83

Theoretical Computer Science 64 (1989) 331–342
North-Holland

NOTE

ON NEČIPORUK'S THEOREM FOR BRANCHING PROGRAMS

Noga ALON[*] and Uri ZWICK

Department of Mathematics and Computer Science, Sackler Faculty of Exact Sciences, Tel-Aviv University, Tel-Aviv, Israel 69978

Communicated by M. Paterson
Received January 1988
Revised May 1988

Abstract. Nečiporuk's theorem yields lower bounds on the size of branching programs computing specific boolean functions. Specifically, if f is a boolean function, $V_1, \ldots, V_p$ is a partition of the set of variables of f, and $r_{V_i}(f)$ is the number of different restrictions of f to V_i, then the size of every branching program which computes f is at least

$$c \cdot \sum_{i=1}^{p} \frac{\log r_{V_i}(f)}{\log\log r_{V_i}(f)}$$

where c is some positive constant.

In this note we determine the largest monotone non-decreasing function $t(\cdot)$ for which Nečiporuk's theorem remains true when the above sum is replaced by $\sum_{i=1}^{p} t(r_{V_i}(f))$. We show that $t(m) \sim \frac{1}{2}\log m/(\log\log m)$ and obtain explicit formulae for it.

1. Introduction

We start with some basic definitions.

Definition 1.1 (*Boolean functions, assignments, restrictions*). A *boolean function* is a function $f:\{0,1\}^V \to \{0,1\}$ where V is a finite set. Suppose that $U \subseteq V$. A function $\alpha: U \to \{0,1\}$ is called an *assignment*. Denote by $f_\alpha:\{0,1\}^{V-U} \to \{0,1\}$ the function obtained from f by assigning the variables of U the values specified by α. The function f_α is called a *restriction* of f to $V-U$. Denote by $r_U(f)$ the number of different restrictions of f to U.

Definition 1.2 (*branching programs*). A *branching program* P is a directed acyclic graph, with a special vertex s, called the *source*, and two other special vertices called *sinks*. All the non-sink vertices are labeled by variables from the set $\{x_1, \ldots, x_n\}$ and the two sinks are labeled, respectively, by the constants 0 and 1. Every non-sink vertex has fan-out two and the edges leaving it are labeled by 0 and 1. Each

[*] Research supported in part by Allon Fellowship and by a Bat Sheva de Rothschild Grant.

assignment of values $b_1, \ldots, b_n$ to the input variables $x_1, \ldots, x_n$ defines a unique computation path which starts at s, leaves every non-sink vertex labeled by x_i through the edge labeled by b_i, and ends in a sink labeled by $f(b_1, \ldots, b_n)$. The function f is said to be the function computed by the program P. When we want to emphasize that s is the source of the program P we denote the program by (P, s).

The size of the program P is defined to be the number of non-sink vertices in it. If f is a boolean function, we denote by $BP(f)$ the minimal size of a branching program which computes f.

If P is a branching program, U is a subset of its variables, α is an assignment of values to the variables of U, and v, u are two vertices in P, we say that $v \rightarrow^{\alpha} u$ iff the unique computation path which starts at v, leaves every non-sink vertex labeled by a variable $x \in U$ through the edge labeled by $\alpha(x)$ and ends in a sink or a non-sink vertex labeled by a variable not in U passes through u.

The *descendants* of a vertex v in a branching program P are all the vertices in P (including v itself) to which there are directed paths from v.

Nečiporuk's theorem for branching programs states the following.

Theorem 1.3 ([2], see also [4]). *If $f:\{0,1\}^V \to \{0,1\}$ is a boolean function and if $V_1, \ldots, V_p$ is a partition of V then*

$$BP(f) \geq c \cdot \sum_{i=1}^{p} \frac{\log r_{V_i}(f)}{\log \log r_{V_i}(f)}$$

where c is some fixed positive constant.

(Here and throughout the paper all the logarithms are natural.)

Definition 1.4 (*The set Θ*). Denote by Θ the set of all functions $\bar{t}(\cdot)$ for which the relation $BP(f) \geq \sum_{i=1}^{p} \bar{t}(r_{V_i}(f))$ holds for every function f and every partition $V_1, \ldots, V_p$.

