ELSEVIER Discrete Applied Mathematics 85 (1998) 223–238

DISCRETE APPLIED MATHEMATICS

Neither reading few bits twice nor reading illegally helps much

S. Jukna[a,*], A. Razborov[b,1]

[a] *Department of Computer Science, University of Trier, D-54286 Trier, Germany and Institute of Mathematics, Akademijos 4, 2600 Vilnius, Lithuania*
[b] *Steklov Mathematical Institute, Gubkina 8, 117966, GSP-1, Moscow, Russia*

Received 12 March 1996; received in revised form 23 January 1998; accepted 2 February 1998

Abstract

We first consider the so-called $(1, +s)$-*branching programs* in which along every consistent path at most s variables are tested more than once. We prove that any such program computing a characteristic function of a linear code C has size at least $2^{\Omega(\min\{d_1, d_2/s\})}$, where d_1 and d_2 are the minimal distances of C and its dual $C^{\perp}$. We apply this criterion to explicit linear codes and obtain a super-polynomial lower bound for $s = o(n/\log n)$.

Then we introduce a natural generalization of read-k-times and $(1, +s)$-branching programs that we call *semantic branching programs*. These programs correspond to *corrupting* Turing machines which, unlike eraser machines, are allowed to read input bits even *illegally*, i.e. in excess of their quota on multiple readings, but in that case they receive in response an unpredictably corrupted value. We generalize the above-mentioned bound to the semantic case, and also prove exponential lower bounds for semantic read-once nondeterministic branching programs. © 1998 Elsevier Science B.V. All rights reserved

Keywords: Complexity; Lower bounds; Branching programs; Switching-and-rectifier networks; Corrupting machines

1. Introduction

We consider the usual model of *branching programs* (b.p.). This model captures in a natural way the deterministic space whereas nondeterministic branching programs (n.b.p.) do the same for the nondeterministic mode of computation. A similar model of *switching-and-rectifier networks* (s.r.n.) appeared already in pioneering work of Shannon and was extensively studied in the Russian literature since early 50th. The best lower bound for unrestricted n.b.p., however, remains the lower bound of $\Omega(n^{3/2}/\log n)$

* Corresponding author. E-mail: jukna@ti.uni-trier.de. Supported by DFG grant Me 1077/10-1.
[1] Supported by grant # 96-01-01222 of the Russian Basic Research Foundation, and by an AMS-FSU grant.

0166-218X/98/$19.00
PII S0166-218X(98)00042-0

proved by Nečiporuk in 1966 [11]. A survey of known lower bounds for these models can be found in [13].

In order to learn more about the power of branching programs, various restricted models were investigated. One of the most intensively studied 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 [10] corresponds to so-called *eraser* Turing machines, and the first super-polynomial lower bounds for 1-b.p. were obtained in [18, 19]; see also [1, 3, 6, 8] for further results in that direction. Exponential lower bounds for 1-n.b.p. were proven in [2, 4, 5, 7]. However, any attempts to get such bounds for 2-b.p. bitterly failed (so far).

One possible explanation of this failure might be that the restriction of being read-k-times is somewhat "unstructured" and, as such, is difficult to capture in an argument. Its stronger and more constructive version requires that in *every* path, be it consistent or not, every variable appears at most k times: the corresponding branching programs were called in [2] *syntactic*. This restriction is much easier to capture and analyze, and, indeed, strong lower bounds for syntactic k-b.p. (for an arbitrary but fixed k) were independently established in [2] (for the nondeterministic case) and in [12] (for the deterministic one); see also [5]. As a matter of fact, the difference between syntactic and ordinary programs disappears in the read-once case, and this provides us with some intuition as to why already the next case $k=2$ (= the first non-syntactic case) presents a new level of difficulties. Another piece of evidence that "syntactic" is a rather strong restriction is given by the exponential separation between syntactic and non-syntactic models established in [5] by exhibiting an explicit function which can be computed by a read-once switching-and-rectifier network [2] of size $O(n^{3/2})$ but requires (syntactic) 1-n.b.p. of exponential size.

Another idea to get closer to the 2-b.p. case is 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.

For *syntactic* $(1,+s)$-b.p., where $s=s(n)\leqslant o(n^{1/3}/\log^{2/3} n)$, exponential lower bounds were proved in [15, 16]. Ref. [14], improving upon [20], established (implicitly) the lower bound $\exp(\Omega(n/(s+1)\log n)^{1/2})$ on the size of non-syntactic $(1,+s)$-b.p. computing some function in ACC. This is super-polynomial in n as long as $s=o(n/(\log n)^3)$.

In the first part of this paper we apply some of the techniques of [20, 14] to show that any $(1,+s)$-b.p. computing the characteristic function of a linear code C has size at least $2^{\Omega(\min\{d_1,d_2/s\})}$, where d_1 and d_2 are the minimal distances of C and its dual $C^{\perp}$, respectively (Corollary 4). We then apply this criterion to concrete linear codes. For a Reed–Muller code this yields the bound $\exp(\Omega(n/(s+1))^{1/2})$ (Theorem 5), and for a Bose–Chaudhuri–Hocquenghem code the bound becomes $\exp(\Omega(\min\{\sqrt{n},n/s\}))$ (Theorem 6). This is super-polynomial in n for any $s=o(n/\log n)$. Whereas we have only a slight numerical improvement over [20, 14], the combinatorial part of our

[2] This is the weakest natural nondeterministic model that is non-syntactic, and no non-trivial lower bounds are known for it. See Section 4 for the definition and a more thorough discussion.

bound is much easier, essentially trivial (modulo some known deep facts from coding theory).

In the second part of this paper we introduce a stronger and, perhaps, more natural [3] version of eraser machines that we call *corrupting machines*. The corresponding restrictions in the non-uniform setting of branching programs are even more tightly associated with the actual computation than for ordinary b.p.: in this sense our new model is opposite to syntactic b.p. in whose definition the computation plays no role at all. For that reason we call the non-uniform version of corrupting machines *semantic* branching programs: these are apparently stronger than their ordinary counterparts, although we have not been able to prove any separation between them.

