A Subexponential Lower Bounds for Branching Programs Restricted with Regard to Some Semantic Aspects

Žák, Stanislav
1997

INSTITUTE OF COMPUTER SCIENCE

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

A subexponential lower bound for branching programs restricted with regard to some semantic aspects

Stanislav Žák

Technical report No. 728

October 15, 1997

Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Prague 8, Czech Republic
phone: (+4202) 66 05 36 90 fax: (+4202) 85 85 789
e-mail: stan@uivt.cas.cz

INSTITUTE OF COMPUTER SCIENCE

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

A subexponential lower bound for branching programs restricted with regard to some semantic aspects

Stanislav Žák[1]

Technical report No. 728
October 15, 1997

Abstract

Branching programs (b.p.s) or binary decision diagrams are a general graph-based model of sequential computation. The b.p.s of polynomial size are a nonuniform counterpart of LOG. Lower bounds of different kinds of restricted b.p.'s are intensively investigated. The restrictions based on the number of tests of variables during any computation (along any path in the case of syntactic b.p.s , resp.) are very important. Various superpolynomial or even exponential lower bounds are known for 1-branching programs, $(1,+k)$-branching programs, and syntactic k-branching programs.

In the paper we present a restriction of another type - the so called gentle branching programs (g-b.p.s) together with the following results.

1. For each 1-b.p. P there is a gentle 1-b.p. P' computing the same function such that $|P'| \leq 3|P|$.
2. Some Boolean functions which are superpolynomially (or even exponentially) hard for 1-b.p.s are polynomially easy for g-b.p.s.
3. The same holds for Boolean functions which are superpolynomially hard for $(1,+k)$-b.p.s or for syntactic k-b.p.s.
4. The functions whose sequences cause the hierarchies for $(1,+k)$-b.p.s and for syntactic k-b.p.s (both with respect to k) are polynomially easy for g-b.p.s.
5. We find a function in $\mathcal{P}$ which is superpolynomially hard for g-b.p.s. The proof is based on a lower bound theorem.

Both, the lower bound theorem and the definition of gentle branching programs are derived from a deeper consideration of the phenomenon of branching of computations.

Keywords

Branching programs, lower bounds

[1]This research was supported by the GA CR, Grant. No. 201/95/0976 "Hypercomplex" and partly by INCO-Copernicus Contract IP960195 ALTEC-KIT.

1 Introduction

1.1 Branching programs

A branching program (b.p.) is a computational model for representing Boolean functions. The input of a branching program is a vector consisting of the values of n Boolean variables. The branching program itself is a directed acyclic graph with one source. The out-degree of each node is at most 2. Every branching node , i.e. a node of out-degree 2, is labeled by an input variable (or equivalently: by an iput 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 variable assigned to the node is tested and the out-going edge labeled by the actual value of the variable is chosen. Finaly, a sink is reached. Its label determines the value of the function for the given input. By the size of a branching program P we mean the number $|P|$ of its nodes.

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

Some generalization of b.p.s have been studied. Let us mention nondeterministic b.p.s where each branching node may have more than two out-going edges. The polynomial sized nondeterministic b.p.s represent a nonuniform variant of $\mathcal{NLOG}$. Let us mention also the ω-b.p.s [13], and the probabilistic b.p.s [1].

In the present paper we use the classical deterministic branching programs. We work in one of the main streams of branching programs investigation which consists in proving lower bounds for restricted branching programs, it means for branching programs of some special properties.

An example of a restriction from the early beginning of investigation are branching programs of restricted width [3] or oblivious branching programs of limited depth. [11].

A special type of b.p.s are the so called ordered binary decision diagrams (OBDD) where along each computation the tests preserve a fixed ordering of variables and each variable is tested at most once. OBDD's play an important role as a tool for representing Boolean functions in many computer applications.

1.2 Restricted branching programs

The most commonly used restrictions are based on the number of tests during a computation or along a path in b.p.s in the question. So, the read-once branching programs (1-b.p.s), where during each computation each variable is tested at most once, are well-known. The lower bounds range from $2^{c\sqrt{n}}$[23][24][5] through $2^{n/logn}$ [8] to 2^{cn} [12][2][22] and $2^{n-o(n)}$ [19].There are lower bounds also in [17][1],[7].

There are some attempts to prove lower bounds for large classes of restricted branching programs. The real-time branching programs are allowed to do at most n tests during each computation. The related lower bounds are in [6],[25],[12].

The syntactic k-branching programs are allowed to have at most k tests on each variable along any (consistent or nonconsistent) path. The lower bounds are in [14],[4],[9] and the hierarchy with respect to k in [15],[16].

The $(1,+k)$-branching programs are allowed to test at most k variables repeatedly during each computation. The lower bounds are in [26][18][10] and the hierarchy in [20]. (The hierarchy for the syntactic case is proven in [21].)

We present another attempt in this direction with results as stated in the Abstract. However, the problem of the superpolynomial lower bound for 2-b.p.s (from 1983) remains open till now.

1.3 The basic ideas and the results

Our results are derived from two informal ideas.

Let us introduce the first one. For the branching program in question given any input we want to catch what information about the contents of the input bits is remembered (or forgotten) at any moment of the computation. We represent such a knowledge by a word of the length n over the alphabet $\{0,1,+,\#\}$ where $+,\#$ stand for "unknown".

From some reasoning about TM's it follows that the definition of crosses $(+)$ has to satisfy the next two, very natural conditions:

a) Immediately before a test on a bit i, i is crossed.

b) Immediately after a test on a bit i, i is non-crossed.

From this requirement it follows that the knowledge depends not only on the node reached during the computation but that different inputs reaching the same node may have remembered different information. This notion of remembered and forgotten information formally defined in Section 3 enables us to formulate and to prove our two lower bound method theorems.

The second informal idea concerns a situation where two computations reach the

same node v of the b.p. and after v they never branch. In this case we see that the bits on which these inputs differ have no influence on the common part of their computations and hence no influence on the reached sink. At v these computations must have achieved a partial result. In Section 3 we try to catch this strange phenomenon by defining when a bit of an input has #. This allows us to define the distribution of inputs over any branching program which is a key-notion of the definition of gentle branching programs in Section 4. There we also prove the relation of 1-b.p.'s and gentle b.p.'s as indicated in the Abstract.

In Section 5 we prove a subexponential lower bound $2^{n^{1/8}}$ on the size of gentle branching programs for a Boolean function which is computable within polynomial time on Turing machines. The proof is based on the theorems from Section 3 mentioned above.

In Section 6 we prove an $O(n)$ upper bound (on gentle branching programs) for the Boolean function f which is defined as follows: $f(A\vec{x}) = 1 \; iff \; A\vec{x} = \vec{0}$ (the input bits are arranged into a matrix and a vector). From [9][18] it is possible to derive that f is subexponentially hard for syntactic k- and for $(1,+k)$-branching programs. Moreover we prove an $O(nlog_2 n)$ upper bound for functions which are used in the Okolnishnikovova proof of the hierarchy of syntactic k-branching programs [15],[16]. Further, we prove a polynomial upper bound $O(n^2)$ for multipointer functions which are used in [20] for the proof of the hierarchy of $(1,+k)$-branching programs.

In Section 7 we prove an $O(n^{3/2})$ upper bound for the parity of the number of triangles in a graph (this function requires $2^{c(n)}$ on read-once b.p.s [2][22]).

Further, we prove a $O(n^2)$ upper bound for half-cliques-only function and a $O(n^3)$ upper bound for the Ablayev-Karpinski function [1]. The last upper bound is proven for the multiplication. Its size is $O(n^3)$ (cf. [17]).

2 Preliminaries

First we introduce some technicalities concerning words. For a binary word (a binary string) $m \in \{0,1\}^n$ by m_i we mean its i-th symbol. If $A \subseteq \{1,2,...,n\}$, by mA we mean the assignment of bits from A consistent with m. If R is a predicate on $\{1,2,...,n\}$ by m_R we mean mA where $A = \{j \mid R(j)\}$. E.g. $m_{\neq i}, m_{<i}$ where $1 \leq i \leq n$. If $A, B \subseteq \{1,...,n\}$, $A \cap B = \emptyset$ and a (b,resp.) is an assignment of A (B,resp.) then by $[ab]$ we mean the assignment of $A \cup B$ such that $[ab]A = a$, $[ab]B = b$. E.g. $[m_{\leq i} m_{>i}] = m$. For a word m by $|m|_1$ we mean the number of its ones; similarly $|m|_0$.

Secondly we introduce some notions concerning branching programs. Let P be a branching program, $m \in \{0,1\}^n$ be an input word. By $comp(m)$ (the computation on m) we understand the sequence of nodes of P which starts in the source of P and ends in a sink such that in each branching node v with a label i $comp(m)$ chooses the successor v' such that the label of the edge (v,v') is equal to m_i. If a node $v \in comp(m)$ we

say that $comp(m)$ goes through v or simply that m goes through v or that m reaches v. Similarly we say that m goes through an edge.

If v is a node of a branching program P then by P_v we mean the branching program with the source in v. If $M \subseteq \{0,1\}^n$ then by PM we mean the branching program which is given by deleting of all nodes and edges (from P) for which no $m \in M$ goes through.

Further, let us remember the operation of development of a branching program P (from its source) into a tree. By a development of P from its node v we understand the development of P_v.

By a subcube we mean the set M of inputs such that there is I, $I \subseteq \{1,...,n\}$ and $M = \{[uv]|v \in \{0,1\}^I\}$ where u is an assignment of $\{1,...,n\} - I$. The bits from I are called free bits, the other are called fixed bits. For a sink v of P M is called an original subcube if all inputs from M reach v and the free bits of M are never tested. For a node $v \in P$ by an agregate computation we mean the set of computations (the set of inputs) which follow the same branch of P_v till its sink.

3 Windows

By a *window* we mean a string $w = w_1...w_n \in \{0,1,+,\#\}^n$. By the length lw of such a window w we understand the number of i's such that $w_i \in \{0,1\}$ (the number of noncrossed bits).

We say that a window w is a window over a word $u \in \{0,1\}^n$ iff for each its noncrossed bit i $w_i = u_i$ holds.

Definition 3.1 *Let P be a branching program (over n variables). Let v be a node of P and M be a subset of the set of all inputs which (starting in the source of P) reach v. Let $m \in M$.*

Then we define the window $w(m,v,M,P)$ over m in the node v with respect to M as follows : (Starting at v) we develop the program P_v into a tree. In this tree we perform all computations starting from v which are given by the inputs from M. We omit all the edges and the nodes of the tree which are not reached by any of these computations. We omit all non-branching nodes. Now let us consider the branch b which is followed by the computation $comp(m)$.

1. We assign crosses (+) to all bits of $w(m,v,M,P)$ which are labels of nodes of the branch b.

2. Now we consider the set L of all inputs (from M) which follow b till its leaf. We assign the cross (+) to each bit which is not tested during any computation (from the source to the sink) on any word from L.

3. Further, we assign a double-cross # to each remaining bit i of $w(m,v,M,P)$ for which in L there is an input $m' \in M$ such that $m_i \neq m'_i$.

4. The remaining bits of $w(m,v,M,P)$ have the same contents as in m.

Comment. a) We see that all inputs following the same branch b have the same set of crossed (+) bits, of double-crossed (#) bits and that all non-crossed bits have the same contents.

b) The larger M the lesser number of non-crossed bits.

Before stating the main theorems we will prove some facts to become familiar with the definition.

For a moment due to some technical reasons we shall use a slightly modified branching programs such that on each path for each node with in-degree > 1 or out-degree > 1 both its immediate predecessors and successors have in-degree = out-degree = 1.