In this note we determine the largest monotone non-decreasing function $t(\cdot)$ in Θ. In Section 2 we define a function $T(\cdot)$ and define $t(\cdot)$ to be its integral inverse. We then show that $t \in \Theta$ and study the asymptotic behavior of $T(\cdot)$ and $t(\cdot)$. We prove that $t(m) \sim \frac{1}{2} \log m/(\log \log m)$. At the end of the section we obtain explicit formulae for $T(\cdot)$ (and therefore also for $t(\cdot)$). In Section 3 we show that if $\bar{t} \in \Theta$ then $\bar{t}(T(n)) \leq t(T(n)) = n$ for every $n \geq 1$. In particular, $t(\cdot)$ is the largest monotone non-decreasing function in Θ.

This shows that the function $\frac{1}{2} \log m/(\log \log m)$ is indeed the asymptotically best function which can be used in Theorem 1.3. The function $t(\cdot)$ defined in this note is slightly better than the functions used in the previous presentations of Nečiporuk's theorem. In [4], for example, a function $\bar{t}'(\cdot)$ which satisfies $\bar{t}'(m) \sim \frac{1}{3} \log m/(\log \log m)$ was used.

It is known that the bounds obtained using Nečiporuk's method cannot grow faster than $O(n/\log n)^2$. Sequences of functions for which the lower bounds using Nečiporuk's method have this asymptotic behaviour were built by Nečiporuk [2] and Paul [3].

Nečiporuk's method can also be used in order to obtain lower bounds on the formula complexity of boolean functions. More specifically, for every basis Ω there exists a positive constant c_Ω such that for every function f and every partition $V_1, \ldots, V_p$ of f's variables

$$L_\Omega(f) \geq c_\Omega \sum_{i=1}^{p} \log_2 r_{V_i}(f) - 1,$$

where $L_\Omega(f)$ is the formula complexity of f over Ω. Denote by B_k the basis which contains all the functions of k or less variables. Denote by $c_k = c_{B_k}$ the best constant for which the above relation always holds. In [5], the behavior of the sequence c_k is studied. It is shown there that $c_2 = 1/\log_2 5$, $c_3 = 1/(2\log_2 6)$, $c_4 = 1/(3\log_2 6)$. Furthermore, $c_5, \ldots, c_8$ are expressed as the limits of easily computed sequences. For the next values of c_k extremely tight approximations are obtained and it is shown that $c_k \sim \log_2 k/k^2$. In this note we combine methods similar to the ones used in [5] together with some new ideas.

2. The functions $T(\cdot)$ and $t(\cdot)$

The functions $T(\cdot)$ and $t(\cdot)$ are defined as follows.

Definition 2.1 (*The functions* $T(\cdot)$ *and* $t(\cdot)$)

$$T(n) = \max\left\{ r_V(f) \;\middle|\; \begin{array}{l} \text{there exists a branching program which computes } f \\ \text{in which exactly } n \text{ vertices are labeled by variables from } V \end{array} \right\}.$$

A moment's reflection shows that the value of $T(n)$ is finite although no bound was placed on the number of vertices labeled by variables not in V. (It certainly does not exceed the total number of boolean functions on V.)

The function $t(\cdot)$ is defined to be the integral inverse of $T(\cdot)$, that is:

$$t(m) = \min\{n \geq 1 \mid T(n) \geq m\}, \quad m \geq 1.$$

The motivation behind these definitions becomes clear in the next simple theorem.

Theorem 2.2. *If* $f: \{0,1\}^V \to \{0,1\}$ *is a boolean function and* $V_1, \ldots, V_p$ *is a partition of* V *then* $BP(f) \geq \sum_{i=1}^{p} t(r_{V_i}(f))$. *In other words* $t \in \Theta$.

Proof. Let P be a branching program of size $BP(f)$ which computes f. Denote by n_i the number of vertices in P labeled by variables from V_i. It is clear that $\sum_{i=1}^{p} n_i = BP(f)$. Notice now that $r_{V_i}(f) \leq T(n_i)$ and therefore $n_i \geq t(r_{V_i}(f))$. Summing up we get that $BP(f) = \sum_{i=1}^{p} n_i \geq \sum_{i=1}^{p} t(r_{V_i}(f))$. $\square$

In order to estimate and then to compute the function $T(\cdot)$ we introduce the notion of *canonical programs.*

Definition 2.3 (*Canonical programs*). A *canonical branching program* P is a branching program with n non-sink vertices each one of them labeled by a different variable from the set $\{x_1, \ldots, x_n\}$, and in which all the edges are directed from "small to large", i.e., if $x_i \to x_j$ is an edge in P then $i<j$. Notice that every function which can be defined using a canonical program can also be defined using a canonical program whose source is x_1. (If x_k is the source of the program then simply redirect the two edges emanating from x_1 into x_k.) Thus we assume that the source of each canonical program is x_1. Denote by CP_n the set of all canonical programs of size n. Denote by CF_n the set of all functions defined by the programs in CP_n. It is easy to see that $|CF_n| \leq |CP_n| = [(n+1)!]^2$.