One of our motivations for introducing semantic b.p. is the common belief that working in a "right" model can significantly advance us to the task of proving lower bounds for the original (weaker, but more awkward) model. In pursuit of this goal we generalize the results about $(1,+s)$-b.p. to the semantic case (Theorem 9), and also we prove exponential lower bounds for semantic 1-n.b.p. (Theorems 13 and 15). Our methods tend to examine multiple readings along rejecting (rather than accepting) computations, and we hope that this approach may turn out to be helpful for the future research in the area.

2. Lower bounds for $(1, +s)$-branching programs

We will use the following notation. A *partial input* is a mapping $a:[n]\to\{0,1,*\}$ where $[n]=\{1,\dots,n\}$. If $a(i)=*$ we say that the ith bit in a is *unspecified* (or undefined). By $S(a)$ we denote the set of all specified bits, i.e. $S(a)=\{i\in[n]\colon a(i)\neq *\}$. For (partial) inputs $a_1,a_2,\dots,a_s$ such that all $S(a_j)$ are pairwise disjoint, $[a_1,a_2,\dots,a_s]$ is the input specifying bits from $\bigcup_{j=1}^{s} S(a_j)$ and defined by the equality $[a_1,a_2,\dots,a_s](i)=a_j(i)$ for $i\in S(a_j)$. The *length* $|a|$ of a is the number of bits in $S(a)$. For two partial inputs a and b, let $D(a,b)$ be the set of all bits where they both are defined and have different values. Given a boolean function $f(x_1,\dots,x_n)$, every partial input a (treated for this purpose as a restriction) defines the subfunction $f|_a$ of f in $n-|a|$ variables in a usual manner. A *minterm* (*maxterm*) of f is a partial input a for which $f|_a\equiv 1$ ($f|_a\equiv 0$, respectively), and which is minimal in the sense that unspecifying every single value $a(i)\in\{0,1\}$ already violates this property. Given a boolean function f, we say that:

- f is *d-rare* if $|D(a,b)|\geqslant d$ for every two different totally defined inputs a,b such that $f(a)=f(b)=1$;
- f is *m-dense* if $|a|\geqslant m$ for every maxterm a of f.

We adopt the standard definition of a branching program (b.p.), see e.g. [17, Section 14]. The *size* $|P|$ of a b.p. P is the number of nodes. For a partial input $a:[n]\to\{0,1,*\}$,

[3] Especially in the context of quantum computations, although we have not been able to draw any direct analogies.

comp(a) is the path in P consistent with a until we reach a node where the first test of $*$ is made. $P|_a$ is the naturally defined program in $n - |a|$ variables that computes $f|_a$, where f is the function computed by P. More specifically, $P|_a$ is obtained from P by removing all edges inconsistent with a and contracting all edges consistently testing a specified bit in a (so that exactly edges and nodes testing an unspecified bit are left intact). If the input a is totally defined, *comp*(a) leads to one of the sink nodes, and $P|_a$ is a trivial single-node program.

A b.p. P is *read-k-times* (k-b.p. for short) if for every (total) input a every variable appears at most k times along *comp*(a). P is $(1,+s)$ if the number of variables tested more than once along *comp*(a) does not exceed s, for every (total) a.

2.1. General bounds

The following general bound was implicitly proved (but not stated exactly in this form) in [14]:

Theorem 1. *Let* $0 \leqslant d, m, s \leqslant n$ *be arbitrary integers. Every* $(1,+s)$*-branching program computing a d-rare and m-dense function must have size at least*

$$2^{(\min\{d, m/(s+1)\}-1)/2}.$$

For completeness we include here its independent proof. Recall first the main technical statement from [14, 20] concerning so-called "forgetting pairs" of inputs.

Definition 2. Let a, b be (partial) inputs with $S(a)=S(b)$. Given a branching program P, the pair a,b is called a *forgetting pair* (for P) if there exists a node w such that w belongs to both *comp*(a) and *comp*(b), and both computations read all the variables with indices in $D(a,b)$ at least once before reaching w.

Given a b.p. P, one can get a forgetting pair by following all the computations until $r := \lfloor \log_2 |P| \rfloor + 1$ different bits are tested along each of them. Since $|P| < 2^r$, at least two of these paths must first split and then stick in some node. Take the corresponding partial inputs a_1' and b_1' and extend them to a_1 and b_1 such that $S(a_1)=S(b_1)=S(a_1')\cup S(b_1')$ and $D(a_1,b_1)\subseteq S(a_1')\cap S(b_1')$. This way we get a forgetting pair of inputs $a_1 \neq b_1$ both of which are defined on the same set of at most $|S(a_1')\cup S(b_1')| \leqslant 2r-1$ bits. We can now repeat the argument for $P|_{a_1}$ and obtain next forgetting pair of inputs $[a_1,a_2]$ and $[a_1,b_2]$, etc. We can continue this procedure for s steps until $s(2r-1) \leqslant s(2\log_2|P|+1)$ does not exceed the minimum number of different variables tested on a computation of P. This proves the following.

Proposition 3 (Žák [20] and Savický and Žák [14]). *Let P be a branching program in which every computation reads at least m different variables. Let s be a natural number in the interval* $1 \leqslant s \leqslant m/(2\log_2|P|+1)$. *Then there exist pairwise disjoint sets*

$I_j \subseteq [n]$ for $j = 1, \dots, s$ and partial inputs $a_j \neq b_j$ with $S(a_j) = S(b_j) = I_j$ such that for all $j = 1, 2, \dots, s$ we have:

(i) $|I_j| \leqslant 2\log_2 |P| + 1$,

(ii) *the inputs $[a_1, \dots, a_j]$ and $[a_1, \dots, a_{j-1}, b_j]$ form a forgetting pair. Moreover, nodes $w_1, \dots, w_s$ fulfilling Definition 2 for these pairs can be chosen in such a way that they appear on the path comp($[a_1, \dots, a_s]$) in the non-decreasing order.* [4]

Proof of Theorem 1. Suppose the contrary that some $(1, +s)$-b.p. P computes a d-rare and m-dense function and has size less than $2^{(\min\{d, m/(s+1)\}-1)/2}$. We can assume w.l.o.g. that $d \geqslant 2$ (otherwise the bound becomes trivial), and this implies that every minterm of f has size $n \geqslant m$. Hence, in order to force f to either 0 or 1 we must specify at least m positions, therefore every computation of P must read at least m different variables. Since $|P| \leqslant 2^{(m/(s+1)-1)/2}$, we can apply Proposition 3 (with $s := s+1$) and find I_j, a_j, b_j $(1 \leqslant j \leqslant s+1)$ with properties (i) and (ii). From (i) and the bound on $|P|$ we have $|I_j| < \min\{d, m/(s+1)\}$, and this implies that the partial input $[a_1, \dots, a_{s+1}]$ specifies strictly less than m variables. Since f is m-dense, $[a_1, \dots, a_{s+1}]$ can be extended to a totally defined input a such that $f(a) = 1$.

