A Lower Bound Method for Branching Programs and Its Application

Žák, Stanislav
2012

Institute of Computer Science
Academy of Sciences of the Czech Republic

A Lower Bound Method for Branching Programs and Its Application

Stanislav Žák

Technical report No. 1171

January 2012

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: +420 266 051 111, fax: +420 286 585 789,
e-mail:ics@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

A Lower Bound Method for Branching Programs and Its Application[1]

Stanislav Žák[2]

Technical report No. 1171

Abstract:

We introduce a new lower bound method for branching programs (b. p.'s). It is based on the observation that sometimes in the branching programs there are some local effects ensuring some minimal size of the local part of the b. p. in question.

The idea of the method is to convert such local effects to the lower bound for the global size of the b. p. in question. We define a kind of such local effects and we prove a theorem converting these effects into a lower bound for each b. p. containing them.

Further we define a Boolean function and we demonstrate that the defined effects are present within a human-like branching program computing this function. Moreover we prove that such effects are present in each "small" read-once branching program assumed in the proof by contradiction. By application of the theorem we obtain that each such b. p. must be "large". In other words we obtain a lower bound for read-once b. p.'s computing this function. The bound is of a subexponential size, i. e. of the standard quality. The proof of the lower bound demonstrates that our method is an effective one.

Keywords:
branching programs, lower bound techniques

[1]This research was partially supported by the projects GA ČR P202/10/1333 and RVO: 67985807.
[2]stan@cs.cas.cz

A Lower Bound Method for Branching Programs and Its Application

Stanislav Žák*

Institute of Computer Science, Academy of Sciences of the Czech Republic,
P. O. Box 5, 18207 Prague 8, Czech Republic, stan@cs.cas.cz

Abstract. We introduce a new lower bound method for branching programs (b. p.'s). It is based on the observation that sometimes in the branching programs there are some local effects ensuring some minimal size of the local part of the b. p. in question.
The idea of the method is to convert such local effects to the lower bound for the global size of the b. p. in question. We define a kind of such local effects and we prove a theorem converting these effects into a lower bound for each b. p. containing them.
Further we define a Boolean function and we demonstrate that the defined effects are present within a human-like branching program computing this function. Moreover we prove that such effects are present in each "small" read-once branching program assumed in the proof by contradiction. By application of the theorem we obtain that each such b. p. must be "large". In other words we obtain a lower bound for read-once b. p.'s computing this function. The bound is of a subexponential size, i. e. of the standard quality. The proof of the lower bound demonstrates that our method is an effective one.

Key words: branching programs, lower bound techniques

1 Introduction

The main problem of the theory of branching programs is to prove the superpolynomial lower bound for a concrete Boolean function from P (or at least NP). This would solve the long-standing problem $LOG \neq ?P$. The main stream of the research has followed the line to prove the superpolynomial lower bounds for restricted branching programs, and gradually to make the restrictions less strong. The most famous restriction was the read-once branching programs. There are many lower bound results concerning read-once branching programs which have arose in the eighties and nineties (see [8]).

This period was followed by the second phase of proofs of superpolynomial lower bounds for branching programs strongly less restricted than read-once

* This research was partially supported by the projects GA ČR P202/10/1333 and RVO: 67985807.

branching programs [3], [7], [6]. The milestone was the result of Ajtai [1] and then [2] with superpolynomial lower bounds for branching programs computing within linear time and time $n.log\ n$, resp.

The present period faces the challenge to prove a stronger result than that of Ajtai. In this text we work in the direction to cross over this barrier. The main deficit of the theory of branching programs is the lack of any method (potentially) yielding lower bounds for general branching programs. We attempt to build up such a method. Our research is based on an observation that in branching programs it is possible to find some local effects ensuring that the branching program is "relatively large" at least in the local places in question. The planned method would derive the size of the branching program (i. e. the lower bound) from such local effects.

In the present text we define a kind of such local effects and we prove a theorem converting these effects into a lower bound for each b. p. containing them. For verifying that the chosen effects are realistic, we define a Boolean function (suspicious that it is not in LOG) and we demonstrate that the defined effects are present within a human-like branching program computing this function. Moreover we prove that such effects are present in each "small" read-once branching program assumed in the proof by contradiction. By application of the theorem we obtain that each such b. p. must be "large". In other words we obtain a lower bound for read-once b. p.'s computing this function. The bound is of a subexponential size, i. e. of the standard quality. The proof of the lower bound demonstrates that our method is an effective one.

This text is a starting point for the research destined for construction of the general method based on "local" effects.

2 Technical Preliminaries

By a branching program (b.p.) P (over binary inputs of length n) we mean a finite, oriented, acyclic graph with one source (in-degree = 0) where all nodes have out-degree = 2 (so-called branching or inner nodes) or out-degree = 0 (so-called sinks). The branching nodes are labeled by variables x_i, $i = 1, ..., n$, one out-going edge is labeled by 0 and the other by 1, the sinks are labeled by 0 or by 1. If a node v is labeled by x_i we say that x_i is tested at v.

For an input $a = a_1...a_n \in \{0,1\}^n$ by $comp(a)$ we mean the sequence of nodes starting at the source of P and ending in a sink. In the sequence for each i, $1 \leq i \leq n$, at any node with label x_i the next node is pointed by the edge with label a_i.

A special case of b.p. with in-degree = 1 in each node (with exception of the source) is called decision tree. Let T be a decision tree and let v be its node labeled by x_i. Let T_0,T_1 be the trees rooted by two edges e_0, e_1 out-going v and labeled by 0 and 1. If we redirect the edges e_0 or e_1 to a node in T_0 or T_1, resp., (and remove all nodes and edges which become unreachable from the root of T) we obtain a decision subtree of T.

If a node $v \in comp(a)$ we say that a reaches v. If a and b reach v and immediately below v they reach different nodes we say that $comp(a)$ and $comp(b)$ diverge in v (or shortly a and b diverge at v). Similarly for more than two inputs.

