ELSEVIER
Theoretical Computer Science 207 (1998) 329–342
Theoretical Computer Science

# Randomized Boolean decision trees: Several remarks

Nikolai K. Vereshchagin*,[1]

*Department of Mathematical Logic, Faculty of Mechanics and Mathematics, Moscow State University, Moscow 119899, Russia*

**Abstract**

Assume we want to show that (a) the cost of any randomized decision tree computing a given Boolean function is at least $c$. To this end, it suffices to prove that (b) there is a probability distribution over the set of all assignments to variables of that function with respect to which the average cost of any deterministic decision tree computing that function is at least $c$. Yao (1977) showed that this method is universal for proving lower bounds for randomized errorless decision trees, that is, that (a) is equivalent to (b). In the present paper we prove that this is the case also for randomized decision trees which are allowed to make errors. This gives the positive answer to the question posed in Yao (1977).

In the second part of the paper we exhibit an example when randomized directional decision trees (defined in Yao (1977)) to evaluate read once formulae are not optimal. We construct a formula $F_n$ of $n$ Boolean variables such that the cost of the optimal directional decision tree computing $F_n$ is $\Omega(n^\alpha)$ and there is an undirectional randomized decision tree computing $F_n$ of cost $O(n^\beta)$ for some $\beta<\alpha$. 

*Keywords:* Boolean decision trees; Randomized algorithm; Directional algorithms; Undirectional algorithms

## 1. Introduction

Boolean decision trees model is the most simple model to compute Boolean functions. In this model, the primitive operation made by an algorithm is evaluating an input Boolean variable. So the cost of a (deterministic) algorithm is the number of variables it evaluates on a worst case input. It is easy to find the deterministic complexity of all explicit Boolean functions (for most functions it is equal to the number of variables).

* E-mail: ver@mech.math.msu.su.

[1] This research was in part supported by the grant MQT300 from the International Science Foundation and by INTAS project N 93-0893.

0304-3975/98/$19.00 
*PII* S0304-3975(98)00071-1

A randomized Boolean decision tree algorithm is allowed to flip coins (without any charge for flipping). We consider two different notions of whether a randomized algorithm $R$ computes a function $f$: (1) the algorithm $R$ computes the function $f$ if on any input $a$ it outputs $f(a)$ with probability 1 (in this case we say that $R$ *computes* $f$) and (2) the algorithm $R$ is allowed to output a wrong answer with some probability $\varepsilon < \frac{1}{2}$, that is, for any input $a$, $\mathrm{Prob}[R(a) \neq f(a)] \leqslant \varepsilon$ (in this case we say that $R$ *$\varepsilon$-computes* $f$). The minimal cost of randomized algorithm computing $f$ according to the first (second) definition is called Las Vegas (Monte Carlo) complexity of $f$. The Monte Carlo complexity depends on $\varepsilon$, therefore we will call it *$\varepsilon$-complexity*. Thus, the Las Vegas complexity is the 0-complexity.

Due to simplicity of the computational model the analogs of many question being open for Turing machine complexity can be solved for Boolean decision trees and many of them are nontrivial.[2] For example, we have $\mathrm{P}^{\mathrm{dt}} \neq \mathrm{NP}^{\mathrm{dt}} \neq \text{Co-}\mathrm{NP}^{\mathrm{dt}}$, $\mathrm{P}^{\mathrm{dt}} = \mathrm{NP}^{\mathrm{dt}} \cap \text{Co-}\mathrm{NP}^{\mathrm{dt}} = \mathrm{BPP}^{\mathrm{dt}} = \mathrm{AM}^{\mathrm{dt}} \cap \text{Co-}\mathrm{AM}^{\mathrm{dt}}$ for decision trees analogs $\mathrm{P}^{\mathrm{dt}}$, $\mathrm{NP}^{\mathrm{dt}}$, $\text{Co-}\mathrm{NP}^{\mathrm{dt}}$, $\mathrm{BPP}^{\mathrm{dt}}$, $\mathrm{AM}^{\mathrm{dt}}$, $\text{Co-}\mathrm{AM}^{\mathrm{dt}}$ of the classes P, NP, Co-NP, BPP, AM, Co-AM [1–3, 5, 10]. An interesting unresolved question is whether $\mathrm{NP}^{\mathrm{dt}} = \mathrm{AM}^{\mathrm{dt}}$.

The known inequalities between the defined complexities are the following: the $\varepsilon$-complexity does not exceed the deterministic complexity; the deterministic complexity does not exceed squared Las Vegas complexity [1, 2, 10] and cubed $\varepsilon$-complexity [5] (within to a multiplicative constant depending on $\varepsilon$). It was shown by Snir [9] that the Las Vegas complexity can be less than deterministic complexity: he presented a function of $n$ variables having deterministic complexity $n$ and Las Vegas complexity $n^{0.753\ldots}$. There is unknown at present whether the Monte Carlo complexity can be essentially less than Las Vegas complexity. That is it is unknown whether their ratio is bounded by a constant.

In the present paper, we are interested in how to prove lower bounds for both randomized complexities. The following way to do this was presented by Yao [11]. Suppose there is a probability distribution $\mu$ on the set of all inputs to $f$ with respect to which the average cost of any deterministic algorithm computing $f$ is at least $c$. Then one can show that the Las Vegas complexity of $f$ is at least $c$. This method was used in [7] to prove exact lower bounds for certain Boolean function.

A similar method can be used to obtain lower bounds for the Monte Carlo complexity. Namely, assume there is a probability distribution $\mu$ with respect to which the average cost of any deterministic algorithm having the agreement probability with $f$ at least $1 - 2\varepsilon$ is $c$ or more. Then one can show that the $\varepsilon$-complexity of $f$ is at least $c/2$ [11].