As I_j's are pairwise disjoint and P is $(1, +s)$, there exists j, $1 \leqslant j \leqslant s+1$, such that all variables with indices from I_j are tested at most once along *comp*(a). Now, let w be the node that corresponds to the forgetting pair

$$[a_1, \dots, a_{j-1}, a_j],\ [a_1, \dots, a_{j-1}, b_j]$$

accordingly to Definition 2; clearly, w is on *comp*(a). All variables with indices from $D(a_j, b_j) \subseteq I_j$ are already tested along *comp*(a) before w, hence no such variable is tested after w, and the computation on the input c obtained from a by replacing a_j with b_j cannot diverge from *comp*(a) after the node w. Therefore, $f(c) = f(a) = 1$. But this, along with $|I_j| < d$, contradicts the d-rareness of f. The proof of Theorem 1 is complete. □

This theorem is especially useful for (characteristic functions of) linear codes, i.e. for linear subspaces of $GF(2)^n$. Say that a subset $C \subseteq \{0,1\}^n$ is *d-rare* or *m-dense* if such is the characteristic function of C.

C is d-rare if and only if the minimal distance of C (treated as a code over $GF(2)$) is at least d.

m-density of C means that for any subset of coordinates $S \subseteq [n]$ with $|S| < m$ and for each vector $v \in \{0,1\}^S$, there is at least one vector in C whose projection onto S coincides with v. It follows that a linear code C (over $GF(2)$) is m-dense iff the minimal distance of its dual $C^\perp$ is at least m. Indeed, the set of all projections of strings in C onto S is a linear subspace in $\{0,1\}^S$, and this subspace is proper if and only if all strings $a \in C$ satisfy a non-trivial linear relation $\sum_i \xi_i a_i = 0 \bmod 2$ whose

[4] This extra property of $w_1, \dots, w_s$ will be used only in Section 3.

support $\{i\colon \xi_i = 1\}$ is contained in S. But, by definition, $C^\perp$ consists exactly of all relations ξ satisfied by C, and its minimal distance is exactly the minimal possible cardinality of a set S for which the projection of C onto $\{0,1\}^S$ is proper.

Hence, Theorem 1 implies:

Corollary 4. *Let C be a linear code with minimal distance d_1, and let d_2 be the minimal distance of the dual code $C^\perp$. Then every $(1,+s)$-branching program computing the characteristic function of C has size at least*

$$2^{(\min\{d_1, d_2/(s+1)\}-1)/2}.$$

2.2. Lower bounds for explicit codes

Reed–Muller codes. Recall that the rth-order binary Reed–Muller code $R(r,\ell)$ of length $n = 2^\ell$ is the set of graphs of all polynomials in ℓ variables over $GF(2)$ of degree at most r. This code is linear and has minimal distance $2^{\ell-r}$.

Theorem 5. *Let $n = 2^\ell$, $0 \leqslant s \leqslant n$ and $r = \lfloor 1/2(\ell + \log_2(s+1)) \rfloor$. Then every $(1,+s)$-branching program computing the characteristic function of the Reed–Muller code $R(r,\ell)$ has size at least* $\exp(\Omega(n/(s+1))^{1/2})$.

Proof. It is known (see, e.g. [9, p. 375]) that the dual of $R(r,\ell)$ is $R(\ell - r - 1, \ell)$. Hence, in the notation of Corollary 4 we have $d_1 = 2^{\ell - r} \geqslant \Omega(\sqrt{n/(s+1)})$ and $d_2 = 2^{r+1} \geqslant \Omega(\sqrt{n(s+1)})$. The desired bound follows. □

Bose–Chaudhuri–Hocquenghem codes. Let $n = 2^\ell - 1$, and let $C \subseteq \{0,1\}^n$ be a BCH-code with designed distance $\delta = 2t+1$, where $t \leqslant \sqrt{n}/4$. Let d_2 be the minimal distance of its dual $C^\perp$. The Carliz–Uchiyama bound (see, e.g., [9, p. 280]) says that $d_2 \geqslant 2^{\ell-1} - (t-1)2^{\ell/2}$ which is $\Omega(n)$ due to our assumption on t. Since the minimal distance d_1 of a BCH-code is always at least its designed distance δ, we get from Corollary 4

Theorem 6. *Let $n = 2^\ell - 1$, and let C be a BCH-code with designed distance $\delta = 2t+1$, where $t \leqslant \sqrt{n}/4$. Then every $(1,+s)$-branching program computing the characteristic function of C has size* $\exp(\Omega(\min\{t, n/s\}))$. *In particular, if $t \geqslant \omega(\log n)$ then every such program must have super-polynomial size as long as $s \leqslant o(n/\log n)$.*

3. Semantic branching programs

The uniform model corresponding to k-b.p. are so-called *eraser machines*, and a similar definition capturing the $(1,+s)$-case can be given in a straightforward way. It is not clear, however, to which extent the very name “eraser” is justified; perhaps, something like *poisoning* machines would be more natural. Indeed, these machines model the situation when after reaching the quota on the amount of readings, input bits get "poisoned" so that any extra attempt to read them leads to something really bad (short circuit, for example). Accordingly, programs for such machines should be designed in such a way that they avoid this unpleasant situation by any means.

