ELSEVIER

Theoretical Computer Science 172 (1997) 293–301

Theoretical Computer Science

Note

A lower bound on branching programs reading some bits twice

Petr Savický*, Stanislav Žák[1]

Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod vodárenskou věží 2, 182 07 Prague 8, Czech Republic

Received October 1995; revised June 1996
Communicated by A. Razborov

Abstract

By $(1, +k(n))$-branching programs (b.p.'s) we mean those b.p.'s which during each of their computations are allowed to test at most $k(n)$ input bits repeatedly. For a Boolean function computable within polynomial time a trade-off is presented between the size and the number of repeatedly tested input bits of any b.p. P computing the function. Namely, if at most $k(n)$ repeated tests are allowed, where $\log_2 n \leqslant k(n) \leqslant n/(1000 \log_2 n)$, then the size of P is at least $\exp(\Omega(n/(k(n)\log_2 n))^{1/2})$. This is exponential whenever $k(n) \leqslant n^\alpha$ for a fixed $\alpha < 1$ and superpolynomial whenever $k(n) = o(n/\log_2^3 n)$.

The presented result is a step towards a superpolynomial lower bound for 2-b.p.'s which is an open problem since 1984 when the first superpolynomial lower bounds for 1-b.p.'s were proven (Wegener, 1988; Žák, 1984). The present result is an improvement on (Žák, 1995).

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 the 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 represents 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

* Corresponding author. E-mails: {savicky,stan}@uivt.cas.cz.
[1] The research of both authors was supported by GA of the Czech Republic, Grant No. 201/95/0976.

out-going edge labeled by the actual value of the input bit is chosen. Finally, 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.

Branching programs were introduced because of their relation to the $P \overset{?}{=} LOG$ problem. Namely, 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. Besides the hope that these restricted models help to solve the general problem, some kinds of the restricted branching programs are important also as a data structure for representing Boolean functions in some CAD applications, e.g. design or verification of Boolean circuits, see e.g. [11].

One possible kind of restriction of the b.p.'s is to bound the number of repeated tests of each input bit. In a k-b.p., each computation can test each input bit at most k times. Even more restrictive assumption is the following. In a syntactic k-b.p., on each path from the source to a sink, each input bit is tested at most k times. This is more restrictive, since in a b.p., there may be paths that are not followed by any computation.

In 1984 the first superpolynomial lower bounds for 1-b.p.'s were proven [10, 12]. The first step towards the case of 2-b.p.'s were made with real-time b.p.'s, which perform at most n steps during each computation on any input of length n. The results were a quadratic lower bound [3], a subexponential lower bound [13] and an exponential lower bound [6]. For syntactic k-b.p.'s, exponential lower bounds have been proven, see [2, 4, 7]. The results of [2, 4] apply even to nondeterministic k-b.p.'s. However, the problem for 2-b.p.'s remains open.

There are also other generalizations of 1-b.p.'s, such that some input bits are allowed to be tested more than once. Namely, $(1, +k(n))$-b.p. is a b.p., where for each computation, the number of input bits tested more than once is at most $k(n)$. There is no restriction on the number of tests of these at most $k(n)$ input bits. Again, syntactic $(1, +k(n))$-b.p.'s are b.p.'s, where the above restriction is applied to any path from the source to a sink, not only to the computations.

For syntactic $(1, +k(n))$-b.p.'s, where $k(n)$ is bounded by $c\,n^{1/3}/\log_2^{2/3} n$ for an appropriate $c > 0$, an exponential lower bound and tight hierarchies (in $k(n)$) are presented in [8, 9].

In the present paper, we prove that every $(1, +k(n))$-b.p., where $\log_2 n \leqslant k(n) \leqslant n/(1000 \log_2 n)$, computing a function f_n has size at least $\exp\left(\Omega\left(n/(k(n)\log_2 n)\right)^{1/2}\right)$. This is exponential whenever $k(n) \leqslant n^\alpha$ for a fixed $\alpha < 1$ and superpolynomial whenever $k(n) = o(n/\log_2^3 n)$. The function f_n is computable by polynomial size general b.p.'s. Moreover, it is computable in polynomial time and, in fact, it is in uniform ACC. The present result is an improvement of [14], where a lower bound $\exp(\Omega(n/k(n)^2)^{1/4})$ for $(1, +k(n))$-b.p.'s and for another function was proved. The lower bound from [14] is superpolynomial whenever $k(n) = o(n^{1/2}/\log_2^2 n)$.