Yao [11] showed that the described method to obtain lower bounds for Las Vegas complexity is universal. That is, for any $f$ there exists a probability distribution $\mu$ with respect to which the average cost of any deterministic algorithm computing $f$ is at least as large as the Las Vegas complexity of $f$. He asked whether this is the case for the Monte Carlo complexity. In the present paper we answer this question in affirmative:

[2] The analog of polynomial time is polylogarithmic number of probed variables.

we show that if the $\varepsilon$-complexity of $f$ is $c$ then there is a probability distribution $\mu$ with respect to which the average cost of any deterministic algorithm having the agreement probability with $f$ at least $1-\varepsilon/2$ is $c/2$ or more (Theorem 3.3). This shows that the described method to obtain lower bounds for Monte Carlo complexity is universal if we identify bounds differing by a multiplicative constant.

We present also the method by which the exact lower bounds for the Monte Carlo complexity can be proved (Theorem 3.4). Examples of the exact lower bounds for the Monte Carlo complexity were established in [8].

In the second part of the paper we compare different types of randomized algorithms to evaluate the so-called read once formulae.

A read once formula is a formula having connectivities $\wedge, \vee$ of any fanin and having exactly one occurrence of each its variable. The problem of evaluating read once formulae is closely related to the problem of evaluating game trees. (A read once formula with Boolean values assigned to its variables itself can be regarded as a tree of a game.) It is easy to see that deterministic complexity of any read once formula is equal to the number of its variables.

In the paper [7], Saks and Wigderson defined the so-called *directional algorithms* to evaluate read once formulae. The definition is given by induction on the number of connectivities in a read once formula $F$. If $F$ is a single variable then there exists the unique directional algorithm to evaluate $F$: evaluate the variable and return its value.

If $F=\bigwedge_{i=1}^{k} F_i$ or $F=\bigvee_{i=1}^{k} F_i$ a directional algorithm $R$ to evaluate $F$ is specified by a sequence of directional algorithms $R_1,\ldots,R_k$ to evaluate respectively $F_1,\ldots,F_k$ and a probability distribution on the set of permutations of $1,\ldots,k$. The algorithm $R$ is performed by selecting a permutation $\sigma$ of $1,\ldots,k$ according to this distribution and evaluating $F_1,\ldots,F_k$ in $\sigma$ order using $R_1,\ldots,R_k$ until the value of $F$ is known. It is easy to see that any directional algorithm to evaluate $F$ computes $F$.

Saks and Wigderson pointed out that not all read once formulae have an optimal algorithm which is directional. They gave no example of such formula, though. In the present paper we construct for all $n$ a read once formula $F_n$ of $n$ variables with $n^{0.}$ gap between Las Vegas complexity and the cost of optimal directional algorithm (Theorem 4.1).

## 2. Notation and preliminaries

Let us fix a set $\{x_1,\ldots,x_n\}$ of Boolean variables. The symbol $f$ will denote a Boolean function of these variables.

A *Boolean decision tree* is is a finite binary rooted tree whose leaves are labeled by zeros and ones, whose internal vertices are labeled by variables from the set $\{x_1,\ldots,x_n\}$, and for every internal vertex, one of the two outgoing edges is labeled by 0 and the other is labeled by 1. A Boolean decision tree computes the Boolean function $f$ of variables $x_1,\ldots,x_n$ defined as follows. Let $b_1,\ldots,b_n$ are Boolean values. Evidently, there exists a single path in the tree starting at the root and going to a leaf such that

for every pair $\langle u,v\rangle$ of consequent vertices in this path if $u$ is labeled by $x_i$, then the edge $\langle u,v\rangle$ is labeled by $b_i$. The value $f(b_1,\ldots,b_n)$ is defined as the label of the end leaf in this path.

Later we will call Boolean decision trees simply *algorithms*.

Let $A$ be an algorithm and let $a$ be an assignment of Boolean values to the variables $x_1,\ldots,x_n$. Denote by $A(a)$ the output value of $A$ on $a$. By $C(A,a)$ we denote the number of variables probed by $A$ on the assignment $a$. This value is called *the cost of A for a*. The *cost* $C(A)$ of an algorithm $A$ is defined as the number of probed variables for the worst assignment:

$$C(A)=\max_{a\in\mathscr{A}} C(A,a).$$

The cost of an algorithm is equal to its height provided any path in it has at most one occurrence of any variable.

The set of all assignments will be denoted by $\mathscr{A}$.

We denote by $\mathscr{D}$ the family of all Boolean decision trees. Denote by $\mathscr{D}(f)$ the family of deterministic algorithms computing $f$. The deterministic complexity $D(f)$ of the Boolean function $f$, $D(f)$, is defined as the minimal cost of an algorithm computing $f$:

$$D(f)=\min_{A\in\mathscr{D}(f)} C(A).$$

A randomized algorithm is a probability distribution over the family of deterministic algorithms. The probability of an algorithm $A$ with respect to the distribution $R$ is denoted by $R_A$. We interpret $R$ as an algorithm that has probability $R_A$ to proceed exactly as $A$. For a randomized algorithm $R$ and an assignment $a$ the cost of $R$ for $a$, $C(R,a)$, is defined as the expected number of probed variables for $a$:

$$C(R,a)=\sum_{A\in\mathscr{D}} R_A C(A,a).$$

We measure the *cost of a randomized algorithm* as the expected number of probed variables for the worst assignment: $C(R)=\max_{a\in\mathscr{A}} C(R,a)$.

Denote by $\mathrm{supp}(R)$ the set of all deterministic algorithms having positive measure with respect to probability distribution defining randomized algorithm $R$. We say that $R$ *computes* Boolean function $f$ if any deterministic algorithm in $\mathrm{supp}(R)$ computes $f$.