P computes function f_P which on each $a \in \{0,1\}^n$ outputs the label of the sink reached by a.

We say that P computes in time $t(n)$ if each its computation is of the length at most $t(n)$.

The well-known class of restricted b.p.'s are so-called read-once branching programs in which along each computation each variable is tested at most once. Read-once b.p.'s compute in time n, of course.

If $comp(a)$ has a common part with a path p in P we say that a follows p.

By a distribution we mean any mapping D of (a subset of) $\{0,1\}^n$ to (the set of nodes of) P with the property that for each a $D(a)$ is a node of $comp(a)$ ($D(a) \in comp(a)$). The class of the distribution at node v is the set of all a's mapped to v.

Let v be a node of P and let A be a set of some (not necessarily all) inputs reaching v. We say that T is a tree developed in v according to P with respect to A iff the branches of T simply follow (only) the paths of P starting at v and followed by inputs from A till the sinks (in T no joining of paths is allowed, of course). Moreover each edge pointing to a node with out-degree = 1 in T is repointed to its successor. Hence in T each node has out-degree = 2 with exception of its leaves.

By the size of P we mean the number of its nodes. By the complexity of a Boolean function f we mean the size of the minimal b.p.'s computing f.

It is a well-known fact that superpolynomial lower bound on the size of b.p.'s implies superlogarithmic lower bound for space complexity of Turing machines.

3 Combinatorics

We introduce some formal mathematics which will be useful in the next section.

Definition 1. *Let U, P, T, Z be natural numbers, $U < \frac{P}{T} < Z$. Let S be a sequence of T places. Let P pebbles be distributed to the places of S. Let the maximal number of pebbles distributed to one place is Z. Let R be a subsequence of S. By its S-length we mean the distance of its first and its last place in S. We say that R is a U-subsequence*

iff

i) To each place of R at least U pebbles are distributed.
ii) S-length of R is at most U.
iii) R is a maximal subsequence (what concerns the number of its places) of its kind.

Lemma 1. *Let U, P, T, Z, S be as above. Then there is a U-subsequence R of S with the number of its places at least $\alpha \geq \frac{\frac{P}{T}-U}{\frac{Z}{U}-1}$.*

Proof. The number of places in S which have at least U pebbles and which are therefore candidates to R is at least $\frac{P-T.U}{Z-U}$. Let us divide S into $\frac{T}{U}$ intervals of length U and let us consider the number of candidates in each interval. It is clear that in such one interval there are at least $\frac{P-T.U}{Z-U}(\frac{T}{U})^{-1}$ candidates. Hence $\alpha \geq \frac{\frac{P}{T}-U}{\frac{Z}{U}-1}$.

□

4 Lower Bound

First we introduce a definition which will be needed in the proof of the lower bound theorem.

Definition 2. *Let B be a decision tree. Let M be a set of inputs, $M \subseteq \{0,1\}^n$. Let L_B be the number of all leaves of B and L_B^M be the number of all leaves of B reached by inputs from M. By M-ratio of B we mean the number $p_B^M = \frac{L_B^M}{L_B}$.*

Let T be a subtree of B. Let L_T be a number of all leaves of T and $L_T^{M,B}$ be the number of all leaves of T reached (in B !) by inputs from M. By (M,B)-ratio of T we mean the number $p_T^{M,B} = \frac{L_T^{M,B}}{L_T}$.

The next definition is a key one for this text.

Definition 3. *Let $C = \{C_1, ..., C_k\}$ be a partition of a subset of the set of input bits $\{1, ..., n\}$. Let s be a function, $s(n) \leq n$.*
We say that a branching program P is a (C, s)-branching program
iff

there is a set A of input strings of size $2^{s(n)}$ such that for each $a \in A$ and for each $C_1, C_2 \in C, C_1 \neq C_2$ there is a node $v = v_{a,C_1,C_2} \in P$ and an input string x such that
i) a, x reach v.
ii) In v, a bit from C_1 is tested.
iii) In v, x follows a (i. e. follows the same outgoing edge).
iv) below v, x diverges a for the first time by a test to a bit from C_2.
v) Each $b \in A$ diverging a at v may meet a only after the moment when a diverges x.

Theorem 1. *Let P be a (C, s)-reasonable branching program with $|C| \leq \frac{n}{2}$ computing in time $T(n) \geq n$. Then the size of P is at least $2^{\frac{\alpha + s(n) - n - 1}{2}}$ where $\alpha \geq (\frac{\binom{|C|}{2}}{T(n)} - U).(\frac{|C|-1}{U} - 1)^{-1}$ for any $U < \frac{\binom{|C|}{2}}{T(n)}$.*

Proof. Let P be a (C, s)-reasonable branching program computing in time $T(n)$.

Then there is a set A of inputs of the cardinality at least $2^{s(n)}$ with the properties from the previous definition .

For each $a \in A$ we understand the nodes of $comp(a)$ as a sequence of places. Let $v \in comp(a)$ be a node (place) and let C_1 be the partition class of the variable tested in v. By the number of pebbles on v we mean the number of partition classes $C_2 \in C$ for which $v = v_{a,C_1,C_2}$ (cf. the previous definition).

We see that the number of pebbles distributed along any computation $comp(a)$, $a \in A$, is $\binom{k}{2}$ where k is the number of partition classes in C. The maximum of pebbles distributed to one node of $comp(a)$ is at most $k - 1$. According to the lemma from the previous section for any $U < \frac{\binom{k}{2}}{T(n)}$ (which implies $U < \frac{n}{4}$!) we have a U-subsequence of $comp(a)$ with at least $\alpha = \alpha_U$ nodes. The inequality in the statement of the theorem follows from the lemma.(To each node of this sequence at least U pebbles are distributed.)

Let us distribute each $a \in A$ to the first node of its U-subsequence S_a. Let v be the node with a maximal class M of this distribution. Hence $|M| \geq \frac{2^{s(n)}}{|P|}$.