We might try to define "truly" eraser machines as machines erasing every input bit after the quota on the number of its readings is reached, and putting into its place a question mark to be observed during subsequent readings. This is not good since the question marks can be used for storing information on the input tape, and read-once logarithmic space eraser (in this sense) machines can recognize essentially all poly-time computable languages. More precisely, for every $L \in \mathbf{P}$ there exists a polynomial $p(n)$ such that the language $\{x\#0^{p(|x|)}: x \in L\}$ is recognized by such a machine. Actually, this fact looks like an interesting phenomenon, so let us briefly sketch its proof.

We use the second part of the input tape (originally occupied by $p(|x|)$ zeros) for simulating the computation of a poly-time decision algorithm for L on x. Let c_{ij} be the (binary) content of the jth cell on the working tape at the ith stage of the performance of this algorithm on x; $1 \leqslant i \leqslant t$, $1 \leqslant j \leqslant l$. For every i, j we reserve one cell a_{ij} on the second part of the input tape. The simulation proceeds in t stages, and our goal in the ith stage is to read exactly those a_{ij} among $a_{i1}, \ldots, a_{il}$ for which $c_{ij} = 1$. Thus, after the ith stage a_{ij} contains 0 if $c_{ij} = 0$, and contains "?" if $c_{ij} = 1$.

Suppose we have already performed i stages and enforced the desired content of the cells $a_{i1}, \ldots, a_{il}$. Now we process $a_{i+1,1}, \ldots, a_{i+1,l}$. First, we, using the external logarithmic space, simulate the $(i+1)$th step of the original computation and put appropriate question marks into O(1) active cells $a_{i+1,j}$, i.e., into those cells for which $c_{i+1,j}$ may in principle differ from c_{ij}. Then we go over all remaining (inactive) cells and simply "copy" the content of $a_{i,j}$ into $a_{i+1,j}$ by reading the latter cell if and only if we observe "?" in $a_{i,j}$ (the content of a_{ij} is destroyed, but we will not use it in the sequel anyway).

We propose *corrupting machines* as an intermediate model between poisoning and eraser machines which is free of this disadvantage: every language recognizable by a logarithmic space corrupting machine belongs to **LOGSPACE**. Namely, when such a machine attempts an illegal reading (that is, in excess of its quota), nothing bad happens (as with poisoning machines) except that the machine gets a possibly corrupted value. Our machine (unlike "truly" eraser machines) does not know whether the reading was legal or not (so, it cannot use this knowledge to store an extra information), and it is required to output the correct answer at the end of the computation no matter which corruption took place during illegal readings (adversary model).

The corresponding restrictions in the non-uniform setting of branching programs are even more tightly associated with the actual computation than for ordinary b.p. In this sense, the situation is just the opposite to the case of syntactic b.p. whose definition is given entirely in terms of internal combinatorial structure of the program. For that reason we call the non-uniform model corresponding to corrupting machines *semantic branching programs* and immediately proceed to this setting for precise definitions. The interested reader should have no difficulties in adopting them to the uniform version.

3.1. Deterministic case

Let $Q \subseteq \mathbb{N}^n$ be an anti-monotone non-trivial predicate which in the sequel will be called the *quota predicate*. Here $\mathbb{N}$ is the set of nonnegative integers, n is the number of variables, and the anti-monotonicity means that $Q(k_1,\dots,k_n)$ along with $k_1' \leqslant k_1,\dots,k_n' \leqslant k_n$ implies $Q(k_1',\dots,k_n')$. The predicate Q expresses the quota on the amount of legal readings, and the following examples are the most important for us:

$Q_k(k_1,\dots,k_n) \equiv \forall i \in [n]\ (k_i \leqslant k)$ (every variable is read at most k times);

$Q_{(1,+s)}(k_1,\dots,k_n) \equiv |\{i \in [n]\colon k_i \geqslant 2\}| \leqslant s$ (at most s variables are read more than once).

Definition 7. For a path p in a b.p. P and a quota predicate Q, we define a vector $k^{p,Q} = (k_1^{p,Q},\dots,k_n^{p,Q}) \in \mathbb{N}^n$ such that $Q(k^{p,Q})$ by induction on the number of edges in p.

(i) If p is empty then $k^{p,Q} = (0,\dots,0)$.

(ii) Let $p = (q,e)$, and suppose that the head node w of e is marked by x_i.

(a) If $Q(k_1^{q,Q},\dots,k_{i-1}^{q,Q},k_i^{q,Q}+1,k_{i+1}^{q,Q},\dots,k_n^{q,Q})$ then we let

$$k^{p,Q} = (k_1^{q,Q},\dots,k_{i-1}^{q,Q},k_i^{q,Q}+1,k_{i+1}^{q,Q},\dots,k_n^{q,Q}).$$

In that case we say that the reading of x_i at w along the path p is *legal*.

(b) If $\neg Q(k_1^{q,Q},\dots,k_{i-1}^{q,Q},k_i^{q,Q}+1,k_{i+1}^{q,Q},\dots,k_n^{q,Q})$ then we let $k^{p,Q} = k^{q,Q}$ and say that x_i is read *illegally* at w.

Notice that illegal readings do not increment the counter $k^{p,Q}$. This allows our program to function properly between different attempts to read illegally.

Given a b.p. P, a quota predicate Q and a totally defined input $a \in \{0,1\}^n$, we let $Comp^Q(a)$ denote the set of all possible I/O paths such that all *legal* readings along these paths are consistent with a. Obviously, $comp(a) \in Comp^Q(a)$, but $Comp^Q(a)$ may also contain other paths (typically inconsistent). We say that P is *semantic w.r.t. Q* if for every $a \in \{0,1\}^n$ all paths in $Comp^Q(a)$ lead to a sink of the same type (accepting or rejecting) as $comp(a)$. A *semantic read-k-times branching program* is a b.p. semantic with respect to Q_k. A *semantic* $(1,+s)$*-b.p.* is a b.p. that is semantic w.r.t. $Q_{(1,+s)}$.

Remark 8. Notice that every (ordinary) k-b.p. or $(1,+s)$-b.p. is also semantic simply for the reason that there can be no illegal readings, and $Comp^Q(a)$ consists of the single path $comp(a)$. In fact, it is easy to see that the condition $\forall a \in \{0,1\}^n (Comp^Q(a) = \{comp(a)\})$ *characterizes* ordinary programs in the class of semantic programs.