Let us define $e_f(A,a)$ to be equal to 1 if $A(a)\neq f(a)$ and to 0 else. In the sequel we denote by $e_f(R,a)$ the value

$$\mathrm{Prob}[R(a)\neq f(a)]=\sum_A R_A e_f(A,a).$$

Let $e_f(R)=\max_{a\in\mathscr{A}} e_f(R,a)$.

We say that $R$ *$\varepsilon$-computes* $f$ if $e_f(R)\leqslant\varepsilon$. It is easy to see that a randomized algorithm computes $f$ iff it 0-computes $f$.

The randomized complexity $R(f)$ of Boolean function $f$ is defined as the minimal cost of randomized algorithm computing $f$. The existence of an optimal randomized algorithm to compute any Boolean function is shown in [11].

Denote by $R^\varepsilon(f)$ the infimum of costs of randomized algorithms $\varepsilon$-computing $f$. Actually, Theorem 3.4 states that there is an optimal randomized algorithm $\varepsilon$-computing $f$.

The following inequalities between the above-defined complexities are known:

(1) $R^\varepsilon(f) \leqslant R(f) \leqslant D(f)$ for any $\varepsilon \in [0,1]$,

(2) $D(f) \leqslant (R(f))^2$ (proven independently in [1, 2, 10]),

(3) $D(f) \leqslant (1-2\varepsilon)^{-3}(R^\varepsilon(f))^3$ for any $\varepsilon \in [0, \frac{1}{2})$ (proven in [5]).

The randomized complexity can be less than deterministic complexity. The best-known example is due to Snir [9]: there exists a sequence of Boolean functions $f_n$ such that $D(f_n) = n$ and $R(f_n) = \mathcal{O}(n^\alpha)$, where $\alpha = \log_2(1+\sqrt{33})/4 = 0.753\ldots$ (see [7] for details). Snir's example yields also the largest known gap between $D(f)$ and $R^\varepsilon(f)$ (for $\varepsilon < \frac{1}{2}$).

## 3. Converting Yao's inequality

Suppose we want to prove that $R(f) \geqslant c$. This can be done as follows. Let $\mu$ be a probability distribution in the set of all assignments of values to the variables of $f$. Let $A$ be a deterministic algorithm. Denote by $C(A,\mu)$ the average cost of $A$ with respect to $\mu$: $C(A,\mu) = \sum_{a \in \mathscr{A}} \mu_a C(A,a)$. Assume that there exists $\mu$ such that $C(A,\mu) \geqslant c$ for any deterministic algorithm $A$ computing $f$. Then we can prove that $R(f) \geqslant c$. Indeed, let $R$ be randomized algorithm that computes $f$. Let $t$ be the average value of $C(A,\mu)$ when $A$ is taken at random with respect to $R$. Then $t \geqslant c$. On the other hand, $t$ is equal to the mean value of $C(R,a)$ when $a$ is taken at random with respect to the distribution $\mu$. Hence, there is $a \in \mathscr{A}$ such that $C(R,a) \geqslant c$, that is $C(R) \geqslant c$.

Yao observed that the well-known Minimax theorem by von Neumann [4] implies that this method to prove lower bounds on $R(f)$ is universal.

**Theorem 3.1** (Yao [11]). *For any Boolean function $f$,*

$$R(f) = \max_{\mu} \min_{A \in \mathscr{D}(f)} C(A,\mu).$$

Another Yao's theorem provides a method to obtain lower bounds for Monte Carlo complexity. Let $e_f(A,\mu)$ stand for the *non-agreement probability* of $A$ with $f$ with respect to $\mu$: $e_f(A,\mu) = \sum_{a \in \mathscr{A}} \mu_a e_f(A,a)$. Let $\mathscr{D}^\varepsilon_\mu(f)$ denote the set of deterministic algorithms which have non-agreement probability with $f$ at most $\varepsilon$ with respect to $\mu$, i.e.,

$$\mathscr{D}^\varepsilon_\mu(f) = \{A \mid e_f(A,\mu) \leqslant \varepsilon\},$$

**Theorem 3.2** (Yao [11]).

$$R^\varepsilon(f) \geqslant (\tfrac{1}{2}) \sup_{\mu} \min_{A \in \mathscr{D}^{2\varepsilon}_\mu(f)} C(A,\mu)$$

*for any* $0 \leqslant \varepsilon \leqslant \frac{1}{2}$.

Note that the inequality in Theorems 3.2 is proper. Indeed, let $f(x)=x$. It is easy to see that $R^{\varepsilon}(f)=1-2\varepsilon$ for any $\varepsilon\in[0,\frac{1}{2}]$. However, $\max_{\mu}\min_{A\in\mathscr{D}_{\mu}^{2\varepsilon}(f)}C(A,\mu)=0$ for any $\varepsilon\in[\frac{1}{4},\frac{1}{2}]$. So for $\varepsilon=\frac{1}{4}$ the inequality in Theorem 3.2 specializes to $\frac{1}{2}>0$.

Yao asked if one can show a similar inequality in the other direction. To answer this question we prove the following:

**Theorem 3.3.**

$$R^{\varepsilon}(f)\leqslant 2\sup_{\mu}\min_{A\in\mathscr{D}_{\mu}^{\varepsilon/2}(f)}C(A,\mu)$$

*for any* $0\leqslant\varepsilon\leqslant 1$.

This theorem is a corollary of other theorem, giving an equivalent definition of the value $R^{\varepsilon}(f)$ in the style of Theorem 3.1. In its formulation, $\sigma$ and $\tau$ denote probability distribution in the set of all assignments and $S$ denotes a non-negative real. Let $\mathscr{R}^{\varepsilon}(f)$ be the set of all randomized algorithms $\varepsilon$-computing $f$.

