A Randomized Competitive Algorithm for Evaluating Priced AND/OR Trees

Eduardo Sany Laber
e-mail: laber@inf.puc-rio.br

PUC-RioInf.MCC25/03 Agosto 2003

Abstract: Recently, Charikar et. al. investigated the problem of evaluating AND/OR trees, with non uniform costs on its leaves, under the perspective of the competitive analysis. For an AND-OR tree T they presented a $\mu(T)$-competitive deterministic polynomial time algorithm, where $\mu(T)$ is the number of leaves that must be read, in the worst case, in order to determine the value of T. Furthermore, they prove that $\mu(T)$ is a lower bound on the deterministic competitiveness, which assures the optimality of their algorithm.

The power of randomization in this context has remained as an open question. Here, we give a step towards solving this problem by presenting a $0.792\mu(T)$-competitive randomized polynomial time algorithm. This contrasts with the best known lower bound $\mu(T)/2$.

Key Words: Randomized Algorithms, AND-OR Trees, Competitive Analysis.

Resumo: Recentemente, Charikar et. al. investigaram o problema de avaliar árvores AND/OR com custos não uniformes em suas folhas, sob a ótica da análise competitiva. Para uma árvore AND-OR T eles apresentam um algoritmo determinístico $\mu(T)$-competitivo que executa em tempo polinomial, onde $\mu(T)$ é o número de folhas que deve ser lida, no pior caso, para determinar o valor de T. Além disso, eles provam que $\mu(T)$ é um limite inferior para a competitividade determinística, o que assegura a otimalidade do algoritmo proposto.

O poder da aleatorização neste contexto ficou em aberto. Neste artigo, damos um passo na direção de resolver este problema ao propor um algoritmo aleatorizado polinomial $0.792\mu(T)$-competitivo. Isto contrasta com o melhor limite inferior conhecido $\mu(T)/2$

Palavras-chave : Algoritmos Aleatorizados, Árvores AND-OR, Análise Competitiva.

1 Introduction

A game tree is a rooted tree, where every internal node has either a MIN or MAX label and the parent of every MIN (MAX) node is a MAX (MIN) node. Every leaf is associated to a real number, its value. The value of a MAX (MIN) node is recursively defined as the maximum (minimum) among the values of its children. The value of a tree is the value of its root. Game trees play a central role in Artificial Intelligence, particularly in game-playing programs.

The AND/OR tree is a particular case of a game tree, where every leaf has either value 0 or 1. It is easy to see that a MAX node can be thought as an OR gate while a a MIN node can be thought as an AND gate. The AND/OR trees are interesting in its own right since they have applications in mechanical theorem proving. They also appear in textbooks exemplifying how to employ techniques from game theory for proving lower bounds on randomized algorithms [8].

Several authors [6, 9, 12, 11, 10, 13] have considered the problem of determining the value of game trees and AND/OR trees by reading as few as possible leaves. In [3], Charikar et al. investigated this problem under the perspective of the competitive analysis [1]. They consider the more general problem where the cost of reading a leaf x_i is c_{x_i} and the cost of evaluating a tree is the sum of the costs of the leaves that are read in this process. This variant was motivated by possible Internet applications where the costs of the information required to take some decision may vary depending on the acquisition source. What has brought us into this problem, however, are database applications where queries involving no conventional data like images, DNA sequences and tables are handled [5, 4, 2, 7]. These queries differ from the traditional ones, since the processing of attributes like images and sequences are much more expensive than that of usual alpha-numeric registers. Since AND/OR trees model an important class of queries, they are particularly important in this scenario.

Now, we explain the competitiveness metric proposed in [3], which will also be adopted here.