Now we show how to extend Theorem 1 to the semantic case.

Theorem 9. *Let $0 \leqslant d, m, s \leqslant n$ be arbitrary integers. Every semantic $(1,+s)$-branching program computing a d-rare and m-dense function must have size at least* $2^{(\min\{d,m/(2s+1)\}-1)/2}$.

In particular, both our bounds for explicit codes (Theorems 5 and 6) are still valid in the same form for the more general case of semantic $(1,+s)$-b.p.

Proof. We begin as in the proof of Theorem 1 but with the assumption

$$|P| < 2^{(\min\{d,m/(2s+1)\}-1)/2},$$

and construct $I_1, \ldots, I_{2s+1}, a_1, \ldots, a_{2s+1}, b_1, \ldots, b_{2s+1}$ satisfying Proposition 3 (with $s := 2s+1$) and a total extension a of $[a_1, \ldots, a_{2s+1}]$ such that $f(a) = 1$. The rest of that proof basically says that every $D(a_j, b_j) \subseteq I_j$ contains at least one variable tested for the second time along *comp*(a), meaning that P is not a $(1,+2s)$-b.p. In our case, however, we have to derive a contradiction from the fact that P is a semantic $(1,+s)$-b.p., which requires some extra work.

Let c_j be the input obtained from a when we replace a_j with b_j, and let p_j, p'_j be the sub-paths of *comp*(a), *comp*(c_j) respectively ending at the node w_j fulfilling Definition 2 for the forgetting pair $[a_1, \ldots, a_j], [a_1, \ldots, a_{j-1}, b_j]$. Let also q_j be the remaining part of *comp*(a) so that *comp*$(a) = (p_j q_j)$. As in the proof of Theorem 1 we are going to force P to accept at least one of the inputs c_j which, together with $f(a) = 1$, would contradict d-rareness of f. For doing this, it suffices to show that $(p'_j q_j) \in Comp^{Q_{(1,+s)}}(c_j)$ for some $1 \leqslant j \leqslant 2s+1$. Consider two cases.

Case 1. *At least s variables are tested more than once along p_{2s}.* We claim that in this case $(p'_{2s+1} q_{2s+1}) \in Comp^{Q_{(1,+s)}}(c_{2s+1})$. Indeed, p'_{2s+1} is OK since all readings along this path (legal or not) are consistent with c_{2s+1}. Moreover, since w_{2s+1} appears on *comp*(a) after w_{2s} (by property (ii) from Proposition 3), p'_{2s+1} extends p_{2s} which implies that $k^{p'_{2s+1}, Q_{(1,+s)}}$ already contains (exactly) s components that are greater or equal than 2. Thus, every repetitive reading of a new variable along $(p'_{2s+1} q_{2s+1})$ that occurs on q_{2s+1} is illegal. This, in particular, applies to all bits from $D(a_{2s+1}, b_{2s+1})$, and all other readings along q_{2s+1} are consistent with a and, hence, with c_{2s+1}.

Case 2. *Less than s variables are tested more than once along p_{2s}.* We know that every $D(a_j, b_j)$ contains at least one bit which is tested once more after the node w_j. For $1 \leqslant j \leqslant 2s$ denote by w'_j the earliest node along *comp*(a) where the second test of a bit from $D(a_j, b_j)$ is made. The assumption of Case 2 implies that at least $(s+1)$ nodes among $w'_1, w'_2, \ldots, w'_{2s}$ must belong to q_{2s}. Let w'_j be the latest (along *comp*(a)) of these nodes. Note that q_j contains the segment $\bar{q}_j$ of *comp*(a) bounded by w_j and w'_j, and this segment is consistent with c_j. Moreover, at least s variables are already tested more than once along $p'_j \bar{q}_j$ (namely, at nodes from the list $\{w'_1, w'_2, \ldots, w'_{2s}\}$ belonging to q_{2s} and other than w'_j). Now, the same argument as in Case 1 shows that $(p'_j q_j) \in Comp^{Q_{(1,+s)}}(c_j)$.

This completes the proof of Theorem 9. □

3.2. Nondeterministic case

We introduce nondeterminism into branching programs simply by additionally allowing *guessing nodes* of out-degree 2 that are not marked by any variable and have an obvious computational meaning. A nondeterministic branching program (n.b.p.) is *read-k-times* or $(1,+s)$ when this restriction is satisfied by all consistent paths beginning at the source node [13]. [5] Notice that every consistent path in a n.b.p. can always be extended to a consistent path terminating at a sink node, so we could equally well consider in this definition only such I/O paths.

We extend Definition 7 to nondeterministic b.p. in an obvious way. Namely, if $p=(q,e)$ and e goes out of a guessing node, we let $k^{p,Q}=k^{q,Q}$.

In order to define acceptance/rejectance conditions for a n.b.p. P on a string a with respect to some quota predicate Q we introduce a game of two players, B (*brancher*) and C (*corrupter*). This game, which we denote by $G^Q(a)$, develops along a path in P, and it begins at the source node. At a guessing node, B simply chooses one of the two alternatives for the game to proceed. Suppose $G^Q(a)$ arrives at a computational node w along some path q, and let e be the outgoing edge consistent with a. If the reading at w is legal (along the joint path (q,e)), $G^Q(a)$ follows e. Otherwise C chooses one of the two continuations. The game terminates when it arrives at a sink node.

The goal of the brancher is to reach one of the accepting sink nodes, and we say that in this case he *wins*. The goals of the corrupter are defined less clearly: in general, she is interested in creating as much damage by corrupting the computation as possible. This leads us to the following definition:

Definition 10. A n.b.p. P is *semantic with respect to a quota predicate* Q if for every string $a\in\{0,1\}^n$ either B has a winning strategy against C in the game $G^Q(a)$ (a is *accepted*) or B loses in the cooperative version of this game, that is even when C helps him to win (a is *rejected*).

A *semantic k-n.b.p.* [$(1,+s)$*-n.b.p.*] is a n.b.p. semantic with respect to Q_k [$Q_{(1,+s)}$, respectively].

