Electronic Colloquium on Computational Complexity, Report No. 30 (1998)

On Branching Programs With Bounded Uncertainty

Stasys Jukna* Stanislav Žák†

June 9, 1998

Abstract

We propose an information-theoretic approach to proving lower bounds on the size of branching programs (b.p.). The argument is based on Kraft-McMillan type inequalities for the average amount of uncertainty about (or entropy of) a given input during various stages of the computation. We first demonstrate the approach for read-once b.p. Then we introduce a strictly larger class of so-called 'gentle' b.p. and, using the suggested approach, prove that some explicit Boolean functions, including the Clique function and a particular Pointer function (which belongs to AC^0), cannot be computed by gentle program of polynomial size. These lower bounds are new since explicit functions, which are known to be hard for *all* previously considered reading-restricted classes of b.p. (such as $(1,+s)$-b.p. or syntactic read-k-times b.p.) can be easily computed by gentle b.p. of polynomial size.

1 Introduction

We consider the usual model of *branching programs* (b.p.). This model captures the deterministic space in a natural way whereas nondeterministic branching programs

*Department of Computer Science, University of Trier, D-54286 Trier, Germany & Institute of Mathematics, Akademijos 4, LT-2600 Vilnius, Lithuania. E-mail: jukna@ti.uni-trier.de. Research supported by the DFG grant Me 1077/10–1.

†Institute of Computer Science, Academy of Sciences, Pod vodárenskou věží 2, 182 00 Prague 8, Czech Republic. E-mail: stan@uivt.cas.cz. Research supported by the Grant Agency CR Grant No 201/98/0717 and partly by MŠMT ČR Grant No. OK-304 and by INCO-Copernicus Contract IP961095 ALTEC-KIT.

ISSN 1433-8092

(n.b.p.) do the same for the nondeterministic mode of computation. Despite considerable efforts, the best lower bound for unrestricted (nondeterministic) b.p. remains the almost quadratic lower bounds of order $\Omega(n^2/\log^2 n)$ proved by Nečiporuk in 1966 [10]. In order to learn more about the power of branching programs, various restricted models were investigated. Let us briefly sketch the progress in that direction (see, e.g. [14] for more comprehensive survey).

One of the most intensively studied models was that of *read-k-times* programs (k-b.p. or k-n.b.p.) where in each computation every input bit can be tested at most k times. This model introduced in [9] corresponds to so-called *eraser* Turing machines, and the first super-polynomial lower bounds for 1-b.p. were obtained in [19, 18]. In fact, for a related model of "regular resolution" (this is a 1-b.p. with more than two outputs) exponential lower bounds were proved, already 30 years ago, by Tseitin in 1968 [16]. For *non-deterministic read-once b.p.* (1-n.b.p.) first exponential lower bounds were proved in [4]. Now, there is a long series of exponential lower bounds for 1-b.p. However, any attempts to get such bounds for, say 2-b.p., failed.

The case of *syntactic* k-b.p. where in *every* path, be it consistent or not, every variable appears at most k times, is much easier to capture and analyze, and, indeed, strong lower bounds for syntactic k-b.p. and syntactic k-n.b.p. (for $k = O(\log n)$) were established [2, 11, 6, 12, 13]. In particular, it was proved that the characteristic functions of particular linear codes require k-b.p. ([11]) and k-n.b.p. ([6]) of exponential size.

Another idea to go closer to the 2-b.p. case was to allow a limited number of bits be tested more than once. More specifically, $(1, +s)$-branching programs are the usual b.p. where in every consistent path at most s variables are tested more than once. Exponential lower bounds for this model were proved in [20, 15, 7]. Again, characteristic functions of linear codes were shown in [7] to require $(1, +s)$-b.p. of super-polynomial size, as long as $s = o(n/\log n)$.

In this paper we describe one approach to proving lower bounds in which main accent is made upon conditions on 'information flow' during the computations rather than on prohibiting the machine to follow certain paths. The approach is based on a more careful analysis of the "amount of uncertainty" about single inputs during the computations on them.

Uncertainty. Given an input $a \in \{0,1\}^n$, the computation $comp(a)$ on a starts in the source-node with no knowledge about this input. At each step the computation

makes a test "is $a(i) = 0$ or $a(i) = 1$?", and after each test one bit of information about the input a is obtained. However, this information about the $a(i)$ is lost at the node v if there is another input b such that $b(i) \neq a(i)$, the computation $comp(b)$ on b reaches the node v, and after the node v one of the following two events happen: either $comp(b)$ diverges from $comp(a)$ immediately after the test of x_i, or $comp(b)$ follows the computation $comp(a)$ until they both reach the same sink. In both cases the program is uncertain about the value $a(i)$: in the first case it tests this bit once more, whereas in the second - it forgets that bit forever.

We mark those bits $a(i)$, $i \in \{1, \ldots, n\}$ of the input a, for which at least one of these two events happen, and call the resulting (marked) string a 'window' of the input a at the node v. The total number $\mathcal{E}(a)$ of marked bits measures the entropy of (or uncertainty about) a at this node. This form to encode the uncertainty using windows was proposed by the second author in [21].

The approach. We suggest the following general frame to prove lower bounds for branching programs.

In the program P computing a given Boolean function f we stop each computation at a particular node. This way we distribute the inputs among the nodes of P: each class F of this distribution corresponds to one node and consists of those inputs, the computations on which where stoped at that node. Then, using the properties of f, we show that, for each such class F, the average entropy $\mathcal{E}(F) = \frac{1}{|F|} \sum_{a \in F} \mathcal{E}(a)$ is small (this is the hardest step of the whole approach). Kraft-McMillan type inequalities (proved in Sect. 4) immediately imply that then no of these classes F can be large. Hence, there must be many such classes and therefore we need many nodes in P.

The results. For read-once b.p. finding non-trivial upper bounds for the average entropy $\mathcal{E}(F)$ is an easy task (see Proposition 4.6). Looking for larger classes of b.p. where this task is still tractable, we define in Sect. 5 one general property of branching programs – the 'gentleness'. Roughly, a program P is gentle if at some of its nodes some large set F of inputs is classified in a 'regular' manner, where the regularity requires that windows of inputs from F at these nodes have some special form. We then prove the following.

1. Read-once branching programs are gentle (Sect. 6).

2. Explicit functions, which are hard for *all* previously considered restricted models of b.p. (such as the characteristic functions of linear codes), can be easily computed by small gentle b.p. (Sect. 7). This fact is not surprising – it just

indicates that 'gentleness' is a new type of restriction. In our upper bound proofs we use the following possibility: if a function is easy to compute by an *unrestricted* b.p. and it has any combinatorial singularity hidden inside, then this singularity can be hardwired into the program to make it 'gentle'.

3. We isolate a new combinatorial property of Boolean functions – the 'strong stability', and (using the bounds on the average entropy $\mathcal{E}(F)$ established in Sect. 4) prove that any such function requires gentle b.p. of exponential size (Theorem 8.2). This criterion implies that some explicit Boolean functions – the Clique function and a particular Pointer function (which belongs to AC^0) – cannot be computed by gentle programs of polynomial size.

We note that the "gentleness" is only a temporary phenomenon which reflects the level of proofs which we are able to do at this time. The main motivation for studying various restricted computational models is to build up techniques and intuitions about what inherent properties of functions make them hard to compute. In this paper we make one more step in that direction: we propose an information-theoretic technique for proving lower bounds and identify a new combinatorial property (the stability) of functions which make them hard to compute in a "gentle" way.