Lemma 2.4. $T(n) \leq |CF_n| \leq [(n+1)!]^2$.

Proof. Let $f:\{0,1\}^V \to \{0,1\}$ be a function defined by a program P, let $U \subseteq V$ and suppose that exactly n vertices in P are labeled by variables from U and that $r_U(f) = T(n)$.

Relabel the non-sink vertices in P by distinct new variables. Denote the program obtained by P', denote by U' the set of variables which label the vertices formerly labeled by variables from U, and denote by f' the function computed by P'. It is easy to see that $r_{U'}(f') \geq r_U(f)$ and therefore also $r_{U'}(f') = T(n)$. Without loss of generality, we may assume that $U' = \{x_1, \ldots, x_n\}$. We can also order the variables in U' in such a way that if there is a directed path from the vertex labeled by x_i to the vertex labeled by x_j then $i<j$.

Notice now that when we assign constants to the variables not in U' we are left with a canonical program on the variables $x_1, \ldots, x_n$. Thus $T(n) = r_{U'}(f') \leq |CF_n| \leq [(n+1)!]^2$. □

In Lemma 3.3 below we show that in fact $T(n) = |CF_n|$.

Definition 2.5 (*The sets CF_n^* and the function $T^*(\cdot)$*). Denote by CF_n^* the set of CF_n functions which depend on all the variables $x_1, \ldots, x_n$. Denote $T^*(n) = |CF_n^*|$. It is clear that $T(n) = \sum_{k=0}^{n} \binom{n}{k} T^*(k)$.

Lemma 2.6. (i) *Let $P \in CP_n$ be a canonical program. Then (P, x_1) defines a function which depends on all the variables $x_1, \ldots, x_n$ if and only if P contains no parallel edges and the indegrees of all its vertices (except x_1) are nonzero.*

(ii) *Let P, $Q \in CP_n$ be two distinct canonical programs. Denote by f and g the functions defined by (P, x_1) and (Q, x_1). If f and g depend on all the variables $x_1, \ldots, x_n$ then $f \neq g$.*

Proof. (i): Clearly, if (P, x_1) contains a pair of parallel edges, or if some vertex in P different from x_1 has indegree 0, then the function defined by (P, x_1) does not depend on all the variables $x_1, \dots, x_n$.

We prove the converse by induction on n. If $n = 1$ then since P does not contain parallel edges, the function computed by P is either x_1 or $\bar{x}_1$. These functions do depend on the variable x_1.

Suppose now that $n > 1$. Since the indegree of x_2 is nonzero, there exists an $a \in \{0, 1\}$ for which $x_1 \rightarrow^a x_2$. Since P contains no parallel edges, $x_1 \rightarrow^{\bar{a}} x_k$ for some $k > 2$. (Actually, the $\bar{a}$-edge from x_1 may go into one of the sinks. The proof in this case is similar.) Denote by A and B the sets of all descendants of x_2, and of x_k respectively, in P. We know that $A \cup B = \{x_2, \dots, x_n\}$. By the induction hypothesis the function f_a, defined by the program (P, x_2), depends on all the variables of A, and the function $f_{\bar{a}}$, defined by the program (P, x_k) depends on all the variables of B. Since $x_2 \in A - B$, it is clear that $f_a \neq f_{\bar{a}}$ and therefore the function $f = (x_1 \oplus \bar{a}) f_a \vee (x_1 \oplus a) f_{\bar{a}}$, which is the function defined by (P, x_1), depends on all the variables $x_1, \dots, x_n$.