From v we develop the syntactic tree B' according to P ("syntactic" means that we take into account also the branches which are not followed by any input e.g. the branches arising in case of repeated tests on the same variable) till the depth U in P (this depth is below all at least α nodes of S_a for each a). Below this depth we continue the developing of the tree ignoring the repeated tests. On each branch b at the node corresponding to the related sink of P we add some subtree S_b which on each their branches tests all variables not tested below v till now. We know that the length of branches (constructed till now) is at least n and that each such branch is followed by at most one input (of length n). The branches (constructed till now) can be of different length (due to the

possible different number of repeated tests in the first interval of length U along different branches). It is easy to see that the maximal length is $n+\delta$, $\delta \leq U$. To obtain a full tree B' of certain length we add some full tree T_b of appropriate length to each branch b shorter than the longest one of branches constructed till now. As a result we obtain a full binary decision tree B' of depth $n+\delta$, $\delta \leq U$.

We modify B' as follows. Below v till the depth U in the middle of each edge we insert a new node with the same test as in the node from which the edge in question out-goes. On the dead edge of this test we add a dummy full binary tree D of depth $log(U)+\alpha+\delta$. We see that in D U full binary trees of depth $\alpha+\delta$ are hidden. Below the depth U we add dummy subtrees of depth $\delta-l$ where l is the depth of the level of B' in question below U. Maximal l for this operation is δ.
Let B be the resulting tree. The number of its leaves is at most
$2^{n+\delta}$ +
$+2^U.2^{log(U)+\alpha+\delta}$ +
$+\sum_{l=0}^{\delta} 2^l.2^{\delta-l} \leq$
$2^{n+\delta}+2^{3.U+log\ U}+2^{U+log\ U} \leq$
$2^{n+\delta}+2^{\frac{3n}{4}+log(\frac{n}{4})+1}$
$\leq\ 2^{n+\delta+1}$.

It follows that M-ratio of B
$p_B^M \geq \frac{|M|}{2^{n+\delta+1}}$.

It is easy to see that for nodes of B' being in the same depth the subtrees of B rooted in them have the same number of leaves.

For the proof of our theorem it suffices to prove the next lemma.

Lemma 2. *There is a full binary decision tree T of depth $\alpha+\delta$ rooted in v such that*

a) Each its branch (considered as a sequence of nodes) is a subsequence of a branch of B.

b) $p_T^{M,B} \geq p_B^M$.

c) The nodes of T reached (in B) by inputs from M in depth (in T) α are pairwise different in P.

d) The number of leaves of T reached (in B) by inputs from M is the same as the number of nodes of T reached (in B) by inputs from M on level α.

According to c) and d) we have that $|P| \geq L_T^{M,B}$.

Therefore $\frac{|P|}{2^{\alpha+\delta}} \geq \frac{L_T^{M,B}}{L_T} = p_T^{M,B} \geq$ (according to b)) $p_B^M \geq \frac{|M|}{2^{n+\delta+1}} \geq \frac{2^s(n)}{|P|.2^{n+\delta+1}}$.

Hence $|P| \geq (2^{\alpha}.2^{s(n)-n-1})^{\frac{1}{2}}$.

□

The theorem is proven, it remains to prove Lemma 2.

Proof of Lemma 2.

The main point of the construction of the desired tree T is a recursive application of the next procedure *Proc*.

Proc starts with v, M as inputs for its first application. The both subtrees of B rooted by the immediate successors v_0, v_1 of v are taken into account. If one of these subtrees is reached by no input from M we add it to the constructed tree T'. The depth of the added subtree is at least $\alpha + \delta$ - this follows from the construction of B.

In the other case the branches (paths in P) followed by inputs from M starting in v_0 and that ones followed by inputs from M starting in v_1 don't meet in P till the depth U. They may meet only after diverging U different x's by tests in U different classes of C. This follows from the fact that P is a reasonable b. p. and that each $a \in M$ has at least U pebbles at v. It will be used in verifying c) of Lemma later.

For $i = 0, 1$ in v_i and below v_i we distribute each input a from M to the node of B where a has the next node of its U-subsequence S_a. According to the rules specified later, as an immediate successor of v in T' we take one such node w_0 below v_0 (or equal v_0) with maximum M-ratio of a special tree rooted in w_0. This node w_0 and the set of inputs from M distributed in it will be the input for the next application of *Proc*. Similarly below v_1.

Proc will be used α-times along any branch in question, and consequently T' will be constructed till to depth α.

What concerns the choice of an appropriate node below v_0 (resp.v_1), first let us take into account the easy case when on each branch there is at most one node of this distribution (= a node where some $a \in M$ are distributed). Among the full subtrees with roots in the nodes of the distribution we choose that one which has maximal (M, B)-ratio. Below in our proof we will demonstrate that (M, B)-ratio of (the tree rooted in) v_0 is not larger than the ratio of (the tree rooted in) the chosen node. This will be important in verifying b) of Lemma.

The difficult case is when some nodes of this distribution are on the same branch of B. We will still construct trees rooted in the nodes of our distribution with the property that their sets of leaves are pairwise disjoint. This property will imply in the proof below that the maximal (M, B)-ratio of these trees is at

least equal or larger than the (M, B)-ratio of the tree rooted in the corresponding v_i.

Let leaders be nodes (of this distribution) which have no such nodes as predecessors. (Each leader is the first on its branch.) We take into account a partition of the set of nodes of the distribution according to the equivalence "to be below (or equal to) the same leader".

Let $\{w_i\}$ be a class of nodes (a class of the partition) where sets of $a \in M$ $\{M_i\}$ are distributed.