**Theorem 3.4.** *For any* $f$ *and any* $0\leqslant\varepsilon\leqslant 1$ *the following values exist and are equal:* $\min_{R\in\mathscr{R}^{\varepsilon}(f)}C(R)$ *and* $\max_{\tau,\sigma,S}\min_{A\in\mathscr{D}}(C(A,\tau)+S(e_f(A,\sigma)-\varepsilon))$.

To prove Theorem 3.4 we need the duality principle in the theory of linear programming (see, for example, [6]).

**Theorem 3.5** (Duality principle). *Let* $X$ *denote the set of non-negative real solutions to the system of linear inequalities*

$$\sum_{i=1}^{t}a_{ij}x_i\geqslant b_j,\quad j=1,2,\ldots,s. \tag{1}$$

*Let* $Y$ *denote the set of non-negative real solutions to the system of linear inequalities*

$$\sum_{j=1}^{s}a_{ij}y_j\leqslant c_i,\quad i=1,2,\ldots,t. \tag{2}$$

*If both* $X$ *and* $Y$ *are non-empty, then the linear function* $\sum_{i=1}^{t}c_ix_i$ *has minimum value on* $X$, *the linear function* $\sum_{j=1}^{s}b_jy_i$ *has maximum value on* $Y$ *and these values are equal.*

**Proof of Theorem 3.4.** The existence of the randomized algorithm of the cost at most $t$ $\varepsilon$-computing $f$ means that the system

$$\begin{aligned}
&-\sum_{A\in\mathscr{D}}x_AC(A,a)+t\geqslant 0\quad\text{for all } a\in\mathscr{A},\\
&-\sum_{A\in\mathscr{D}}x_Ae_f(A,a)\geqslant-\varepsilon\quad\text{for all } a\in\mathscr{A},\\
&\sum_{A\in\mathscr{D}}x_A=1.
\end{aligned}\tag{3}$$

has a non-negative solution $x_A$, $A \in \mathscr{D}$. It is easy to see that the last equation may be rewritten as inequality $\sum_{A\in\mathscr{D}} x_A \geqslant 1$.

By applying the duality principle we obtain that there exists the minimal $t_{\min}$ for which the system (3) is consistent and that $t_{\min}$ is equal to the maximum value of the function $-\varepsilon\sum_{a\in\mathscr{A}} w_a + z$, on the set of non-negative real $w_a$, $y_a$, $a \in \mathscr{A}$, $z$ satisfying the system of linear inequalities

$$-\sum_{a\in\mathscr{A}} y_a C(A,a) - \sum_{a\in\mathscr{A}} w_a e_f(A,a) + z \leqslant 0 \quad \text{for all } A \in \mathscr{D},$$

$$\sum_{a\in\mathscr{A}} y_a \leqslant 1. \tag{4}$$

We can apply the duality principle as both systems (3) and (4) are consistent.

It is easy to see that the maximum value of $-\varepsilon\sum_{a\in\mathscr{A}} w_a + z$ is achieved if

$$z = \min_{A\in\mathscr{D}}\left(\sum_{a\in\mathscr{A}} y_a C(A,a) + \sum_{a\in\mathscr{A}} w_a e_f(A,a)\right),$$

therefore the maximum value is equal to the maximum value of

$$-\varepsilon\sum_{a\in\mathscr{A}} w_a + \min_{A\in\mathscr{D}}\left(\sum_{a\in\mathscr{A}} y_a C(A,a) + \sum_{a\in\mathscr{A}} w_a e_f(A,a)\right),$$

where $w_a$, $y_a$, $a \in \mathscr{A}$, are non-negative numbers such that $\sum_{a\in\mathscr{A}} y_a \leqslant 1$. It is easy to see that the maximum value is achieved when $\sum_{a\in\mathscr{A}} y_a = 1$.

Let $\tau_a = y_a$, $S = \sum_{a\in\mathscr{A}} w_a$, $\sigma_a = w_a/S$. □

**Example 3.1.** Let us remind that $R^\varepsilon(f) = 1 - 2\varepsilon$ for $f(x) = x$ and for any $\varepsilon \in [0, \frac{1}{2}]$. So by Theorem 3.4 the maximum value of $\min_{A\in\mathscr{D}}(C(A,\tau) + S(e_f(A,\sigma) - \varepsilon))$ is equal to $1 - 2\varepsilon$. The maximum is achieved on the uniform distributions $\tau$ and $\sigma$ and $S = 2$. Indeed, if an algorithm $A$ does not ask the value of $x$ and outputs a constant then $C(A,\mu) = 0$ and $S(e_f(A,\sigma) - \varepsilon) = 1 - 2\varepsilon$. If $A$ evaluates $x$ and outputs its value then $C(A,\tau) = 1$ and $S(e_f(A,\sigma) - \varepsilon) = -2\varepsilon$.

This example can be extended to $\langle\tau,\sigma,S\rangle$ needed to obtain the lower bound by Santha (see [8]) for Monte Carlo complexity of read once formulae.

**Proof of Theorem 3.3.** It suffices, by Theorem 3.4, to show that for any $0 \leqslant \varepsilon \leqslant 1$

$$\max_{\tau,\sigma,S}\min_{A\in\mathscr{D}}(C(A,\tau) + S\cdot[e_f(A,\sigma) - \varepsilon]) \leqslant 2\sup_{\mu}\min_{A\in\mathscr{D}_\mu^{\varepsilon/2}(f)} C(A,\mu).$$

Let us take arbitrary $0 \leqslant \varepsilon \leqslant 1$. We will prove that for all $\tau,\sigma,S$ there is $\mu$ such that

$$\min_{A\in\mathscr{D}}(C(A,\tau) + S\cdot[e_f(A,\sigma) - \varepsilon]) \leqslant 2\min_{A\in\mathscr{D}_\mu^{\varepsilon/2}(f)} C(A,\mu).$$

Let us take arbitrary $\tau, \sigma, S$. Let $\mu$ be equal to the arithmetic mean of the distributions $\tau, \sigma$: $\mu = \tau/2 + \sigma/2$. We will show that

$$C(A,\tau) + S \cdot (e_f(A,\sigma) - \varepsilon) \leqslant 2C(A,\mu)$$

for all $A \in \mathscr{D}_\mu^{\varepsilon/2}(f)$.

