reading at most k variables. These two computations specify two different partial inputs u, v. If a is the node of the b. p. reached by u, then the subprogram starting at a computes $f|_u$. Hence, if u and v lead to the same node, we would have $f|_u = f|_v$. This would be a contradiction with the assumed k–separability. □

Definition 2.3 *A system of n integer weights $w_1, w_2, ..., w_n$ is (s,m)–complete, if $w_i \not\equiv 0 \bmod m$ for all $i = 1,2,...,n$ and for every subset $I \subseteq \{1,2,...,n\}$ of size at least s and any index j, $1 \leq j \leq m$, there are $y_i \in \{0,1\}$ for $i \in I$ such that $\sum_{i\in I} w_i y_i \equiv j \bmod m$.*

Theorem 2.4 *Let $w_1, w_2, ..., w_n$ be (s,n)–complete. Let f be defined as $f(x) = x_i$, where i is determined by the identity $i \equiv \sum_{j=1}^{n} w_j x_j (\bmod n)$. Then f is $(n-s-3)$–separable.*

Proof: In this proof, all congruences $\equiv$ are considered mod n. Let $k = n - s - 3$. Let u, v be two partial computations with at most k fixed positions. We shall prove that $f|_u \neq f|_v$ by considering three cases.

Case 1. Let u and v have distinct sets of fixed positions. We shall prove that $f|_u$ depends essentialy on all variables not fixed by u. By the same argument, $f|_v$ depends on all variables not fixed by v. This implies $f|_u \neq f|_v$, since the two subfunctions depend on different subsets of variables.

Let $u_i = *$. We are going to prove that $f|_u$ depends essentially on x_i by finding two extensions x and x' of u differring only at the i-th position such that $f(x) \neq f(x')$.

For any integer p, let the position p mean the position j, where $j \equiv p \bmod n$ and $1 \leq j \leq n$.

Let j be such that $u_j = *$ and $j \not\equiv i$, $j + w_i \not\equiv i$. Let u' be an extension of u obtained by setting the position i to 0 and by setting the position j and, if the position $j + w_i$ is not fixed, also the position $j + w_i$, in such a way that the positions j and $j + w_i$ are fixed to different values. This is always possible, since the position j is originally not fixed. Since u' contains at most $n - s$ fixed positions and our weight system is (s, n)-complete, it is possible to find an extension x of u' such that $\sum_{k=1}^{n} w_k x_k \equiv j$. We have $f(x) = u'_j$. If x' differs from x only in the position i, i.e. $x'_i = 1$, we have $f(x') = u'_{j+w_i}$. Hence, we have $f(x) \neq f(x')$.

Case 2. Let u and v have the same set I of fixed positions. Let

$$\Delta = \sum_{i \in I} w_i v_i - \sum_{i \in I} w_i u_i.$$

We are going to prove $f|_u \neq f|_v$ by finding an extension x of u and an extension x' of v such that they coincide on the positions not in I and $f(x) \neq f(x')$.

Subcase 2a. Let $\Delta \neq 0$. We extend u and v to u' and v' by setting some positions not fixed in u and v to the same values. The new inputs will be called u' and v'. Let $j \notin I$ be an index of some position. If also $(j + \Delta) \notin I$, we set the positions j and $(j + \Delta)$ of both u and v to different values, i.e. $u'_j = v'_j \neq u'_{j+\Delta} = v'_{j+\Delta}$. Otherwise, we set the position j of both u' and v' so that $u'_j = v'_j$ is different from $v'_{j+\Delta}$. We have at most $k + 2 \leq n - s$ fixed positions. Hence, there is an extension x of u' such that $\sum_{i=1}^{n} w_i x_i \equiv j$. We have $f(x) = u'_j$. Let x' be obtained from x by replacing the values of positions specified by u by the values specified by v. Then $f(x') = v'_{j+\Delta}$ and hence $f(x) \neq f(x')$.