We construct the corresponding trees R_i rooted in w_i. Each R_i contains all branches followed by inputs from M_i. The sets of leaves of R_i's are pairwise disjoint (since in each leaf there is at most one input - from the construction of B). In general the union of R_i's do not cover the whole subtree rooted in the leader of w_i's because in general there are branches not followed by any input from M. To each R_i we potentially add some other branches to save the possibility to continue the construction of the desired full tree T of depth $\alpha + \delta$ in case when the modified R_i is chosen as the tree with maximal (M, B)-ratio.

We potentially modify R_i's according to the following rules:

Let us take R_{i_0} one of R_i's. In B let us follow its branches from its root to its leaves. Let u be a node on some branch b such that only one outgoing edge is followed by inputs from M_{i_0}. In case when u is in B in depth at least $d = U + \delta$ we add the whole subtree of B rooted by the out-going edge in question to R or we do nothing if u plays its role for another R_i which consumes this subtree in question.
In case when u is in the depth at most d the most difficult case is such that the out-going edge in question is followed by inputs from some other M_i's.
We saturate the need of subtree in the direction of the out-going edge in question using the added subtree rooted in the middle of this edge.

Since for each $a \in M$ the U-subsequence S_a is of the S-length at most U each added subtree of depth $log\ (U) + \alpha + \delta$ is large enough to yield U possibly needed full subtrees of desired depth $\alpha + \delta$ (cf. the definition of B). We take the root of the tree with maximum M-ratio as the immediate successor of v.

From T' we construct the desired full binary tree T of depth $\alpha + \delta$ as follows:

To obtain T we modify T' only below the depth (in T') α.

From each node of T' in depth α in T' we follow the branches in B (!). We distinguish three cases A), B), C).

A) If the node in question (on level α and below) of T' is not reached in B by any input from M we simply add a subtree of an appropriate depth to gain the desired depth $\alpha + \delta$ of T.

B) If the node in question has only one out-going edge followed by inputs from M we take its immediate successors in B' as its immediate successors in T. We prolong the dead edge by an subtree of appropriate depth to gain the desired depth $\alpha + \delta$ of T and we follow the edge with inputs from M.

C) If in a node both outgoing edges are followed by inputs from M we consider two subtrees rooted in this node. Each of these two subtrees contains the subtree rooted by one outgoing edge in question and the dummy tree rooted by the middle of the opposite out-going edge (cut in the appropriate depth to gain the desired depth $\alpha + \delta$ of T). We chose that one with maximal number of inputs from M.

Now, it remains to prove that T satisfies the conditions stated in Lemma.

T is rooted in v, of course, and each its branch is of length $\alpha + \delta$. This is ensured by the fact that the length of branches of B is sufficient for the operations by which T is constructed both in the case of branches of B', and in the case of full dummy subtrees added in the step of construction from B' to B, too. Moreover, T is a full binary tree which is ensured by the fact that in the construction of T each node of T defined by an application of $Proc$ is developed in both directions by the next application of $Proc$ or by pending some full subtree instead of one edge out-going the node in question.

Further, it is easy to see that T is a subtree of B simply since each node of T is a node of B (from the construction of T) (cf. condition a) of Lemma).

Now, it remains to verify the conditions b), c), and d) of Lemma.

Lemma 3. *b)* $p_T^{M,B} \geq p_B^M$.

Proof. For any node w of T let M_w be the set of $m \in M$ which in B reach w, T_w be the subtree of T rooted in w, and B_w be the subtree of B rooted in w.

We shall prove that for each node w of T the inequality $p_{T_w}^{M_w,B_w} \geq p_{B_w}^{M_w}$ holds. For v we have $M_v = M,\ B_v = B,\ T_v = T$, hence the statement of the lemma follows.

Lemma 4. *For each node* $w \in T$ *the inequality* $p_{T_w}^{M_w,B_w} \geq p_{B_w}^{M_w}$ *holds.*

Proof. The proof is by induction according to the level of w in T starting with leaves of T.

The first step of induction: Let w be a leaf of T.

Then T_w is a tree consisting of the unique node (the root = the leaf). There are two possibilities:

1) Some $m \in M$ reaches w (in B). Then $p_{T_w}^{M_w,B_w} = \frac{L_{T_w}^{M_w,B_w}}{L_{T_w}} = \frac{1}{1} = 1 \geq p_{B_w}^{M_w}$.

2) The node w of T is not reached by any $m \in M$ in B. Then $p_{T_w}^{M_w,B_w} \geq 0 = \frac{0}{L_{B_w}} = \frac{L_{B_w}^{M_w}}{L_{B_w}} = p_{B_w}^{M_w}$.

The second step of induction: Let w be a node of T and for each node w' of T on any level of T below level of w the inequality $p_{T_{w'}}^{M_{w'},B_{w'}} \geq p_{B_{w'}}^{M_{w'}}$ holds.

What concerns the level in T of w we shall distinguish two possibilities:

I. The level of w in T is larger or equal to α.

We have three cases A), B), C) according to the three cases A), B), C) in the construction of T between depths α and $\alpha + \delta$:

A) w is not reached by any $m \in M$.
In this case $p_{T_w}^{M_w,B_w} = p_{B_w}^{M_w} = 0$

B) Exactly one edge outgoing w is followed by some $m \in M$.
Let w_0, w_1 be the immediate successors of w in T, w_1 be on the dead edge. Then $L_{B_{w_1}}^{M_{w_1}} = 0$ and $w_1 \in B'$ according to the construction of T.

$p_{T_w}^{M_w,B_w} = \frac{L_{T_{w_0}}^{M_{w_0},B_{w_0}} + L_{T_{w_1}}^{M_{w_1},B_{w_1}}}{L_{T_{w_0}} + L_{T_{w_1}}} = \frac{L_{T_{w_0}}^{M_{w_0},B_{w_0}}}{2.L_{T_{w_0}}}$ (since w_0 and w_1 are on the same level of T)

$= \frac{1}{2}.p_{T_{w_0}}^{M_{w_0},B_{w_0}} \geq (by\ induction) \frac{1}{2}.p_{B_{w_0}}^{Mw_0} = \frac{L_{B_{w_0}}^{M_{w_0}}}{2.L_{B_0}} \geq \frac{L_{B_{w_0}}^{M_{w_0}} + L_{B_{w_1}}^{M_{w_1}}}{2.L_{B_0} + 2.L_o} = \frac{L_{B_w}^{M_w}}{L_{B_w}} = p_{B_w}^{M_w}$
where L_0 is the number of leaves of each dummy subtrees added in B in the middle of each edge outgoing w.