Let us fix a set $M \subseteq \{0,1\}^n$, let P be a branching program such that $P = PM$. For each node v of P we put $M_v =_{df} \{m \in M | m \ reaches \ v\}$. In the sequel (before the main theorems) at each node $v \in P$ we will consider the windows with respect to M_v only.

Proposition 3.2 *Let v be a branching node of P with a test on the bit i. Let v_1 be one of two nodes which immediately succeedes v. Then for any $m \in M$ going through v, v_1 the following holds: The windows for m at v and at v_1 differ only in the bit i. At v i is crossed, at v_1 i is non-crossed.*

Proof: Let $m \in M$. Let b be the branch of the tree developped at v which is followed by m. Since $P = PM$ at v the tree has a branching node labeled by i. Hence i is crossed. At v_1 all inputs from M_{v_1} have the same value of i, hence i is non-crossed. The branch b_1 of the tree developped at v_1 (induced by m) has the same branching nodes as b has (with the exception of v with the label i) and its leaf has the same set of inputs as the leaf of b has. Hence the set of crossed and double-crossed bits remains the same (with the exception of i). □

Comment. We can follow a computation on any input m from the source to a sink and consider the changes of the window on m during the computation. The proposition says that a test on a bit implies that exactly one crossed bit becomes non-crossed, which is in a good correspondence with the informal idea that a test is an acquisition of exactly one bit of information.

By a symmetric word we mean any word of the form uv where $v = u^R$. By a pair of symmetric positions we mean any pair (i,j) , $1 \leq i,j \leq n$, where $i = n - j + 1$.

Theorem 3.3 *Let $M = \{0,1\}^n$, let P be a branching program recognizing symmetric words of the length n. Then for each symmetric word m and for each pair (i,j) of symmetric positions during comp(m) there is at least one node v at which, in the window for m (with the respect to M_v), both i,j are non-crossed.*

Comment. The theorem corresponds to the informal idea that for comparing two bits i,j it is necessary that both i,j are known (non-crossed) at the same moment.

Proof:

By contradiction. Let m be a symmetric word and i, j be a pair of symmetric positions such that during $comp(m)$ they are never both non-crossed at the same moment (at the same node). We see that for both i, j there are some nodes at which they are non-crossed since both must be tested during $comp(m)$. Let us assume that the last window on i precedes the last window on j. Let v be the node from $comp(m)$ such that immediately before v i is non-crossed and this is the last moment when i is non-crossed during $comp(m)$. Since at v i is not non-crossed there is an input m' such that $m_i \neq m'_i$ and such that a) m, m' branches after v on i or b) m, m' follow a common path till a common sink. According to a) there is a window on m with the non-crossed i after v. It is impossible. It remains only the case b). From b) it follows that m' is also a symmetric word. After v j becomes non-crossed. Hence after v there is a test on j (according to the previous proposition). Since from v m, m' follow the same path m, m' have the same value on j. For our symmetric words m, m' we have $m'_i \neq m_i = m_j = m'_j$. A contradiction. □

Now we introduce two main lower bound method theorems.

Theorem 3.4 *Let P be a branching program, let v be one of its nodes, let M be a subset of the set of all inputs which (starting at the source) reach v. Then the average number of crosses and double-crosses in $w(m, v, M, P)$ for $m \in M$ is at least* $\log_2 |M|$.

More formally: $\sum_{m \in M}(n - lw(m, v, M, P)/|M| \geq log_2|M|$.

Proof: Let us develop the program $P_v M$ into a tree T_1. Let d_1 be the average number of crosses (+) and double-crosses (#) over all $m \in M$.

We transform T_1 into a tree T_2 which in each of its leaves has only one input from M. In each leaf of T_1 which is reached by two or more inputs from M we start a new subtree such that on each its branch there is a test on each bit which is crossed according to rule 2 of the definition of $w(., v, M, P)$ and then on each branch there are some tests on bits which are double-crossed in $w(., v, M, P)$ until in each leaf of the transformed tree T_2 there is only one input from M.

If we apply the rules 1 - 3 from the definition of windows on T_2, we see that for each $m \in M$, for each bit i , if i is crossed according to T_2 then i was a crossed or double-crossed bit according to T_1. Since T_2 does not give double-crosses we see that for the average number d_2 of crosses given by T_2 the inequality $d_1 \geq d_2$ holds.

Further, we shall transform T_2 into a balanced tree T_3. (By a balanced tree we mean a tree where for each pair of its branches the difference of their lengths is at most one.) We shall also transform the set M into another set M' of inputs such that $|M'| = |M|$. Let d_3 be the average number of crosses given by T_3 to inputs from M'. We shall prove $d_2 \geq d_3 \geq \log_2 |M'| = \log_2 |M|$.

Let us describe one step of the transformation of T_2 into T_3. Suppose there are two branches b_1, b_2 of T_2 such that $|b_1| - |b_2| \geq 2$ where for any branch b $|b|$ denotes the

number of its tests (its length). Let m_1 (m_2, resp.) be the input which follows b_1 (b_2, resp.) till its leaf. We delete m_1 from M. The total number of crosses is decreased by $|b_1|+1$ ($|b_1|$ is the number of crosses on b_1 and 1 is given by the disappearing of the last cross on the input m which branches with m_1 in the last test of b_1). We add such an input m' to M that m' follows b_2 till its leaf where m' branches with m_2. By the last action the total number of crosses is increased by $|b_2|+2$ ($|b_2|+1$ is the number of crosses on m' and 1 is for the additional cross on m_2).

We repeat this procedure until the resulting tree is balanced. We see that $|M| = |M'|$ and $d_2 \geq d_3$.

Now it remains to prove the following proposition.

Proposition 3.5 *Let T_3 be a ballanced tree. Let M be the number of its leaves. Then the average length of branches of T_3 is at least* $\log_2(|M|)$.

Proof: Let k be the length of the shorter branches of T_3. Let m be the number of nodes of T_3 on the level k which are not leaves, $m \in < 0, 2^k - 1 >$.

We see that the number of branches of T_3 is $2^k + m$. Hence, the average length of branches of T_3 is $[(2^k+m).k + 2m]/(2^k+m)$. We have to verify $\log_2(2^k+m) \leq k + 2m/(2^k+m)$. It suffices to verify $2m/(2^k+m) \geq \log_2(1+m/2^k)$. For $m \in < 0, 2^k >$ we put $x = (1+m/2^k)$; hence $x \in < 1, 2 >$.

It suffices to verify $2x - 2 \geq x.\log_2 x$. It is the truth for $x = 1, x = 2$. Moreover for $x \in < 1, 2 > (x.\log_2 x)'' = (\log_2 e)/x > 0$. Hence, $x.\log_2 x$ is a convex function on $< 1, 2 >$. Therefore $2x - 2 \geq x.log_2 x$.

□

□

Theorem 3.6 *Let P be a branching program. Let $\{X_i\}_{i=1}^r$ be a system of sets of some inputs. To each X_i a node v_i of P is assigned such that all $x \in X_i$ go through v_i. Then*
$\log_2 r \geq \log_2 \sum_{i=1}^r |X_i| - n +$ *average length of* $w(x, v_i, X_i, P)$.
More formally: $\log_2 r \geq \log_2 \sum_{i=1}^r |X_i| - n + \sum_{i=1}^r \sum_{x \in X_i} lw(x, v_i, X_i, P) / \sum_{i=1}^r |X_i|$.

Proof:
For $x \in (0, \infty)$ we put $\phi(x) = x.\log_2 x$. We know that $\phi(x)$ is a convex function since $\phi''(x) = (\log_2 e)/x > 0$. According to Jensen's inequality we have $\phi(\sum_{i=1}^r |X_i|/r) \leq (\sum_{i=1}^r \phi(|X_i|))/r$.

Further $\sum_{i=1}^r |X_i|.log_2(\sum_{i=1}^r |X_i|/r) \leq \sum_{i=1}^r |X_i|.\log_2 |X_i|$ and $\log_2 \sum_{i=1}^r |X_i| - \log_2 r \leq (\sum_{i=1}^r |X_i|.\log_2 |X_i|)/\sum_{i=1}^r |X_i|$.

According to Theorem 3.4, the last expression is not greater than the average number of crosses and double-crosses in $w(.,v_i,X_i,P)$ which is equal to $n-average\ length\ of\ windows$. Hence

$\log_2 r \geq \log_2 \sum_{i=1}^{r} |X_i| - n + average\ length\ of\ windows.$

□

Comments. a) Under the assumption that $\{X_i\}_{i=1}^{r}$ cover all 2^n inputs the statement of the Theorem is abreviated as follows:

"$log_2 r \geq\ average\ length\ of\ windows$".

If there is a relation between r and $|P|$ we have a lower bound for $|P|$.

b) The Theorem corresponds to an informal idea that "If we want to have a possibitity to remember much information about many inputs then we need a large memory".

c) Both Theorems form a method for proving lower bounds on the size of general branching programs. We will apply them in Section 5 for our lower bound. There is an open question to find another nontrivial application.

4 Gentle branching programs

Let us introduce the key-notion for the definition of the gentle branching programs.

Definition 4.1 *By the distribution on a branching program we mean a partition of the set of all inputs $\{0,1\}^n$ together with a one-to-one assignment of all classes of the partition to some nodes of the branching program in question. The partition and the assignment are given by the following rule.*

Briefly (with a danger of confusion): We stop each input in the node where its window has a double-cross # for the first time (or in a sink).

Precisely: By Pr we mean the next procedure.

Input: $A \subseteq \{0,1\}^n$. For each computation $comp(m)$, $m \in A$, in each node v let us consider the window $w(m,v,M,P)$ where $M \subseteq A$ is the set of all inputs from A reaching v.

For each $m \in A$ we mark the first v in $comp(m)$ such that $w(m,v,M,P)$ has a bit with a double-cross #.

In the set S of marked nodes we say that $v \in S$ is a maximum one iff there is no $v_1 \in S$, $m_1 \in A$ such that v_1 is marked for m_1 and v_1 precedes v in $comp(m_1)$.

For each maximum node $v \in S$ we define $M_v =_{df} \{m \in A \mid v\ is\ marked\ for\ comp(m)\}$.

Output: $R_A =_{df} \{M_v | v\ is\ maximum\}$,

$A := A - \bigcup R_A$.

We perform Pr starting with $A = \{0,1\}^n$ until $R_A = \emptyset$.

The resulting partition is given as $R =_{df} R_{A_1} \cup ... R_{A_k} \cup \{M_{s_1}, ... M_{s_l}\}$ *where* R_{A_i} *means* R_A *for the* $i-th$ *cycle of* Pr*, and* M_{s_j} *is the set of inputs which reach the sink* s_j *(after the last cycle of* Pr*).*

It is easy to see that the assignment $v \rightarrow M_v \in R$ which we have constructed is one-to-one.

By a *proper* class of the distribution we mean a class from $R_{A_1} \cup ... \cup R_{A_k}$.

It is clear that each improper class M_{s_j} is an original subcube.

Definition 4.2 *For a distribution* $\mathcal{D}$*, by the factorization of* $\mathcal{D}$ *we mean a finer partition given by the factorization of each proper class of* $\mathcal{D}$ *according to the equivalence "to have the same set of (bits with) double-crosses #'s".*

More precisely: Let M *be a proper class of* $\mathcal{D}$*. Let* v *be the node such that* $M = M_v$ *in the* j*-th cycle of* Pr *(for some* j*). We say that* $m_1, m_2 \in M$ *are equivalent*

iff $\forall i (i = 1...n)(w(m_1, v, M, P)_i = \# \leftrightarrow w(m_2, v, M, P)_i = \#)$

where A_j *is the input for the* j*-th cycle of* Pr*.*

Let M, C be two sets of inputs, $C \subseteq M$, and let all inputs from M reach a node v. We say that C preserves the double-crosses of M if for each $u \in C$ and for each $i, 1 \leq i \leq n$ the next implication holds $w(u, v, M, P)_i = \# \Rightarrow w(u, v, C, P)_i = \#$.