Let f be a function over a set of variables $V = \{x_1, x_2, \ldots, x_n\}$ (a game tree can be thought as a function f, where the leaves correspond to the variables in V). Each variable x_i has a non-negative cost c_{x_i} and the vector $c =< c_{x_1}, \ldots, c_{x_n} >$ is called the *cost vector*. Given $U \subset V$, we define the cost of U as the sum of the costs of its variables. A *setting* σ of the variables is the choice of a value for each variable. The partial setting restricted to $U \subset V$ is denoted by $\sigma_{|U}$. A set $U \subset V$ is *sufficient* with respect to σ if the value of f is determined by the partial setting $\sigma_{|U}$. Such a U is a *proof* (certificate) of the value of f under σ. The cheapest proof of the value of f under σ is thus the sufficient set with minimum cost. We use $c^f(\sigma)$ to denote the cost of such a proof.

For example, consider the AND/OR tree T presented in Figure 1. For the setting $\sigma_R = (0, 0, 0, 1, 1)$, we have $c^T(\sigma_R) = 3+5+2$. On the other hand, for the setting $\sigma_S = (0, 1, 1, 1, 1)$, we have $c^T(\sigma_S) = 2+6+4$.

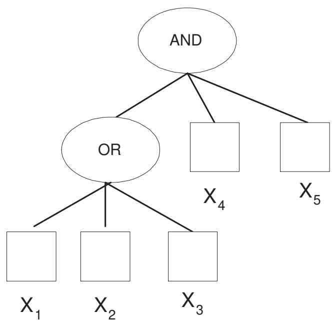

Figure 1: An AND/OR tree T with vector cost $c =< 3, 5, 2, 6, 4 >$.

An evaluation algorithm for f sequentially reads the variables in V. The algorithm stops when the set of variables read so far is sufficient with respect to σ. The cost of the algorithm $\mathcal{A}$ for a setting σ is given by $c^f_{\mathcal{A}}(\sigma)$. As an example, let $\mathcal{A}$ be an algorithm that reads the variables following the sequence

$x_1x_2x_3\ldots$ and skipping those variables that cannot affect, at the current point, the value of f anymore. Thus, for the tree T in Figure 1, $c^T_{\mathcal{A}}(\sigma_R) = 10$, since $\mathcal{A}$ reads the leaves x_1, x_2, x_3. On the other hand, $c^T_{\mathcal{A}}(\sigma_S) = 18$, since in this case it reads x_1, x_2, x_4, x_5.

The competitiveness of $\mathcal{A}$ is defined by

$$\gamma_c^{\mathcal{A}}(f) = max_\sigma \left\{ \frac{c^f_{\mathcal{A}}(\sigma)}{c^f(\sigma)} \right\}$$

The best possible competitive ratio for any deterministic algorithm, then, is

$$\gamma_c^f = \min_{\mathcal{A}} \gamma_c^{\mathcal{A}}(f), \tag{1}$$

where the minimum is got over all possible deterministic algorithms $\mathcal{A}$. For the case where f is an AND/OR tree function, Charikar et. al. [3] present a pseudo-polynomial γ_c^f-competitive deterministic algorithm.

Furthermore, they studied the dependence of the competitive ratio on the structure of f, defining the extremal competitiveness $\gamma(f)$ of f as

$$\gamma(f) = \max_c \gamma_c^f \tag{2}$$

This measure captures somehow the complexity of f, leaving the cost vector at the background. For the case where f is an AND/OR tree T, they show that

$$\gamma(T) = \max\{k(T), l(T)\}, \tag{3}$$

where $k(T)$ and $l(T)$ are, respectively, the number of leaves that must be read in the worst case in order to guarantee that the value of $T(\sigma)$ is 1 or 0. A simple method to calculate these values is described in Section 2.