C) Both edges out-going w are followed by inputs from M.

Let w_0, w_1' be immediate successors of w in B'. Let the number of inputs from M reaching w_0 is the same or larger than this number in case of w_1'. Since w_0, w_1' are on the same level of B' we have $L_{Bw_0} = L_{Bw_1'} =_{df} b$ and consequently $L_{B_{w_0}}^{M_{w_0}} \geq L_{B_{w_1'}}^{M_{w_1'}}$.

Let w_1 be the second immediate successor of w in T. According to the construction of T w_1 is the root of the dummy tree pending in the middle of the edge (w, w_1') in B', $L_{T_{w_1}}^{M_{w_1},B_{w_1}} = 0$. Let L_0 be the number of its leaves.

$p_{T_w}^{M_w,B_w} = \frac{L_{T_w}^{M_w,B_w}}{L_{T_w}} = \frac{L_{T_{w0}}^{M_{w0},B_{w0}} + L_{T_{w1}}^{M_{w1},B_{w1}}}{b+L_0} \geq \frac{2.L_{T_{w0}}^{M_{w0},B_{w0}}}{2.b+2.L_0} \geq \frac{L_{B_{w0}}^{M_{w0}} + L_{B_{w1}'}^{M_{w1}'}}{2.b+2.L_0} = \frac{L_{B_w}^{M_w}}{L_{B_w}} = p_{B_w}^{M_w}$.

II. The level of w is less than α.

Let w_0, w_1 be the immediate successors of w in T while v_0, v_1 be the immediate successors of w in B.

From the induction assumption for $i = 0, 1$ for trees T_{w_i}, B_{w_i} rooted in w_i it follows that
$p_{Tw_i}^{M_{w_i},B_{w_i}} \geq p_{B_{w_i}}^{M_{w_i}}$ holds.

Lemma 5. $p_{Tw_i}^{M_{w_i},B_{w_i}} \geq p_{B_{v_i}}^{M_{v_i}}$

Proof. According to the construction of T below v_i the sets of leaves of the subtrees of B rooted in the nodes to which some classes of inputs from M are distributed, are pairwise disjoint. Since the set of leaves of B_{v_i} is the union of the sets of leaves of the trees in question and possibly some other subtrees below v_i with no inputs from M we see that $p_{B_{v_i}}^{M_{v_i}}$ is not larger than the maximum M-ratio of the trees in question. (From formal point of view the M-ratio is a fraction $\frac{a}{b}$. We use the fact that $\frac{a}{b} \leq \frac{c}{d}$ implies $\frac{a}{b} \leq \frac{a+c}{b+d} \leq \frac{c}{d}$.)

According to the construction of T, the tree rooted in w_i have the maximum M-ratio which on its turn is not larger than $p_{Tw_i}^{M_{w_i},B_{w_i}}$. □

Since w_0, w_1 are on the same level of T we have $L_{T_{w_0}} = L_{T_{w_1}} =_{df} b$ and consequently $L_{T_w} = 2b$.

Further since v_0, v_1 are on the same level of B' we have $L_{B_{v_0}} = L_{B_{v_1}} = d$ and consequently $L_{B_v} = 2d + 2L_0$ where L_0 is the number of leaves of each from both dummy trees rooted in the middle of edges outgoing v in B'.

Further for $i = 0, 1$ let us define $a_i =_{df} L_{T_{w_i}}^{M_{w_i},B_{w_i}}$ and $c_i =_{df} L_{v_i}^{M_{v_i},B_{v_i}}$.

According to the previous lemma for $i = 0, 1$ we have $\frac{a_i}{b} \geq \frac{c_i}{d}$. Hence $p_{T_v}^{M_v,B_v} = \frac{a_0+a_1}{2b} \geq \frac{c_0+c_1}{2d} \geq \frac{c_0+c_1}{2d+2L_0} = p_{B_v}^{M_v}$. □

The condition b) of Lemma is verified.

Lemma 6. *c) The nodes of T reached by inputs from M in depth α in T are pairwise different in P.*

Proof. Let v_0, v_1 be nodes of T in depth α in T.

Then for $i = 0, 1$ there are some $m_i \in M_{v_i}$.

Let v be the last common node for m_0, m_1 in T. According to the construction of T both m_0, m_1 are in M_v, i.e. v is a node of the special subsequences of both m_0, m_1.

In P for example m_0 has in v at least U pebbles.

Hence there are U C_2's such that the corresponding x_{C_2}'s are in v together with m_0, m_1. Each such x_{C_2} follows m_0 at the test in v and later x_{C_2} leaves m_0 by a test in C_2. m_1 may meet m_0 only after the moment when the last x_{C_2} leaves m_0.

v_0 is in depth at most α below v in T and therefore in P v_0 is in the depth at most U below v. But this is an unsufficient number for tests for leaving U x_{C_2}'s.

Hence in P, m_1 is not in v_0 , therefore v_0, v_1 are different in P.

□

Lemma 7. *d) The number of leaves of T reached (in B) by inputs from M is the same as the number of nodes of T reached (in B) by inputs from M on level α.*

Proof. In the second part of the construction of T starting at the level of leaves of T' in the depth α we prolong each node by two outgoing edges till the depth $\alpha + \delta$. The rule is such that exactly one edge followed by inputs from M outgoes each node reached by inputs from M. Cf. cases A), B), C) of the construction.

□

□

The key Lemma 2 is proven.