(ii): The proof is again by induction on n. For $n = 1$ the claim clearly holds. Suppose now that $n > 1$. Let k be the smallest index for which the edges emanating from x_k in P and Q are not identical. Denote by A (respectively by B), the set of all descendants of x_k in P (in Q respectively). Denote by P_A the subprogram of P induced by A and by the sinks, and by Q_B the subprogram of Q induced by B and the sinks. Denote by f_A, g_B the functions computed by (P_A, x_k) and (Q_B, x_k) respectively. There exists an assignment α to the variables $x_1, \dots, x_{k-1}$ such that $f_\alpha = f_A$ and $g_\alpha = g_B$. By part (i) of this lemma f_A and g_B depend on all the variables of A and B respectively. If $A \neq B$ then clearly $f_A \neq g_B$, otherwise the fact that $f_A \neq g_B$ follows from the induction hypothesis. Therefore $f_\alpha \neq g_\alpha$ which implies that $f \neq g$, as required. □

We can now prove the following theorem.

Theorem 2.7. $T(n) \geq T^*(n) > (k!)^{\lfloor n/k \rfloor}(n-k)!$ *for every* $1 \leq k \leq \frac{1}{2}n$.

Proof. We build more than $(k!)^{\lfloor n/k \rfloor}(n-k)!$ different canonical programs with no parallel edges in which the indegrees of all the vertices except x_1 are nonzero. Using the two parts of Lemma 2.6, we get that the functions defined by these programs (when x_1 serves as the source) are all distinct. Therefore $T(n) \geq T^*(n) = |CF_n^*| > (k!)^{\lfloor n/k \rfloor}(n-k)!$.

We assume for simplicity that $k \mid n$ (the general case is similar). The programs are constructed in the following way: We partition the n vertices $x_1, \dots, x_n$ into n/k blocks of k vertices each. Each adjacent pair of vertices in the first block is connected by an 1-edge. The 0-edges emanating from the vertices of the ith block, $i < n/k$, are directed into the vertices of the $(i+1)$st block in such a way that no two of them enter the same vertex. Notice that the indegree of every non-sink vertex is now exactly 1 and that the number of possible choices up to now is $(k!)^{(n/k)-1}$.

In order to complete the programs we have to specify the 0-edges emanating from the last layer (there are $(k+1)!$ choices), and the 1-edges emanating from the last vertex of the first block and from the vertices in the rest of the blocks. The number of choices here is $(n-k+1)!$ (notice that we are not allowed to choose parallel edges). This completes the proof. □

In fact, as pointed out by one of the referees, one can easily prove a result which is slightly stronger than the result of Theorem 2.7. For example, if $n=2^k-1$ we can get

$$T(n) > \left[\left(\frac{n+1}{2}\right)!\right]^2 \cdot \prod_{i=1}^{k} \left(\frac{n+1}{2^i}\right)!$$

using the following argument: partition the n nodes into $k-1$ blocks of size 1, $2, \ldots, 2^{k-1}$. Direct the edges emanating from a block into the next block in such a way that the indegree of each node (except the source) will be exactly one. In other words, construct a complete binary tree. The edges emanating from the last block can now be freely chosen. This result can be easily extended to cover the case where n is not of the specified form. However, the details are somewhat more cumbersome. As we shall see in the next corollary, the present form of Theorem 2.7 is sufficient for our purposes.

Corollary 2.8. $\log T(n) \sim 2n \log n$, $t(m) \sim \frac{1}{2} \log m/(\log\log m)$.

Proof. We know that $(k!)^{\lfloor n/k \rfloor}(n-k)! \leq T(n) \leq [(n+1)!]^2$ for every $1 \leq k \leq \frac{1}{2}n$. Choosing $k \sim n/\log n$ we immediately get that $\log T(n) \sim 2n \log n$ and therefore $t(m) \sim \frac{1}{2} \log m/(\log\log m)$. □

For the sake of completeness we state two formulae which can be used in order to compute the exact values of $T^*(n)$, and therefore also of $T(n)$ and $t(m)$. The proof of these formulae appears in the appendix. Note however that although these formulae determine $t(m)$ precisely for all m, they do not supply immediately the asymptotic behavior of the function $t(\cdot)$, which was determined in the last corollary.