It is easy to see that each class F of the factorization of any class M of the distribution preserves the double-crosses of M.

We say that a class of M of a distribution $\mathcal{D}$ is a *significant* one iff $|M| \geq 2^n/\alpha$. We say that an fd-class F subclass of a class M of a distribution is a *significant* one iff $|F| \geq |M|/\beta$. α, β will be quantified in the next chapter.

Definition 4.3 *Let* F *be an* fd*-class.*

By h_F *we mean the set of bits which are double-crossed; by* t_F *we mean the set of the other bits.*

$H_F =_{df} \{mh_F \mid m \in F\}$.

$T_F =_{df} \{mt_F \mid m \in F\}$.

Definition 4.4 *Let* F *be an* fd*-class at a node* v*. Let* D *be the set of bits which are double-crossed (for all* $f \in F$*). Let* S_F *be the tree induced by* F *at* v*. Let* $V_1, ..., V_t$ *be the agregate computations which coincide with branches of* S_F*. Let* $M_i^1, ..., M_i^{m_i}$ *be the maximum original subcubes which are formed by inputs following* V_i*. By* G_i^j *we mean the set of bits from* D *which are free for* M_i^j *and by* O_i^j *we mean the set of bits from* D *which are fixed for* M_i^j*.*

Definition 4.5 *Let* P *be a branching program. We say that* P *is a gentle branching program if the following holds:*

If there is a significant proper class in the distribution on P then there is another significant proper class (at a node v) containing a significant fd-class F fulfilling the following conditions (R1), (R2), (R3)

(R1) Let the sets W_1, W_2 of bits be defined as follows :
$W_1 =_{df} \{i|(\forall f_1, f_2 \in F)(w(f_1, v, F, P)_i = w(f_2, v, F, P)_i \in \{0,1\})\}$
$W_2 =_{df} \{i|(\exists f \in F)(w(f, v, F, P)_i \in \{0,1\})\}$.
The condition is $|W_2 - W_1| \leq \gamma$. (γ will be quantified in the next section.)

(R2) $\forall h \in H_F \; \forall t \in T_F \; f_P([h,t]) = f_Q(t)$ where $Q = P_v F$.

(R3) For all i, j O_i^j is the same set.

From the definition of windows we know that the condition (R3) implies for all i, j $G_i^j = \emptyset$.

Theorem 4.6 *For each 1-branching program P there is a gentle 1-branching program P' with $|P'| \leq 3|P|$ and $f_P = f_{P'}$.*

Proof: Let P be a 1-branching program.

Proposition 4.7 *Let C be a proper class of the distribution on P. Then C consists of a unique fd-subclass satisfying (R1),(R2) of the definition of gentle programs.*

Proof:
Let C be equal to the set $\{u^i \in \{0,1\}^n | i = 1, ..., s\}$. For each i , $i = 1, ..., s$, let R_i be the set of bits tested during $comp(u^i)$ before v is reached. Let $R =_{df} \bigcup_{i=1}^{s} R_i$.

Proposition 4.8 *For each bit i outside R and for each j, $j = 1, ..., s$, $w(u^j, v, C, P)_i = +$.*

Proof: Let us develop a tree T induced by C from the node v. During $comp(u^j)$ after reaching v the bit i is either read for the first time - therefore in T there is a branching node with the label i and we have the desired + on i, or the bit i is not tested at all - we have a + on i, too.
□

Let us remember the well-known fact that after v no u^j tests any bit from R.

Proposition 4.9 *For each $i \in R$ if i is a double-crossed bit for an $u^j \in C$ then i is a double-crossed bit for all $u^j \in C$.*

Proof: Let $i \in R$ be a double crossed bit for an $u \in C$. Then there is an $u' \in C$ such that u' follows u to its sink, u, u' differ on i and at least one from the inputs u, u' is tested on i before v.

Let us choose any $u^j \in C$. Let $n^j =_{df} u^j(\{1,2....,n\} - R)$. We see that $[uR, n^j]$ and $[u'R, n^j]$ have the same properties (mentioned above) as u, u'. Therefore they have a # on i. Moreover, they follow u^j from v to the sink in question. Therefore u^j has a # on i, too.

□

Corollary 4.10 *All $u^j \in C$ have the same region of the crossed (+), the double-crossed (#) and the non-crossed bits.*

We see that C is also an fd-class satisfying (R1),(R2).

□

Proposition 4.11 *For each 1-branching program P there is a branching program P' such that*

(1) $f_{P'} = f_P$,

(2) $|P'| \leq 3|P|$,

(3) P' is a 1-branching program ,

(4) Each proper class of the distribution on P' satisfies (R3) from the definition of the gentle branching programs.

Proof:

We will transform P into P' by a sequence of elementary steps. Each of them will consist in inserting one redundant test in the program. "Redundant test " means a test where both out-going edges go to the same node. It is clear that (1) $f_{P'} = f_P$ will be satisfied.

Let Q be a 1-branching program arising from P after some transformation steps. Let us describe the elementary step on Q.

For any input u by $comp^D(u)$ we mean the initial part of $comp(u)$ from the source to the node at which u is distributed. We say that an edge e of Q is supported by a path p iff $e \in p$. Let $D(Q)$ be the subgraph of Q given by edges which are supported by $comp^D(u)$, $u \in \{0,1\}^n$. $D(Q)$ is also an oriented acyclic graph with one source. Its special properties are :

a) Its sinks are given by the nodes of Q at which some inputs are distributed.

b) Except of sinks (and the source) all nodes of $D(Q)$ have in-degree 1 in $D(Q)$.

Let us consider the set of all nodes of Q at which a proper class not satisfying (R3) is distributed. Let v be a maximum node of this set (i.e. in P v has no predecessors from this set). Let F be the proper class distributed at v. Let $v_1, ..., v_l$ be the immediate predecessors of v in $D(Q)$. In $D(Q)$ for each $j, 1 \leq j \leq l$ there is a unique path p_j going from the source to v via v_j. Let M^j be the maximum original subcube going through p_j to v. Let G^j be the set of bits from D (see def. 4.4) which are free for M^j.

There is a j_0 such that $G^{j_0} \neq \emptyset$ since F does not satisfy (R3). Let $a \in G^{j_0}$.

On p_{j_0} let y be the first node such that the subprogram Q_y does not contain any test on a. Such a y exists because in Q_v there is no test on a since $a \in D$. y is not the source because in Q a test on a must exist since $a \in D$. Let x be the immediate predecessors of y on p_{j_0}. In Q_x there is a test on a therefore no path from the source to x contains a test on a. Now it is clear that if we insert a redundant test (= two nodes) on a between x and y the resulting program Q' remains to be 1-branching program. At this moment we see that the point (3) of the statement of the Theorem is also satisfied.

Let us consider the distribution on Q'. In comparison with the situation on Q there are changes concerning F. In $D(Q')$ the path p_{j_0} from $D(Q)$ is interrupted at the second node of the test inserted between x and y where a new class F' of the distribution (on Q') arises. It is clear that F' satisfies (R3) since inputs from F' have double-crosses (#) only on a.

Slightly more complicated situation is at the node v. If $l \geq 3$ than at v there is some class of distribution with $l-1$ in-going original subcubes M^j.

In the case $l = 2$ one path p_j with only one M^j remains alone. p_j expands to a tree whose branches end by meeting one another or by meeting another path in $D(Q')$ or by reaching a sink.

We see that $D(P')$ without the nodes with in-degree > 1 forms a tree. The first nodes of inserted redundant tests are placed only in the leaves of this tree. The remaining nodes are from P. Hence, the number of the inserted first nodes is at most twice the maximum number of leaves of any tree in P which is equal to twice $(|P|+1)/2$. Since at least one leaf of our tree is from P we may estimate the number of inserted first nodes by $|P|$. Hence, $|P'| \leq 3|P|$ (the point 2 of the statement of the Theorem).
□

From Proposition 4.7 it follows that P' is a gentle 1-branching program.

The proof of Theorem is completed.
□

5 The lower bound

For the purposes of the present paper we put $\alpha = \beta = 2^{2n^{1/8}}$ and $\gamma = n^{1/8}$.

We shall define a Boolean function J for which we shall prove our lower bound for gentle b.p.'s.

Let us assume that n input bits are organized into a binary $(0,1)$ $(k\sqrt{n} \times \sqrt{n}/k)$-matrix A where $k \in N, k \geq 4$ and moreover $\sqrt{n}/k$ is an odd number. (Such a matrix A will be called "a matrix for J".) The columns of A are indexed by numbers $i \in < -(\sqrt{n}/k-1)/2, (\sqrt{n}/k-1)/2 >$. The column C_0 will be called the starting column.

In the following definition for two vectors (columns) M, C of the same dimension by $M \oplus C$ ($M \wedge C$, resp.) we shall mean the column which is the componentwise sum modulo 2 (componentwise conjunction, resp.) of M and C. For a column M by $|M|_1$ we shall mean the number of 1's in M. Similarly $|M|_0$.

On natural numbers larger than $(\sqrt{n}/k-1)/2$ for $q = 5n^{1/8}$ we define some intervals $N_1, ..., N_q$ and a binary labeling of numbers from these intervals.

For i, $1 \leq i \leq q$, the labeling on N_i forms the word $(0^i 1^i)^{12\gamma}$. We see that each two positions in N_i which differ by about i have different labels and that the summary length of $N_1, ..., N_q$ is at most $24\gamma q^2$. (The area of intervals N_i is adjacent to the matrix for J.)

Further the interval $< k\sqrt{n}/4 - (\sqrt{n}/k-1)/2; k\sqrt{n}/4 + (\sqrt{n}/k-1)/2 >$ will be called the marked area MA. We see that the inequality

$24\gamma q^2 \leq k\sqrt{n}/4 - \sqrt{n}/k + 1$

holds. It gives more than a sufficient space for $\bigcup_{i=1}^{q} N_i$ between the matrix for J and MA.

Outside of $\bigcup_{i=1}^{q} N_i \cup MA$ the odd positive numbers greater than $(\sqrt{n}/k-1)/2$ will be labelled by 1, the even ones by 0. The negative numbers will be labelled in the symmetric way with only the exception that bits symmetric to those from MA will be labelled by zeroe's.

Definition 5.1 *Let A be a matrix for J. Let s be an index of a column of A, M be a column of dimension $k\sqrt{n}$ and $\Box \in \{\oplus, \wedge\}$.*

We define $Jump(M, s, \Box) = (M', s', \Box')$ as follows :

$M' =_{df} M \Box C_s$,

Let $p =_{df} (|M'|_1 - |M'|_0)/2$. If $s + p \in < -(\sqrt{n}/k-1)/2; (\sqrt{n}/k-1)/2 >$ then $s' = s + p$ and $\Box' = \oplus$.

If $s + p$ is in MA, then $s' = s + p - k\sqrt{n}/4$ and $\Box' = \wedge$.

If $s+p$ is outside of the matrix for J and not in MA then $s' = s+p$ and $\Box'$ remains undefined.

In the sequel M will be called the "memory column" incoming to C_s and M' will be called the "memory column" out-going from C_s.

We see that if s' is also an index of a column of A we can iterate $Jump$ on $(M', s', \Box')$.