Recently, the lower bounds of the present paper were improved for characteristic functions of some well-known binary linear codes, see [5]. The largest lower bound is achieved there for BCH codes and it is $\exp\left(\Omega\left(\min\{n^{1/2}, n/k(n)\}\right)\right)$. This is super-polynomial for any $k(n) = o(n/\log_2 n)$.

2. The lower bound

First, we shall discuss some notation. As an input to a b.p. we shall consider also partial inputs. This means that the input bits may have values 0, 1 and $*$. In the last case, we say that the corresponding input bit is not specified by the (partial) input. For an input u let *comp*(u) denote the sequence of nodes of the b.p. visited during the computation on u. For a partial input u, we say that the computation for u leads to a node w, if w is the first branching node on the computation for u, which tests a bit not specified by u.

If u is a vector, then its ith coordinate is denoted by $u(i)$. By subscripts, we distinguish different vectors. If $u \in \{0, 1\}^n$, then $\|u\|$ denotes the number of ones in u.

Assume, P is a b.p. computing a function f and let f' be a subfunction of f obtained by setting some input bits of f to constants. By changing every branching node of P labeled by an input bit involved in the setting to a node with out-degree one and with an appropriate successor, we obtain a b.p. P' computing f'. The b.p. P' will be called a restriction of P.

Definition 2.1. Let $I = \{1, ..., n\}$ be the set of indices of the input bits. Let $u_1, u_2, \ldots, u_s$ be (partial) inputs and let for all $i = 1, 2, \ldots, s$, $A_i \subseteq I$ be the set of indices of the input bits specified in u_i. Let A_i be pairwise disjoint. Then, let $[u_1, u_2, \ldots, u_s]$ be the (partial) input specifying the bits with the indices from $\bigcup_{i=1}^{s} A_i$ such that if $j \in A_i$, then $[u_1, u_2, \ldots, u_s](j) = u_i(j)$.

Definition 2.2. Let a, b be (partial) inputs with the same set of specified bits. Let $D = \{i : a(i) \neq b(i)\}$. The pair a, b is called a forgetting pair, if there is a node w in the branching program such that w belongs to both *comp*(a) and *comp*(b) and both computations read all the bits with indices in D at least once before reaching w.

Lemma 2.3. *Let c be the size of a branching program P and let every computation of P read at least d different bits. Let $s \geqslant 0$ be a natural number such that $(2\log_2 c + 1)(s+1) \leqslant d$. Then there exist pairwise disjoint sets $A_i \subset \{1, 2, \ldots, n\}$ for $i = 1, \ldots, s+1$ and partial inputs $a_i : A_i \to \{0, 1\}$ and $b_i : A_i \to \{0, 1\}$, $a_i \neq b_i$ such that for all $i = 1, 2, \ldots, s+1$ we have*

(i) $|A_i| \leqslant 2\log_2 c + 1$,

(ii) *the inputs $[a_1, \ldots, a_{s+1}]$ and $[a_1, \ldots, a_{i-1}, b_i, a_{i+1}, \ldots, a_{s+1}]$ form a forgetting pair.*

Proof. Let $r = \lfloor \log_2 c \rfloor + 1 > \log_2 c$. By our assumption, every computation of P reads at least r input bits, since $r \leqslant 2\log_2 c + 1 \leqslant d$. We shall construct a sequence

$U_0, U_1, \dots, U_r$ of sets of partial inputs in such a way that for every $i = 0, 1, \dots, r$, the set U_i consists of 2^i inputs with exactly i specified bits. In particular, U_0 contains only the totally undefined input. Given U_i, the set U_{i+1} is constructed as follows. For each $u \in U_i$, follow the computation for u until the first bit not specified by u is required. Let j be its index. Then, include into U_{i+1} the inputs u_0 and u_1 extending u by setting the jth bit to 0 and 1, respectively. Now, since $2^r > c$, there are at least two distinct inputs in U_r, say v_1 and v_2, such that the computations for both these inputs lead to the same node. Note that, by construction of U_r, there is a bit specified by both v_1 and v_2, but with different values in v_1 and v_2. Moreover, for both $i = 1, 2$, the computation for v_i reads all the bits specified by v_i.

