A large lower bound for 1-branching programs

Petr Savický, Stanislav Žák*
Institute of Computer Science, Academy of Sciences of Czech Republic,
Pod vodárenskou věží 2, 182 07 Praha 8, Czech Republic
e-mail: {savicky,stan}@uivt.cas.cz

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 input bit at most once. There is a series of lower bounds for 1–b. p.'s. The largest known lower bound was $2^{n/2000}$ for a function of n variables, see [11]. In the present paper, a lower bound of $2^{n-o(n)}$ is given.

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 n input bits. 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 index of an input bit 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 input bit assigned to the node is tested and the out-going edge labeled by the actual value of the input bit 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. 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. One of the most basic restrictions are read once branching programs (1–b. p.), where the restriction is such that during each computation each input bit is tested at most once.

There is a series of lower bounds for 1–b. p.'s. For the Boolean function called half-clique only, a lower bound $2^{\Omega(\sqrt{n})}$, where n is the number of input bits, was proved in [16]. Independently, for the clique function with appropriate parameters, a lower bound $2^{\Omega(\sqrt{n})}$ was proved in [14]. A lower bound $2^{\sqrt{n/2}-O(1)}$ appeared in [3] for testing Hamiltonian cycle

*The research of both authors was supported by GA of the Czech Republic, grant No. 201/95/0976.

and perfect matching in a graph. In [1] a lower bound $2^{n/c}$ for a large c for the function parity of the number of triangles. This lower bound was improved in [11] to a lower bound $2^{n/2000}$ for the same function. In [11] a technique for proving lower bounds for 1–b. p.'s is described that generalizes most of the previous results on 1-b. p.'s. In [7], a bound $2^{\Omega(\sqrt{n})}$ is proved for multiplication.

Besides 1–b. p.'s more general models are investigated. Namely, k–b. p.'s and syntactic k–b. p.'s. In a k–b. p., every computation can test each input bit at most k times. Syntactic k–b. p.'s are k–b. p.'s, where the restriction of at most k tests of each input bit is applied not only to valid computations, but to all paths from the source to a sink. 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 k–b. p.'s for k up to $\Omega(\log n)$, exponential lower bounds are known, see [2], [4], [6]. The results [2], [4] apply even to nondeterministic syntactic k–b. p.'s. For general (nonsyntactic) k–b. p.'s, proving exponential lower bounds for explicit functions is an open problem even for $k = 2$.

Another generalization of 1–b. p.'s are $(1,+k)$–b. p.'s. In a $(1,+k)$–b. p., in any computation, at most k of input bits may be tested more than once. For such b. p.'s, exponential lower bounds are known, see [5], [8] and [17].

The 1–b. p.'s are interesting also as a data structure for representing the Boolean functions. If a function f is represented by a small 1–b. p., it is possible to test effectively if the function f is not a zero function (satisfiability test). It is also possible to count the number of assignments x satisfying $f(x) = 1$. Both these problems may be solved in time polynomial in the size of the representation. Moreover, there is a probabilistic test of equivalence of two 1–b. p.'s, working in time polynomial in the size of the input b. p.'s. Since for some other operations there are no efficient algorithms, even more restricted models are used, namely OBDDs (ordered binary decision diagrams). OBDDs are used e.g. for representing the Boolean functions in some CAD applications like design or verification of Boolean circuits. As a review paper see [15].

In the present paper, a lower bound $2^{n-o(n)}$ on the size of 1–b. p.'s is proved for an explicit function of n variables.

2 Integer weight systems

In this section, we prove existence of integer weight systems that are used in the next section to define a Boolean function.

Definition 2.1 A system of m integer weights $w_1, w_2, \ldots, w_m$ is (s,n)–complete, if for every subset $I \subseteq \{1,2,\ldots,m\}$ of size at least s and any index i, $1 \leq i \leq n$, there are $y_j \in \{0,1\}$ for $j \in I$ such that $\sum_{j \in I} w_j y_j \equiv i \pmod{n}$.