2 Branching programs

Given a set of bits $I \subseteq [n] \rightleftharpoons \{1, \ldots, n\}$, an *assignment on* I is a mapping $a : I \to \{0,1\}$; here I is the *domain* of a. Assignments on the whole set $[n]$ are called *input vectors* (or simply *inputs*). A *projection* of a onto a subset $J \subseteq I$ is an assignment $a\restriction_J$ which coincides with a on all the bits in J. If a and b are two assignments with disjoint domains I and J, then (a,b) is an assignment on $I \cup J$ which coincides with a on I, and with b on J. Given a boolean function $f(x_1, \ldots, x_n)$, every assignment $a : I \to \{0,1\}$ (treated for this purpose as a restriction) defines the subfunction f_a of f in $n - |I|$ variables which is obtained from f by setting x_i to $a(i)$ for all $i \in I$.

We need also to fix some (standard) notation concerning branching programs. A branching program (b.p.) is a directed acyclic graph with one source. The out-degree of each (non-sink) node is 2. Every node is labeled by an input variable x_i (or equivalently: by a bit i) and the two out-going edges are labelled by tests "$x_i = 0$" and "$x_i = 1$". The (two) sinks (out-degree 0 nodes) are labeled by 0 and 1. By the *size* of a branching program P we mean the number $|P|$ of its nodes. The *computation*

$comp(a)$ on an input $a \in \{0,1\}^n$ is a sequence of nodes of P which starts in the source of P and at each node v labelled by i, $comp(a)$ follows that edge going from v which corresponds to the test $x_i = a(i)$. If the computation $comp(a)$ goes through a node v, then we also say that the input a *reaches* this node. By $comp_v(a)$ we will denote the part of the computation $comp(a)$ starting from the node v. The program *computes* a Boolean function f if, for every input $a \in \{0,1\}^n$, the computation $comp(a)$ reaches a sink labelled by $f(a)$.

3 Windows

Let P be a branching program and v be a node in P. Let $F \subseteq \{0,1\}^n$ be an arbitrary subset of inputs, each of which reach the node v.

Definition 3.1 The *window* $w(a,v,F)$ of input $a \in F$ at the node v with respect to the set F is a string of length n in the alphabet $\{0,1,+,\#\}$ which is defined according to the following three rules (cf. Fig. 1). Let $F(a)$ be the set of those inputs $b \in F$ for which $comp_v(b) = comp_v(a)$.

1. We assign a simple-cross $(+)$ to the i-th bit of a if
   - either there is a $b \in F$ such that the first divergency of $comp_v(a)$ and $comp_v(b)$ is caused by a test on i (in this case we call that cross the *down-cross*),
   - or the bit i is not tested along any computation $comp(b)$ for $b \in F(a)$ (in this case we call that cross the *up-cross*).
2. We assign a *double-cross* $(\#)$ to the i-th bit of a if it was not crossed according to the first rule, and if $a(i) \neq b(i)$ for some input $b \in F(a)$.
3. The remaining bits of $w(a,v,F)$ are non-crossed (i.e. specified) and their values are the same as in a.

We have defined windows only at nodes but they can be easily extended to windows at edges as well. Let $e = (u,v)$ be an edge and $a \in F$ be an input, going through this edge. By a window $w(a,e,F)$ of a at the edge e with respect to F we mean the window $w(a,v,F_e)$ of a at v with respect to the set F_e of all those inputs from F, which go through the edge e. In this case we will also say that $w(a,v,F_e)$ is the window of a *immediately before* the node v (with respect to F).

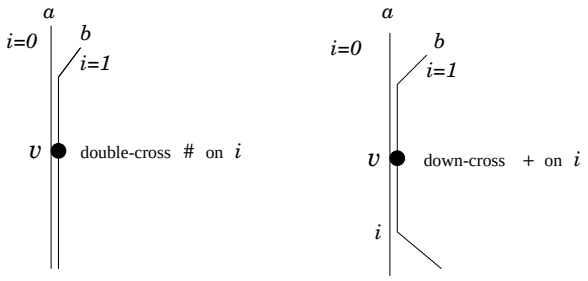

Figure 1: Schematic picture for two types of crosses.

Remark 3.2 If some bit is double-crossed (#) in a window of $a \in F$ at some node v with respect to F, then this bit is not tested and remains double-crossed in the windows of a at *each* subsequent node w of $comp(a)$ with respect to the set of all inputs from F reaching w. (Double-crossed bits are "forgotten forever".)

Remark 3.3 The larger is F the smaller is the number of non-crossed bits in the windows relative to F.

Remark 3.4 If $a, b \in F$ and $comp_v(a) = comp_v(b)$ then the windows of a and b (at v with respect to F) have the same sets of down-crosses, of up-crosses (+) and of double-crosses (#), and all non-crossed bits and down-crossed bits have the same contents in both a and b.

This observation implies that double-crossed bits may be used to 'cut-and-paste' computations in the following sense.

Proposition 3.5 *If* $a, b \in F$ *and* $comp_v(a) = comp_v(b)$ *then* $P(b\restriction_I, a\restriction_{\overline{I}}) = P(a)$ *where* $I \rightleftharpoons D(a) = D(b)$.

Proof. The fact that both a and b belong to F means, in particular, that the computations on these two inputs both reach the node v. Since after v these two computations do not diverge, we have that $P(a) = P(b)$. On the other hand, by Remark 3.4, we have that $comp(b\restriction_I, a\restriction_{\overline{I}}) = comp(b)$. Hence, $P(b\restriction_I, a\restriction_{\overline{I}}) = P(b) = P(a)$, as desired.■

4 General bounds for windows length

The number of crosses in the windows for inputs from $F \subseteq \{0,1\}^n$ measures the amount of uncertainty about these inputs when the corresponding computations meet in one node. The next theorem shows that the 'average uncertainty' is at least $\log_2 |F|$.

Theorem 4.1 *Let P be a branching program and v a node in it. Let $F \subseteq \{0,1\}^n$ be a set of inputs, each of which reaches the node v. For $a \in F$, let k_a be the number of bits which are crossed in the window of a at v with respect to F. Then*

$$\sum_{a \in F} 2^{-k_a} \leqslant 1 \tag{1}$$

and

$$\sum_{a \in F} k_a \geqslant |F| \cdot \log_2 |F|. \tag{2}$$

Proof. Our first goal is to establish a 1-1 connection between the inputs from F and branches in a particular binary tree. By a *binary tree* we mean a branching program, whose underlying graph is a tree. By a *branch* in such a tree we mean a path p from the root to a leaf; its *length* $|p|$ is the number of nodes in it minus 1 (i.e. the leaf is ignored).