Definition 5.2 *Let A be a matrix for J. The value $J(A)$ is given as follows :*

Starting with the values $M = 0^{k\sqrt{n}}$, $s = 0$ and $\Box = \oplus$ we iterate Jump until the number of iterations is n or the last s' is outside of $A(\cup MA)$. In the case of n iterations we define $J(A) = 0$. In the other case $J(A)$ is defined as the label of s'.

It is easy to see that J is computable on Turing machines within a polynomial time ; $J \in \mathcal{P}$.

Theorem 5.3 *Each gentle branching program computing the function J is of a size greater than $c(n) = 2^{n^{1/8}}$.*

Proof: By contradiction. Let P be a gentle program computing J, the size of P is $c(n)$. Let us perform the distribution on P and its factorization.

Since the number of sinks is at most $c(n)$ there are at most $c(n)$ improper classes of our distribution. Each improper class consists of only one original subcube otherwise there would be double-crossed bits and the class would be proper. Each original subcube S in question has at most $(\sqrt{n}/k - 1)/2 + 1$ free bits on C_0 since in the other case there would be a possibility to reach both output values after the first jump which is in contradiction with the fact that all inputs from S reach the same sink. Hence, S has at least $b =_{df} k\sqrt{n} - (\sqrt{n}/k - 1)/2 - 1$ fixed bits.

Therefore, the improper classes cover at most $c(n)2^{n-b}$ inputs. We see that there is a proper class of the distribution of the cardinality at least
$(2^n - c(n).2^{n-b})/c(n) \geq 2^n/\alpha$.
Such a proper class is a significant one. Since P is a gentle branching program at a node v, there is a significant proper class with a significant fd-subclass F fulfilling the conditions (R1),(R2),(R3) of the definition of gentle branching programs. The cardinality of F is at least $f \geq 2^n/(\alpha.\beta)$.

Let $D, S_F, V_i, M_j^i, m_i, O_j^i$ be as in the Definition 4.4. We will prove that $D = \emptyset$; this will be a contradiction.

For each i all inputs from V_i have the same region O_i of non-crossed bits and also the same region R_i of bits with crosses +'s.

Let $o_i =_{df} |O_i|$, $d =_{df} |D|$, $r_i =_{df} |R_i|$. For s an index of a column of the matrix for J we define $O_{s,i} =_{df} O_i \cap C_s$, $D_s =_{df} D \cap C_s$, $R_{s,i} =_{df} R_i \cap C_s$ and further $o_{s,i} =_{df} |O_{s,i}|$, $d_s =_{df} |D_s|$, and $r_{s,i} =_{df} |R_{s,i}|$.

We asume w.l.o.g. that each node preceding v in P has out-degree = 1.

Proposition 5.4 *Let $a \in O_i^j$, $i \in <1, ..., t>, j \in <1, ..., m_i>$.*
Then immediately before v for all inputs from M_i^j the bit a is a non-crossed one.
More formally: Let v_l be such a node immediately preceding v that all inputs from M_i^j go through v_l. Let X_l be the set of all inputs from F which reach v_l. Then for all $m \in M_i^j$ $(w(m, v_l, X_l, P))_a \in \{0, 1\}$.

Proof: By contradiction. There is an input from M_i^j which has a) a cross + or b) a double-cross # on a immediately before v.

Case a) On a a cross may be placed if a test on a is expected - but this is impossible since $a \in D$, or if during the computation from the source to the sink a is not tested - but this is in contradiction with the fact that the original subcube M_i^j is a maximum one.

Case b) If an $m \in M_i^j$ would have a double-cross # on a before v, then m is distributed to a node before v.

□

Let o_i^j be the cardinality of O_i^j. $O_{s,i}^j =_{df} O_i^j \cap C_s$, $o_{s,i}^j$ its cardinality.

Proposition 5.5 $\sum_{i,j} |M_i^j|.(o_i^j + o_i)/f \leq \log_2 c(n) + n - \log_2 f$.

Proof: Let us consider our f inputs from F. Immediately before v they reach nodes $v_1, ..., v_z$ (each with out-degree = 1). These nodes define a partition of F into classes $X_1, ..., X_z$.

According to the Theorem 3.6 we have

$(\log_2(c(n)) \geq) \log_2 z \geq \log_2 f - n + \sum_{l=1}^{z} (\sum_{x \in X_l} lw(x, v_l, X_l, P))/f$.

Therefore

$\log_2 c(n) + n - \log_2 f \geq \sum_{l=1}^{z} (\sum_{M_i^j \subseteq X_l} \sum_{x \in M_i^j} lw(x, v_l, X_l, P))/f$

$\geq \sum_{l=1}^{z} (\sum_{M_i^j \subseteq X_l} \sum_{x \in M_i^j} (o_i^j + o_i))/f$ according to the previous proposition

$= \sum_{l=1}^{z} \sum_{M_i^j \subseteq X_l} |M_i^j|.(o_i^j + o_i)/f$

$= \sum_{i,j} |M_i^j|.(o_i^j + o_i)/f$.

□

Corollary 5.6 $d \leq \log_2 c(n) + n - \log_2 f \leq 5n^{1/8}$.

Proof: (R3) implies that for all i, j $o_i^j = d$. □

Corollary 5.7 $|W_1| \leq log_2 c(n) + n - log_2 f \leq 5n^{1/8}$.

Proposition 5.8 *If* $D_0 = \emptyset$, *then* $\forall s \neq 0 \quad D_s = \emptyset$.

Proof: By contradiction. Let us suppose $\exists s \neq 0 \quad D_s \neq \emptyset$. We are able to choose h_1, h_2 two assignements of D_s given by H_F such that or i) $a =_{df} (|h_1|_1 - |h_1|_0)/2 - (|h_2|_1 - |h_2|_0)/2 > 0$ or ii) there is a bit j (a row j) in D_s on which h_1, h_2 differ.

Case i).

We know from (R1) that $|W_2| \leq \gamma + log_2 c(n) + n - log_2 f \leq 6n^{1/8}$ and from the Corollary 5.6 that $|D| \leq log_2 c(n) + n - log_2 f \leq 5n^{1/8}$. Hence, there is an interval I of columns without any bit from $W_2 \cup D$ containing $2(\gamma + log_2 c(n) + n - log_2 f) \leq 12n^{1/8}$ columns.

On S_F it is easy to see that for any specification of all bits outside of $W_2(\cup D)$ there is a number of branches of S_F which respect this specification. Some possible branches may differ on some bits from W_2. If we choose such a branch b it is clear that in any case t induced by b is in T_F and therefore $J([h_1,t]) = J([h_2,t])$ (according to the condition (R2) of the definition of gentle). We shall construct a specification which induces only such $t's \in T_F$ that on $[h_1,t][,[h_2,t]$ the jumps go from C_0 to I then to C_s then to the interval N_a and such that the positions reached by $[h_1,t],[h_2,t]$ differ by a. Hence, $J([h_1,t]) \neq J([h_2,t])$. This will be our contradiction.

Outside of I, C_0, C_s, we choose an arbitrary assignment, say all zeroes.

For a set X of input bits by $R(X)$ we mean the set of rows containing at least one bit from X. For a set R of rows and a set C of columns by $R\diamond C$ we mean the set of input bits which are in a row from R and in a column of C.

We choose a set R_1 of rows such that $R_1 \cap (R(W_2) \cup R(D)) = \emptyset$ and $|R_1| = |I|$. We divide the rows except of $R(W_2 \cup D)$ and R_1 into two sets R_2, R_3 such that $(|R_2| + |R_1| - |R_3| - |R(W_2 \cup D)|)/2 = p_1 + k\sqrt{n}/4$ where p_1 is the leftmost position of I. We give one's into $R_2\diamond C_0$,$R_1\diamond C_0$, zeroe's into $R_3\diamond C_0$, $R(W_2 \cup D)\diamond C_0 - W_2$. For any t induced by this specification it is clear that the first jump is to I (via MA) and that the next memory will be created by $\wedge$.

On I first we give zeroes to $I\diamond(R(W_2 \cup D))$. We give one's to $C_{p_1}\diamond R_1$, to $C_{p_1+j}\diamond R_1$ we give $|R_1| - j$ one's and j zeroes (we have defined the content of $I\diamond R_1$).

Further, we choose a set R_4 of rows such that $R_4 \subseteq R_2$, $p_1 + (|R_1| + |R_2 - R_4| - |R_3| - |R_4| - |R(W_2 \cup D)|)/2 = s$. It is clear that such R_4 exists since R_2 is sufficiently large. We give ones to $(R_2 - R_4)\diamond I$ and we give zeroes to $R_4\diamond I$. It is clear that for any t induced by this specification of bits of I the next jump is to C_s. Explanation: Zeroe's in $I\diamond R(W_2)$ eliminate the incertainty about the contents of $C_0\diamond W_2$. The specification of $I\diamond R_1$ eliminate the difference of positions of columns in I. Moreover zeroe's in $I\diamond R(D)$ ensure that the memory column incoming to C_s has zeroes on $R(D_s)$.

In any case the memory column M incoming to C_s has zeroes on $R(W_2 \cup D), R_4, R_3$ and M has one's on $R_2 - R_4$. The values of M on R_1 depends on the case. The new out-going memory column M' is created by $\oplus$.

First we give some values to $C_s\diamond R_1$, for instance zeroes. For any specification of $C_s\diamond(R_2 \cup R_3)$ it is clear that for any $t \in T_F$ induced by any branch of S_F which respects this specification

a) There is an uncertainty concerning the jumped position of the size of at most $|R(W_2 \cup D) \cup R_1| < 4(\gamma + log_2 c(n) + n - log_2 f) \leq 24n^{1/8}$.

b) The positions for $[h_1,t],[h_2,t]$ differ about a since the incoming memory column M has zeroes on $R(D)$.

So, we specify $C_s \diamond (R_2 \cup R_3)$ in such a way that M' with supposed zeroes on $R(W_2 \cup D) \cup R_1$ would jump to the leftmost position of the interval N_a. In fact, in $R(W_2 \cup D) \cup R_1$ there are some ones in M', but the jumped position still remains in N_a. Hence $[h_1, t], [h_2, t]$ have different values - contradiction.

Case ii)

We proceed in a similar way as in case i) till to the jump on C_s. We choose another sufficiently large interval K of columns. From C_s we jump via MA to K. On K we arrange zeroes and one's in a similar way as in the previous case i) on I with a substantial change that on the row $j \in R(D_s)$ on which h_1, h_2 differ we don't give zeroes but ones. The effect of this change is that for any $t \in T_F$ induced by any branch of S_F which respect our specification the positions jumped by $[h_1, t], [h_2, t]$ will differ by one. Therefore it suffices to arrange K in such a way that in any case the next jump is to the interval N_1.

□

Proposition 5.9 $D_0 = \emptyset$

Proof: By contradiction. Let us suppose $D_0 \neq \emptyset$. We are able to choose h_1, h_2 two assignements of D_0 given by H_F such that i) $a =_{df} (|h_1|_1 - |h_1|_0)/2 - (|h_2|_1 - |h_2|_0)/2 > 0$ or ii) there is a bit j (a row j) in D_0 on which h_1, h_2 differ.

Case i).

Outside of W_2 and D we specify the bits of C_0 in such a way that with supposed zeroes in $W_2 \cap C_0$ and in D_0 the first jump would be to the leftmost position of the interval N_a. N_a is sufficiently large , so for any content of $W_2 \cap C_0$ and D_0 the first jump must be in N_a. For any $t \in T_F$ induced by any branch of S_F which respect our specification the positions jumped by $[h_1, t], [h_2, t]$ will differ about a. Hence, $J([h_1, t]) \neq J([h_2, t])$. A contradiction with (R2) of the definition of the gentle b.p.'s.