5 Multisyms

We start our searching for the first application of the lower bound theorem from the previous section. The candidate to be a hard Boolean function is the function of so-called multisyms below.

For appropriate n's we understand the binary inputs of length n as matrices $m \times k$ where $m.k = n$. We say that some t columns are covered by a row r if the bits of these t columns on row r have the same value 0 or 1 (they are monochromatic on r). We say that such a matrix is a t-multisym if each choice of t columns is covered by a row.

It is easy to see that for any constant t t-multisyms are in P and that for any unbounded and reasonably constructible function $t(n)$ $t(n)$-multisyms are in co-NP.

For the purposes of this text we shall use only 2-multisyms, simply multisyms.

This function was already used in [4] for testing lower bound techniques based on the notion of windows (cf. next section) for the case of read-once b.p.'s. A superpolynomial lower bound has been achieved.

We often use notation $m = \epsilon(n).log\ n$ and $k = \frac{n}{\epsilon(n).log\ n}$. It is easy to see that for $\epsilon(n) \geq 2$ the number of multisyms is at least 2^{n-1}. Indeed the number of non-multisyms is at most

$$\binom{k}{2}.2^m.2^{n-2m} \leq (\frac{n}{\epsilon(n).log\ n})^2.2^{n-m} \leq 2^{n+2.log\ (\frac{n}{\epsilon(n).log\ n})-\epsilon(n).log\ n} \leq 2^{n-1}.$$

By $\beta(n)$-strong multisyms we mean multisyms with at least $\beta(n).log\ n$ (covering) monochromatic rows for each pair of columns.

For $\beta(n) \leq \frac{\epsilon(n)}{2.log(\epsilon(n))+2.loglog\ n}$ the number of $\beta(n)$-strong multisyms is at least 2^{n-1}. Indeed, from the assumption it follows that

$\beta(n) \leq \frac{\epsilon(n).log\ n}{2.log(\epsilon(n).log\ n).log\ n}$ and further

$\beta(n) \leq \frac{\epsilon(n).log\ n-1-2.log\ n-log(\epsilon(n))-loglog\ n}{log(\epsilon(n).log\ n).log\ n}$. Hence

$\beta(n) \leq \frac{\frac{m-1-2.log\ n}{log\ m}-1}{log\ n}$ which implies

$2.log\ n + log\ m.(\beta(n).log\ n + 1) \leq m - 1$ and further

$2^{2.log\ n}.m^{\beta(n).log\ n+1} \leq 2^{m-1}$. Hence

$2^{n-1} \geq n^2.2^{n-m}.m^{\beta(n).log\ n+1} \geq \binom{k}{2}.2^{n-m}.m^{\beta(n).log\ n} \geq$

$\geq \binom{k}{2}.2^{n-2.m}.2^m.\sum_{i=0}^{\beta(n).log\ n-1}\binom{m}{i}$.

The last expression is an upper bound on the number of strings which are not $\beta(n)$-strong multisyms. Hence the number of $\beta(n)$-strong multisyms is at least 2^{n-1}.

By a canonical branching programs computing multisyms we mean any branching program P consisting from a chain of subprograms $(P_{i,j})$ for $i, j = 1...k,\ i \neq j$. Each program $P_{i,j}$ is responsible for verifying the covering of the pair of columns (C_i, C_j) by at least one row. Each $P_{i,j}$ has two sinks - simply to separate the matrices with covered (C_i, C_j) from the others. The first sink of $P_{i,j}$ ("the pair C_i, C_j is covered") is the source of the next subprogram $P_{i',j'}$, the second sink of $P_{i,j}$ ("the pair C_i, C_j is not covered") is one of sinks of P ("non-multisyms"). Each such $P_{i,j}$ is a chain of microprograms M_r for each row r. M_r is responsible for testing of covering of (C_i, C_j) by row r. (M_r tests equality of

two bits.)

Lemma 8. *Each canonical branching program is a $(C, s(n))$-branching program for $C = (C_1, ..., C_k)$ where C_i's are columns, and $s(n) = n - 1$.*

Proof. Let P be a canonical branching program computing multisyms. We want to verify that P satisfies the conditions of definition of (C, s)-branching programs for some C, s.

We see that the set $C = \{C_1, ..., C_k\}$ of all columns is in fact a partition of the set of input bits. Further we define $s(n) = n - 1$. Let A be the set of all multisyms. We know that (under trivial assumptions) $|A| \geq 2^{n-1}$.

Let a be a multisym, let C_1, C_2 be a pair of columns. Let us take the subprogram $P_{1,2}$ of P corresponding to the pair C_1, C_2 and within it let us take the first microprogram M_r (responsible for a row r) such that r covers C_1, C_2 in a. In M_r a leaves $P_{1,2}$. Let v be the input node of M_r. Moreover let us take a nonmultisym x not covering the pair (C_1, C_2) but covering all other pairs of columns. x must reach v, and let us suppose that in v x follows a. We see that conditions i), ... , v) of the definitions of (C, s)-branching programs are satisfied. □

6 Windows

In this section we introduce a complexity tool which is based on the intuitive idea of remembering and forgetting. We shall use it for proving our lower bound in the next section.

Definition 4. *Let P be a branching program, v be its node. Let A be a subset of the set of all inputs reaching v. From v we develop a tree $T_{v,A}$ according to P with respect to A. From the level of sinks we arbitrarily(!) test appropriate bits in such a way that in these tests both out-going edges are followed by inputs from A (till the moment when in each leaf of the resulting tree $T_{v,A}$ there is exactly one input from A).*

For each $a \in A$ we define the window $w(a, v, A)$ on a at v with respect to A in such a way that $w(a, v, A)_i = +$ if and only if in $T_{v,A}$ there is a test on bit i along the branch followed by a. (On the other -non-crossed- bits $w(a, v, A)$ equals a.)

The length of a window is the number of its non-crossed bits.