Claim 4.2 *There is a binary tree $T = T_{v,F}$ with $|F|$ leaves, and there is a 1-1 correspondence $F \ni a \mapsto p_a$ between the inputs from F and the branches of T such that $|p_a| \leqslant k_a$ for all $a \in F$.*

Proof of Claim 4.2. Starting at the node v, we develop the program P into the tree rooted in v. In this tree we perform all computations starting from v which are given by the inputs from F. We delete from this tree all the nodes which are reachable by *no* of the inputs from F. After that we omit all non-branching nodes. Observe that for every input $a \in F$, the bits tested along the corresponding branch of the resulting tree T_1 are exactly the bits which are down-crossed by $(+)$ in $w(a, v, F)$. To capture the remaining crosses, we transform T_1 into a tree T_2, each leaf of which is reachable by only one input from F. At each leaf of T_1, which is reached by two or more inputs from F, we start a new subtree such that on each its branch there are tests on bits, which are up-crossed $(+)$, and then on bits which are double-crossed $(\#)$ in the windows of corresponding inputs at v. This way, the length of every branch in

T_2 is at most the total number of crossed bits in the windows of those inputs from F which follow this branch. Since, by Remark 3.4, non-crossed bits of inputs going to the same leaf of T_1, are the same and have the same value in all windows, each leaf of the transformed tree T_2 is reached by only one input from F, as desired. ■

To get the first inequality (1), we combine this claim with the well-known Kraft-McMillan inequality from Information Theory about the codeword length for prefix codes: if $C = \{c_1, \ldots, c_m\}$ are binary strings, no of which is a prefix of another, and l_i is the length of c_i, then $\sum_{i=1}^{m} 2^{-l_i} \leqslant 1$. Since the branches of $T = T_{v,F}$ clearly form a prefix code (each of them ends in the leaf) and are in 1-1 correspondence with the inputs from F, Kraft's Inequality immediately yields the desired estimate: $\sum_{a \in F} 2^{-k_a} \leqslant \sum_{p \in T} 2^{-|p|} \leqslant 1$.

To get the second inequality (2) (which was also derived in [21] using different argument), we relate the length of branches in a binary tree with the number of its leaves. For a binary tree T, let $|T|$ be the number of its leaves, and let $\lambda(T)$ be the total length of its branches, i.e. $\lambda(T) = \sum_p |p|$ over all branches p in T. By Claim 4.2, $\sum_{a \in F} k_a \geqslant \lambda(T)$, where $T = T_{v,F}$. Since $|F| = |T|$, inequality (2) follows directly from the following simple claim.

Claim 4.3 *For any binary tree T, $\lambda(T) \geqslant |T| \cdot \log |T|$.*

Proof of Claim 4.3. Induction on $|T|$. Basis ($|T| = 2$) is trivial. Take now a binary tree T with more than 2 leaves and let T_1 and T_2 be the subtrees of T, whose roots are immediate successors of the root of T. By induction hypothesis

$$\begin{aligned} \lambda(T) &= (\lambda(T_1) + |T_1|) + (\lambda(T_2) + |T_2|) \\ &\geqslant |T_1| \cdot \log |T_1| + |T_2| \cdot \log |T_2| + |T| . \end{aligned}$$

For $x > 0$ the function $f(x) = x \log x$ is convex (its second derivative is $f''(x) = (\log_2 e)/x > 0$). For convex functions Jensen's Inequality says that $f(\sum_i \alpha_i x_i) \leqslant \sum_i \alpha_i f(x_i)$ as long as $\sum_i \alpha_i = 1$ and $0 \leqslant \alpha_i \leqslant 1$. Applying this inequality to the previous estimate with $x_1 = |T_1|$, $x_2 = |T_2|$ and $\alpha_1 = \alpha_2 = 1/2$, we get

$$\begin{aligned} \lambda(T) &\geqslant (|T_1| + |T_2|) \cdot \log \frac{|T_1| + |T_2|}{2} + |T| \\ &= |T| \cdot \log \frac{|T|}{2} + |T| = |T| \log |T| , \end{aligned}$$

as desired.■

The *length* of a window is the number of non-crossed bits in it. Theorem 4.1 can be used to estimate the 'average length' of windows in terms of program size.

Let P be a branching program, V is the set of its nodes and $A \subseteq \{0,1\}^n$ be a set of inputs. A *distribution* of A (among the nodes of P) is a mapping $\varphi : A \to V$ which sends each input $a \in A$ to some node of the computation $comp(a)$. Given such a distribution, the *average length* of windows (of inputs from A) is the sum

$$H(A,\varphi) \rightleftharpoons \frac{1}{|A|}\sum_{a\in A} \ell_a,$$

where ℓ_a is the length of the window $w(a,v,F)$ of a at the node $v = \varphi(a)$ with respect to the set $F \rightleftharpoons \{b \in A : \varphi(b) = v\}$ of all those inputs, which are mapped to the same node; we call this set F the *class* of distribution at v. We can also distribute the inputs from A among the edges of P. In this case the average length of windows is defined in the same way with ℓ_a being the length of the window of a at the corresponding edge.

Theorem 4.4 *Let P be a branching program, $A \subseteq \{0,1\}^n$ a set of inputs and φ be any distribution of these inputs among the nodes of P. Then*

$$H(A,\varphi) \leqslant \log|P| + n - \log|A|.$$

If φ distributes the inputs from A among the edges of P then the same upper bound holds with $|P|$ replaced by $|E|$ where $E = \varphi^{-1}(A)$ is the set of edges to which at least one input is sent.

Proof. Let $v_1,\ldots,v_r$ be the nodes to which A is mapped by φ, and let F_j be the set of those inputs from A which are mapped to v_j. The sets $F_1,\ldots,F_r$ form a partition of A. For every $a \in A$, $n-\ell_a$ is the number of crossed bits in the window $w(a,v,F_j)$ of a at the node v_j with respect to the set F_j containing a. Thus, inequality (2) implies that $\sum_{a\in F_j}\ell_a \leqslant |F_j|(n-\log|F_j|)$ for every $j = 1,\ldots,r$. Hence,

$$\begin{aligned} H(A,\varphi) &= \frac{1}{|A|}\sum_{j=1}^{r}\sum_{a\in F_j}\ell_a \leqslant \frac{1}{|A|}\sum_{j=1}^{r}|F_j|(n-\log|F_j|) \\ &= n - \sum_{j=1}^{r}\frac{|F_j|}{|A|}\log\frac{|F_j|}{|A|} - \log|A| \leqslant n + \log r - \log|A|. \end{aligned}$$

The last inequality here follows from the fact that, for $p_j \rightleftharpoons |F_j|/|A|$, the sum $-\sum_{j=1}^{r} p_j \log p_j$ is exactly the entropy of the partition of A into r blocks, and hence,

does not exceed $\log r$, with the equality when blocks are of equal size. Since $|P| \geqslant r$, we are done.■

In the proof above we use the fact that the sets $F_1, \ldots, F_r$ are mutually disjoint, so that we have a natural connection to the entropy function. Notice however, that disjointness is not severe requirement. Although in this paper we will use only Theorem 4.4, let us mention its modification for the case of non-disjoint sets ([21]).