Let C_1 be the set of bits tested by $comp(v_1)$ and C_2 be the set of bits tested by $comp(v_2)$. Let $A_1 = C_1 \cup C_2$. Since $C_1 \cap C_2 \neq \emptyset$, $|A_1| \leqslant 2r - 1 \leqslant 2\log_2 c + 1$. Let a partial input a_1 extend v_1 in such a way that the bits with indices in A_1 not determined in v_1 are equal to the values of these bits in v_2. Similarly, b_1 extends v_2 so that the bits undefined in v_2 have the same value as in v_1. Hence, a_1 and b_1 may differ on the bits with indices from $C_1 \cap C_2$ whereas the bits with indices from the symmetric difference of C_1 and C_2 are specified in both a_1 and b_1 and are equal. One can easily see that $comp(a_1)$ follows $comp(v_1)$ and $comp(b_1)$ follows $comp(v_2)$ until $comp(a_1)$ and $comp(b_1)$ join in some node w. All the bits, where a_1 and b_1 differ, are read on both computations before reaching w. Hence, a_1 and b_1 form a forgetting pair.

If A_j, a_j and b_j for all $j = 1, \dots, i$, where $i < s + 1$, are already constructed, we continue in the following way: consider only those inputs which equal a_j on A_j for all $j = 1, \dots, i$. These inputs define a restriction P_i of P. Since $|\bigcup_{j=1}^{i} A_j| \leqslant (2\log_2 c + 1)s \leqslant d - 2\log_2 c - 1$ and $r \leqslant 2\log_2 c + 1$, each computation of P_i tests at least r input bits. Otherwise, during a computation of P less than d bits are tested. This would be a contradiction to the assumptions of the lemma. Now, the construction of $U_0, U_1, \dots, U_r$ applied in the first part of the proof to P is applied to P_i. Using this, a_{i+1}, b_{i+1} and A_{i+1} are defined in the same way as a_1, b_1 and A_1 above. Note that, by construction, A_{i+1} contains only input bits not in $\bigcup_{j=1}^{i} A_j$.

It follows from the construction that (i) is satisfied. Moreover, the construction of a_i and b_i implies that the computations for the partial inputs $[a_1, \dots, a_i]$ and $[a_1, \dots, a_{i-1}, b_i]$ lead to the same node. Hence, the inputs $[a_1, \dots, a_{s+1}]$ and $[a_1, \dots, a_{i-1}, b_i, a_{i+1}, \dots, a_{s+1}]$ form a forgetting pair. $\square$

Definition 2.4. For every nonzero natural numbers m, t let $g_{m,t}$ be the Boolean function of $(m+1)t$ variables defined as follows. The input of $g_{m,t}$ is treated as an m by t matrix A over $GF(2)$ and a vector u over $GF(2)$ of length t. Then, let $g_{m,t}(A, u) = 1$ if and only if $Au = 0$.

In order to prove that $g_{m,t}$ is hard for $(1, +s)$-b.p. for some s, we will find an appropriate m by t matrix A and then we prove a lower bound on the size of any $(1, +s)$-b.p. computing $g_{m,t}(A, u)$ as a function of u only. The properties of the matrix A needed for this are stated in the following definition.

Definition 2.5. An m by t matrix A will be called hard, if the two following conditions are satisfied:

(a) For every nonzero $u \in \{0,1\}^t$, if $\|u\| < m/\log_2 t$, then $Au \neq 0$.

(b) For every partial input u with at most $\lfloor t/4 \rfloor$ specified coordinates, there exists a total input u' extending u such that $Au' = 0$.

Since the multiplication of a vector u by a fixed matrix A is a linear operation, the property (a) is equivalent to the following: If u, v are distinct vectors, $Au = 0$ and the Hamming distance of u and v is less than $m/\log_2 t$, then $Av \neq 0$. In terms of the linear codes, this means that the code $\{u : Au = 0\}$ has the minimum distance at least $m/\log_2 t$.

Lemma 2.6. *Let m and t be some nonzero natural numbers such that $m > \log_2 t$ and a hard m by t matrix exists. Let s be such that $4m(s+1) \leqslant t \log_2 t$. Then every $(1,+s)$-b.p. computing $g_{m,t}$ has the size at least $2^{((m/\log_2 t)-1)/2}$.*

Proof. Assume, A is a hard m by t matrix. Assume that a b.p. P computes $g_{m,t}$ and each of its computations reads at most s input bits repeatedly. Let P' be the restriction of P computing $g_A(u) = g_{m,t}(A,u)$. By definition, the function g_A is a function of t variables. Clearly, P' is not larger than P and it is also $(1,+s)$-b.p. Let the size of P' be c and let the minimum number of input bits read on a computation path be d. Then $d > t/4$, because every partial input u of at most $t/4$ specified bits may be extended to an input x with $g_A(x) = 1$ and also to x' with $g_A(x') = 0$. The existence of x follows from (b). The input x' may be chosen as an extension of u differing from x in one coordinate. By (a), we then have $g_A(x') = 0$.