The window $w(a, v, B)$ is said to be an A-natural one iff B is the set of all inputs from A reaching v. For A the set of all inputs we use only "natural" instead of "A-natural".

Comments.

i) We see that the definition of windows is ambiguous due to the arbitrary last part of the construction of $T_{v,A}$ beginning at the level of sinks. We may do so since the next applications are based on the notion "the length of window".

ii) In the definition if we replace "node v" by "edge e" we obtain the window assigned to the edge e.

iii) For each a in a given set A comparing the window on a at v with the respect to A and the window on a at an out-going edge e leaving v with respect to the subset of A corresponding to e we see that the rule "one test, (exactly) one cross is removed" is satisfied.

iv) It is clear that the simple thing holds: "The larger A, the larger number of branches in the tree, the larger number of crosses, the shorter windows".

For the theory of windows the following theorems are important. Possible proofs can be found in [6].

Theorem 2. *Let us have r binary trees. Let l be the average length of their branches and S be the sum of (the numbers of) their leaves. Then $l \geq log_2\ S\ - log_2\ r$.*

Theorem 3. *Let P be a branching program and A be a set of inputs of length n distributed in (the set of nodes of) P. Let $A_1,\ ...\ A_r$ be all classes of this distribution. Then*

$log_2\ (size\ of\ P) \geq log_2\ r \geq log_2\ |A| - n + avelw$

where $avelw$ is the average length of windows of inputs from A according to A_i's, $i = 1,\ ...\ r$.

In the next section we shall use the following theorem.

Theorem 4. *Let P be a branching program, A be a set of inputs of length n and z be a number.*

Then there is a set $A_z \subseteq A$ of cardinality at least $(1 - \frac{1}{z}).|A|$ such that for each $a \in A_z$ all A-natural windows on a along $comp(a)$ are of length at most $z.(log_2\ (size\ of\ P) - log_2\ |A| + n)$.

Proof. We distribute each $a \in A$ to the node of its (first) A-natural window of maximal length. We see that for each $a \in A$ the window according to (the corresponding class of) this distribution is not shorter than the maximal A-natural window on a at that node in question and therefore it is not shorter than any A-natural window on a along whole $comp(a)$.

We see that at most $\frac{1}{z}.|A|$ inputs from A have windows according to our distribution of length at least $z.avelw$. Hence at least $(1-\frac{1}{z}).|A|$ of inputs from A have windows according to our distribution of length at most $z.avelw$. Hence - as stated above - $(1-\frac{1}{z}).|A|$ of inputs from A have each their A-natural windows (along the whole computations) of length at most $z.avelw \leq z.(log_2\ (size\ of\ P) - log_2\ |A| + n)$ (according to the previous theorem). Q.E.D.

Comment.

1. Informally, the theorem says that for any large A all a's from A but a fraction of them have short natural windows.

2. An open problem is whether the theorem holds with "A-natural windows" replaced by "A_z-natural windows".

7 The Application

The next theorem reduces the proof by contradiction of superpolynomial lower bound for read-once b.p.'s to the problem of superpolynomial lower bound for (C,s)-b.p.'s (cf. Theorem 1).

Theorem 5. *Let P be a read-once branching program computing multisyms with $\epsilon(n) = n^{\frac{2}{7}}$. Then $size(P) \geq 2^{n^{\frac{1}{8}}}$.*

Proof. By contradiction. Let us denote $x = 2^{n^{\frac{1}{8}}}$ and let $size(P) < x$. Let us suppose that $\beta(n) \leq \frac{\epsilon(n)}{2.log(\epsilon(n))+2.loglog\ n}$.

Under this assumption we shall prove that P is a (C,s)-branching program for the set of $\beta(n)$-multisyms, $s(n) = n-1$ and $C = \{C_1, ..., C_k\}$ where C_i's are all the columns of the input matrix. Consequently according to the Theorem 1 we shall obtain that the size of P is at least x which will give the desired contradiction.

According to Theorem 4 there are at least 2^{n-2} of $\beta(n)$-strong multisyms such that each their natural window in P is of length at most $2.(log\ x + 1)$. Let S be the set of such multisyms.

For any multisym a and for each pair of columns C_1, C_2 we define a pair of nodes $v_{a,C_1,C_2}, w_{a,C_1,C_2}$ *in* $comp(a)$ (in brief v, w) as follows: Let r be the first row monochromatic on C_1, C_2 such that both its bits (on C_1, C_2) are both tested along $comp(a)$. v_{a,C_1,C_2} is the node of the first (in $comp(a)$) test in question, w_{a,C_1,C_2} is the node of the second one.

Lemma 9. *Let $a \in S$, let C_1, C_2 be columns.*

Then there is no input c satisfying both next a), b).

a) c meets a at the moment when $comp(a)$ still has not covered C_1, C_2 and still has touched (by any test) at most $2.(log\ x\ +\ 1)$ pairs covering C_1, C_2 in a.

b) There is a bit $i, i \in C_1 \cup C_2, a(i) \neq c(i)$, $a(i)$ tested by $comp(a)$ before meeting $comp(a)$ and $comp(c)$.

Proof. By contradiction. Suppose such an c exists. We construct c' as a prolongation of c (it means we define values of the bits not tested by $comp(c)$ before meeting with $comp(a)$). Outside of C_1, C_2 we give the same values which are in a. Hence c' and a cannot diverge by tests outside of C_1, C_2.

Fact 1. On C_1, C_2 each bit tested by $comp(c)$ before the meeting point is also tested by $comp(a)$ before the meeting point.