Theorem 4.5 *Let P be a branching program, and $v_1, \ldots, v_r$ be some of its nodes. Let F_j be a set of some inputs reaching the node v_j, $j = 1, \ldots, r$, and let*

$$H(F_1, \ldots, F_r) \rightleftharpoons \frac{1}{N} \sum_{j=1}^{r} \sum_{a \in F_j} \ell_{a,j}$$

where $N \rightleftharpoons \sum_{j=1}^{r} |F_j|$ and $\ell_{a,j}$ is the length of the window $w(a, v_j, F_j)$. Then

$$H(F_1, \ldots, F_r) \leqslant \log_2 |P| + n - \log_2 N.$$

Proof. We have seen (cf. the proof of Claim 4.3) that the function $f(x) \rightleftharpoons x \log x$ is convex. So, by Jensen's inequality, $f\left(\sum_j |F_j|/r\right) \leqslant \left(\sum_j f(|F_j|)\right)/r$. Hence, $(N/r) \cdot \log(N/r) \leqslant \sum (|F_j| \log |F_j|)/r$ and therefore, $\log N - \log |P| \leqslant \log N - \log r = \log(N/r) \leqslant \left(\sum |F_j| \cdot \log |F_j|\right)/N \leqslant n - H(F_1, \ldots, F_r)$ where the last inequality follows from the inequality (2).■

Theorems 4.4 and 4.5 suggest the following general frame to obtain lower bound on the size of P in terms of windows: if it is possible to distribute some large set of inputs $A \subseteq \{0,1\}^n$ among some nodes of P so that the average window-length is $\geqslant h$, then the program P must have size exponential in $\log |A| - n + h$.

In general, bounding the (average) window length is a hard task. On the other hand, for *read-once* branching programs (1-b.p.) this can be done easily. A Boolean function f is *m-mixed* if for any subset I of m bits and any two different assignments $a, b : I \to \{0,1\}$ we have that $f_a \neq f_b$; here, as usually, f_a denotes the subfunction of f obtained by setting the variables x_i with $i \in I$, to $a(i)$. It is well known (see, e.g. [4]) that any such function requires 1-b.p. of size 2^m. Most of exponential lower bounds for 1-b.p. were obtained using this criterion. Let us show how this result can be derived using the proposed frame in terms of windows.

Proposition 4.6 *If f is m-mixed then any read-once branching program for it has size at least 2^m.*

Proof. Define the distribution φ of all inputs from $A \rightleftharpoons \{0,1\}^n$ among the nodes of P by sending each input a to the $(m+1)$-st node $v = \varphi(a)$ of the computation $comp(a)$. Let $I(a)$ be the set of bits tested along the computation $comp(a)$ until the node v; hence $|I(a)| = m$.

Claim 4.7 *For every $a \in A$, no of the bits from $I(a)$ is crossed in the window of a at v with respect to the set $F = \varphi^{-1}(v)$.*

To prove the claim, assume the opposite that some bit $i \in I \rightleftharpoons I(a)$ is crossed. Since i was tested before v, this cross cannot be an up-cross; since P is read-once, the bit i is not tested after v, and hence, this cross cannot be a down-cross. So, bit i is double-crossed, which means that some other input b such that $b(i) \neq a(i)$, also reaches the node v. Since P computes an m-mixed function, there must be an assignment $c : \overline{I} \to \{0,1\}$ such that $P(a\restriction_I, c) \neq P(b\restriction_I, c)$. But this is impossible because (due to read-once condition), no bit from I is tested along the computation $comp(c)$ after the node v, and hence, the computations on both these two inputs reach the same sink. ■

By the claim, $H(A,\varphi) \geqslant m$, which, together with Theorem 4.4, implies that $|P| \geqslant 2^{H(A,\varphi)-n+\log|A|} \geqslant 2^m$, as desired.■

5 Gentle programs

We have seen that for 1-b.p., bounding the (average) window length is an easy task. In this section we describe one more general situation where it becomes tractable. This situation requires some additional knowledge about the *form* of windows.

Let P be a branching program and v be a node in P. Throughout this section, let $F \subseteq \{0,1\}^n$ be an arbitrary (but fixed) set of inputs which reach the node v, i.e. the computations on inputs from F go through the node v; in this case we say also that F is *classified at* v. We will always assume that the set F is *closed* in the following natural sense: $a \in F$, $b \in \{0,1\}^n$ and $comp(b) = comp(a)$ imply $b \in F$.

Let a be an input from F. Depending on what is the window $w(a,v,F)$ for this input a at the node v with respect to F, we define the following subsets of $\{1,\ldots,n\}$.

$N(a) \rightleftharpoons$ the set of all non-crossed bits;

$D(a) \rightleftharpoons$ the set of all double-crossed (#) bits;

$S(a)$ $\rightleftharpoons$ the set of those bits $i \in D(a)$, which were non-crossed in the window for a *immediately before* the node v. i.e. which were non-crossed in the window for a at the corresponding edge, feeding into v.

Let also

N $\rightleftharpoons$ the set of all bits which are non-crossed and have the same value in the windows at v of *all* inputs from F (the *common specified part* of F), and

D $\rightleftharpoons$ the set of all bits which are double-crossed in the windows at v of *all* inputs from F (the *core* of F)

Definition 5.1 We say that F is classified at v *in a regular manner* with *fluctuation* γ and *deviation* δ if its core $D \neq \emptyset$ and, for every input $a \in F$, $|N(a) \setminus N| \leqslant \gamma$ and $max\{|D(a) \setminus D|, |D(a) \setminus S(a)|\} \leqslant \delta$.

The fluctuation tells that the "mixed" non-crossed part of $N(a)$ has at most γ bits, whereas the deviation ensures that at least $|D(a)| - \delta$ bits of a where double-crossed at the node v for the first time.

Definition 5.2 A branching program P is *gentle* on a set of inputs $A \subseteq \{0,1\}^n$ with fluctuation γ and deviation δ if there is a distribution $\varphi : A \to V$ of these inputs among the nodes of P such that each (non-empty) class $F = \{a \in A : \varphi(a) = v\}$ of this distribution is classified at the corresponding node v in a regular manner with the fluctuation γ and deviation δ. We also say that a program is α-*gentle* if it is such on some set of at least $2^{n-\alpha}$ inputs.

Parameters α, γ and δ range between 0 and n, and reflect the 'degree of gentleness': the smaller they are the more gentle the program is. In the next section we will show that read-once branching programs (1-b.p.) are very gentle: for them $\alpha \leqslant 1$ and $\gamma = \delta = 0$.

6 Read-once programs are gentle

Recall that a branching program is *read-once* (1-b.p.) if along every path every bit is tested at most once. Let $I(p)$ be the set of bits that are tested along some path p. A 1-b.p. is *uniform* if: (i) for a path p beginning at the source, the set $I(p)$ depends

only on the terminal node v of p (accordingly, we denote it by $I(v)$), and (ii) for every sink v, $I(v)$ contains all variables. As observed in [11], the uniformity is actually not a serious restriction. Namely, by adding some "dummy tests" (i.e. tests where both out-going edges go to the same node), every 1-b.p. can be made uniform; the size increases by at most a factor of n.

Theorem 6.1 *Let P be a uniform read-once b.p. Then, for every set $A \subseteq \{0,1\}^n$, $|A| \geqslant 3$, the program P is gentle on all but two inputs from A with deviation $\delta = 0$ and fluctuation $\gamma = 0$. In particular, P is gentle with $\alpha \leqslant 1$.*

Proof. Let V be the set of nodes of P. Define the distribution $\varphi : A \to V$ inductively as follows: $\varphi(a) = v$ if v is the first node along the computation $comp(a)$ at which $comp(a)$ meets another computation $comp(b)$ on some input $b \in A \setminus \{a\}$ which follows a different path from the source to v and which is still not mapped (by φ to a node before v).

Since P is uniform, each of the (two) sinks can be reached by at most one input which is not mapped to no of the nodes along its computation (including the sink). Hence the number of mapped inputs is at least $|A| - 2$.

We want to prove that each class of the distribution φ is classified at the corresponding node in a regular manner with the fluctuation 0 and deviation 0.