The lower bound on the size of P' is proved by contradiction. Assume that

$$2 \log_2 c + 1 < m/\log_2 t. \tag{1}$$

Since $(s+1) \leqslant (t \log_2 t)/(4m)$ by the assumptions of the lemma, we obtain

$$(2 \log_2 c + 1)(s+1) \leqslant t/4. \tag{2}$$

This together with $d > t/4$ implies that the assumptions of Lemma 2.3 are satisfied for P' and g_A. Consider the partial inputs $[a_1,\ldots,a_{s+1}]$ and $[a_1,\ldots,a_{i-1},b_i,a_{i+1},\ldots,a_{s+1}]$ guaranteed by the lemma. By (2), $[a_1,\ldots,a_{s+1}]$ specifies at most $t/4$ input bits. Hence, (b) from the definition of a hard matrix implies that it is possible to extend $[a_1,\ldots,a_{s+1}]$ to an input $x = [a_1,\ldots,a_{s+1},a]$ with $g_A(x) = 1$. For $i = 1,\ldots,s+1$ let $y_i = [a_1,\ldots,b_i,\ldots,a_{s+1},a]$. The inputs x and y_i form a forgetting pair. Moreover, the Hamming distance of x and y_i is at most $2\log_2 c + 1 < m/\log_2 t$. Hence, by (a) we have $g_A(y_i) = 0$. Consider the node w from Definition 2.2 applied to x and y_i. This node is reached by the computation for x and also for y_i. However, since $g_A(x) \neq g_A(y_i)$, the computations for x and y_i reach different sinks. Hence, there is a node w' reached by both computations, such that either $w' = w$ or w' is reached after w and such that in w' an input bit differing x and y_i is read. Since this bit was read also before reaching w, it is read twice in both computations. Since the sets of input bits, where x and y_i differ are disjoint for different i's, the computation for x reads at least $s+1$ input bits twice. This is a contradiction. Hence, (1) is not satisfied and we have $\log_2 c \geqslant (m/\log_2 t - 1)/2$. □

Lemma 2.7. *For every large enough natural number t and any natural number m satisfying $6 \log_2 t \leqslant m \leqslant t/8$, there exists a hard m by t matrix A.*

Proof. For the proof of the existence of A we will replace (b) by the following stronger property.

(b′) The linear span of any $t - \lfloor t/4 \rfloor$ columns of A is the whole space $\{0,1\}^m$.

By the following argument, (b′) $\Rightarrow$ (b). If u' is a total input extending a partial input u, then Au' is a sum of two groups of columns in A. The first group are the columns corresponding to nonzero coordinates in the partial input u and the second group correspond to nonzero coordinates extending u to u'. Because of (b′), for any u of at most $\lfloor t/4 \rfloor$ specified coordinates it is possible to choose the extension u' in such a way that the two groups of columns have the same sum. Then, the total sum is zero. This implies (b).

The existence of A with (a) and (b′) will be established by proving that a matrix chosen at random satisfies these properties with positive probability. Assume, the entries of A are equal to 1 with probability $\frac{1}{2}$ and that they are independent.

Let B_u be the event $Au = 0$. If u is a fixed nonzero vector, then Au is uniformly distributed over $\{0,1\}^m$. Hence, the probability of B_u is 2^{-m}. The probability of the disjunction of B_u over some set of vectors u is at most the sum of the probabilities of corresponding B_u. Hence, the probability that (a) is not satisfied is at most 2^{-m} times the number of nonzero u, $\|u\| < m/\log_2 t$. It is easy to verify that for every l at most $(t+1)/3$ we have, using also Stirling's formula,

$$\sum_{j=1}^{l} \binom{t}{j} \leqslant 2\binom{t}{l} \leqslant 2\left(\frac{te}{l}\right)^l.$$