Case ii).

We proceed as in the case ii) of the previous proof. We find a sufficiently large interval K of columns (outside of $W_2 \cup D$). Then we specify the bits in such a way that the jumps go from C_0 to K via MA, from K to N_1.

□

From the last two propositions it follows that $D = \emptyset$. A contradiction. The theorem is completely proven.

□

6 Upper bounds

6.1 The function $f(A\vec{x})$

Let us suppose that our n input bits are organized into an $(l \times m)$-matrix A and into an $(m \times 1)$-vector $\vec{x}$. We define $f(A\vec{x}) = 1$ iff $A\vec{x} = \vec{0}$ (for all j, $1 \leq j \leq l$, $\sum_{i=1}^{m} a_{ji} x_i = 0$).

For appropriate choices of l, m f is superplynomially hard for $(1, +k)$-branching programs[18]. This follows also from [10]. From [9] it follows that f is superpolynomially hard also for nondeterministic syntactic k-branching programs. It is clear that f is computable within polynomial time, $f \in P$.

Theorem 6.1 *There is a gentle branching program P which computes f and which is of the size $O(n)$.*
Proof:

Let us describe a natural branching program P which computes f. P is a concatenation of branching programs $P_1 ..., P_l$ where for each $j = 1...l$ P_j is responsible for the multiplication of the j-th row A_j of A with the vector $\vec{x}$. Each P_j has two sinks; in one of them - with the label 0 - the computations on inputs for which $A_j x = 1$ stop, the other sink is sticked with the source of P_{j+1}. Hence, P has l sinks with the label 0 and only one with the label 1.

Now, let us describe P_j. P_j is a levelled branching program of the width 2. P_j has m levels each with two tracks. In the upper tracks of l_i - there are two nodes with label x_i (with the only exception of the upper track of l_1 where only one node exists - the source of P_j labeled by x_1). In the lower track of l_i there are two nodes labeled by a_{ji}. On the last l_{m+1} level there are only two sinks.

In the upper track of each level l_i, $m \geq i \geq 2$, one of the nodes with label x_i is called 0-node, the second one is called 1-node. In the upper track of the first level there is only the 0-node.

The 0-node in the upper track of the level i represents the inputs for which $\sum_{s=1}^{i-1} a_{js} x_s = 0$ (i.e. it is reached by computations on these inputs). A similar rule for 1-node.

For $a \in \{0, 1\}$ the 0-edge outgoing from the a-node on the level i reaches the a-node of the level $i + 1$. The 1-edge reaches one of the two nodes in the lower track of the level i labeled by a_{ji}. The edges out-going from each node in the lower track reach the 0-node or 1-node of the level $i + 1$. They are arranged in such a way that the rule concerning a-nodes is satisfied.

All P_j's (and therefore P too) are completely described. We see that P computes f and that P is of the size of at most $l|P_j| \leq l.6m \leq 6n$.

Now, we shall prove that P is a gentle branching program. Let us concentrate on P_1.

Definition 6.2 $n_0^1 =_{df} 2^n$, *for* $i = 2, ..., m$, $a \in \{0, 1\}$, n_a^i *is the number of inputs which reach the a-node of the level i.* n_0^{m+1} *is the number of inputs which leave P_1 by the sink sticked with the source of P_2,* n_1^{m+1} *for the other sink of P_1.*

Proposition 6.3 $n_0^2 = 3n_1^2$.

Proof: The 0-node of the level 2 is reached by exactly those inputs $A\vec{x}$ which have $x_1 = 0$ or $x_1 = 1$ and $a_{1,1} = 0$. □

Proposition 6.4 *For* $i = 2, ..., m$ *if* $n_0^i \geq n_1^i$, *then* $n_0^{i+1} \geq n_1^{i+1}$.

Proof:

$n_0^{i+1} = 3/4.n_0^i + 1/4.n_1^i \geq 3/4n_1^i + 1/4n_0^i = n_1^{i+1}$.

The equalities follow from the fact that 0-edges from the a-node of the level i go directly to the a-node of the level $i+1$ (for both $a = 0, 1$), and that 1-edge outgoing from the a-node of the level i goes to a node in the lower track of the level i where a branching on a_{1i} is performed. □

Corollary 6.5 $n_0^{m+1} \geq n_1^{m+1}$.

Let us perform the distribution on P. From our point of view the first three nodes with in-degree > 1 will be interesting - the 0-node of level 2 v_{02}, the 0-node v_{03} and the 1-node v_{13} of the level 3. Let M be the set of inputs such that $\sum_{i=1}^{m} a_{1i}x_i = 0$. We know that $|M| \geq 2^n/2$.

Proposition 6.6 *At* v_{02} *no* $m \in M$ *will be distributed there.*

Proof: By contradiction. Le us suppose that there is an $m \in M$ which is distributed at v_{02}. All m' which follow m from v_{02} to a sink are in M too.

For each $x_i, i = 1, ..., m$ in v_{02} and after it all m' following m go through at least one node with the test on x_i (x_1 is tested at the source of P_2). Hence m has no # on $x_1, ..., x_m$.

For each $a_{i,j}$ different from $a_{1,1}$ after v_{02} all m' following m simultaneously test or don't test $a_{i,j}$. Hence there is no # on $a_{i,j} \neq a_{1,1}$. The only candidate for # is $a_{1,1}$.

There are two possibilities .

a) $m_{x_1} = 0$

Then for all m' which follow m from v_{02} to a sink $m'_{x_1} = 0$ holds due to the test on x_1 at the source of P_2. Hence for all such m' $a_{1,1}$ is not tested at all, therefore there is + on $a_{1,1}$.

b) $m_{x_1} = 1$

Then for all m' which follow m from v_{02} to a sink $m'_{x_1} = 1$ holds (due to the test on x_1 at the source of P_2). Since they all have $x_1 = 1$ and they all reach v_{02}, they all must have $a_{1,1} = 0$. Hence, $a_{1,1}$ is noncrossed.

We see that there is no # on m at v_{02} with respect to all inputs reaching v_{02}. A contradiction.

□

Proposition 6.7 *At v_{02} there is a unique class of distribution containing all $m \in \{0,1\}^n - M$ going through v_{02}, the double-crosses # are on bits x_1, a_{11}.*

Proof: If we develop the tree from v_{02} we see that all inputs, which in P_1 go to the 0-sink, have double-crosses on x_1, a_{11}. Therefore they are distributed at v_{02}. □

Proposition 6.8 *At v_{03} there is a unique fd-class (of the distribution) F_0 containing only inputs from M. These inputs have values $x_1 = 1, x_2 = 1$ and $((a_{11} = 0$ and $a_{12} = 0)$ or $(a_{11} = 1$ and $a_{12} = 1))$.*

Proof: All inputs from $\{0,1\}^n - M$ which go through v_{03} and which are not distributed at v_{02} have the values $x_1 = 1, a_{11} = 1, x_2 = 1, a_{12} = 1$. Therefore at v_{03} there is no # on these inputs.

Let us develop a tree T from v_{03}. Since in P_2 x_1 and x_2 are tested repeatedly, at any leaf of T the inputs reaching it have the same values of x_1, x_2.

The case $x_1 = 0, x_2 = 0$. The bits a_{11}, a_{12} are not tested at all, therefore on a_{11}, a_{12} there are +'s (not #'). No double-crosses.

The case $x_1 = 0, x_2 = 1$. a_{11} is not tested at all. A cross + on a_{11}. Since v_{03} is reached on a_{12} there must be the value $a_{12} = 0$. No double-crosses.

The case $x_1 = 1, x_2 = 0$. a_{12} is not tested at all - there is a cross on a_{12}. Since v_{03} is reached, then there must be the value $a_{11} = 0$.

The case $x_1 = 1, x_2 = 1$. v_{03} is reached by inputs from M with the values $x_1 = 1, x_2 = 1, a_{11} = 0, a_{12} = 0$ or $x_1 = 1, x_2 = 1, a_{11} = 1, a_{12} = 1$. There are double-crosses (#) on $\{a_{11}, a_{12}\}$.

□

Proposition 6.9 *At v_{13} there are two fd-classes. The first one contains inputs from $\{0,1\}^n - M$, the double-crosses are on $\{x_2, a_{12}\}$. The second one F_1 contains inputs from M, the double-crosses are on $\{a_{11}, a_{12}\}$. These inputs have values $x_1 = 1, x_2 = 1, a_{11} = 0, a_{12} = 1$ or $x_1 = 1, x_2 = 1, a_{11} = 1, a_{12} = 0$.*

Proof: Let us develop a tree T from the node v_{13}. We don't consider the inputs from $\{0,1\}^n - M$ which are distributed at v_{02}. Each remaining input which goes to the 0-sink of P_1 has the values $x_1 = 1, a_{11} = 1$. Also we see that there are #'s on x_2, a_{12}.

Now let us consider the inputs from M reaching v_{13}. Since in P_2 the bits x_1 and x_2 are tested repeatedly, at any leafs of T the inputs from M reaching it have the same values of x_1, x_2.

The case $x_1 = 0, x_2 = 1$. a_{11} is not tested, on a_{11} there is a cross. a_{12} must be equal to 1. No double-crosses.

The case $x_1 = 1, x_2 = 0$. a_{12} is not tested, a + on a_{12}. a_{11} must be equal to 1. No double-crosses.

The case $x_1 = 1, x_2 = 1$. For reaching v_{13} it must hold $a_{11} = 0, a_{12} = 1$ or $a_{11} = 1, a_{12} = 0$. There are double-crosses on $\{a_{11}, a_{12}\}$.
□

Let F be the largest class from F_0, F_1.

Proposition 6.10 *The cardinality of F is at least $2^n/20$.*

Proof:
We divide all inputs into the groups $G_1, ..., G_{16}$ according to the equalities $x_1 = c_1, a_{11} = c_2, x_2 = c_3, a_{12} = c_4,\ c_i \in \{0, 1\}$. Ten of them $G_1, ..., G_{10}$ reach v_{03}, $G_{11}, ..., G_{16}$ reach v_{13}. For each $i, i = 1, ..., 10$ we define $G_i^0 = G_i \cap M$. For $i = 1, ..., 10$ all G_i^0 have the same cardinality since in and after v_{03} the computation depends only on $x_3, ..., x_m, a_{1,3}, ..., a_{1,m}$. Similarly for $G_{11}^0, ..., G_{16}^0$.
Let M_0 (M_1, resp.) be the set of inputs from M which go through v_{03} (v_{13} ,resp.). $|M_0| + |M_1| = |M| \geq 2^n/2$ (see Corollary 6.5).

At v_{03} the inputs from two groups from $G_1^0, ..., G_{10}^0$ form F_0. At v_{13} the inputs from two groups from $G_{11}^0, ..., G_{16}^0$ form F_1.

Hence, $|F_0| = |M_0|/5$, $|F_1| = |M_1|/3$. If $|M_0| \geq |M_1|$ then $|F_0| = |M_0|/5 \geq |M|/10 \geq 2^n/20$. If $|M_1| \geq |M_0|$ then $|F_1| = |M_1|/3 \geq |M|/6 \geq 2^n/12$. Hence, $|F| > 2^n/20$.
□

Corollary 6.11 *At least one of the distribution classes at v_{03} and v_{13} is a significant one.*

Now it suffices to prove that F is a significant fd-class which satisfies (R1),(R2),(R3) from the definition of the gentle programs.