Let F be a class of the distribution at a node v. We are going to describe the window on each input from F (with respect to F at v). At first we see that there are no up-crosses since P is uniform. Let I be the set of bits tested along at least one computation $comp(a)$, $a \in F$, on the path from the source to v. All bits outside I are down-crossed (in windows of all inputs from F) since P is uniform. No bit from I is tested at v and below v since P is read-once. Hence the bits in I can be only double-crossed or non-crossed.

Let us define $D \rightleftharpoons \{i \in I : \exists a, b \in F\ a(i) \neq b(i)\}$. By the definition of F, $D \neq \emptyset$. It is also clear that for any input from F the bits in $I \setminus D$ are non-crossed. We want to prove that for each input $a \in F$, D is the set of its double-crossed bits. For any $i \in D$ there must be a $b \in F$ such that $a(i) \neq b(i)$. Let us consider the combined input $c = (b\restriction_I, a\restriction_{\overline{I}})$. This input follows b from the source to v, hence is in F, too. After v, it follows the computation on a till the sink. Hence a has a double-cross on i. This shows that D is the (nonempty) core of F and that the fluctuation of F is 0 (since all inputs from F have the same set of noncrossed bits).

It remains to verify that the deviation of F is 0, i.e. that $S(a) = D(a)$ for all $a \in F$. This follows directly from the fact that the window on each $a \in F$ immediately before v has no double-crosses, since otherwise the computation on a would have to be met before v by the computation on some other input and therefore a would be distributed before v.■

7 Functions with small gentle programs

For a branching program to be gentle it is sufficient that it has some 'gentle enough' fragment – a node (or a set of nodes) at which some large set of inputs is classified in a regular enough manner. Assume that the function f can be represented in the form $f = g \wedge h$ (or $f = g \vee h$) so that h has a (unrestricted!) b.p. of size t whereas the first component g has a b.p. of size s which is gentle on some subset A of $g^{-1}(0)$ (resp., of $g^{-1}(1)$). Then, by connecting the 1-sink (resp., 0-sink) of the (gentle) b.p. for g to the source of the b.p. for h, we obtain a b.p. for the original function f which is α-gentle for $\alpha \leqslant n - \log|A|$, and has size $s + t$. Thus, to design the desired gentle b.p. for f, it is enough, by Theorem 6.1, that its first component g has small uniform 1-b.p. and the set $g^{-1}(0)$ (or $g^{-1}(1)$) is large enough.

These simple observations allows one to design small gentle b.p.'s for a lot of known functions. In particular, it is easy to show (see Proposition 9.1 in the appendix) that, if a function f has at least one minterm or maxterm of length l, then f can be computed by an $(l+1)$-gentle b.p. of size $2l + 1 + t$ where t is the unrestricted b.p. size of f.

Here we restrict ourselves by one particularly important class of Boolean function – characteristic functions of linear codes (see Appendix for more examples). Let $C \subseteq \{0,1\}^n$ be a linear code (i.e. a linear subspace of $GF[2]^n$), and let $f_C(x)$ be its characteristic function, i.e. $f_C(x) = 1$ iff $x \in C$. It is known that for some explicit linear codes $C \subseteq \{0,1\}^n$, their characteristic functions f_C require syntactic k-b.p. ([11]), syntactic k-n.b.p. ([6]) and $(1, +s)$-b.p. ([7]) of super-polynomial size, as long as $k = o(\log n)$ or $s = o(n/\log n)$. Thus, these functions are hard for *all* restricted models of branching programs, considered so far.

Proposition 7.1 *For every linear code $C \subseteq \{0,1\}^n$, the function $f_C(x)$ has an α-gentle branching program of size $O(n^2)$ with $\alpha \leqslant 2$ and $\gamma = \delta = 0$.*

Proof. Let $R \subseteq \{0,1\}^n$ be the set of rows in the parity-check matrix of C; hence $x \in C$ iff $\langle x, r\rangle = 0$ for all $r \in R$. Fix a row $r \in R$, and let $g \rightleftharpoons \langle x, r\rangle \oplus 1$. Since the scalar product $\langle x, r\rangle$ is just a parity function, it has a (standard) uniform 1-b.p. of linear size. Since $|g^{-1}(0)| = 2^{n-1}$, Theorem 6.1 implies that this program is α-gentle with $\alpha \leqslant 2$ and $\gamma = \delta = 0$. Since $f_C = g \wedge f_C$ and f_C has an obvious unrestricted b.p. of size $O(n^2)$, the combined program computes f_C and is also gentle, as desired.■

8 Stable functions are hard

What functions are hard for gentle programs? We have seen that functions, which were hard for previous restricted models of b.p., can be easily computed in a gentle manner. This is not surprising because for gentleness the presence of any 'gentle enough' singularity is sufficient. This fact just means that gentleness is a new property of b.p., and that combinatorial properties of Boolean functions, which make them hard for known restricted models of branching programs – like 'mixness' for 1-b.p. [4], 'degree' for 1-n.b.p. [4, 7], or 'density' and 'rareness' for (1,+s)-b.p. [7] and syntactic k-n.b.p. [6] – do not work for gentle b.p.

Mixness is quite universal property: it is known that almost all Boolean functions in n variables are m-mixed for $m = n - (1+\varepsilon)\log n$. Besides mixness, a stronger property of 'stability' was introduced in [3, 4]. A function f is *m-stable* if for any subset I of m bits and for any bit $i \in I$ there is an assignment $c : \overline{I} \to \{0,1\}$ on the remaining bits and a constant $\varepsilon \in \{0,1\}$ such that $f_c \equiv x_i \oplus \varepsilon$, i.e. the subfunction f_c depends only on the i-th variable (and does not depend on variables x_j, $j \in I \setminus \{i\}$). Every m-stable function is also m-mixed, and hence, requires 1-b.p. of size 2^m.

8.1 The general lower bound

In this section we prove that similar result holds also in the case of gentle branching programs with a somewhat stronger stability condition. Namely, we additionally require that the condition $f_c \equiv x_i \oplus \varepsilon$ cannot be destroyed by toggling some small number of bits of c.

The *Hamming distance* $\mathrm{dist}(x, y)$ between two assignments $x : I \to \{0,1\}$ and $y : J \to \{0,1\}$ is the number of bits $i \in I \cap J$ for which $x(i) \neq y(i)$.

Definition 8.1 Say that f is *strongly (m, d)-stable* if for any subset of bits I with

$|I| \leqslant m$ and any bit $i \in I$ there is an assignment $c : \overline{I} \to \{0,1\}$ and a constant $\varepsilon \in \{0,1\}$ such that $f_{c'} \equiv x_i \oplus \varepsilon$ for any assignment $c' : \overline{I} \to \{0,1\}$ of Hamming distance at most d from c.

Theorem 8.2 *If f is strongly (m,d)-stable for some $d \geqslant \gamma + \delta$, then any α-gentle branching program P computing f with fluctuation γ and deviation δ, has size larger than $2^{m-\alpha-\delta-1}$.*

Proof. Since P is α-gentle, there is a set of inputs $A \subseteq \{0,1\}^n$ of cardinality $|A| \geqslant 2^{n-\alpha}$ and a distribution $\varphi : A \to \{v_1, \ldots, v_r\}$ of these inputs among some nodes $v_1, \ldots, v_r$ of P such that every class $F_j \rightleftharpoons \{a \in A : \varphi(a) = v_j\}$ of this distribution is classified at the corresponding node v_j in a regular manner with fluctuation γ and deviation δ.