Lemma 2.2 *Let p, q, r and b be integers, p and q relatively prime. Moreover, let $b/2 \leq p < q \leq b$ and $0 \leq r < b$. Then there are nonnegative integers $x, y \leq b$ such that $xp - yq = -r$.*

Proof: By a 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, for any integer i, the pair $x_0 + iq$ and $y_0 + ip$ is also a solution of the equation from the lemma. Let i_0 be the minimal i such that $x_0 + iq \geq 0$. Let $x =_{\mathrm{df}} x_0 + i_0 q$ and $y =_{\mathrm{df}} y_0 + i_0 p$. We have $0 \leq x \leq q - 1 \leq b - 1$ and $y = xp/q + r/q < b + 1$. Since y is an integer, the lemma follows. □

Lemma 2.3 *Let b be a positive integer and let $w_1, w_2, \ldots, w_m$ be prime numbers. Let $b/2 \le w_1 \le \ldots \le w_m \le b < n$ and let the m numbers w_i consist of d different primes, each with at least $\lfloor m/d \rfloor$ and at most $\lceil m/d \rceil$ occurrences. Let s be an integer satisfying $s \ge \max\{bd + \lceil m/d \rceil, 2b + 2n/b\}$. Then the system $w_1, w_2, \ldots, w_m$ is (s, n)-complete.*

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

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

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

Now, let i be such that $1 \le i \le n$. Let t be such that $0 \le t < n$ and $t \equiv i - bq \pmod{n}$. Let $C \subseteq I - A - B$ be a minimal subset of indices such that $\sum_{j \in C} w_j \ge t$. Such a set C exists, since $\sum_{j \in I-A-B} w_j \ge b/2(s - 2b)$ and this is at least n by our assumption on s. Let $r = \sum_{j \in C} w_j - t$. Since C was minimal and $w_j \le b$ for all $j = 1, 2, \ldots, m$, we have $r < b$. By Lemma 2.2, there are $x, y \le b$ satisfying $xp - yq = -r$. Now, let y_j be 1 for x indices j from the set A, for $b - y$ indices from B and all indices from C. Let y_j be zero otherwise. Then, we have $\sum_{j \in I} w_j y_j = xp + (b-y)q + t + r \equiv i$. $\square$

For simplicity of notation, we shall introduce the following abbreviation. Let

$$s(n) = \lceil 2\, n^{2/3} \ln^{1/3} n \rceil.$$

In fact, the multiplicative constant 2 in this definiton may be replaced by any number $c > \sqrt[3]{9/2} = 1.65\ldots$. In order to simplify the proofs, we shall work with 2.

Theorem 2.4 *For every n large enough, there is an $(s(n), n)$-complete weight system $w_1, w_2, \ldots, w_n$ computable in time polynomial in n.*

Proof: Use Lemma 2.3 with $b = \lceil 0.8\, n^{1/3} \ln^{2/3} n \rceil$ and $d = \lceil n^{1/3} \ln^{-1/3} n \rceil$. By the prime number theorem, for any positive ε and any b large enough, there are at least $(1-\varepsilon)b/\ln b$ different primes between 1 and b and at most $(1+\varepsilon)(b/2)/\ln(b/2)$ of them are below $b/2$. It is easy to verify that for some small positive ε and every n large enough, we have

$$(1-\varepsilon)b/\ln b - (1+\varepsilon)(b/2)/\ln(b/2) = 1.2\,(1+o(1))(1-3\varepsilon)n^{1/3} \ln^{-1/3} n \ge d.$$

Hence, there are at least d different primes between $b/2$ and b. Since testing primality of any number less or equal to n may be trivially done in time polynomial in n, it is possible to find the required number of primes in time polynomial in n. In order to complete the proof of the lemma, it is sufficient to verify that $bd + \lceil n/d \rceil = 1.8(1 \pm o(1))n^{2/3} \ln^{1/3} n \le s(n)$ and $2b + 2n/b = 2.5(1 \pm o(1))n^{2/3} \ln^{-2/3} n \le s(n)$. $\square$

3 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 input bit is fixed to the specified value, while a $*$ means that the input bit remains free. We say that a partial input u is defined on I, if $u_i \in \{0,1\}$ for all $i \in I$ and $u_i = *$ otherwise.

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 $u_i \in \{0,1\}$ for all $i = 1, 2, \dots, n$.