Using this and the fact that the last expression is nondecreasing in l, if l is considered as a real variable between 0 and t, we obtain that the probability that (a) is not satisfied is at most

$$2\binom{t}{\lfloor m/\log_2 t \rfloor} 2^{-m} \leqslant 2\left(\frac{et\log_2 t}{m}\right)^{m/\log_2 t} 2^{-m}. \tag{3}$$

By the assumptions of the lemma, $m \geqslant 2e \log_2 t$. This implies that the fraction inside the bracket in (3) is at most $t/2$. Hence, (3) is bounded by

$$2\left(\frac{t}{2}\right)^{m/\log_2 t} \cdot 2^{-m} = 2 \cdot 2^{-m/\log_2 t} \cdot 2^m \cdot 2^{-m} \leqslant \frac{1}{32}. \tag{4}$$

Now, let us consider the property (b′). A group of columns generates the whole space $\{0, 1\}^m$ iff it is not contained in any subspace of dimension $m - 1$, i.e. in a set of the form $\{z : \langle v,z\rangle = 0\}$, where v is a nonzero vector from $\{0,1\}^m$ and $\langle\cdot,\cdot\rangle$ denotes the scalar product mod 2. Consider the following property.

(b″) Every nonzero vector $v \in \{0,1\}^m$ has nonzero scalar product with at least $\lfloor t/4\rfloor + 1$ columns of A.

We shall prove that (b″) and (b′) are equivalent. To prove (b′) $\Rightarrow$ (b″), assume that (b″) is not satisfied. Then, there is a group of $t - \lfloor t/4\rfloor$ columns of A with zero scalar product with some nonzero v. This contradicts (b′), since these columns do not generate the whole $\{0,1\}^m$. To prove (b″) $\Rightarrow$ (b′), consider some group of $t - \lfloor t/4\rfloor$ columns of A. By (b″) and the characterization of subspaces from above, for any subspace of $\{0,1\}^m$ of dimension $m-1$, there is a column in this group not contained in the subspace. This implies (b′).

The probability of (b″) may be estimated as follows. Fix some nonzero $v \in \{0,1\}^m$ and let X_i be the scalar product of v and the ith column of A. The X_i''s are independent and $\Pr(X_i = 1) = \frac{1}{2}$. Let $h = t/2 - \lfloor t/4\rfloor \geqslant t/4$. The probability that (b″) is not satisfied for the given particular v is, by Chernoff inequality (see e.g. [1]),

$$\Pr\left(\textstyle\sum_{i=1}^{t} X_i \leqslant \lfloor t/4\rfloor = t/2 - h\right) \leqslant e^{-2h^2/t} \leqslant e^{-t/8}$$

Since there are $2^m - 1$ nonzero vectors v, the probability that (b″) is not satisfied for at least one of them is the probability of the disjunction of $2^m - 1$ events of probability at most $e^{-t/8}$. Hence, the probability that (b″) is not satisfied is at most $(2^m - 1)e^{-t/8}$. This is at most $(2/e)^{t/8}$, since $m \leqslant t/8$. Hence, the probability that (b″) is not satisfied tends to zero with increasing t. Together with (4), this implies that the probability that a random matrix does not satisfy some of the conditions (a) and (b″) is less than 1 for t large enough. This proves the existence of the required hard matrix A. □

Definition 2.8. For every natural number $n \geqslant 6$, let the Boolean function f_n be defined as follows. The first $2\lceil \log_2 n\rceil$ bits are considered as the binary representation of natural numbers m and t. If $m = 0$ or $t = 0$ or $(m+1)t > n - 2\lceil \log_2 n\rceil$, the function f_n is 0. Otherwise, f_n is equal to $g_{m,t}$ applied to the next $(m+1)t$ bits after the representations of m and t.

Theorem 2.9. *Let* $\log_2 n \leqslant s(n) \leqslant n/(1000 \log_2 n)$. *Then for every n large enough, the size of any* $(1,+s(n))$*-b.p. computing the function* f_n *is at least*

$$\exp\left(\Omega\left(\frac{n}{s(n)\log_2 n}\right)^{1/2}\right).$$

This is superpolynomial, whenever $s(n) = o(n/\log_2^3 n)$.

Proof. Let P be a $(1,+s(n))$-b.p. computing f_n. In order to prove the lower bound on the size of P, we choose appropriate numbers m and t such that $(m+1)t \leqslant n - \lceil 2\log_2 n\rceil$. Then, we set the first $2\lceil \log_2 n\rceil$ input bits to the binary representation of m and t. As the result of this restriction, we obtain a $(1,+s(n))$-b.p. P'. By definition of f_n, P'

computes $g_{m,t}$. Then, the size of P' is estimated using Lemmas 2.6, 2.7. Since P' is a subprogram of P, the lower bound obtained for P' in this way is also valid for P.