Semantic b.p. make a subclass of semantic n.b.p. (with respect to the same quota predicate Q). In this case there is no brancher, and C is doomed to fail in the solitaire game $G^Q(a)$, both for accepted and rejected inputs.

Ordinary (read-k-times or $(1,+s)$) n.b.p. also make a subclass of semantic n.b.p. (cf. Remark 8). This is because C never has a chance to participate in the game, due to the structure of the program, and the game itself proceeds only along consistent paths.

Finally, note that if a semantic program accepts or rejects according to Definition 10, it also accepts (or rejects) in the usual sense. Indeed, it is easy to see that acceptance/

[5] One natural modification of this definition will be discussed in the next section.

rejectance conditions from Definition 10 turn into ordinary ones in the partial case when the corrupter is *passive*, i.e. refrains from corrupting the computation by always choosing the continuation consistent with a.

For a Boolean function f and an integer d we denote by $\mathrm{cov}(f,d)$ the minimal h for which there exist monomials $u_1,\dots,u_h$, of d literals each, such that $f \leqslant u_1 \vee \dots \vee u_h$. Our general bound for semantic 1-n.b.p. looks as follows:

Theorem 11. *Let f be a d-rare function, $d \geqslant 1$. Then every semantic read-once non-deterministic branching program computing f has size at least* $\mathrm{cov}(f,d-1)$.

Proof. We can assume w.l.o.g. that $d \geqslant 2$ (otherwise the bound becomes trivial). Let P be a semantic 1-n.b.p. computing some d-rare function f. Fix arbitrarily one consistent[6] accepting path p_a for every accepted input a. Since $d \geqslant 2$, p_a must read all variables at least once. Let $p_a = (p_a' p_a'')$, where p_a' is a segment of p_a along which exactly $(d-1)$ variables are tested (at least once), and let w_a be the terminal node of p_a'. For each node w in $W := \{w_a\colon f(a) = 1\}$ select arbitrarily one path from all the paths p_a' with $w_a = w$, and denote this path by p_w. Let u_w be the monomial of $(d-1)$ literals corresponding to that path p_w. We are going to finish the proof by showing that $f \leqslant \bigvee_{w \in W} u_w$.

For this we will exploit one particular property of semantic read-once n.b.p. (not shared already by $(1,+1)$-n.b.p.). Namely, in the cooperative mode of the game $G^{Q_1}(a)$, B and C can follow *every* path p (consistent or not) for some input $a_p \in \{0,1\}^n$. This input a_p is simply constructed by letting $a_p(i)$ to be the result of the *first* reading of x_i along p. The input a_p is in general partial, but when p leads to an accepting sink, and the function f computed by the program is known to be 2-rare, a_p must be an accepted input (since B and C can win by cooperating, a_p can not be rejected), and it must be totally defined (from 2-rareness).

Suppose now that $f(b) = 1$, and w is the terminal node of p_b'. We claim that $u_w(b) = 1$.

Indeed, otherwise the input a_p corresponding to the path $p = (p_w p_b'')$ would be an accepted input different from b (since readings along p_w have priority in defining a_p). On the other hand, all bits from $D(b,a_p)$ must be tested along p_w. To show this, notice that every bit i not tested along p_w is tested for the first time only on p_b''. Let ε be the result of the earliest reading of x_i along p_b''. Then $b(i) = \varepsilon$ since p_b'' is consistent with b, and $a_p(i) = \varepsilon$ by construction of a_p. Hence, $i \notin D(b,a_p)$.

Thus, $u_w(b) = 0$ could happen only if P would accept two different inputs a_p and b with $|D(b,a_p)| < d$, which is impossible by d-rareness of f. This completes the proof of the fact $u_w(b) = 1$, and the proof of Theorem 11. $\square$

The following easy lemma provides a lower bound on $\mathrm{cov}(f,d)$ in terms of density.

---

[6] Such path exists since the brancher must have a winning strategy on a also in the case when the corrupter plays passively.

Lemma 12. *For an m-dense function f in n variables,*

$$\mathrm{cov}(f,d) \geqslant \exp\left(\Omega\left(\frac{md}{n}\right)\right).$$

Proof. Let $f \leqslant \bigvee_{i=1}^{h} u_i$, where u_i are monomials of d literals and $h = \mathrm{cov}(f,d)$. Hit this inequality with a restriction ρ assigning random (0–1) values to randomly chosen $(m-1)$ variables. Then

$$\begin{aligned}\mathbf{P}[u_i|_\rho \not\equiv 0] &\leqslant \mathbf{P}\left[|S(u_i) \cap S(\rho)| < \frac{md}{2n}\right] + \mathbf{P}\left[u_i|_\rho \not\equiv 0 \,\middle|\, |S(u_i) \cap S(\rho)| \geqslant \frac{md}{2n}\right] \\ &\leqslant \exp\left(-\Omega\left(\frac{md}{n}\right)\right)\end{aligned}$$

and $\mathbf{P}[f|_\rho \not\equiv 0] = 1$ since f is m-dense. On the other hand,

$$\mathbf{P}[f|_\rho \not\equiv 0] \leqslant \sum_{i=1}^{h} \mathbf{P}[u_i|_\rho \not\equiv 0].$$

The statement follows. □

Theorem 11 and Lemma 12 imply the lower bound $\exp(\Omega(md/n))$ on the size of semantic 1-n.b.p. computing a d-rare and m-dense function. In particular, this gives an $\exp(\Omega(\sqrt{n}))$ bound for BCH-codes:

Theorem 13. *Let $n = 2^\ell - 1$, and let C be a BCH-code with designed distance $\delta = 2t + 1$, where $t \leqslant \sqrt{n}/4$. Then every semantic read-once nondeterministic branching program computing the characteristic function of C has size* $\exp(\Omega(t))$.

The following theorem extends the lower bound argument used in [4, 2] (for ordinary 1-n.b.p.) to semantic 1-n.b.p., and works for Boolean functions which are not sufficiently rare for Theorem 11 to give a strong lower bound.