Subcase 2b. Let $\Delta = 0$. Let $j \in I$ be such that $u_j \neq v_j$. There is an extension x of u such that $\sum_{i=1}^{n} w_i x_i \equiv j$. Let x' be obtained by replacing the values specified by u by the values specified by v. By our assumptions, we have $\sum_{i=1}^{n} w_i x'_i \equiv j$. Hence $f(x') = v_j \neq u_j = f(x)$. □

Lemma 2.5 *Let p, q and r be integers, p and q relatively prime. Moreover, let $b/2 \leq p \leq q \leq b$ and $0 \leq r \leq b$. Then there are nonnegative integers $x, y \leq b + 1$ such that $xp - yq = -r$.*

Proof: By the well-known theorem, for every p and q there are x_0 and y_0 such that $x_0 p - y_0 q$ equals the largest common divisor of p and q. Since p and q are relatively prime, there are some x'_0 and y'_0 satisfying $x'_0 p - y'_0 q = 1$ and by choosing $x_0 = -r x'_0$ and $y_0 = -r y'_0$, we obtain $x_0 p - y_0 q = -r$. Then, every pair $x_0 + iq$ and $y_0 + ip$ is also a solution of the above equation. Let i_0 be the minimal i such that $x_0 + iq \geq 0$. Let $x =_{\text{df}} x_0 + i_0 q$ and $y =_{\text{df}} y_0 + i_0 p$. We have $0 \leq x \leq q - 1 \leq b - 1$ and $y = xp/q + r/q \leq b - 1 + 2$. □

Theorem 2.6 *Let b and $w_1, \ldots, w_n$ be integers. Let $b/2 \leq w_1 \leq \ldots w_n \leq b < m$ and let the n numbers w_i consist of k different primes, each with at least $\lfloor n/k \rfloor$ and at most $\lceil n/k \rceil$ occurrences. Let s be larger then $\max\{bk, b(k-1) + \lceil n/k \rceil, 2(b+1) + 2m/b\}$. Then the system $w_1, \ldots, w_n$ is (s, m)–complete.*

Proof: Let $I \subseteq \{1, 2, \ldots, n\}$, $|I| = s$ be a set of indices of some numbers among w_i.

If each of the primes has at most b occurrences in I, we would have $s \leq bk$. This is a contradiction with the choice of s. Hence, some of the primes, say p, has at least $b+1$ occurrences. Let $A \subseteq I$ be a set of indices of $b+1$ occurrences of p.

The number p has at most $\lceil n/k \rceil$ occurrences among w_i. If all the other occur at most b times in I, we would have $s \leq \lceil n/k \rceil + (k-1)b$. This is a contradiction with the choice of s. Hence, some of the primes, say q has at least $b+1$ occurrences. Let $B \subseteq I$ be a set of $b+1$ occurrences of q. W.l.o.g. we may assume that $p < q$.

Now, let i be such that $1 \leq i \leq m$. Let t be such that $0 \leq t < m$ and $t \equiv i - (b+1)q \pmod{m}$. Let $C \subseteq I - A - B$ be a minimal subset of indices such that $\sum_{j \in C} w_j \geq t$. Such a set C exists, since $\sum_{j \in I-A-B} w_j \geq b/2(s - 2(b+1))$ and this is at least m by our assumption on s. Let $r = \sum_{j \in C} w_j - t$. Since C was minimal, we have $r \leq b$. By Lemma 2.5, there are $x, y \leq b+1$ satisfying $xp - yq = -r$. Now, let y_j be 1 for x indices j from the set A, for $b+1-y$ indices from B and all indices from C. Let y_j be zero otherwise. Then, we have $\sum_{j \in I} w_j y_j = xp + (b+1-y)q + t + r \equiv i$. $\square$

Theorem 2.7 *For every fixed $\varepsilon > 0$ and every n large enough, there is a weight system $w_1, w_2, \ldots, w_n$ constructible in time polynomial in n that is (s, n)–complete for $s = \lceil (2+\varepsilon) n^{2/3} \ln^{1/3} n \rceil$.*