Let us first consider one of these classes $F \in \{F_1, \ldots, F_r\}$, and let $v \in \{v_1, \ldots, v_r\}$ be the corresponding node, at which this class is classified. Let also $N = N_F$ be the common specified part of F.

Claim 8.3 *For every input $c \in \{0,1\}^n$ there is an input $w \in F$ which outside the set $D(w) \cup N$ differs from c in at most γ bits.*

Proof. We construct the desired input w as follows. Starting at the node v, we develop the program into the tree rooted in v. In this tree we perform all computations starting from v which are given by the inputs from F. We delete from this tree all the nodes (together with corresponding subtrees) which are reachable by *no* of the inputs from F. Let T be the resulting tree. One branch p_c of this tree is consistent with c if we take into account only the tests made at outdegree-2 nodes (the branching nodes of T). Let $L \subseteq F$ be the set of all inputs from F which follow p_c. By Remark 3.4, these inputs have the same sets of double-crossed bits, of up-crossed bits, of down-crossed bits and of non-crossed bits. On down-crossed bits they have the same value as c has. Since up-crossed bits are free bits of these inputs and since F is closed, there is an input $w \in L$ which equals c also on up-crossed bits. Therefore, the inputs w and c may differ only on bits, which were either double-crossed or non-crossed (in the window of w at v with respect to F). Hence, outside the set $D(w) \cup N$, these inputs can differ in at most $|N(w) \setminus N| \leqslant \gamma$ bits.■

Let E be the set of edges entering the node v, and consider a new distribution $\psi : F \to E$ which sends every input $a \in F$ to the edge $\psi(a)$ which the input a

goes through before it comes to v. Let $S(a)$ stand for the set of those bits in $D(a)$, which were non-crossed in the window of a immediately before the node v, i.e. in the window of a at the incoming edge $e = \psi(a)$. Let as before, $N = N_F$ be the common specified part of F. Since $S(a) \cap N = \emptyset$, $|S(a)| + |N|$ does not exceed the total length ℓ_a of the window of a at the corresponding edge e. This gives a lower bound $H(F,\psi) = \frac{1}{|F|}\sum_{a\in F} \ell_a \geqslant |N| + \frac{1}{|F|}\sum_{a\in F} |S(a)|$ on the average window length of inputs from F at edges feeding into the node v. We will use this to prove the following lower bound.

Claim 8.4 $H(F,\psi) > m - \delta$.

Proof. In fact, we will prove a stronger fact that *all* the windows are long enough, namely that $|S(a)| > m-\delta-|N|$, for every input $a \in F$. By the previous observation, this immediately gives the desired lower bound on $H(F,\psi)$.

Assume the opposite and take an input $a \in F$ for which $|S(a)| \leqslant m - \delta - |N|$. Consider the set of bits $I \rightleftharpoons D \cup N$ where $D = \bigcap_{a\in F} D(a)$ is the core of F. Since $S(a) \subseteq D(a)$ and $|D(a) \setminus S(a)|$ cannot exceed the deviation δ, we have that $|D(a)| \leqslant |S(a)| + \delta$. Hence, $|I| \leqslant |D(a)| + |N| \leqslant |S(a)| + \delta + |N| \leqslant m$. Since F is classified at v in a regular manner, its core D is non-empty. Take an arbitrary bit $i \in D$. Since $|I| \leqslant m$ and our function f is strongly (m,d)-stable, there must be an input $c \in \{0,1\}^n$ and a constant $\varepsilon \in \{0,1\}$ such that

$$f(x,c') = x(i) \oplus \varepsilon \tag{3}$$

for any assignment $x : I \to \{0,1\}$ and any assignment $c' : \overline{I} \to \{0,1\}$ such that $\mathrm{dist}(c', c\restriction_{\overline{I}}) \leqslant d$. According to Claim 8.3 there is an input $w \in F$, which outside the set $J \rightleftharpoons D(w) \cup N$ differs from c in at most γ bits. On the other hand, since the bit i belongs to the core D, it was double-crossed also in the window for the input w. Hence, there must be an input $b \in F$ such that $b(i) \neq w(i)$ and $comp_v(b) = comp_v(w)$. By Proposition 3.5, the program P must output the same value on both inputs w and $(b\restriction_J, w\restriction_{\overline{J}})$ (because inputs w and b coincide on N). But outside the set $I = D \cup N$ both these inputs differ from c in at most $\mathrm{dist}(w\restriction_{\overline{J}}, c\restriction_{\overline{J}}) + |D(w) \setminus D| \leqslant \gamma + \delta \leqslant d$ bits. This gives the desired contradiction because then, taking $c' \rightleftharpoons w\restriction_{\overline{I}}$ and setting $x = w\restriction_I$ and $x = b\restriction_I$ in (3), we have that $f(w) = w(i) \oplus \varepsilon \neq b(i) \oplus \varepsilon = f(b\restriction_I, w\restriction_{\overline{I}})$.■

Using this claim we complete the proof of the theorem as follows. Let $E_1, \ldots, E_r$ be the sets of edges, feeding into the nodes $v_1, \ldots, v_r$, and let $F_1, \ldots, F_r$ be the

corresponding classes, distributed (by φ) to these nodes. Theorem 4.4 together with Claim 8.4 implies that $\log|E_j| > m - \delta - n + \log|F_j|$ for every $j = 1, \ldots, r$. Since the sets E_j of edges are mutually disjoint and $\sum_{j=1}^{r}|F_j| = |A| \geqslant 2^{n-\alpha}$, the desired lower bound on the total number $|P|$ of nodes in P follows: $2|P| \geqslant \sum_{j=1}^{r}|E_j| > 2^{m-\delta-n}\sum_{j=1}^{r}|F_j| \geqslant 2^{m-\alpha-\delta}$.■

8.2 Explicit Stable Functions

The *clique function* $\mathrm{Clique}_{n,k}$ has $\binom{n}{2}$ Boolean variables, encoding the edges of an n-vertex graph, and outputs 1 iff this graph contains at least one complete subgraph on k vertices.

Proposition 8.5 $\mathrm{Clique}_{n,k}$ *is strongly* (m,d)*-stable for any* m *and* d *such that* $d \leqslant k-2$, $m+d \leqslant \binom{k}{2}$ *and* $2m + (d+1)(k-2) \leqslant n$.

Proof. To show the desired stability, take a set I of m edges and an edge $e = (u,v) \in I$. Our goal is to find a set of edges $C \subseteq \overline{I}$ such that, for any a set of edges $J \subseteq \overline{I}$, $|J| \leqslant d$, the set $(C \setminus J) \cup \{e\}$ has a k-clique but $(C \cup J) \cup (I \setminus \{e\})$ has no such clique.

Take $d+1$ mutually disjoint $(k-2)$-cliques $U_1, \ldots, U_{d+1}$, and join them with both ends of the edge $e = (u,v)$. Condition $n - 2m \geqslant (d+1)(k-2)$ ensures that we can choose these cliques so that, for every $1 \leqslant i \leqslant d+1$, no vertex of U_i is incident with an edge of I. Let $C \rightleftharpoons Q_1 \cup Q_2 \cup \cdots \cup Q_{d+1}$ where Q_i is the set of all edges, connecting the vertices of U_i with each other and with both ends u and v of the edge e. Note that each of the graphs Q_i is "almost k-clique": it lacks only the edge e. We claim that the graph C has the desired properties.