Let us choose arbitrary $A \in \mathscr{D}_\mu^{\varepsilon/2}(f)$. We have by definition $e_f(A,\mu) < \varepsilon/2$. Since $e_f(A,\mu) = e_f(A,\sigma)/2 + e_f(A,\tau)/2$, we have $e_f(A,\sigma) < \varepsilon$, hence,

$$C(A,\tau) + S \cdot [e_f(A,\sigma) - \varepsilon] \leqslant C(A,\tau) = 2C(A,\mu) - C(A,\sigma) \leqslant 2C(A,\mu). \qquad \square$$

## 4. Examples of read once formulae for which undirectional algorithms are better than directional ones

We need first to recall some notions and results from [7].

Let $R$ be a randomized algorithm computing the value of read once formula $F$. Denote by $C_0(R)$ $[C_1(R)]$ the $\max_{a:F(a)=0} C(R,a)$ $[\max_{a:F(a)=1} C(R,a)]$. Let $d_0(F)$ $[d_1(F)]$ denote the minimum of $C_0(R)$ $[C_1(R)]$ over all directional algorithm $R$ evaluating $F$. Let $d(F) = \max(d_0(F), d_1(F))$.

Saks and Wigderson proved that there is a directional algorithm $R$ for which $C_0(R) = d_0(F)$ and $C_1(R) = d_1(F)$ and found the following recurrences for $d_0(F)$ and $d_1(F)$:

$$d_1(G \wedge H) = d_1(G) + d_1(H),$$

$$d_0(G \wedge H) = \max\left\{ d_0(G), d_0(H), \frac{d_0(G)d_1(G) + d_0(H)d_1(H) + d_1(G)d_1(H)}{d_1(G) + d_1(H)} \right\},$$

$$d_0(G \vee H) = d_0(G) + d_0(H),$$

$$d_1(G \vee H) = \max\left\{ d_1(G), d_1(H), \frac{d_0(G)d_1(G) + d_0(H)d_1(H) + d_0(G)d_0(H)}{d_0(G) + d_0(H)} \right\}.$$

In the sequel, we will use the following easy observation:

$$d_0(G) > \frac{d_0(G)d_1(G) + d_0(H)d_1(H) + d_1(G)d_1(H)}{d_1(G) + d_1(H)}$$

iff $d_0(G) > d_1(G) + d_0(H)$ (in this case $d_0(G \wedge H) = d_0(G)$).

Our goal is to construct for all $n$ a read once formula $F_n$ of $n$ variables with $n^\varepsilon$ gap between $R(F_n)$ and $d(F_n)$.

### 4.1. The first example

A list of variables like $v_1, v_2, \ldots, v_k$ will be denoted by $\vec{v}$.

Consider the following example. Let $F(\vec{x}) = \bigvee_{i=1}^{11} x_i$, $G = (F(\vec{x}) \wedge z) \wedge (F(\vec{y}) \wedge u)$.

It is easy to see that the optimal way to evaluate $F(\vec{x})$ is to evaluate variables in a random order until 1 is found. So we get $d_0(F) = 11$ and $d_1(F) = 6$.

As $d_0(F) > d_1(F) + d_0(z)$, by applying the above recurrences we obtain $d_0(F(\vec{x}) \wedge z) = 11$, $d_1(F(\vec{x}) \wedge z) = 7$, and hence $d_0(G) = 14.5$, $d_1(G) = 14$.

Let us construct an algorithm $R$ to evaluate $G$ such that $C_0(R) = C_1(R) = 14$. The randomized algorithm $R$ is performed as follows: evaluate first $F(\vec{x})$ and $F(\vec{y})$ in a random order (using the optimal algorithm to evaluate $F$) and then evaluate $z$ and $u$ in a random order (the evaluating stops if 0 is found).

Obviously, $C_1(R) = 14$. Let us find $C_0(R)$.

If $F(\vec{x}) = 0$ or $F(\vec{y}) = 0$ for an assignment, then the cost of $R$ for that assignment is at most $\frac{1}{2}(11 + 6) + \frac{1}{2} \cdot 11 = 14$ achieving 14 when $F(\vec{x}) = 0$ and $F(\vec{y}) = 1$. If $F(\vec{x}) = F(\vec{y}) = 1$ and $z = 0$ or $u = 0$ then the cost of $R$ is at most $6 + 6 + \frac{1}{2}(1 + 1) + \frac{1}{2} = 13.5$. So $C_0(R) = 14$.

So directional algorithms are not optimal to evaluate the formula $G$. However, this example has the following minor point. We used the fact that the family of directional algorithm to evaluate the formula $x \wedge z \wedge y \wedge u$ (using an AND of fanin 4) is wider than the family of directional algorithm to evaluate the formula $(x \wedge y) \wedge (z \wedge u)$. Namely, a directional algorithm to evaluate $x \wedge z \wedge y \wedge u$ is allowed to probe variables in the order, say, $z, u, x, y$ but no directional algorithm to evaluate $(x \wedge y) \wedge (z \wedge u)$ is allowed to do so.

We shall construct now an example without this minor point, that is, we shall construct a formula in which ANDs and ORs alternate. Let us use the same idea as in the above example.