By contradiction. Let i_1 be a bit on C_1, C_2 tested by $comp(c)$ but not tested by $comp(a)$ before the meeting point. Let i'_1 be a bit associated with it on the same row on C_1, C_2. Since P is a read-once b.p. i_1 must not be tested below the meeting point. We construct a', a'' two prolongations of a. Outside of C_1, C_2 they equal a. On C_1, C_2 on pairs of bits on the same rows with exception of i_1, i'_1 we give values $01, 10$ so that each pair C, C_1 and C, C_2 is covered (we have at our disposition at least $\beta(n).log\ n - log\ x$ free pairs of bits). In i_1 we give different values for a', a'', in i'_1 we give the same value for both a', a''. a', a'' differ only on i_1 - therefore they will not diverge because on i_1 they must not test since i_1 was tested by c. They reach the same sink. On the other hand -due to the arrangement on i_1, i'_1- one of them is a multisym and the other not. A contradiction.

On $6.log\ n$ pairs of bits on rows on C_1, C_2 we may give values $10, 01$ in such a way that both a and c' cover each C, C_1 and C, C_2 for each $C \neq C_1, C_2$.

Fact 2. c' covers all pairs C, C' for $C, C' \neq C_1, C_2$.

By contradiction. On C_1, C_2 on the remaining pairs of bits we give values 00. a, c' reach the same sink but only a is a multisym. A contradiction.

Fact 3. c covers C_1, C_2.

By contradiction. Let c not cover C_1, C_2. On the remaining bits of C_1, C_2 with exception of j associated with i we construct a common prolongation in such a way that the resulting c' does not cover C_1, C_2. To j we give the value $a(i)$. a', c' reach the same sink but only a' is a multisym. A contradiction.

We see that c' covers all pairs of columns. On the other hand on C_1, C_2 there is a common prolongation a' of a such that a', c' reach the same sink with

arrangement on bits i, j such that a' is not multisym. A contradiction. The proof of our lemma is closed.

□

Lemma 10. *The number of pairs covering C_1, C_2 (in a) touched by (tests of) comp(a) before w is at most $2.(log\ x + 1)$.*

Proof. By contradiction. Let this number be larger. Then many bits from the pairs in question are tested by *comp*(a) before w and therefore many bits are introducing natural windows on a before w. But $a \in S$, therefore each its natural window is of length at most $2.(log\ x + 1)$. Hence a c must meet a soon and close some bit in the window on a. But this is impossible according to the previous lemma. A contradiction.

□

Let us verify that P is a (C, s)-branching program. For $a \in S$ an $\beta(n)$-multisym and for any pair of C_1, C_2 we have defined the nodes $v_{a,C_1,C_2}, w_{a,C_1,C_2} \in$ *comp*(a) in brief v, w.

Let us define y as follows. Outside of C_1, C_2 $a = y$. On C_1, C_2 y equals a on all bits tested by *comp*(a) before w, on the bit tested at w y differs a and on the remaining bits y is defined arbitrarily in such a way that y is a nonmultisym.

Now it suffices to prove that any c diverging a in v never meets a before w nor in w. But this follows from the previous lemmas.

Hence P is a (C, s)-branching program. According to Theorem 1

the size of P is at least $2^{\frac{\alpha\ +s(n)-n-1}{2}}$ where $\alpha \geq (\frac{\binom{|C|}{2}}{T(n)} - U).(\frac{|C|-1}{U} - 1)^{-1}$ for any $U < \frac{\binom{|C|}{2}}{T(n)}$.

We have $s(n) = n - 2$, $T(n) = n$ and we take $U = \frac{\binom{|C|}{2}}{2.T(n)}$. We know that $|C| = k = \frac{n}{\epsilon(n).log\ n}$.

Then $\alpha \geq \frac{\frac{P}{T} - U}{\frac{Z}{U} - 1} = \frac{(\frac{P}{2T})^2}{Z - \frac{P}{2T}} = \frac{(\frac{k.(k-1)}{4n})^2}{k - 1 - \frac{k.(k-1)}{4n}} \geq \frac{k^2(k-1)}{16n^2} \geq \frac{k^3}{17n^2} = \frac{\frac{n^3}{(\epsilon(n).log\ n)^3}}{17.n^2} = \frac{n}{17(\epsilon(n).log\ n)^3}$.

Hence the size of P is at least $2^{\frac{n}{35(\epsilon(n).log\ n)^3}}$. A contradiction. The assumption $size(P) < x$ is false.

It remains to verify for which $\beta(n)$ the steps of the proof are valid. We need $(\beta(n).log\ n - log\ x \geq 6.log\ n$ in the proof of Fact 1 in the proof of Lemma 9 which is equivalent to the inequality $\beta(n) \geq \frac{n^{\frac{1}{8}}}{log\ n} + 6$.

This inequality is not contrary to the choice of $\beta(n)$ at the starting point of the proof.

Hence. we achieve a lower bound $2^{n^{\frac{1}{8}}}$.

□

References

1. M. Ajtai - A non-linear time lower bound for Boolean branching programs, Proc. of 40th IEEE Ann. Symp. on Foundations of Computer Science, 1999, pp. 60-70
2. P. Beame, M. Saks, X. Sun, and E. Vee - Super-linear time-space tradeoff lower bounds for randomized computation, Proceedings of the 41st Annual Symposium on Foundations of Computer Science, Redondo Beach, CA, November 2000, IEEE, New York, 169179.
3. A. Borodin, A. Razborov and R. Smolensky - On lower bounds for read-k-times branching programs, Computational Complexity 3, 1993, 1 - 18.
4. S. Jukna, S. Žák - On branching programs with bounded uncertainty, Proc. of ICALP'98, LNCS 1443,Springer, Berlin, 259-270.
5. S. Jukna, S. Žák - Some notes on the information flow in read-once branching programs, in Proc. of 27th Ann. Conf. on Current trends in Theory and Practice of Informatics, LNCS 1963, Springer, Berlin, 2000, pp. 356-364
6. S. Jukna, S. Žák - On uncertainty versus size in branching programs, Theoretical Computer Science 290 (2003), 1851-1867.
7. P. Savický and S. Žák - A hierarchy for (1,+k)-branching programs with respect to k, LNCS 1295, MFCS97, 478-487.
8. I. Wegener - Branching programs and binary decision diagrams, SIAM, 2000