1.1 Our Result

The main open direction according to Charikar et. al [3] is understanding the power of randomization in this context. Here, we give an important step in this direction. Given an algorithm $\mathcal{A}$, its randomized competitiveness is defined by

$$\delta_c^{\mathcal{A}}(f) = max_\sigma E\left[\frac{c^f_{\mathcal{A}}(\sigma)}{c^f(\sigma)}\right] = max_\sigma \frac{E[c^f_{\mathcal{A}}(\sigma)]}{c^f(\sigma)}$$

The optimal randomized competitiveness is defined by

$$\delta_c^f = \min_{\mathcal{A}} \delta_c^{\mathcal{A}}(f) \tag{4}$$

Finally, the extremal randomized complexity of f is defined by

$$\delta(f) = \max_c \delta_c^f \tag{5}$$

In [3], the following is observed.

Theorem 1 *If T is an AND/OR tree, then*

$$\delta(T) \geq \frac{1 + \max\{k(T), l(T)\}}{2}$$

Clearly, $\delta(T) \leq \gamma(T) = \max\{k(T), l(T)\}$, since any deterministic algorithm can be viewed as a randomized algorithm. Here, we show that

$$\delta(T) \leq 0.792 \max\{(k(T), l(T)\}$$

This result is proved through the analysis of a randomized polynomial algorithm that combines three key ideas: an optimal way to evaluate AND/OR trees with depth at most 2; a "binarization" of trees with unrestricted depth and a variation of the WeakBalance algorithm proposed in [3] which specially handles nodes whose children are roots of trees with depth at most 1. The main question that remains open is whether

$$\delta(T) = (1 + \max\{(k(T), l(T)\})/2$$

or not.

1.2 Related Work

Given an AND/OR trees T, it is known that any deterministic algorithm, in the worst case, must evaluate all leaves of T before determining its value.

Tarsi [12] considered the problem of minimizing the expected number of evaluated leaves for a distribution probability in which every leaf has probability p of having value 1. He has proved that certain class of deterministic algorithm are optimal for balanced trees (a class that includes uniform trees).

For binary trees, where every internal node has exactly two children and every leaf is at distance $2k$ from the root Snir [11] presents a randomized algorithm which reads at most $n^{0.793}$ leaves in the average, where $n = 2^{2k}$ is the total number of leaves. In [10], Saks and Wigderson prove that this algorithm is optimal. For general AND/OR trees, they present techniques for generating upper an lower bounds on the expected number of leaves that need to be read.

1.3 Paper Organization

The paper is organized as follows. In Section 2, we introduce some additional notation and state some facts that will be useful throughout this text. In Section 3, we prove that $\delta(T) = 1+\max\{k(T), l(T)\}/2$ for every AND/OR tree T with depth at most 2. In Section 4, we prove that $\delta(T) \leq 0.792 \max\{k(T), l(T)\}$, the main result of this paper. Finally, in Section 5, we make our final comments.

2 Notations and Basic Facts

Let T be a rooted tree with costs on its leaves. Define $h(T)$ as the depth of T, that is, the longest path from the root of T to a leaf. If T is a leaf, $h(T) = 0$. Given a node x in T, let T_x be the maximal (w.r.t node inclusion [1]) subtree of T rooted in x. We use c_T to denote the sum of the costs of the leaves of T. Throughout this text we use r to denote the root of T and $T_1, \ldots, T_k$ to denote the subtrees rooted at the children of r.

A general AND/OR tree (G-AND/OR tree) T is a rooted tree where every internal node has either an AND or OR label. Furthermore, to each leaf x of T it is associated a cost c_x and a bit value. The value of an AND internal node is 1 if all of its children have value 1 and it is 0, otherwise. The value of an OR internal node is 0 if all of its children have value 0 and it is 1, otherwise. In some occasions, we use the term variables of T to refer to the leaves of T. The value of T for a setting σ is denoted by $T(\sigma)$. As an example, for the setting $\sigma_R = (0, 0, 0, 1, 1)$ in Figure 1, we have $T(\sigma_R) = 0$. We say that two G-AND/OR trees T and T' are equivalent if $T(\sigma) = T'(\sigma)$ for every σ. Whenever the context is clear we use abuse the notation by using σ to refer to the partial setting restricted to the leaves of T_x.