Theorem 2.9

$$T^*(n) = (n+1)! \cdot n! - \sum_{k=1}^{n-1} \left\langle {n \atop k} \right\rangle T^*(k) \qquad n \geq 1$$

$$= \sum_{k=0}^{n-1} (-1)^k \left\{ {n \atop k} \right\} (n-k+1)!(n-k)! \quad n \geq 1,$$

where

$$\left\langle {n \atop k} \right\rangle = \left\langle {n-1 \atop k-1} \right\rangle + n(n-1)\left\langle {n-1 \atop k} \right\rangle \qquad k>1,\ n>1,$$

$$\left\{ {n \atop k} \right\} = (n-k+1)(n-k)\left\{ {n-1 \atop k-1} \right\} + \left\{ {n-1 \atop k} \right\}, \quad k>0,\ n>1$$

and

$$\left\langle {1 \atop k} \right\rangle = \begin{cases} 1 & \textit{if } k=1, \\ 0 & \textit{otherwise}; \end{cases} \qquad \left\{ {1 \atop k} \right\} = \begin{cases} 1 & \textit{if } k=0, \\ 0 & \textit{otherwise}. \end{cases}$$

3. Odd–even branch

The way the functions $T(\cdot)$ and $t(\cdot)$ were defined insures that $t \in \Theta$ (Theorem 2.2). In order to prove that if $t'(m) > t(m)$ for some m then $t' \notin \Theta$ or, pushed to the limit, to prove that if

$$t'(m_0) > t(m_0) \quad \text{and} \quad \bar{t}(m) = \begin{cases} t'(m_0) & \text{if } m = m_0, \\ 0 & \text{otherwise} \end{cases}$$

then $\bar{t} \notin \Theta$, we try to construct a sequence of functions $\{f_n\}_{n=1}^{\infty}$ such that the set of variables of f_n can be partitioned into $p_n + 1$ subsets $V_{n,1}, \ldots, V_{n,p_n}, U_n$ in such a way that $r_{V_{n,i}}(f) = m$ for all $1 \leq i \leq p_n$ and such that $p_n t(m)/BP(f_n) \to^{n\to\infty} 1$. In this section we are able to construct such sequences when $m = T(n)$ for some n. For these constructions we turn the famous odd–even sorting networks (see for example [1]) into branching programs.

Notice that if $t'(T(k)) > t(T(k)) = k$ for some k and $t'(1), t'(2), t'(3), t'(4) \geq 1$ then the fact that $t' \notin \Theta$ can be proved in a much simpler way. Simply build a canonical program $P \in CP_n$ which defines a function f for which there exists a subset V of the variables of size k for which $r_V(f) = T(k)$. Now, complete V into a partition of all the variables by defining singletons $U_1, \ldots, U_{n-k}$ for the remaining variables. Since $1 \leq r_{U_i}(f) \leq 4$ for every $1 \leq i \leq n-k$ we get that

$$BP(f) = n = k + (n-k) < t'(T(k)) + (n-k) \leq t'(r_V(f)) + \sum_{i=1}^{n-k} t'(r_{U_i}(f))$$

and therefore $t' \notin \Theta$.

The result we prove is of course more general. We begin by defining the odd–even branching programs.

Definition 3.1 (*odd–even branch*). For every $n \geq 1$ and every even $m \geq 2$ define $\mathrm{OEB}_{m,n}$ to be the branching program whose graph is composed of n layers of m vertices each, in which the connections between the layers are as shown in Fig. 1. More formally, denote the ith vertex in the jth layer by $x_{i,j}$ where $1 \leq i \leq m$ and