Proof: Let $b = \lceil n^{1/3} \ln^{2/3} n \rceil$ and $k = \lceil n^{1/3} \ln^{-1/3} n \rceil$. By the prime number theorem, for every n large enough, there are at least $0.95n/\ln n$ and at most $1.05n/\ln n$ primes less than n. Hence, there are at least $0.95b/\ln b - 1.05(b/2)/\ln(b/2) \geq 0.42b/\ln b$ primes between b and $b/2$. It is easy to verify that, for every n large enough, we have $k < 0.42\, b/\ln b$. Hence, there are at least k different primes between $b/2$ and b. Testing of primality of numbers less than n is trivialy computable in time polynomial in n. Hence, in time polynomial in n, we can find the first k primes between $b/2$ and b. By taking $\lceil n/k \rceil$ or $\lfloor n/k \rfloor$ of each of them appropriately, we obtain a system $w_1, w_2, \ldots, w_n$. It is easy to verify that $\max\{bk, b(k-1) + n/k, 2(b+1) + 2n/b\} \leq (2+\varepsilon) n^{2/3} \ln^{1/3} n$. Hence, Theorem 2.6 implies that the system $w_1, w_2, \ldots, w_n$ constructed above is (s, n)–complete for $s = \lceil (2+\varepsilon) n^{2/3} \ln^{1/3} n \rceil$ and every n large enough. $\square$

Now, the main result follows immediately from Theorems 2.2, 2.4 and 2.7.

Theorem 2.8 *There is a sequence of Boolean functions $\{f_n\}_{n=1}^{\infty}$ that is in P and such that f_n is a function of n variables and for every n large enough, every 1–b. p. computing f_n has size at least 2^{n-s}, where $s = 3n^{2/3} \ln^{1/3} n$.*

Bibliography

[1] L. Babai, P. Hajnal, E. Szemeredi and G. Turan, A lower bound for read-once-only branching programs, *Journal of Computer and Systems Sciences*, vol. 35 (1987), 153–162.

[2] A. Borodin, A.Razborov and R. Smolensky, On Lower Bounds for Read-k-times Branching Programs, *Computational Complexity* 3 (1993) 1 – 18.

[3] P. E. Dunne, Lower bounds on the complexity of one–time–only branching programs, In *Proceedings of the FCT, Lecture Notes in Computer Science*, 199 (1985), 90–99.

[4] S. Jukna, A Note on Read-k-times Branching Programs, *RAIRO Theoretical Informatics and Applications*, vol. 29, Nr. 1 (1995), pp. 75–83.

[5] S. Jukna, A. A. Razborov, Neither Reading Few Bits Twice nor Reading Illegally Helps Much, preprint

[6] E. A. Okolnishkova, Lower bounds for branching programs computing characteristic functions of binary codes (in Russian), *Metody diskretnogo Analiza,* 51 (1991), 61–83.

[7] S. J. Ponzio, A lower bound for integer multiplication with read-once branching programs, *Proceedings of 27's Annual ACM Symposium on the Theory of Computing*, Las Vegas, 1995, pp. 130–139.

[8] P. Savický, S. Žák, A Lower Bound on Branching Programs Reading Some Bits Twice, to appear in TCS.

[9] D. Sieling, New Lower Bounds and Hierarchy Results for Restricted Branching Programs, TR 494, 1993, Univ. Dortmund, to appear in *J. of Computer and System Sciences.*

[10] D. Sieling and I. Wegener, New Lower bounds and hierarchy results for Restricted Branching Programs, in *Proc. of Workshop on Graph-Theoretic Concepts in Computer Science WG'94*, Lecture Notes in Computer Science Vol. 903 (Springer,Berlin, 1994) 359 – 370.

[11] J. Simon, M. Szegedy, A New Lower Bound Theorem for Read Only Once Branching Programs and its Applications, *Advances in Computational*

Complexity Theory (J. Cai, editor), DIMACS Series, Vol. 13, AMS (1993) pp. 183–193.

[12] I. Wegener, On the Complexity of Branching Programs and Decision Trees for Clique Functions, *JACM* 35 (1988) 461 – 471.

[13] I. Wegener, Efficient data structures for the Boolean functions, *Discrete Mathematics* 136 (1994) 347 – 372.

[14] S. Žák, An Exponential Lower Bound for One-time-only Branching Programs, in *Proc. MFCS'84*, Lecture Notes in Computer Science Vol. 176 (Springer, Berlin, 1984) 562 – 566.

[15] S. Žák, A superpolynomial lower bound for $(1, +k(n))$- branching programs, in *Proc. MFCS'95*, Lecture Notes in Computer Science Vol. 969 (Springer, Berlin, 1995) 319 – 325.