We have to verify that (i) deleting the edges J we cannot destroy all the k-cliques $Q_1 \cup \{e\}, \ldots, Q_{d+1} \cup \{e\}$, and (ii) adding the edges of J to the set $C \cup (I \setminus \{e\})$ we do not get a k-clique. The first item (i) holds because $e \notin J$ and, due to the condition $|J| \leqslant d$, some of the cliques $Q_i \cup \{e\}$ has no edge in J. For the second item (ii), recall that $e \notin J$. So, the graph $(I \setminus \{e\}) \cup J$ cannot have a k-clique because there are to few edges: $|(I \setminus \{e\}) \cup J| \leqslant |I \cup J| - 1 \leqslant m + d - 1 < \binom{k}{2}$. Hence, the only possibility to get a k-clique in $(I \setminus \{e\}) \cup J \cup C$ is to take some vertex $w \notin \{u,v\}$ and connect it (using the edges of J) with one of the ends u or v of the edge e and with all the vertices in some $(k-2)$-clique U_i. This requires at least $k-1$ new edges. (The

alternative would be to take two different vertices w_1 and w_2 and connect them with some of U_i; this would require $2k-3$ new edges). But we do not have enough edges in J to do this because $|J| \leqslant k-2$.■

Corollary 8.6 *For $k \leqslant \sqrt{n/2}$ and $\alpha \leqslant k^2/4$, any α-gentle program computing* $\text{Clique}_{n,k}$ *with parameters $\gamma+\delta \leqslant k-2$, has size $2^{\Omega(k^2)}$. For maximal k, the bound is $2^{\Omega(n)}$ with $\alpha = \Omega(n)$ and $\gamma+\delta = \Omega(\sqrt{n})$.*

Proof. If $k \leqslant \sqrt{n/2}$ then, by Proposition 8.5, $\text{Clique}_{n,k}$ is (m,d)-stable for $m = \binom{k}{2} - k + 2$ and $d = k-2$, and the desired lower bound follows directly from Theorem 8.2.■

The Clique function is NP-complete. Bellow we describe an explicit strongly stable function which belongs to AC^0.

Let s and k be such that $ks^2 = n$ and $k \geqslant \log n$. Arrange the n variables $X = \{x_1, \ldots, x_n\}$ into a $k \times s^2$ matrix; split the i-th row ($1 \leqslant i \leqslant k$) into s blocks $B_{i1}, B_{i2}, \ldots, B_{is}$ of size s each, and let ε_i be the OR of ANDs of variables in these blocks. The *pointer function* $\pi(X)$ is defined by: $\pi(X) = x_j$ where j is the number (between 1 and n), whose binary code is $(\varepsilon_1, \ldots, \varepsilon_k)$.

Proposition 8.7 *The pointer function $\pi(X)$ is strongly (m,d)-stable for any m and d such that $m+d \leqslant s-1$.*

Proof. Take a set of bits I with $|I| \leqslant m$ and a bit $i_0 \in I$. Let $(\varepsilon_1, \ldots, \varepsilon_k)$ be the binary code of i_0. Define an assignment $c : \overline{I} \to \{0,1\}$ as follows: for $j \notin I$, set $c(j) = \varepsilon_t$ where t is the number of a row containing the variable x_j.

Now, take an arbitrary set of bits $J \subseteq \overline{I}$ with $|J| \leqslant d$. Since $|I \cup J|$ is strictly less than s, we have that: (i) in every row at least one block is disjoint from $I \cup J$, and (ii) each block contains at least one bit outside the set by $I \cup J$. So, independent of actual values of bits in $I \cup J$, the values of $\varepsilon_1, \ldots, \varepsilon_k$ remain the same. Thus, for any two assigments $a, b : I \to \{0,1\}$ and any assignment $c' : \overline{I} \to \{0,1\}$, with $\text{dist}(c', c) = |J| \leqslant d$, both inputs (a, c') and (b, c') point to the same variable x_{i_0}. Hence $\pi(X)$ is stronly (m,d)-stable.■

Corollary 8.8 *For $s = \lceil (n/\log n)^{1/2} \rceil$ any gentle program computing the function $\pi(X)$ with parameters $\alpha+\gamma+\delta \leqslant n^{1/2-\varepsilon}$, has size* $\exp\left(\Omega(n/\log n)^{1/2}\right)$.

References

[1] F. Ablayev, M. Karpinski, On the Power of Randomized Branching Programs. In: *Proc. of ICALP'96*, Lect. Notes in Comput. Sci., **1099** (Springer, 1996), 348–356.

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

[3] P. E. Dunne, Lower bounds on the complexity of one-time-only branching programs. In: *Proc. of FCT'85*, Lect. Notes in Comput. Sci., **199** (Springer, 1985), 90–99.

[4] S. Jukna, Lower bounds on the complexity of local circuits. In: *Proc. of MFCS'86*, Lect. Notes in Comput. Sci., **233** (Springer, 1986), 440–448. [Journal version: Entropy of contact circuits and lower bounds on their complexity, *Theoret. Comput. Sci.*, **57** (1988), 113–129.]

[5] S. Jukna, The effect of null-chains on the complexity of contact schemes. In: *Proc. of FCT'89*, Lect. Notes in Comput. Sci., **380** (Springer, 1989), 246-256 .

[6] S. Jukna, A note on read-k-times branching programs, *RAIRO Theoretical Informatics and Applications*, **29**:1 (1995), pp. 75-83.

[7] S. Jukna, A. A. Razborov, Neither Reading Few Bits Twice nor Reading Illegally Helps Much, ECCC Report Nr. 37, 1996 (to appear in *Discrete Appl. Math.*).

[8] M. Krause, C. Meinel, and S. Waack, Separating the eraser Turing machine classes $L_e, NL_e, co-NL_e$ and P_e, *Theoret. Comput. Sci.*, **86** (1991), 267–275.

[9] W. Masek, *A fast algorithm for the string editing problem and decision graph complexity.* Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1976.

[10] E. I. Nečiporuk, On a Boolean function, *Soviet Mathematics Doklady* 7:4 (1966), 999–1000.

[11] E. A. Okolnishnikova, Lower bounds for branching programs computing characteristic functions of binary codes. In: *Metody discretnogo analiza*, **51** (1991), 61–83 (in Russian).

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

[13] E. A. Okolnishnikovova, On the Hierarchy of Nondeterministic Branching k-Programs. In: *Proc. of FCT'97*, Lect. Notes in Comput. Sci., (Springer, 1997).

[14] A. A. Razborov, Lower bounds for deterministic and nondeterministic branching programs. In: *Proc. of FCT'91*, Lect. Notes in Comput. Sci., **529** (Springer, 1991), 47–60.

[15] P. Savický and S. Žák, A lower bound on branching programs reading some bits twice, *Theoret. Comput. Sci.*, **172** (1997), 293–301.

[16] G. C. Tseitin, On the complexity of derivations in propositional calculus. In: *Studies in Mathematics and Mathematical Logic, Part II,* ed. A. O. Slisenko (1968), 115–125.

[17] I. Wegener, *The complexity of Boolean functions*, Wiley-Teubner, 1987.