### 4.2. *The second example*

Let

$$F(\vec{x}) = \bigvee_{i=1}^{39} x_i,$$

$$H(\vec{z}) = [(z_1 \vee z_2) \wedge (z_3 \vee z_4)] \vee [(z_5 \vee z_6) \wedge (z_7 \vee z_8)],$$

$$I(\vec{x}, \vec{z}) = F(\vec{x}) \wedge H(\vec{z}),$$

$$J(\vec{x}, \vec{z}, t) = I(\vec{x}, \vec{z}) \vee t,$$

$$G(\vec{x}, \vec{z}, t, \vec{y}, \vec{u}, s) = J(\vec{x}, \vec{z}, t) \wedge J(\vec{y}, \vec{u}, s).$$

It is easy to see that $d_0(F) = 39$ and $d_1(F) = 20$. By applying the above recurrences we obtain $d_0(H) = 5.5$, $d_1(H) = 4\frac{3}{8}$, $d_0(I) = 39$, $d_1(I) = 24\frac{3}{8}$, $d_0(J) = 40$, $d_1(J) = \frac{7925}{320}$, $d_0(G) = \frac{33525}{640} = 52.38\ldots$, $d_1(G) = \frac{7925}{160} = 49.53\ldots$.

Let us construct a randomized undirectional algorithm $R$ to evaluate $G$ having smaller cost. The algorithm $R$ is performed as follows. Evaluate first $t$ and $s$. Then evaluate $F(\vec{x})$ and $F(\vec{y})$ in a random order (using the optimal algorithm to evaluate $F$) and then evaluate $H(\vec{z})$ and $H(\vec{u})$ in a random order (the evaluating stops if the value of $G$ is found). It is easy to see that the both values $C_0(R, \langle \vec{x}, \vec{z}, t, \vec{y}, \vec{u}, s \rangle)$, $C_1(R, \langle \vec{x}, \vec{z}, t, \vec{y}, \vec{u}, s \rangle)$ are achieved for assignments in which $t = s = 0$.

If $F(\vec{x})=F(\vec{y})=H(\vec{z})=H(\vec{u})=1$, then the cost of $R$ is at most $d_0(t)+d_0(s)+2d_1(F)+2d_1(H)=1+1+2\times 20+2\times 4\frac{3}{8}=50\frac{3}{4}$, and the value $50\frac{3}{4}$ is achieved. So $C_1(R)=50\frac{3}{4}$.

Let us find $C_0(R)$. If $F(\vec{x})=0$ or $F(\vec{y})=0$, then $C(R,\langle\vec{x},\vec{z},t,\vec{y},\vec{u},s\rangle)=d_0(s)+d_0(t)+\frac{1}{2}(d_0(F)+d_1(F))+\frac{1}{2}d_0(F)=1+1+\frac{1}{2}(39+20)+\frac{1}{2}\times 39=51$. If $F(\vec{x})=F(\vec{y})=1$ and $H(\vec{z})=0$ or $H(\vec{u})=0$ $C(R,\langle x,y,z,u\rangle)$ does not exceed $d_0(s)+d_0(t)+2d_1(F)+\frac{1}{2}(d_0(H)+d_1(H))+\frac{1}{2}d_0(H)=1+1+2\times 20+\frac{1}{2}(5.5+4\frac{3}{8})+\frac{1}{2}\times 5.5=49\frac{11}{16}$.

So $C_0(R)=51$.

Thus $C(R)=51<d_0(G)=52.38\ldots$ . Therefore, any optimal randomized algorithm to evaluate $G$ is not directional.

### 4.3. *The example of read once formula with $n^\varepsilon$ gap between the cost of optimal directional and undirectional algorithms*

**Theorem 4.1.** *There are $0<\beta<\alpha$ and $c,d>0$ such that the following holds. For all $n$ there exists a read once formula $G_n$ of $n$ variables having alternate ANDs and ORs such that $d(G_n)>cn^\alpha$ and $R(G_n)<dn^\beta$.*

**Proof.** Let us make use of the formula $G$ from the last example. Let us define the sequence of read once formulae $\{G_i\}$ by letting $G_0=v$ (a variable) and $G_{i+1}=G(G_i(\vec{v}^1),G_i(\vec{v}^2),\ldots,G_i(\vec{v}^{96}))$. Here $v^j$ stands for the list of variables $v_1^j,v_2^j,\ldots,v_{k_i}^j$, where $k_i=96^i$ is the number of variables in the formula $G_i$.

The values $d_0(G_i)$ and $d_1(G_i)$ can be found inductively using Saks and Wigderson's recurrences. This routine task is done in the Addendum. Here is the result:

$$\begin{aligned} d_0(G_{i+1}) &= \tfrac{32503}{640}d_0(G_i)+\tfrac{511}{320}d_1(G_i),\\ d_1(G_{i+1}) &= \tfrac{6903}{320}d_0(G_i)+\tfrac{511}{80}d_1(G_i). \end{aligned} \tag{5}$$

Let us define an undirectional algorithm $R_i$ to evaluate $G_i$ by induction on $i$. To perform $R_0$ evaluate the variable and output its value.

The algorithm $R_{i+1}$ is performed as follows: run the above algorithm $R$ evaluating $G$ but instead evaluating variables of $G$ run $R_i$.

Let us denote by $a_0(i)$ $[a_1(i)]$ the value $C_0(R_i)$ $[C_1(R_i)]$.