For a set of inputs $A \subseteq \{0,1\}^n$ and an integer $0 \leqslant k \leqslant n$, we define the *kth degree* $d_k(A)$ as the maximum number of inputs in A, all of which have 1's on some fixed set of k coordinates. An input a is a *lower one* of a Boolean function f if $f(a) = 1$ and $f(b) = 0$ for all inputs $b \neq a$ such that $b \leqslant a$. *Lowest ones* are lower ones with the smallest number of 1's.

Theorem 14. *Let f be a Boolean function, A be the set of its lowest ones and r be the number of* 1*'s in them. Then, for every $0 \leqslant k \leqslant r$, every semantic read-once nondeterministic branching program computing f has size at least* $|A|/(d_k(A)d_{r-k}(A))$.

Proof. Let P be a semantic 1-n.b.p. computing f. Given an input $a \in A$, let p_a be any accepting path which is followed by the game $G^{\mathcal{Q}_1}(a)$ when the corrupter always chooses to continue along the edge marked by 0, totally disregarding real values of bits. p_a may be inconsistent but it has one nice property: for each bit i, the variable x_i appears positively on p_a *exactly* $a(i)$ times. Let $p_a = (p'_a p''_a)$, where p'_a is a segment of p_a with exactly k positive readings. We denote the corresponding set of bits by I_a, and let J_a denote the set of remaining $r - k$ bits in $a^{-1}(1)$. For a node w of P, let A_w denote the set of all inputs $a \in A$ such that w is the terminal node of p'_a. We are going to finish the proof by showing that $|A_w| \leqslant d_k(A) d_{r-k}(A)$ for every node w.

Fix some node w of P, and let $\mathscr{I} = \{I_a \colon a \in A_w\}$, $\mathscr{J} = \{J_b \colon b \in A_w\}$. Consider an arbitrary pair $I \in \mathscr{I}$, $J \in \mathscr{J}$, and denote by $(I \vee J)$ the input defined by $(I \vee J)(i) = 1$ iff $i \in I \cup J$. Choose some $a, b \in A_w$ such that $I = I_a$, $J = J_b$, and let the input a_p correspond to the path $p = (p'_a p''_b)$ as in the proof of Theorem 11. Then, clearly, $a_p \leqslant (I \vee J)$. Moreover, a_p is accepted because p leads to an accepting sink. But since $|I| + |J| = r$ and r is the smallest number of 1's in an accepted input, this is possible only when $I \cap J = \emptyset$ and $(I \vee J) = a_p \in A$. Let us emphasis that this conclusion holds for *every* pair $I \in \mathscr{I}$, $J \in \mathscr{J}$.

With this observation in mind, we fix an arbitrary $J \in \mathscr{J}$ and notice that $\{(I \vee J) \colon I \in \mathscr{I}\}$ is a set of different inputs from A, all of which have 1's on J. Hence, $|\mathscr{I}| \leqslant d_{r-k}(A)$ (provided $\mathscr{J} \neq \emptyset$). Similarly, $|\mathscr{J}| \leqslant d_k(A)$ which implies $|\mathscr{I}| \cdot |\mathscr{J}| \leqslant d_k(A)$ $d_{r-k}(A)$. Finally, every $a \in A_w$ is uniquely determined by the pair (I_a, J_a), therefore $|A_w| \leqslant |\mathscr{I}| \cdot |\mathscr{J}|$. This completes the proof of the desired inequality $|A_w| \leqslant d_k(A) d_{r-k}(A)$, and of Theorem 14. □

We demonstrate the theorem by a lower bound for explicit functions in AC^0. The *exact-perfect-matching* function is a Boolean function EPM_n in n^2 variables, encoding the edges of a bipartite graph with parts of size n; the function computes 1 iff the input graph is a perfect matching. The *isolated vertex* function is a Boolean function $ISOL_{2n}$ in $\binom{2n}{2}$ variables, encoding the edges of an undirected graph on $2n$ vertices; the function computes 1 iff the input graph has no isolated vertices. That is,

$$ISOL_{2n} = \bigwedge_{i=1}^{n} \left(\bigvee \{x_{\{i,j\}} \colon 1 \leqslant j \leqslant n,\ j \neq i\} \right).$$

It is clear that both these functions are in AC^0. Moreover, it is known that EPM_n has a read-once switching-and-rectifier network (see the next section for definition) of size $O(n^3)$ but cannot be computed by a 1-n.b.p. of polynomial size [4]. Note also that neither of these two functions is d-rare even for $d = 5$, so Theorem 11 cannot give any super-polynomial lower bounds for them.

Lowest ones for EPM_n and $ISOL_{2n}$ are perfect matchings. The first function has $n!$ lowest ones and, for every $1 \leqslant k \leqslant n$, the kth degree of them is exactly $(n-k)!$. The second function has $(2n)!/2^n \cdot n!$ lowest ones, and the kth degree is $(2n-2k)!/2^{n-k} \cdot (n-k)!$. By Theorem 14 we get that these functions are hard for semantic 1-n.b.p.:

Theorem 15. *Neither EPM_n nor $ISOL_{2n}$ can be computed by a semantic read-once nondeterministic branching program of size smaller than $\binom{n}{\lfloor n/2 \rfloor}$.*

4. Conclusion and open problems

In this paper we have further (after [14]) simplified the original lower bound argument of [20] and applied it to explicit linear codes. The most interesting open question certainly consists in modifying that argument in order to make some variable be read for the third time, i.e. in trying to prove super-polynomial lower bounds for the read-twice case.

Our knowledge about the power of n.b.p. is even more depressing: for this model the $(1, +1)$ case is still open. In fact, there are no non-trivial lower bounds even for a weaker model of *read-once switching-and-rectifier networks* (1-s.r.n.). In [13] these were defined in such a way that they are equivalent to 1-n.b.p. Since now we are interested in outlining challenges in the area, we adopt here the following simpler definition that leads to a stronger model.

Definition 16. A *switching-and-rectifier network* is a directed graph (not necessarily acyclic!) with one distinguished source node s and several accepting sink nodes. Some of its edges receive labels of the form "$a(i)=0$", "$a(i)=1$", whereas other edges are left free (consistent with any input). An input a is *accepted* if there exists at least one path from s to one of the sink nodes consistent with a, and *rejected* otherwise. The switching-and-rectifier network is *read-once* (1-s.r.n.) if every variable is tested at most once along every consistent path [7] beginning at s.