First, let us consider the case $F = F_0$. We see that F_0 is a significant subclass of F_0.

ad(R1). Let us develop the tree S_F from v_{03}. We see that $x_1 = 1, x_2 = 1$ are noncrossed bits on each branch of S_F. The other bits which may sometimes be non-crossed are $a_{1,m}, x_m$.

ad (R2).

(R2) is clearly satisfied since all $t \in T_F$ have $x_1 = 1, x_2 = 1$. Hence, the result does not depend on $a_{1,1} = 0, a_{1,2} = 0$ or $a_{1,1} = 1, a_{1,2} = 1$.

ad (R3).

Let D be the set of double-crossed bits . We know that all branches of S_F have $D = \{a_{1,1}, a_{1,2}\}$. From Proposition 6.7 it is clear that in D there are no original subcubes. Hence, for all i, j $O_i^j = D$.

Similarly for the case $F = F_1$.

The theorem is proven.

□

6.2 Okolnishnikovova function $F_{n,s}$

Let us describe the functions $F_{n,s}$, n, s natural numbers, $n > s$, s divides n, which are the witness functions for the hierarchy of syntatic k-branching programs with respect to k ([15],[16]).

There are $N = \binom{n}{s}$ variables, each of them is indexed by a subset a, $|a| = s$, of the set $I_n = \{1, ..., n\}$. For $i = 1...n$, $W_i =_{df} \{a | i \in a \subseteq I_n, |a| = s\}$. We see that $|W_i| = \binom{n-1}{s-1}$. We define $F_{n,s} =_{df} \bigwedge_{i=1}^{n} (\bigvee_{a \in W_i} x_a)$.

Theorem 6.12 *There is a gentle program P (on $N = \binom{n}{s}$ variables) such that a) P computes $F_{n,s}$, b) $|P| \leq O(N \log N)$.*

Proof:

We shall describe a branching program P of the size $N \log N$ which computes $F_{n,s}$ such that P is a gentle branching program.

Let P_i be a program of a standard form which computes the function $\bigvee_{a \in W_i} x_a$. P_i has two sinks; in the source of P_i there is a test on the first x_a, the 1-edge of this test ends in the 1-sink, the 0-edge leads to the test of the next variable (or in the case of the test on the last variable to the 0-sink). On its turn the next variable will be managed in the same way. We see that $|P_i| = \binom{n-1}{s-1} + 2$.

Roughly speaking, P will be a concatenation (a conjunction) of P_i's with some additional properties.

We choose a variable x. Let all P_i's testing x have the tests on x at their sources. Moreover all P_i's with tests on x will form the initial sequence of our concatenation of all P_i's. (The number of P_i's with a test on x is s.)

From technical reasons after this initial sequence we insert a chain of redundant tests of all variables different from x into the concatenation of all P_i's. The tests of this chain are of specific form so that both 0-edge and 1-edge of a test of a variable go to the same node where the next variable is tested (in the case of the last test they go to the source of the first subprogram from the remaining P_i's).

Later we will define a linear ordering on variables tested in the initial sequence of P_i's.

Now we see that P computes $F_{n,s}$ and that $|P| \leq n|P_i| + N \leq O(N \log N)$.

Let us investigate the distribution on P. Let $v_1, ..., v_s$ be the sinks of P_i's from the initial sequnce such that v_i's are sources of the next parts of P. $v_1, ..., v_{s-1}$ are the unique candidates for a class of distribution before v_s (since they are the unique nodes before v_s with indegree > 1). If an input of $m \in \{0,1\}^N$ belongs to a class of the distribution at a node $v_1, ..., v_{s-1}$, then m does not have # on x since in $v_1, ..., v_{s-1}$ a test on x follows. Moreover, $comp(m)$ does not reach v_s since the tests on all variables different from x follow. Hence, each m which is distributed at $v_1, ..., v_{s-1}$ ends in a 0-sink of one of $P_1, ..., P_s$. The number of such m's is not greater than $s.2^N/2^{\binom{n-1}{s-1}}$.

The class of the distribution at v_s is a larger one. Each input m with $m_x = 0$ which reaches v_s has a double-cross # on x since v_s is also reached by the input m' which differ from m only on x. Further, at v_s the double-cross # may be only on x - this follows from the fact that in the next chain there are tests on all variables different from x.

We see that the class of the distribution at v_s is a significant proper class which consists from the unique fd-subclass F.

It is clear that F satisfies (R2) from the definition of gentle programs. It remains to be verified that (R1),(R3) are satisfied, too.

First we shall complete the specification of the construction of P. We define a linear ordering $\prec$ on the set of all variables (different from x). The initial part of $\prec$ is given by the (arbitrary) ordering of tests in P_1. In P_2 , first the tests on variables which are tested also in P_1 respect $\prec$. Moreover they precede the tests on the remaining variables. In $\prec$, these remaining variables follow after the previous ones and they respect the test ordering in P_2. In each P_i, the variables from the previous P_j's are tested in respect to $\prec$ at first, then new variables are tested and $\prec$ is extended.

Let $y_1, ..., y_s$ be the variables which in $P_1, ...P_s$ are tested as the last variables. Let us suppose that we have $m \in F$. For each variable $y \neq x$ we shall find out whether y is crossed (+) at v_s or not.

If y is not tested during $comp(m)$ between the source of P_1 and the node v_s, then y is crossed since there is m' , $m'_y \neq m_y$ and $m'_{\neq y} = m_{\neq y}$. Both $comp(m), comp(m')$ reach v_s and they must branch on y in the chain after v_s.

Now, we have $m \in F$, $y \neq x$, and y is tested during $comp(m)$ between the source of P_1 and v_s.

We have two cases A) $m_x = 0$, B) $m_x = 1$.

ad B). Since $m_x = 1$ and $m \in F$ there is an m' such that $m'_x = 0$ and m' reaches

v_s. From v_s $comp(m)$ and $comp(m')$ follows the same path, hence they have the same set of crosses +'s. We see that B) is the same as A) for m'.

ad A). We have $m \in F$, $m_x = 0$. Let $y \neq x$ be a variable tested during $comp(m)$ before v_s.

a) the case $m_y = 0$. There is an input m' such that $m'_y = 1$ and $m'_{\neq y} = m_{\neq y}$.

We see that $m'_x = 0$ and m' reaches v_s (m' has more one's than m). Therefore, $m' \in F$ (see above). m and m' branch on y in the chain after v_s. Hence, m, m' have + on y.

b) the case $m_y = 1$. There is an input m' such that $m'_{\prec y} = m_{\prec y}$ and $m'_y = 0$ and $m'_{y\prec}$ is an unary word over $\{1\}$. If $y \neq y_1, ..., y_s$ (last variables), then m' reaches v_s. Since $m'_x = 0$ we have $m' \in F$. m and m' branches on y in the chain after v_s. Hence, both m, m' have + on y.

We see that for all input $m \in F$ all non-crossed bits are from the set $\{y_1, ..., y_s\}$. Since $s < N^{1/8}$, (R1) is satisfied.

(R3) is satisfied trivially since $D = \{x\}$ and in any computation on any input x is tested before v_s.

□

6.3 Multipointer functions

In [20] it was proven for $k(n)$ up to $k(n) = 2^{-1}.n^{1/8}.log^{-1/4}n$ that multipointer functions $f_{n,k(n)}$ causes the hierarchy for $(1, +k(n))$-branching programs - it means that $f_{n,k(n)}$ is polynomially easy for $(1, +k(n))$-branching programs but superpolynomially hard for $(1, +k(n) - 1)$-branching programs. We shall prove that $f_{n,k(n)}$ (for $k(n)$ till $n^{1/8}$) are polynomially easy for the gentle branching programs.

We define the functions $f_{n,k}$ as follows: First we describe them informally. The n variables are divided into k blocks of length m. For every $j = 1, 2, ..., k$, a weighted sum of the bits of block j determines an index i_j of some of the input bits. Then the value of the function is the parity of the bits determined by i_j for $j = 1, 2, ..., k$. The exact definition of $f_{n,k}$ requires some technical notation.

For every natural number n, let $p(n)$ be the smallest prime greater than n. Consider the set $\{1, 2, ..., n\}$ as a subset of $Z_{p(n)}$, the field of the residue classes modulo $p(n)$. Then for every $t \in Z_{p(n)}$ let $\omega(t) = t$ if $t \in \{1, 2, ..., n\}$ and $\omega(t) = 1$ otherwise.

Definition 6.13 *For every $t = (t_1, ..., t_k) \in \{1, ..., n\}^k$ and every $x \in \{0, 1\}^n$, let $Par(x, t) =_{df} x_{t_1} \oplus ... \oplus x_{t_k}$.*

Definition 6.14 *Let k divide n and let $m = n/k$. Let $\psi_{n,k} : \{0, 1\}^n \to \{1, 2, ..., n\}^k$ be defined as follows. For every x let $\psi_{n,k}(x) =_{df} (t_1, ..., t_k)$ where for every $j = 1, 2, ..., k$*

$t_j = \omega(\sum_{i=1}^{m} i x_{(j-1)m+i})$ *where the sum is evaluated in* $Z_{p(n)}$. *Moreover let* $f_{n,k}(x) =_{df} Par(x, \psi_{n,k}(x))$.

Theorem 6.15 *For* $k < n^{1/8}$ $f_{n,k}$ *is computable on a gentle branching program of the size* $O(n^2)$.

Proof:

Given $k \in N$ we shall construct a branching program P computing $f_{n,k}$. The structure of P will reflect k blocks of input bits with the corresponding pointers.

P will consist from k-parts P_i, $i = 1, ..., k$. Each P_i will have two input nodes (with the exception of P_1 which has only one input node - the source of P) and two output nodes. The output nodes of P_k are the sinks of P. The two output nodes of P_i are sticked with the input nodes of P_{i+1} for all $i = 1, ..., k-1$. One of the output node of P_i is called the 0-node , the other is called the 1-node. For $a \in \{0, 1\}$ it is the truth that the a-node of P_i is reached by all inputs for which the value of the parity of the first i bits pointed out by pointers from the first i blocks is equal to a.

P_1 has $m + 1$ levels. On each level from the first m ones one bit is tested. Each output node of the i-th level represents one magnitude of the contribution of the first i bits to the pointer. On the last level $(m + 1)$ the bits which are pointed out are tested. The outgoing edges are sticked to the a-nodes of P_1.

The other P_i's consist from two copies of P_1 (labelled by bits from the i-th block). The only change is on the $m + 1$-st levels where the outgoing edges must point to the correct a-node.

We see that that P computes $f_{n,k}$ and that the size of P is not greater then $(2k - 1)|P_1| \leq O(n^2)$.

It remains to prove that P is a gentle branching program. We assume that at the top of P_1 there is a small irregularity. The initial part of P_1 consists of the full tree of depth 3 where at the root of the tree the variable x_2 is tested, at its successors x_3 is tested and the last tests are on x_5. The leaves of the branches $(x_2 = 1, x_3 = 1, x_5 = 0)$ and $(x_2 = 0, x_3 = 0, x_5 = 1)$ are sticked into one node v (they give the same contribution to the pointer on the first block). Moreover we suppose that at the level m of P_1 the variable x_m is tested.

Now let us perform the distribution on P. The node v is one of the first candidates for nodes at which some clases are distributed. Let us consider an input p going through v and assume that $p_2 = 1, p_3 = 1, p_5 = 0$. Let us take the input p' with $p'_2 = 0, p'_3 = 0, p'_5 = 1$ such that p' follows p to a sink. We see that p, p' have #'s on x_2, x_3, x_5. Since during the path through P all variables different from x_2, x_3, x_5 are tested p, p' must equal on these variables. Moreover p, p' must avoid possible tests on x_2, x_3, x_5 at the last levels of P_i's. Let us estimate the number of such p - i.e. the

cardinality of the class F of the distribution at v.