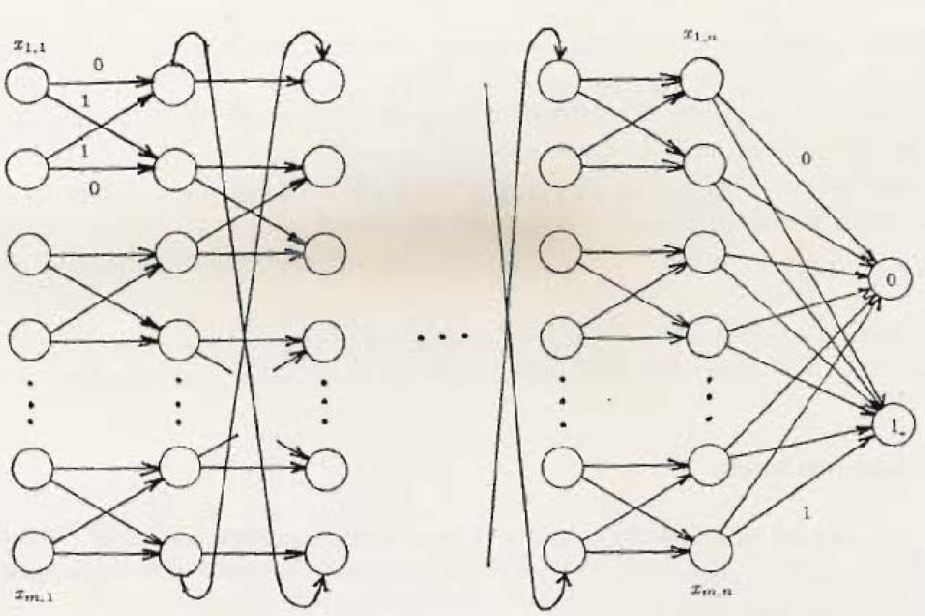

Fig. 1. Odd-even branching program.

$1 \le j \le n$. If $j < n$ then the 0-edge emanating from $x_{i,j}$ goes to $x_{i,j+1}$ and the 1-edge from $x_{i,j}$ goes to $x_{i+(-1)^{i+j},j+1}$ (the first index here is taken modulo m). The 0-edges of the vertices in the last layer go into the 0-sink and the 1-edges go into the 1-sink (as shown in Fig. 1). The non-sink vertices of $\mathrm{OEB}_{m,n}$ are labeled by distinct variables. For simplicity we use $x_{i,j}$ to denote also the variable labeling the vertex $x_{i,j}$. The source of $\mathrm{OEB}_{m,n}$ is $x_{1,1}$. The function computed by $\mathrm{OEB}_{m,n}$ is denoted by $f_{m,n}$.

Lemma 3.2. *Let $m \ge 1$ be an even integer. For every function $\pi:\{1,\ldots,m\}\to\{1,\ldots,m\}$ (not necessarily a permutation) there exists an assignment α of constants to the variables $\{x_{i,j} \mid 1 \le i \le m,\ 1 \le j \le m\}$ of $\mathrm{OEB}_{m,m+1}$ under which $x_{i,1} \to^{\alpha} x_{\pi(i),m+1}$ for every $1 \le i \le m$.*

Proof. If π is a permutation then the claim follows immediately from the fact that Odd–Even Sort is a sorting network.

Suppose now that π is not a permutation. Define the sets $A_i = \pi^{-1}(i)$ for $1 \le i \le m$. Choose a permuation π' such that if $A_i \ne \emptyset$ then $i \in \pi'(A_i)$ and such that π' is monotone decreasing on every A_i. We know that there exists an assignment α under which $x_{i,1} \to^{\alpha} x_{\pi'(i),m+1}$ for $1 \le i \le m$. We may assume that the paths $x_{i,1} \to^{\alpha} x_{\pi'(i),m+1}$ do not use the 1-edges $x_{1,i} \to x_{m,i+1}$ or $x_{m,i} \to x_{1,i+1}$ which are not present in the ordinary odd–even Sort. Since π' is decreasing on every A_i, we get that every pair of paths from the set $\{x_{j,1} \to^{\alpha} x_{\pi'(j),m+1} \mid j \in A_i\}$ must cross one another. In particular, if $A_i \ne \emptyset$ and $\pi'(k) = i$ then all the paths from $\{x_{j,1} \to^{\alpha} x_{\pi'(j),m+1} \mid j \in A_i\}$ cross the path $x_{k,1} \to^{\alpha} x_{i,m+1}$. Suppose now that $j \in A_i$ and that the paths $x_{j,1} \to^{\alpha} x_{\pi'(j),m+1}$ and $x_{k,1} \to^{\alpha} x_{i,m+1}$ cross one another between rows a and $a+1$, and between layers b