For any input of length n, say x, and any integer j, let x_j denote x_k, where $1 \le k \le n$ and $k \equiv j \pmod{n}$.

Definition 3.1 *Let f be a Boolean function. We say that f is k–separable, if for every set $I \subseteq \{1,2,\dots,n\}$ such that $|I| \le n-k$ and for every two different partial inputs u, v defined on I we have $f|_u \ne f|_v$.*

Theorem 3.2 *Let n and s be integers and let a weight system $w_1, w_2, \dots, w_n$ be (s,n)–complete. Let f be defined as $f(x) = x_i$, where i is the unique index determined by the identity $i \equiv \sum_{j=1}^{n} w_j x_j (\mathrm{mod}\ n)$. Then, the function f is $(s+2)$–separable.*

Proof: Throughout this proof, the symbol $\equiv$ means a congruence relation modulo n. Denote $k = s+2$. Moreover, let $I \subseteq \{1,2,\dots,n\}$, $|I| \le n-k$ and let u, v be different partial inputs defined on I. Let

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

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

Case 1: $\Delta \not\equiv 0$.
First, we extend u and v to u' and v' by setting one or two positions not fixed in u and v. Choose any $j \notin I$. If also $j+\Delta \notin I$, u' and v' are created from u and v by setting the position j to 0 and the position $j+\Delta$ to 1. Hence, in this case, we have $u'_j = v'_j \ne u'_{j+\Delta} = v'_{j+\Delta}$. If $j+\Delta \in I$, the position j of both u' and v' is set in such a way that $u'_j = v'_j \ne v'_{j+\Delta}$. We have at least $k-2 = s$ positions that are still not specified in both u' and v'. Since the numbers $w_1, w_2, \dots, w_n$ are (s,n)–complete, 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 y be a partial input extending v such that the bits with indices outside I have the same value in x and y. Then, $f(y) = v'_{j+\Delta}$ and hence $f(x) \ne f(y)$.

Case 2: $\Delta \equiv 0$.
Let $j \in I$ be such that $u_j \ne v_j$. There is an extension x of u such that $\sum_{i=1}^{n} w_i x_i \equiv j$. Let y be a partial input extending v such that the bits with indices outside I have the same value in x and y. By our assumptions, we have $\sum_{i=1}^{n} w_i y_i \equiv j$. Hence $f(y) = v_j \ne u_j = f(x)$. $\square$

As in [11], for any node v of a b. p., let v^+ be the set of indices of input bits read in v or on some directed path starting in v. For a partial input x, let $S(x)$ be the set of specified bits in x. The following theorem is a consequence of Theorem 2.4 in [11]. For convenience of the reader, we present the proof of this theorem adopted for the special case.

Theorem 3.3 *Any 1–b. p. computing a k–separable function has size at least 2^{n-k}.*

Proof: Let P be a 1–b. p. computing a k–separable function f. For an input x, let $e(x)$ be the edge (u,v) such that both u and v belong to the computation path for x and $|u^+| \ge k+1$ and $|v^+| \le k$. For each x, such an edge exists and it is unique. Moreover, let x^* be the restriction of x to the bits read in the computation for x before reaching the node v. Hence, x^* is the minimal restriction of the input x that guarantees that the computation goes through the edge (u,v).

Fix some input x_0. We are going to count the inputs x satisfying $e(x) = e(x_0)$. To this end, we shall prove that for every x we have $e(x) = e(x_0)$ if and only if x extends x_0^*, and $S(x_0) = n - k$.

By definition of x_0^*, it is clear that the computation for any input x extending x_0^* goes through the edge $e(x_0)$. Hence, in this case, we have $e(x) = e(x_0)$.

In order to prove the other direction of the equivalence, assume that x is such that $e(x) = e(x_0)$, but x does not extend x_0^*. Hence, there is a node w reached in the computation path for both x and x_0 before reaching u and such that the two computations leave w by different edges. Hence, the bit tested at w is specified in both x^* and x_0^*, but has different values in these two partial inputs. We shall derive from this a contradiction with the assumed k–separability of f.