Let

$$m = \left\lfloor \frac{1}{4} \cdot \sqrt{\frac{n \log_2 n}{s(n)}} \right\rfloor,$$
$$t = \left\lfloor 3 \cdot \sqrt{\frac{ns(n)}{\log_2 n}} \right\rfloor.$$

Clearly, $(m+1)t \leqslant 3/4 \cdot n + o(n) \leqslant n - 2\lceil \log_2 n \rceil$. Hence, by setting the first $2\lceil \log_2 n \rceil$ bits as described above, we obtain a b.p. P' computing $g_{m,t}$.

In order to apply Lemma 2.7, we have to verify $6 \log_2 t \leqslant m \leqslant t/8$. Clearly, $\sqrt{n} \leqslant t \leqslant n$. Hence, $\frac{1}{2} \cdot \log_2 n \leqslant \log_2 t \leqslant \log_2 n$. It follows that

$$\frac{m}{\log_2 t} = (1 - o(1)) \cdot \frac{1}{4} \cdot \frac{\sqrt{\frac{n \log_2 n}{s(n)}}}{\log_2 t} \geqslant \frac{1}{5} \cdot \sqrt{\frac{n}{s(n) \log_2 n}}. \tag{5}$$

By substituting the maximal possible value of $s(n)$, we obtain

$$\frac{m}{\log_2 t} \geqslant \frac{1}{5} \cdot \sqrt{1000} \geqslant 6.$$

Since, moreover, $m/t = (1 \pm o(1)) \cdot \frac{1}{12} \cdot (\log_2 n)/s(n) < \frac{1}{8}$, the assumptions of Lemma 2.7 are satisfied. Hence, there is an m by t hard matrix.

In order to use Lemma 2.6, it remains to verify its assumption on $s = s(n)$, namely $4m(s(n)+1) \leqslant t \log_2 t$. We have

$$\frac{t \log_2 t}{4m} = (1 \pm o(1)) \cdot 3 \cdot \frac{s(n)}{\log_2 n} \log_2 t \geqslant (1 \pm o(1)) \cdot \frac{3}{2} \cdot s(n) \geqslant s(n) + 1.$$

Lemma 2.6 implies that the size of P' and hence also of P is $\exp(\Omega(m/\log_2 t))$. This implies the theorem using (5). □

Note that the function $g_{m,t}$ is computable by branching programs of linear size and f_n by polynomial branching programs. Both these functions are in P.

References

[1] B. Bollobás, *Random Graphs* (Academic Press Inc., London, 1985).

[2] A. Borodin, A. Razborov and R. Smolensky, On lower bounds for read-k-times branching programs, *Comput. Complexity* **3** (1993) 1–18.

[3] M. Ftáčnik and J. Hromkovič, Nonlinear lower bound for real-time branching programs, *Comput. Artificial Intelligence* **4** (1985) 353–359.

[4] S. Jukna, A note on read-k-times branching programs, *RAIRO Inform. Theor. Appl.*, 29 (1) (1995) 75–83.

[5] S. Jukna and A. Razborov, Neither reading few bits twice nor reading illegally helps much, TR96-037, ECCC, Trier.

[6] K. Kriegel and S. Waack, exponential lower bounds for real-time branching programs, in: *Proc. FCT'87*, Lecture Notes in Computer Science, Vol. 278 (Springer, Berlin, 1987) 263 - 267.

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

[8] D. Sieling, New lower bounds and hierarchy results for restricted branching programs, Tech. Report 494, Univ. Dortmund 1993; *J. of Comput. System Sci., to appear.*

[9] D. Sieling and I. Wegener, New lower bounds and hierarchy results for restricted branching programs, in: *Proc. Workshop on Graph-Theoretic Concepts in Computer Science WG'94*, Lecture Notes in Computer Science, Vol. 903 (Springer,Berlin, 1994) 359–370.

[10] I. Wegener, On the complexity of branching programs and decision trees for clique functions, *JACM* **35** (1988) 461–471.

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

[12] 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.

[13] S. Žák, An exponential lower bound for real-time branching programs, *Inform. and Control* **71** (1/2) (1986) 87–94.

[14] 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.