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

[19] S. Žák, An exponential lower bound for one-time-only branching programs. In: *Proc. of MFCS'84*, Lect. Notes in Comput. Sci., **176** (1984), 562–566.

[20] S. Žák, A superpolynomial lower bound for $(1, +k(n))$-branching programs. In: *Proc. of MFCS'95*, Lect. Notes in Comput. Sci., **969** (Springer, 1995), 319–325.

[21] S. Žák, A subexponential lower bound for branching programs restricted with regard to some semantic aspects, ECCC Report Nr. 50, 1997.

9 Appendix: More upper bounds

A 1-*term* (0-*term*) of a Boolean function f is a (partial) assignment $a : I \to \{0,1\}$ for which $f\restriction_a \equiv 1$ ($f\restriction_a \equiv 0$, respectively). A *minterm* (*maxterm*) of f is its 1-term (0-term) a which is minimal in the sense that unspecifying every single value $a(i) \in \{0,1\}$ already violates this property.

Proposition 9.1 *Let f be a Boolean function whose (unrestricted) branching program P has size s. Let $a : I \to \{0,1\}$ and $b : J \to \{0,1\}$ be 0-terms (or 1-terms) of f such that there is an $i \in I \cap J$ with $a(i) \neq b(i)$. Then f can be computed by an α-gentle b.p. of size at most $s + |I \cup J|$ with parameters $\alpha \leqslant |I \cup J|$, $\delta \leqslant \max\{|I \setminus J|, |J \setminus I|\}$ and $\gamma = 0$.*

Proof. By the observation, made in Section 7, it is enough to design a gentle b.p. P' such that sticking one sink of P' with the source of P we still obtain a branching program computing f. Assume w.l.o.g. that a, b are 0-terms of f (the case of 1-terms is dual). Let F be the set of all inputs from $\{0,1\}^n$ which are consistent with at least one of the 0-terms a and b, and let P' be the 1-b.p. computing the Or of these two 0-terms. It is clear that P' computes f on F. Moreover, the whole program P' consists of two paths which split at the source on the test on i and then meet only at the 0-leaf v. Since these paths meet at v for the first time, for all $f \in F$ at v the double-crosses (#) appear on the same set of bits $D = (I \cup J) \setminus \{j \in I \cap J : a(j) = b(j)\}$, and each of these bits was either non-crossed or up-crossed (+) immediately before v; moreover, at each of the two immediate predecessors of v, at most $\max\{|I \setminus J|, |J \setminus I|\}$ of the bits from D could be up-crossed. Hence, at v F is a regular set with fluctuation $\gamma = 0$ and deviation $\delta \leqslant \max\{|I \setminus J|, |J \setminus I|\}$. Since $|F| \geqslant 2^{n-|I \cup J|}$, the whole program is α-gentle with $\alpha \leqslant |I \cup J|$, and has size $s + |P'| \leqslant s + |I \cup J|$.■

9.1 Exact-Perfect-Matching

The *exact-perfect-matching* function is a Boolean function EPM_n in $n = m^2$ variables, which, given an $m \times m$ matrix $X = (x_{ij})$ with entries in $\{0,1\}$, computes 1 iff X is a permutation matrix, i.e. if there is a permutation σ of $\{1, \ldots, m\}$ such that $x_{i,j} = 1$ iff $\sigma(i) = j$. This function is in AC^0 but is known to require (even semantic) 1-n.b.p. of size $2^{\Omega(\sqrt{n})}$ ([5, 8, 7]).

Proposition 9.2 *The function EPM_n has a 4-gentle branching program of size $O(n^{3/2})$ with fluctuation $\gamma = 0$ and deviation $\delta = 0$.*

Proof. The function $EPM_n(X)$ has a trivial unrestricted b.p. of size $O(n^{3/2})$ (test if every row has at least one 1, and then test if every column has at least $n - 1$

zeroes). So, it is enough to take two maxterms $\begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}$ and $\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$ on the first four variables $\begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix}$ and apply Proposition 9.1.■

9.2 Balanced Strings

Let $a \in \{0,1\}^n$ be a binary string, with each bit coloured in Red or Blue. The *red substring* [*blue substring*] of a is obtained from a by deleting all blue [respectively, all red] bits. We say that such a string is *balanced* if its red and blue substrings are equal. Coloured bits can be encoded by a strings of the length two $\{00, 10, 01, 11\}$ where the first bit represents the bit itself and the second one represents its colour. This way we get a Boolean function f on $2n$ variables, which outputs 1 iff the corresponding (coloured) string is balanced.

In [1] it is proven that the four-letter variant of f requires 1-b.p. of size at least $2^{\Omega(n)}$.

Proposition 9.3 *The function f is computable on a 4-gentle branching program P of size $O(n^3)$ with fluctuation $\gamma = 0$ and deviation $\delta = 0$.*

Proof. There is a branching program P of size $O(n^3)$ which computes f by the following trivial procedure: Search for the next red bit and search for the next blue bit, and compare. So, it is enough to take two maxterms 0011 and 1001 on the first four variables, and apply Proposition 9.1.■

9.3 The Isolated Point Function

Let us consider the functions $f_{m,s}$, which where used in [12, 13] as the witness functions for the hierarchy of syntactic read-k-times branching programs with respect to k.

A *hypergraph* over a set of points $[m] = \{1, \ldots, m\}$ is a family $\mathcal{F}$ of subsets of $[m]$. The hypergraph is *s-uniform* if each its member contains exactly s points. A point i is *isolated* in $\mathcal{F}$ if $i \notin S$ for all $S \in \mathcal{F}$. Given a hypergraph $\mathcal{F}$, we ask if it has an isolated point. The Boolean function $f_{m,s}$ has $n = \binom{m}{s}$ variables x_S, one for each s-element subset S of $[m]$, and outputs 1 iff the corresponding s-uniform hypergraph

has no isolated points. That is

$$f_{m,s} \rightleftharpoons \bigwedge_{i=1}^{m} \left(\bigvee_{i \in S \in \mathcal{F}} x_S \right).$$

For every $s \leqslant m$,$s|m$, this function has an obvious syntactic read-s-b.p. of size $m\binom{m-1}{s-1} = s\binom{m}{s} = sn$. On the other hand, it is proved in [13] that, as long as $k \leqslant C \ln n / \ln \ln n$ $(C < 1/2)$ and $k \leqslant s/\log s$, the function $f_{m,s}$ cannot be computed by a k-n.b.p. of size smaller than $2^{\Omega(n^{1-2C})}$.

Proposition 9.4 *The function $f_{m,s}$ is computable by an α-gentle branching program of size $O(sn)$ with $\alpha = O((\log n)/s)$, deviation $\delta \leqslant 1$ and fluctuation $\gamma = 0$.*

Proof. Take any $m+1$ s-element sets $T_1, S_1, \ldots, S_m$ such that $T_1 \neq S_1$, $1 \in T_1 \cap S_1$ and $i \in S_i$ for all $i = 2, \ldots, m$. The corresponding monomials $x_{S_1} x_{S_2} \cdots x_{S_m}$ and $\overline{x}_{S_1} x_{T_1} x_{S_2} \cdots x_{S_m}$ are 1-terms of $f_{m,s}$ and their union has only $2m+1 = O((\log n)/s)$ literals. Proposition 9.1 yields the desired gentle b.p. for $f_{m,s}$.■

Some other upper bounds can be deduced from results in [21].

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'