Let $e(x) = e(x_0) = (u, v)$ and let i be the label of the node u. Then, both $S(x_0^*) \setminus \{i\}$ and $S(x^*) \setminus \{i\}$ are disjoint from u^+. Let $I = S(x_0^*) \cup S(x^*)$. We have $|I| \leq n - |u^+| + 1 \leq n - k$. Consider any extension x_0' of x_0^* and any extension y of x^* such that both these extensions specify all the bits with indices from I. Since x_0^* and x^* are incompatible, x_0' and y are different. On the other hand, the computations for both x_0' and y go through the node v. Since I and v^+ are disjoint, both the computations depend after the the visit of the node v only on bits with indices outside I. Hence, $f|_{x_0'} = f|_y$. This is a contradiction with k–separability of f.

To prove $|S(x_0^*)| = n - k$, assume for a moment that $|S(x_0^*)| < n - k$. Then, there is an index i of an input bit that is neither in $S(x_0^*)$ nor in v^+. Let y and y' be the extensions of x_0^* obtained by setting the bit i to 0 and 1 respectively. The computations for both y and y' go through (u, v). Hence, as above, we have two different settings of at most $n - k$ input bits giving the same subfunction of f. This is again a contradiction with k–separability of f.

Hence, there are exactly 2^k extensions of x_0^* and hence 2^k inputs x satisfying $e(x) = e(x_0)$. This is satisfied for any input x_0 and, hence, the number of edges in the b. p. is at least 2^{n-k}. This implies that the size of P is at least 2^{n-k-1}, since the out-degree of every node is at most 2.

In fact, one can prove the lower bound 2^{n-k} for the number of nodes of P using the following. The set of edges $e(x)$ for all $x \in \{0,1\}^n$ has the property that no two of these edges participate in a directed path. An easy argument shows that if G is a directed acyclic rooted graph such that the out-degree of each node is at most two and S is a subset of the edges with the above property, then $|S| \leq |G|$. $\Box$

Now, the main result follows immediately from Theorems 3.3, 3.2 and 2.4.

Theorem 3.4 *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 $s = \lceil 2\, n^{2/3} \ln^{1/3} n \rceil + 2$.*

It is interesting to compare this lower bound with the maximal complexity of 1–b. p.'s. Every function has a 1–b. p. of size $2^{n - \log n + O(1)}$ and there is a nonconstructive proof that some functions require size $2^{n - \log n - O(1)}$, see [13].

4 Remarks and an open problem

In [12], Szemeredi presented a proof of the following. If n is a prime, then $0, 1, 2, \ldots, n-1$ is an (s, n)–complete system for $s = n^{1/2} \log^{O(1)} n$. This yields an improvement on the present result.

Moreover, he proved in [12] that, if n is a prime, then an (s, n)–complete system may exist only if $s \geq \sqrt{n}$. Hence, a lower bound better than $2^{n-\sqrt{n}}$ may be obtained using the technique of the present paper, only if an (s, n)–complete system for a small s may be constructed using the assumption that n is a composite number. There is also a possibility to replace the weighted sum of input bits by a more general function $\phi : \{0,1\}^n \to \{1, 2, \ldots, n\}$. The following two conditions on such a function ϕ for some $s < \sqrt{n}$ are sufficient to prove a good lower bound:

1. If at most $n - s$ input bits are set to some constants, the restricted function ϕ has still all the values from $\{1, 2, \ldots, n\}$ in its range.
2. There is a polynomial $p(n)$ such that for every $I \subseteq \{1, 2, \ldots, n\}$, $|I| = n - s$, there is at most $p(n) - 1$ different settings of the bits with indices from I giving different subfunctions of ϕ on the remaining bits.

If a function ϕ satisfies these two conditions, then every 1–b. p. for the Boolean function $f(x) = x_{\phi(x)}$ has size at least $2^{n-s}/p(n)$. The proof uses the lower bound technique of [11] and it is omitted. It is an open problem if an explicitly defined function ϕ satisfying the two conditions for some $s < \sqrt{n}$ may be constructed. The weighted sum of input bits described in the present paper satisfies these two conditions with $s = s(n)$ and $p(n) = n + 1$.

References

[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] E. Szemeredi, personal communication.

[13] I. Wegener, *The complexity of Boolean functions*, Wiley-Teubner Series in Computer Science, 1987.

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

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

[16] 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.

[17] 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.