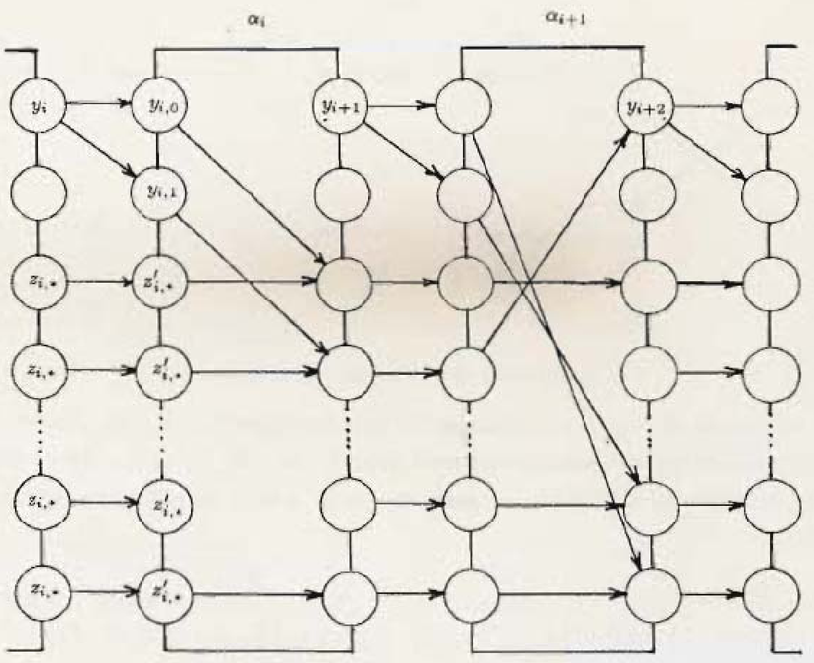

Fig. 3. Building the assignment α.

satisfies $y_{k0} \rightarrow^{\alpha_k} y_{a(k)}$, $y_{k1} \rightarrow^{\alpha_k} y_{b(k)}$, $z'_{k,0} \rightarrow^{\alpha_k} y_0$, $z'_{k,-1} \rightarrow^{\alpha_k} y_{-1}$ (again y_0 and y_{-1} are the sinks). The union of all these assignments forms the desired assignment α. □

Theorem 3.4. *If $\bar{t} \in \Theta$, then $\bar{t}(T(k)) \leqslant k$.*

Proof. Choose $m > (2k+7)/3$ and consider the sequence of functions $\{f_{m,n}\}_{n=1}^{\infty}$. The previous lemma shows that the variables of $f_{m,n}$ can be partitioned into subsets $V_1, \ldots, V_{p_n}$, U in such a way that $|V_i| = k$, $r_{V_i}(f_{m,n}) = T(k)$ for $1 \leqslant i \leqslant p_n$ and such that $(kp_n/mn) \rightarrow^{n\to\infty} 1$.

If $\bar{t} \in \Theta$ then we have

$$mn \geqslant BP(f_{m,n}) \geqslant \sum_{i=1}^{p_n} \bar{t}(r_{V_i}(f_{m,n})) = p_n \bar{t}(T(k))$$

or $\bar{t}(T(k)) \leqslant (mn/p_n) \rightarrow^{n\to\infty} k$ as required. □

Corollary 3.5. *If $\bar{t} \in \Theta$ and $\bar{t}$ is monotone non-decreasing, then $\bar{t}(k) \leqslant t(k)$ for every $k \geqslant 1$.*

Proof. If $\bar{t} \in \Theta$, $T(k-1) < m \leqslant T(k)$ and $\bar{t}(m) > t(m) = t(T(k))$ then we also have $\bar{t}(T(k)) \geqslant \bar{t}(m) > t(m) = t(T(k))$ which is a contradiction to the last theorem. □

Acknowledgment

We would like to thank the referees for their helpful comments.

Appendix

Proof of Theorem 2.9. According to Lemma 2.6, $T^*(n)$ is equal to the number of different canonical programs which define functions that depend on all the variables $x_1, \ldots, x_n$. We know that these functions do not contain parallel edges, and that the source of each one of them is x_1. The total number of canonical programs satisfying these two conditions is $(n+1)! \cdot n!$. From this number we have to subtract the number of canonical programs which satisfy the above two conditions but define functions that do not depend on all the variables $x_1, \ldots, x_n$.

Let A be a subset of $\{x_1, \ldots, x_n\}$ with $x_1 \in A$. If (P, x_1) is a program which defines a function that depends on all the variables of A and only on them, then it is clear that the edges emanating from A are directed into other vertices in A or into the sinks. If the function is to depend on all the variables of A, these edges can be chosen in $T^*(|A|)$ different ways. The edges emanating from the vertices not in A may now be chosen arbitrarily. The only constraint is that no parallel edges are allowed. These edges can be therefore chosen in $\prod_{x_j \notin A} (n-j+2)(n-j+1)$ different ways. Therefore

$$T^*(n) = (n+1)! \cdot n! - \sum_{\substack{A \subseteq \{x_1,\ldots,x_n\} \\ x_1 \in A}} T^*(|A|) \cdot \prod_{x_j \notin A} (n-j+2)(n-j+1)$$

$$= (n+1)! \cdot n! - \sum_{k=1}^{n-1} T^*(k) \cdot \sum_{\substack{A \subseteq \{x_1,\ldots,x_n\} \\ x_1 \in A \\ |A|=k}} \prod_{x_j \notin A} (n-j+2)(n-j+1).$$