The following recurrences are proved in the Addendum:

$$\begin{aligned} a_0(i+1) &= 50.5a_0(i)+0.5a_1(i),\\ a_1(i+1) &= 44\tfrac{1}{4}a_0(i)+6\tfrac{1}{2}a_1(i). \end{aligned} \tag{6}$$

It is well known that the solution to the system (5) has the form

$$d_0(i)=p_1\lambda_1^i+p_2\lambda_2^i,\quad d_1(i)=q_1\lambda_1^i+q_2\lambda_2^i,$$

where $\lambda_1, \lambda_2$ are the characteristic values of the matrix

$$\begin{pmatrix} \frac{32503}{640} & \frac{511}{320} \\ \frac{6903}{160} & \frac{511}{80} \end{pmatrix} = \begin{pmatrix} 50\frac{503}{640} & 1\frac{191}{320} \\ 43\frac{23}{160} & 6\frac{31}{80} \end{pmatrix}$$

The computation yields $\lambda_1 = 4.88\ldots$, $\lambda_2 = 52.28\ldots$ .

The solution to system (6) has the from

$$a_0(i) = r_1\mu_1^i + r_2\mu_2^i, \qquad a_1(i) = s_1\mu_1^i + s_2\mu_2^i,$$

where $\mu_1 = 6.00\ldots$, $\mu_2 = 50.99\ldots$ are the characteristic values of the matrix

$$\begin{pmatrix} 50.5 & 0.5 \\ 44\frac{1}{4} & 6\frac{1}{2} \end{pmatrix}.$$

All we need is the inequality $\lambda_2 < \mu_2$. This inequality can be proved without complicated computation as follows. Let us prove that $\lambda_2 < 51 < \mu_2$. To this end, let us substitute the number 51 in the characteristic polynomial of both matrices. This substitution can be done by subtracting 51 from diagonal elements of matrices and computing the determinants of the resulting matrices:

$$\begin{Vmatrix} -\frac{137}{640} & 1\frac{191}{320} \\ 43\frac{23}{160} & -44\frac{49}{80} \end{Vmatrix} < 0$$

and

$$\begin{Vmatrix} -0.5 & 0.5 \\ 44.25 & -44.5 \end{Vmatrix} > 0.$$

We shall write $f(i) = \Theta(g(i))$ if there are constants $c_1, c_2$ such that $c_1 g(i) \leqslant f(i) \leqslant c_2 g(i)$.

Thus, we have

$$d_0(G_i) = \Theta(\lambda_2^i), \qquad d_1(G_i) = \Theta(\lambda_2^i),$$

$$a_0(i) = \Theta(\mu_2^i), \qquad a_1(i) = \Theta(\mu_2^i).$$

Recall that the number of variables $n_i$ in $G_i$ is $96^i$. So we have

$$d(G_i) = \Theta(n^{\log_{96}\lambda_2}) \quad \text{and} \quad R(G_i) = \mathcal{O}(n^{\log_{96}\mu_2}). \qquad \square$$

If $n$ is not of the form $96^i$ then the formula $F_n$ can be constructed by adding fictitious variables.

More precise computation shows that our example yields the difference $0.0058\ldots$ between $\alpha$ and $\beta$ in the formulation of Theorem 4.1. A refinement of that example gives the difference $0.0077\ldots$ . It is interesting whether the difference between two exponents

can be significantly greater, say, greater than 0.05. If this is not the case then we can restrict ourselves with directional algorithms for evaluation read once formulae being sure that our loss is very small. This would be nice because the optimal directional algorithm can be found in polynomial time given a formula. In the case of binary gates (or, when the fanin of gates is bounded by a constant) this was shown in [7]. In the case of unbounded fanin a polynomial algorithm was constructed by A. Evfimjevsky (unpublished).

## Addendum

Let us prove the equalities (5). To prove them we need the inequality $d_0(G_i) \geqslant d_1(G_i)$. For $n=0$ we have $d_0(G_0)=d_1(G_0)=1$. It is easy to verify that if $d_0(G_i) \geqslant d_1(G_i)$ and $d_0(G_{i+1})$, $d_1(G_{i+1})$ satisfy (5) then $d_0(G_{i+1}) \geqslant d_1(G_{i+1})$. So we can prove the equalities (5) assuming that $d_0(G_i) \geqslant d_1(G_i)$.

Let us define $F_{i+1}, H_{i+1}, I_{i+1}, J_{i+1}$ as formulae obtained in the same way from $F, H, I, J$ as $G_{i+1}$ is obtained from $G$. For example, $F_{i+1}(\vec{x}^1, \ldots, \vec{x}^{39}) = \bigvee_{j=1}^{39} G_i(\vec{x}^j)$. By applying three times the Saks and Wigderson's recurrences, we obtain

$$d_0(H_{i+1}) = 4.5d_0(G_i) + d_1(G_i),$$

$$d_1(H_{i+1}) = 2.125d_0(G_i) + 2.25d_1(G_i).$$

We have

$$d_0(F_{i+1}) = d_0\left(\bigvee_{i=1}^{39} G_i(\vec{v}^i)\right) = 39d_0(G_i),$$

$$d_1(F_{i+1}) = d_1\left(\bigvee_{i=1}^{39} G_i(\vec{v}^i)\right) = 19d_0(G_i) + d_1(G_i),$$

$$d_0(I_{i+1}) = d_0(F_{i+1} \wedge H_{i+1}) = d_0(F_{i+1}) = 39d_0(G_i).$$

The last equality is true since

$$d_0(F_{i+1}) \geqslant d_1(F_{i+1}) + d_0(H_{i+1}),$$

which follows from the assumption $d_0(G_i) \geqslant d_1(G_i)$. Further,

$$d_1(I_{i+1}) = d_1(F_{i+1} \wedge H_{i+1}) = d_1(F_{i+1}) + d_1(H_{i+1})$$

$$= 21.125d_0(G_i) + 3.25d_1(G_i).$$

Thus, we have

$$\begin{aligned} d_0(J_{i+1}) &= d_0(I_{i+1} \vee G_i) = d_0(I_{i+1}) + d_0(G_i) = 40 d_0(G_i), \\ d_1(J_{i+1}) &= d_0(I_{i+1} \vee G_i) \\ &= \frac{d_0(I_{i+1})d_1(I_{i+1}) + d_0(G_i)d_1(G_i) + d_0(I_{i+1})d_0(G_i)}{d_0(I_{i+1}) + d_0(G_i)} \\ &= \tfrac{6903}{320} d_0(G_i) + \tfrac{511}{160} d_1(G_i). \end{aligned}$$