We see that from the pair of inputs which differ only in the last variable of the first block at most one of them goes through a test on one from the variables x_2, x_3, x_5 at the last level of P_1 since the possible contribution of the last variable to the pointer is very large. The same fact holds for all remaining blocks. Hence the cardinality of F is at least $(2^n/4).2^{-1}.2^{k-1} \geq 2^n/2^{2n^{1/8}}$. Therefore F is a significant class of the distribution.

We see that F is also an fd-class. It remains to verify the conditions (R1),(R2),(R3) from the definition of the gentle branching programs.

Ad (R1). In each P_i we want to avoid the tests on x_2, x_3, x_5. This can be ensured by fixing the last variable in each block. Hence at most these $n^{1/8}$ variables are noncrossed in any computation on input from F.

Ad (R2). Since after v the inputs from F are not tested on x_2, x_3, x_5 the reached sink does not depend on these variables.

Ad (R3). Clearly.
□

7 Other upper bounds

7.1 Parity of triangles in the graph

Let $G = (V, E)$ be a graph,$V = \{v_i\}_{i=1}^m$, $E \subseteq V \times V$. G defines a binary matrix $A = (a_{ij})_{i=1,j=1}^m$ where $a_{ij} = 1$ iff $(v_i, v_j) \in E$.

Since G is unoriented, A is symmetric. By a code of G we mean the binary string $e = a_{12}...a_{1m}a_{23}...a_{2m}...a_{j,j+1}...a_{jm}...a_{m-1,m}$.

Let f be a Boolean function such that $f(e) = 1$ iff e is a code of a graph which has an odd number of triangles.

For f the lower bounds $2^{n/c}$ on the size of 1-branching programs have been proven in [2],[22].

Theorem 7.1 *There is a gentle branching program P of size $O(n^{3/2})$ which computes f.*

Proof: In P the following algorithm is implemented.

We consider all possible triangles (on m vertices), each (possible) triangle is considered only once.

We fix an (possible) edge, say e_1, arbitrarily. In the first phase of the algorithm we consider all (possible) triangles over e_1. In the second phase, we consider the remaining triangles.

P will have k levels P_i, $i = 1...k$ where k is the number of all possible triangles. Each level P_i will be responsible for considering a triangle.

Each P_i has two inputs, two outputs and some internal nodes (with exception of P_1 with one input node - the source of P). The output nodes of P_{i-1} will be sticked with the input nodes of P_i. The output nodes of P_k are sinks of P.

For $i = 1...k$, one output node of P_i is called 0-node, the other one the 1-node. The following rule is fulfilled : Starting at the source of P, each input which among the first i considered possible triangles has an even (odd, resp.) number of actual triangles achieves the 0-node (the 1-node, resp.) of P_i.

We see that P computes f. It remains to specify the internal nodes of each P_i. For each i the input nodes of P_i will be the roots of two copies of the full ballanced tree of depth 3 with tests on variables which represent the edges of which the i-th triangle consists. The out-going edges in the depth 3 point to 0- and 1-node of P_i in such a way that the rule concerning the 0- and 1-nodes is satisfied.

We see that the size of P is at most $O(n^{3/2})$.

Now we are going to prove that P is a gentle b.p.. Let i_0 be the last level of P where the edge e_1 is tested. Let us perform the distribution on P and let us consider the situation at the 0-node and 1-node of P_{i_o}. In P only the output nodes of P_i's have in-degree > 1, hence only at the output nodes the double-crosses (#) may arise. From the construction of P it is clear that in $P_{i_0+1}, ..., P_k$ all computations have tests on all variables except e_1.Hence in $P_1, ..., P_{i_0}$ the #'s may be at most on e_1. Moreover, in $P_1, ..., P_{i_0-1}$ the # on e_1 is impossible since a test on e_1 follows in P_{i_0}.

Let us investigate the possibility of # on e_1 at the output nodes of P_{i_0}. Let m be an input. m has # on m_1 iff there is m' such that a) $m'_1 \neq m_1$, b) $comp(m')$ follows $comp(m)$ from the node in question to a sink. In the output nodes of P_{i_0} the condition b) implies that $m_{\neq 1} = m'_{\neq 1}$ since in the remaining part of P $comp(m)$ tests all variables different from e_1.

W.l.o.g. we suppose that $m_1 = 1, m'_1 = 0$. We see that m is the code of such a graph G_m that deleting the edge e_1 from G_m the parity of the number of triangles does not change since $comp(m')$ goes to the same sink as $comp(m)$. Hence, in G_m, the number of triangles with the edge e_1 is even. Hence, # may only be at the 0-node of P_{i_0}.

On the other hand, the condition "$m_1 = 1$ and in G_m there is an even number of triangles with e_1" is a sufficient one for a # on m_1.

Let F be the set of all inputs which have a # on e_1 at the 0-node of Pi_0. Later we shall see that F is a significant class of the distribution. It is clear that F is also an fd-class and that for F the condition (R2) from the definition of gentle b.p.'s is satisfied.

For verifying (R1), from the 0-node of P_{i_0} let us develop the tree S_F induced by F.
Since during each computation each variable different from e_1 is tested there are exactly two inputs m, m', $m_1 \neq m'_1, m_{\neq 1} = m'_{\neq 1}$ in each leaf of S_F.

Let us arbitrarily choose a branch b of S_F. We shall investigate the set of non-crossed bits of inputs following b.

What concerns variables which have not been tested in $P_1, ..., P_{i_0}$ there will be a cross on each of them. It is clear since after P_{i_0} each computation goes through nodes with tests on all (till this moment) non-tested variables.

Let $m \in F$ be an input following b, let $x \neq e_1$ be a variable tested in $P_1, ...P_{i_0-1}$. Let y be such a variable that the edges represented by e_1, x, y form a triangle. W.l.o.g. we suppose $m_1 = 1$.

If $m_y = 0$ we define an input m' as follows : $m'_{\neq x} =_{df} m_{\neq x}$ and $m'_x \neq m_x$. Since $y = 0$ it is clear that m' has the same number of triangles over e_1 as m does (even) and therefore $m' \in F$. m' will branch with m on x therefore m has a cross on x.

If $m_y = 1$, then we choose m' such that
(i) $m'_{e_1} =_{df} 1$,
(ii) $m'_x \neq_{df} m_x$,
(iii) m' equals m on variables which are tested on b before x,
(iv) On the other variables with exception of those which represent the i_0-th triangle over e_1 we define m' arbitrarily.
(v) On the remaining two variables we define m' in such a way that m' has an even number of triangles over e_1.
Now we see that $m' \in F$, m' follows m along b till to the test on x. Hence m has a $+$ on x.
We see that on each branch b of S_F the only candidates for non-crossed bits are the two bits which are tested at the level i_0. Hence (R1) is satisfied.

By the way we have obtained the fact that the cardinality of F is at least 2^{n-3}. Hence F is a significant class of the distribution on P.

(R3) is satisfied clearly.
□

7.2 Half-cliques-only

By a half-clique-only we understand an unoriented graph where one half of vertices form the full graph and the other vertices are isolated.($G = (V, E), V = V_1 \cup V_2, |V_1| = |V_2|, E = V_1 \times V_1$).

In 1984 [24] it was proven that the Boolean function f which gives 1 exactly on the codes of half-cliques-only is subexponentially hard for 1-branching programs. In 1993 [4] it was proven that f is subexponentially hard even for nondeterministic 1-branching programs.

By a code of a graph we mean the same string $e = s_1, ..., s_{m-1}$ as in the previous subsection where $s_j = a_{j,j+1}, ..., a_{j,m}$.

Let us notice that the code of any half-clique-only has some special properties :
a) Exactly $m/2 - 1$ segments s_i , $i = 1, ..., m$, contain at least one symbol 1 ,
b) For $i < j$ if $a_{i,j} = 1$ then the segment s_j is an suffix of the segment s_i.

Let us introduce a natural algorithm computing f.
1. Search for the first segment s_i containing a symbol 1; s_i is pointed out.
2. Move 1 to a counter C.
3. Perform $Proc$ until $C = m/2 - 1$.
4. Let s_j be the last segment which is pointed out. Check whether the segment s_k is a zero-sequence for each $k, k > j$.

$Proc$:
a) Let s be the segment which is pointed out. Search for the first $j_0, j_0 > j$ such that $a_{j,j_0} = 1$ (the leading one).
b) Check whether s_k is a zero sequence for each $k, j < k$.
c) Check whether s_{j_0} is a suffix of s_j.
d) s_{j_0} is pointed out.
e) Add 1 to C.

It is easy to see that such an algorithm can be implemented in a branching program P_1 of the size of at most $O(m^4) \leq O(n^2)$.

Theorem 7.2 *f is computable on a gentle branching program P of the size $O(n^2)$.*

Proof:

At the top of P there is a tree of the depth 3 with branching on $a_{1,2}, a_{1,3}, a_{2,3}$. The branches $a_{12} = 1$ $a_{1,3} = 0$ $a_{2,3} = 1$ and $a_{1,2} = 1$ $a_{1,3} = 1$ $a_{2,3} = 0$ are sticked into one node v which is a sink of P with the label 0. The other branches are sticked to a node which is the source of P_1.

It is clear that P computes f and that $|P| \leq O(n^2)$. It remains to prove that P is a gentle program.

The sink v is reached by $2^n/4$ inputs. Each of them has #'s on $a_{1,3}, a_{2,3}$, 1 on $a_{1,2}$ and +'s on the other bits. Hence, these inputs form a significant proper class of the distribution. It easy to see that this class is an fd-class fullfilling the conditions (R1),(R2),(R3) from the definition of gentle branching programs.

□

7.3 Ablayev-Karpinski's function

In [] there is defined a function f on words over four-letter alphabet $\{0, 1, \bar{0}, \bar{1}\}$ as follows : $f(u) = 1$ iff $p(u) = \bar{p}(u)$ where $p, \bar{p}$ are projections such that $p(u_1u_2) = p(u_1)p(u_2)$ and $p(0) = 0, p(1) = 1$ and $p(\bar{0}) = p(\bar{1}) = \Lambda$ (= the empty word). Similarly for $\bar{p}$.

In [1] the authors prove that f is superpolynomially hard for nondeterministic four-letter 1-branching programs. We use a variant of f where the four letter alphabet is encoded by strings of the length two $\{00, 10, 01, 11\}$ where the first bit represents the value and the second one represents the type.

There is a natural algorithm A computing f which is based on the procedure: "Search for the next value of the first type and search for the next value of the second type, and compare".

Such an algorithm can be implemented in a branching program P_1 of the size of at most $O(n^3)$.

Theorem 7.3 *f is computable on a gentle branching program P of size $O(n^3)$.*

Proof: At the top of P there is a tree on depth four which branches on the first four variables. The branches induced by inputs with prefixes $0011, 1001, 0110, 1100$ are sticked into a node v which is a sink of P with the value 0. The other branches are sticked into the other node which is the source of P_1.

It is clear that P computes f and that the size of P is at most $O(n^2)$. It remains to prove that P is a gentle program.

The sink v is reached by $2^n/4$ inputs. At v each of them has #'s on the first four bits. The other bits of these inputs have +'s. We see that the class of these inputs is a significant proper class of the distribution, moreover it is also a significant fd-class fullfilling the conditions (R1),(R2),(R3) from the definition of gentle branching programs.

□

7.4 Multiplication

S. Ponzio [17] has proven a $2^{c\sqrt{n}}$ lower bound for 1-branching programs computing a function f which is closely connected with the multiplication. For the same function f we shall prove a $O(n^3)$ upper bound for gentle branching programs.