Thus, 1-n.b.p. can be viewed as specially structured 1-s.r.n., and, as we already observed, a separation between them is provided by the EPM_n function [4]. Moreover, the example from [5] somehow suggests that methods previously known for 1-n.b.p. (including our Theorems 11 and 14) seem to be inherently too weak to deal with 1-s.r.n., and the latter model probably requires some new machinery.

We have introduced semantic branching programs and proved in this framework exponential lower bounds for $(1, +s)$-b.p. (when $s = o(n/\log n)$) and 1-n.b.p. These are exactly at the border of our knowledge about ordinary branching programs. In this connection, it would be interesting to prove (or disprove) that semantic b.p. are strictly stronger than their ordinary counterparts. This could be done, say, by exhibiting a function that can be computed by a poly-size semantic 1-b.p. or 1-n.b.p. but requires super-polynomial size in the corresponding ordinary model.

One more natural class of nondeterministic models (both in ordinary and semantic settings) is obtained when we relax the rejectance condition. More specifically, for ordinary programs we only require that for every accepted input there exists at least one accepting path obeying the quota on the amount of reading (but paths violating this quota are also allowed, both accepting and rejecting). For semantic n.b.p. we simply relax the rejectance condition to its ordinary form (B is required to loose only

[7] Ref. [13] used here a broader notion of "almost consistent" path, and it is the only place where our definitions diverge.

in cooperation with the *passive* corrupter). Let us call these nondeterministic models *strong*. We remark that we do not know of any lower bounds for strong 1-n.b.p. (even ordinary), and that in fact strong 1-n.b.p. can be easily shown to include 1-s.r.n.

The overall conclusion is that 1-s.r.n. seems to be the "minimal" nondeterministic model for which no non-trivial lower bounds are known, and it is also remarkable that at the same time it is the weakest non-syntactic model. Thus, proving exponential lower bounds for 1-s.r.n. (along with proving such bounds for 2-b.p.) is the next logical challenge in the area.

Acknowledgements

We are indebted to both anonymous referees of this paper for many valuable remarks.

References

[1] L. Babai, P. Hajnal, E. Szemerédi, G. Turán, A lower bound for read-once-only branching programs, J. Comput. System Sci. 35 (1987) 153–162.
[2] A. Borodin, A. Razborov, R. Smolensky, On lower bounds for read-k times branching programs, Comput. Complexity 3 (1993) 1–18.
[3] P.E. Dunne, Lower bounds on the complexity of one-time-only branching programs, in: Proc. FCT'85, Lecture Notes in Computer Science, vol. 199, Springer, Berlin, 1985, pp. 90–99.
[4] S. Jukna, Lower bounds on the complexity of local circuits, in: Proc. MFCS'86, Lecture Notes in Comput. Science, vol. 233, Springer, Berlin, 1986, pp. 440–448. [Journal version: S. Jukna, Entropy of contact circuits and lower bounds on their complexity, Theoret. Computer Sci. 57 (1988) 113–129.]
[5] S. Jukna, A note on read-k-times branching programs, Informatique Theorique Appl./Theoret. Inform. Appl. 29 (1) (1995) 75–83.
[6] M. Krause, Exponential lower bounds on the complexity of local and real-time branching programs, J. EIK 24 (3) (1988) 99–110.
[7] M. Krause, C. Meinel, S. Waack, Separating the eraser turing machine classes $L_e, NL_e, co-NL_e$ and P_e, Theoret. Computer Sci. 86 (1991) 267–275.
[8] K. Kriegel, S. Waack, Lower bounds on the complexity of real-time branching programs, in: Proc. FCT'87, Lecture Notes in Computer Science, vol. 278, Springer, Berlin, 1987, pp. 90–99.
[9] F.J. MacWilliams, N.J.A. Sloane, The Theory of Error-correcting Codes. Elsevier, North-Holland, 1977.
[10] 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.
[11] E. I. Nechiporuk, Ob odnoĭ bulevskoĭ funktsii, DAN SSSR 169(4) (1966) 765–766
E. I. Nečiporuk, On a Boolean function, Sov. Math. Doklady 7 (4) (1966) 999–1000.
[12] E. A. Okolnishnikova Nizhnie otsenki slozhnosti realizatsii kharakteristicheskikh funktsiĭ dvoichnykh kodov binarnymi programmami, Metody diskretnogo analiza 51 (1991) 61–83.
E. A. Okolnishnikova, Lower bounds for branching programs computing characteristic functions of binary codes, Metody Discretnogo Analiza 51 (1991) 61–83 (in Russian).
[13] A.A. Razborov, Lower bounds for deterministic and nondeterministic branching programs, in: Proc. FCT'91, Lecture Notes in Computer Science, vol. 529, Springer, Berlin, 1991, pp. 47–60.
[14] P. Savický, S. Žák, A lower bound on branching programs reading some bits twice, Theoret. Computer Sci. 172 (1997) 293–301.
[15] D. Sieling, New lower bounds and hierarchy results for restricted branching programs, J. Comput. System Sci. 53 (1) (1996) 79–87.
[16] D. Sieling, I. Wegener, New lower bounds and hierarchy results for Restricted Branching Programs, in: Proc. of Workshop on Graph-Theoretic Concepts in Computer Science WG'94, Lecture Notes in Computer Science, vol. 903, Springer, Berlin, 1994, pp. 359–370.

[17] I. Wegener, The Complexity of Boolean Functions, Wiley–Teubner, New York, 1987.
[18] I. Wegener, On the complexity of branching programs and decision trees for clique functions, J. ACM 35 (1988) 461–471.
[19] S. Žák, An exponential lower bound for one-time-only branching programs, in: Proc. MFCS'84, Lecture Notes in Computer Science, vol. 176, Springer, Berlin, 1984, pp. 562–566.
[20] S. Žák, A superpolynomial lower bound for $(1, +k(n))$-branching programs, in: Proc. MFCS'95, Lecture Notes in Computer Science, vol. 969, Springer, Berlin, 1995, pp. 319–325.