The last equality to be true we need the inequalities

$$d_1(I_{i+1}) \leqslant d_0(I_{i+1}) + d_1(G_{i+1}), \qquad d_1(G_{i+1}) \leqslant d_0(G_{i+1}) + d_1(I_{i+1}),$$

which are true under assumption $d_0(G_i) \geqslant d_1(G_i)$. Finally,

$$\begin{aligned} d_0(G_{i+1}) &= d_0(J_{i+1}) + 0.5 d_1(J_{i+1}) = \tfrac{32503}{640} d_0(G_i) + \tfrac{511}{320} d_1(G_i), \\ d_1(G_{i+1}) &= 2 d_1(J_{i+1}) = \tfrac{6903}{160} d_0(G_i) + \tfrac{511}{80} d_1(G_i). \end{aligned}$$

It is easy to see that the inductive assumption $d_0(G_i) \geqslant d_1(G_i)$ implies $d_0(G_{i+1}) \geqslant d_1(G_{i+1})$. So we have

$$\begin{aligned} d_0(G_{i+1}) &= \tfrac{32503}{640} d_0(G_i) + \tfrac{511}{320} d_1(G_i), \\ d_1(G_{i+1}) &= \tfrac{6903}{320} d_0(G_i) + \tfrac{511}{80} d_1(G_i). \end{aligned}$$

Let us prove Eq. (6).

Let us remind that both values $C_0(R)$ and $C_1(R)$ are achieved for assignments in which $t = s = 0$.

Obviously, $a_1(i+1) = 2a_0(i) + 2 \times 19 a_0(i) + 2a_1(i) + 2 \times 2\frac{1}{8} a_0(i) + 2 \times 2\frac{1}{4} a_1(i) = 44\frac{1}{4} a_0(i) + 6\frac{1}{2} a_1(i)$.

Let us find $a_0(i+1)$. If at least one of two copies of $F_i$ is zero, then the cost of $R$ for that assignment is at most $2a_0(i) + \frac{1}{2}(39a_0(i) + 19a_0(i) + a_1(i)) + \frac{1}{2} \times 39 a_0(i) = 50.5 a_0(i) + 0.5 a_1(i)$. If both copies of $F_i$ are one and at least one copy of $H_i$ is zero then the cost of $R$ is at most $2a_0(i) + 2(19a_0(i) + a_1(i)) + \frac{1}{2}(4.5a_0(i) + a_1(i) + 2\frac{1}{8} a_0(i) + 2\frac{1}{4} a_1(i)) + \frac{1}{2}(4.5a_0(i) + a_1(i)) = 45\frac{9}{16} a_0(i) + 4\frac{1}{8} a_1(i)$. Assume that $a_0(i) \geqslant a_1(i)$ (this is proved by induction). Then $50.5 a_0(i) + 0.5 a_1(i) \geqslant 45\frac{9}{16} a_0(i) + 4\frac{1}{8} a_1(i)$ and we get $a_0(i+1) = 50.5 a_0(i) + 0.5 a_1(i)$.

It is easy to see that the assumption $a_0(i) \geqslant a_1(i)$ implies that $a_0(i+1) \geqslant a_1(i+1)$. Thus,

$$\begin{aligned} a_0(i+1) &= 50.5 a_0(i) + 0.5 a_1(i), \\ a_1(i+1) &= 44\tfrac{1}{4} a_0(i) + 6\tfrac{1}{2} a_1(i). \end{aligned}$$

## References

[1] M. Blum, R. Impagliazzo, General oracle and oracle classes, Proc. 28th Annual IEEE Symp. on Foundation of Computer Science, New York, May 1987, pp. 118–126.

[2] J. Hartmanis, L. Hemachandra, Complexity classes without machines: on complete languages for UP, Theoret. Comput. Sci. 58 (1988) 129–142 (Preliminary version appeared in: Internat. Colloq. on Automata, Languages and Programming, 1986, Lecture Notes in Computer Science, vol. 226, 1986, pp. 123–135.).

[3] R. Impagliazzo, M. Naor, Decision trees and downward closures, 3rd Annual Conf. on Structure in Complexity Theory, 1988, pp. 29–38.

[4] V. Neumann, Zur teorie der gesellschaftspiele, Math. Ann. 100 (1928) 295–320.

[5] N. Nisan, Probabilistic versus deterministic decision trees and CREW PRAM complexity, Proc. 21th Ann. ACM Symp. on Theory of Computing, 1989, pp. 327–335.

[6] C. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[7] M. Saks, A. Wigderson, Probabilistic Boolean decision trees and the complexity of evaluating game trees, Proc. 27th Ann. IEEE Symp. on Foundation of Computer Science, 1986, pp. 29–38.

[8] M. Santha, On the Monte Carlo boolean decision tree complexity of read-once formulae, Proc. 6th Ann. Conf. on Structure in Complexity Theory, 1991, pp. 180–187.

[9] M. Snir, Lower bounds for probabilistic linear decision trees, Theoret. Comput. Sci. 38 (1985) 69–82.

[10] G. Tardos, Query complexity or why is it difficult to separate $NP^A \cap Co\text{-}NP^A$ from $P^A$ by a random oracle, Combinatorica 9 (1990) 385–392.

[11] A.C.-C. Yao, Probabilistic computations: toward a unified measure of complexity, Proc. 18th Ann. IEEE Symp. on Foundation of Computer Science, 1977, pp. 222–227.