Denote

$$\left\langle {n \atop k} \right\rangle = \sum_{\substack{A \subseteq \{x_1,\ldots,x_n\} \\ x_1 \in A \\ |A|=k}} \prod_{x_j \notin A} (n-j+2)(n-j+1)$$

$$= \sum_{\substack{A \subseteq \{1,\ldots,n\} \\ 1 \in A \\ |A|=k}} \prod_{j \notin A} (n-j+2)(n-j+1)$$

$$= \sum_{\substack{B \subseteq \{1,\ldots,n-1\} \\ |B|=n-k}} \prod_{j \in B} (j+1)j.$$

It is easy to check that $\langle {n \atop k} \rangle$ satisfy the required recursive relations.

Alternatively, we can count, using the principle of inclusion and exclusion, the number of canonical programs with x_1 as the source in which the indegrees of all the vertices, except x_1, are nonzero. Denote by A_i the set of the canonical programs in CP_n with no parallel edges in which the indegree of x_i is zero. It is easy to check that if $1 < i_1 < i_2 < \cdots < i_k \leq n$ then

$$|A_{i_1} \cap \cdots \cap A_{i_k}| = (n-k+1)! \cdot (n-k)! \cdot \prod_{j=1}^{k} (n-k+j-i_j+2)(n-k+j-i_j+1).$$

Therefore

$$\sum_{1<i_1<i_2<\cdots<i_k\leq n} |A_{i_1}\cap\cdots\cap A_{i_k}| = (n-k+1)!\cdot(n-k)!\cdot\left\{{n \atop k}\right\},$$

where

$$\left\{{n \atop k}\right\} = \sum_{1<i_1<i_2<\cdots<i_k\leq n}\prod_{j=1}^{k}(n-k+j-i_j+2)(n-k+j-i_j+1)$$

and therefore according to the principle of inclusion and exclusion

$$\begin{aligned} T^*(n) &= |A_2^c\cap\cdots\cap A_n^c| \\ &= (n+1)!\cdot n! - \sum_{k=1}^{n-1}(-1)^k \sum_{1<i_1<i_2<\cdots<i_k\leq n}|A_{i_1}\cap\cdots\cap A_{i_k}| \\ &= \sum_{k=0}^{n-1}(-1)^k\left\{{n \atop k}\right\}(n-k+1)!\cdot(n-k)!. \end{aligned}$$

Again, it is easy to check that $\{{n \atop k}\}$ satisfy the stated recursive relations. □

Using the above formulae we get

$$\begin{array}{lll} T^*(1)=2, & T^*(2)=8, & T^*(3)=56, \\ T^*(4)=608, & T^*(5)=9440, & T^*(6)=198272, \\ T^*(7)=5410688, & T^*(8)=186043904, & T^*(9)=7867739648 \end{array}$$

and also that

$$\begin{array}{lll} T(1)=4, & T(2)=14, & T(3)=88, \\ T(4)=890, & T(5)=13132, & T(6)=265286, \\ T(7)=7020256 & T(8)=235455602, & T(9)=9754845460. \end{array}$$

References

[1] D.E. Knuth, *The Art of Computer Programming, Vol. 3, Sorting and Searching* (Addison-Wesley, New York, 1973).

[2] E.I. Nečiporuk, A Boolean function, *Dokl. Akad. Nauk SSSR* **169**(4) (1966) (Russian); *Sov. Math. Dokl.* **7**(4) (1966) 999–1000 (English translation).

[3] W.J. Paul, A 2.5n-lower bound on the combinatorial complexity of boolean functions, *SIAM J. Comp.* **6**(3) (1977) 427–443.

[4] I. Wegener, *The Complexity of Boolean Functions*, Wiley–Teubner Series in Computer Science (1987).

[5] U. Zwick, Optimizing Nečiporuk's theorem, Technical Report 86/1987, Department of Computer Science, Tel-Aviv Univeristy.