Given $x, y \in \{0,1\}^m$, $x = x^m x^{m-1} ... x^2 x^1$, $y = y^m y^{m-1} ... y^2 y^1$ we define $f(xy) =_{df} z^m$ where z^m is the m-th digit (from the right) of the binary number $z = x \times y \in \{0,1\}^{2m}$, $z = z^{2m} z^{2m-1} ... z^2 z^1$.

Theorem 7.4 *f is computable on a gentle branching program P of the size $O(n^3)$.*

Proof: In P we will implement the well-known algorithm for the multiplication. The algorithm sums the rows $R_i = r_{im} .... r_{i1} 0^{i-1}$ for $i = 1...m$ where $r_{ij} =_{df} x^j \times y^i$. We will consider the columns $C_j = c_{1j} ... c_{m,j}$ where $c_{ij} =_{df} r_{i,j-i+1}$ for $j - i + 1 > 0$ and $c_{ij} =_{df} 0$ otherwise. The value $f(xy)$ is given as the sum of the last digit of the sum of C_m and of the last digit of the transition from the previous columns $C_{m-1}, ..., C_1$.

The structure of P will reflects the columns $C_1, ..., C_m$ and the magnitude of the respective transitions.

P will consists of the parts $P_1, ..., P_m$. With exceptiom of P_m, P_j's are of the regular form as follows:

Each P_j has some input nodes ad some output nodes. Each input node represents one magnitude of the transition after the sum of the columns $C_1, ..., C_{j-1}$. It means that all inputs xy having the transition of the same magnitude (after $C_1, ..., C_{j-1}$) go through the same input node of P_j. We see that P_1 has only one input node - the source of P. The output nodes of P_j are sticked with the input nodes of P_{j+1}.

Each P_j, $j = 1, ..., m-1$ consists of l_j levels. Each level L_i, $i = 1...l_j$ corresponds to the value c_{ij}. Each level L_i, $i = 1, ..., l_j$ has some input nodes and some output nodes. Each input node represents one magnitude of the sum of the transition after the columns $C_1, ..., C_{j-1}$ and of $c_{1j}, ..., c_{i-1,j}$. The output nodes of L_i are input nodes of L_{i+1}.

At the level L_i it is necessary to find out the value c_{ij}. Each input node of L_i is the root of the full tree of the depth 2. The tree tests the variable x^{j-i+1} at its root and the variable y^i at the two immediate successors of the root. The leaves of these trees are sticked into the output nodes of L_i.

Now P is described till P_{m-1}. Since the maximal magnitude of the transition is not larger than $2m$ the inequality $\sum_{j+1}^{m-1} |P_j| \leq O(n^3)$ holds.

It remains to describe P_m. Each input node of P_m (=an output node of P_{m-1}) has out-degree = 1. In any case the outgoing edge leads to a node a if the last (the

rightmost) digit of the transition after $C_1, ..., C_{m-1}$ is zero or to a node b in the other case. Starting with a, b P_m consists of levels L_i , $i = 1, ..., l_m$. (We know that $l_m = m$.) Each level L_i has two input nodes and two output nodes. Each input node of L_i represents one value of the last digit of the sum of the transition after $C_1, ..., C_{m-1}$ and of $c_{1m}, ..., c_{i-1,m}$. The output nodes of L_i are sticked with the input nodes of L_{i+1}.The output nodes of L_m are the sinks of P.

Now we see that P will compute f.

At each input node of L_i it is necessary to find out the value of c_{im}. This is ensured by two full trees of the depth 2 with roots in the input nodes. In both trees at the root the variable x^{m-i+1} is tested and at the both succesors of the root we test the variable y^i. With exception of L_1 the leaves of both trees are sticked into the output nodes of L_i.

Inside of L_1 firstly, in the tree with the root in a (b,resp.) we stick the branches $x^m = 0, y^1 = 0$ and $x^m = 1, y^1 = 0$ to a node c (d, resp.). Then we give edges leading from c, d and the remaining leaves of both trees to the output nodes of L_1.

We see hat $|P| \leq O(n^3)$. It remains to prove that P is a gentle branching program.

Let us perform the distribution on P. For all inputs no bit is double-crossed in $P_1, ..., P_{m-1}$ since each bit is tested in P_m. The nodes c, d in L_1 of P_m are the first candidates for nodes with a proper class of the distribution.

Proposition 7.5 *Each input xy going through c (d, resp.) has a double-cross (#) on x^m and only on x^m at c (d, resp.).*

Proof: W.l.o.g. let xy be an input going through c. At a there is the first test on x^m . Hence also the input $\bar{x}y$ goes through c where x and $\bar{x}$ differ only on x^m (y^1 is non-crossed, $y^1 = 0$). Since the test at a is the last test on x^m, after c xy and $\bar{x}y$ never branch and therefore they have a # on x^m. After c each input goes through tests on all remaining bits different from x^m, y^1. Hence on these bits there is no # at c.
□

2^{n-1} inputs go through nodes c, d. Hence at least one of the classes of the distribution at c and at d is a significant one. W.l.o.g. we assume that there is a significant class F at c. We see that F is also an fd-subclass and therefore it suffices to prove that F satisfies the conditions (R1),(R2),(R3) from the definition of gentle branching programs.

Ad (R1). Let us consider the tree S_F induced by F at c.

Proposition 7.6 *Let xy be an input from F following a branch b of S_F. Then all bits different from $x^1, x^2, x^3, x^m, y^1, y^{m-2}, y^{m-1}, y^m$ are crossed in $w(xy, c, F, P)$.*

Proof: By contradiction. Let e be a non-crossed bit of $w(xy, c, F, P)$ different from $x^1, ..., y^m$. We shall construct another input $x'y'$ such that a) $x'y' \in F$, b) in S_F $x'y'$ follows xy till the test on e where $xy, x'y'$ will branch. (Hence xy will have a cross on e.) This will be our contradiction.

We choose x'^m arbitrarily, we put $y'^1 =_{df} 0$. Then (in P) we follow xy from c to the test on e - this path defines the value of pairs x'^{m-1}, y'^2; x'^{m-2}, y'^3 ; ... . In $x'y'$ on e we give the opposite value (as in xy). In any case $x'^1, x'^2, x'^3 y'^m, y'^{m-1}, y'^{m-2}$ remain undefined. If some other bits of $x'y'$ are also undefined we define them arbitrarily. Further we define $x'^1 = 1$, $x'^2 = 1$, x^3, y'^m arbitrarily. y'^{m-1} and y'^{m-2} are defined in such a way that the last digit of the transition after $C_1, ..., C_{m-1}$ (on $x'y'$) is equal to 0.

We can argue that this is possible as follows: The starting point is to choose $y'^{m-1} = y'^{m-2} = 0$. If the last digit of the transition for C_m is equal to one we change y'^{m-1}, y'^{m-2}. We know that y'^{m-1} is taken into account only in C_{m-1} (the corresponding bit is x'^1) and that y'^{m-2} is taken into account only in C_{m-1} (the corresponding bit is x'^2) and in C_{m-2} (the corresponding bit is x'^1). Let $z' =_{df} x' \times y'$. If under the condition $y'^{m-1} = y'^{m-2} = 0$ we have $z'^{m-1} = 1$ we change y'^{m-1} to 1. If $z'^{m-1} = 0$ and $z'^{m-2} = 0$ we put $y'^{m-1} = y'^{m-2} = 1$. If $z'^{m-1} = 0$ and $z'^{m-2} = 1$ we put $y'^{m-1} = 0, y'^{m-2} = 1$.

Hence $x'y'$ goes through a and moreover since $y'^1 = 0$ $x'y'$ goes through c and it has # on x'^m. Therefore $x'y' \in F$ - the condition a) from our plan of the proof is satisfied. Also b) is satisfied. A contradiction.

□

From the proposition it follows that the only candidates for non-crossed bits are $x^1, x^2, x^3, y^{m-2}, y^{m-1}, y^m$. Hence (R1) is satisfied.

Ad (R2). Let us choose $h \in H_F$, $t \in T_F$. Let us consider $comp([ht])$ in P. We shall prove that $[ht]$ goes through c and after c the computation depends only on t. Since $t \in T_F$ there is an input $xy \in F$ such that $xyt_F = t$ (where t_F contains all bits except of x^m). The input xy goes from the source to a. Before a there is no test on x^m. Hence $[ht]$ follows the same path from the source to a. Sice $xy \in F$ $y^1 = 0$ holds - therefore also $[ht]$ goes to c. After c only the bits from t_F are tested, hence the computation depends only on t. We see that (R2) is satisfied.

Ad (R3) The bit which is double-crossed at c is tested before c at a for all inputs from F. Hence (R1) is satisfied.

We see that P is a gentle branching program. The Theorem is proven.

□

Acknowledgements.

I thank K. Bendová, S. Jukna, P. Pudlák for their discusions. I am grateful to P. Savický for his help with the main proofs of Section 3.

Bibliography

[1] F.Ablayev, M. Karpinski , On the power of Randomized Branching Programs , Proc. of ICALP'96, Lecture Notes in Computer Science 1099, Springer 1996, 348 - 356.

[2] 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.

[3] D. A. Barrington , Bounded-width Polynomial Size Branching Programs Recognize Exactly those Languages in NC^1, Proc. 18. ACM STOC, 1 - 5.

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

[5] 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.

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

[7] A. Gál , A simple function that requires exponential size read-once branching programs , to appear in Combinatorica.

[8] S. Jukna, Entropy of Contact Circuits and Lower Bounds on Their Complexity, Theoretical Computer Science, 57 (1988), pp. 113 - 129.

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

[10] S. Jukna, A. A. Razborov, Neither Reading Few Bits Twice nor Reading Illegally Helps Much, TR96-037,ECCC, Trier.

[11] M.Krause, S. Waack, On Oblivious Branching Program of Linear Length, Berlin, 1989

[12] K. Kriegel , S. Waack , Exponential lower bounds for real-time branching programs , Proc. FCT'87. Lecture Notes in Computer Science, Vol. 278, Springer 1987, 263 - 267.

[13] C. Meinel, S. Waack , Separating Complexity Classes Related to Bounded Alternating ω-Branching Programs , Math. Systems Theory 28, 21 - 39 (1995)

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

[15] E. A. Okolnishnikovova , Comparing the complexity of binary k-programs (in Russian), Diskretnyj analiz i issledovanije operacij, 1995, Vol. 2, No. 4, pp. 54 - 73.

[16] E. A. Okolnishnikovova , On the Hierarchy of Nondeterministic Branching k-Programs, FCT'97

[17] 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.

[18] P. Savický, S. Žák , A lower bound on branching programs reading some bits twice, Theoretical Computer Science 172, 1997, pp. 293 - 301.

[19] P. Savický, S. Žák , A large lower bound for 1-branching programs, TR96-036, ECCC, Trier.

[20] P. Savický, S. Žák , A hierarchy for (1,+k)-branching programs with respect to k , Proc. MFCS'97, Lecture Notes in Computer Science 1295, Springer 1997, 478 -487.

[21] D. Sieling , New Lower Bounds and Hierarchy Results for Restricted Branching Programs , TR 494, 1993, Univ. Dortmund, to appear in JCSS.

[22] 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.

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

[24] S. Žák , An exponential lower bound for one-time-only branching programs, Proc. MFCS'84, Lecture Notes in Computer Science 176, Springer 1984, 562 - 566.

[25] S. Žák , An exponential lower bound for real-time branching programs, Inform. and Control 71, 1986 , 87 -94.

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