[1] el: verificar

An AND/OR tree is an G-AND/OR tree where the parent of every AND (OR) node is an OR (AND) node and each internal node has at least two children. A single leaf is a *trivial* AND/OR tree. It is easy to verify the following fact

Fact 1 *Every G-AND/OR tree T is equivalent to an AND/OR tree T' such that $h(T') \leq h(T)$.*

The functions $k(T)$ and $l(T)$ can be calculated as follows. If T is a leaf then $k(T) = l(T) = 1$. If r is an AND node, then $k(T) = \sum_{i=1}^{k} k(T_i)$ and $l(T) = \max_{i=1\dots,k} l(T_i)$. Similarly, if r is an OR node, then $l(T) = \sum_{i=1}^{k} l(T_i)$ and $k(T) = \max_{i=1\dots,k} k(T_i)$. For example, in the tree of Figure 1, we have $k(T) = l(T) = 3$.

In order to make this reading easier we provide a list of notations.

$c^T(\sigma)$: cost of the cheapest proof for the value of T when the setting is σ
c_T: sum of the costs of the leaves of T
$c^T_{\mathcal{A}}(\sigma)$: cost incurred by algorithm $\mathcal{A}$ to determine the value of T when the setting is σ
$T(\sigma)$: value of T when the setting is σ
$h(T)$: depth of T

3 Evaluating Trees of Depth at Most 2

In this section, we prove the following theorem

Theorem 2 *If T is an AND/OR tree and $h(T) \leq 2$, then*

$$\delta(T) = \frac{1 + \max\{(k(T), l(T)\}}{2}$$

In [10], Saks and Wigderson defined the class of directional algorithms. An algorithm is directional if it reads the leaves of T following a depth first search in T, in which the next child of the current node to be visited is randomly selected according to some probability distribution.

The proof of Theorem 2 follows from the analysis of EVAL, a directional algorithm presented below. What makes EVAL interesting is the probability distribution employed, in which the next subtree to be visited is selected with a probability that depends on the square of the inverse of the sum of its leaves costs.

We have the following lemma.

Lemma 3 *Let $Pr[S, i, j]$ be the probability of EVAL selecting the index i before the index j. Then,*

$$Pr[S, i, j] \leq (w_i)/(w_i + w_j)$$

Proof: We use induction on the size of S. Clearly, if $|S| = 2$, the result holds. We have that

$$Pr[S, i, j] = \frac{w_i}{W} + \sum_{k \in S | k \notin \{i,j\}} \frac{w_k}{W} Pr[S - \{k\}, i, j]$$

By inductive hypothesis, we have that

$$Pr[S, i, j] = \frac{w_i}{W} + \frac{w_i}{w_i + w_j} \sum_{k \in S | k \notin \{i,j\}} \frac{w_k}{W} = \frac{w_i}{W} + \left(\frac{w_i}{w_i + w_j}\right)\left(\frac{W - w_i - w_j}{W}\right) = \frac{w_i}{w_i + w_j}$$

□

```
EVAL(T: AND/OR tree )
    If T is a leaf then Read T and Return the value of T
    S ← {1,...,k} (*)
    For each i ∈ S make w_i ← 1/(c_{T_i})^2
    While S ≠ ∅ do
        W ← Σ_{j∈S} w_j
        Selects an index i from S with probability w_i/W
        If the root of T is an OR gate then
            If EVAL(T_i)=1 then Return 1
        Else
            If EVAL(T_i)=0 then Return 0
        S ← S − {i}
    End While
        If the root of T is an OR gate then Return 0
        Else Return 1
```

Figure 2: Eval Algorithm

Lemma 4 *Let T be an AND/OR tree with depth at most 1 and let σ be a setting for T. If $T(\sigma) = 1$, then*

$$\frac{E[c^T_{EVAL}(\sigma)]}{c^T(\sigma)} \leq \frac{1 + l(T)}{2}$$

On the other hand, if $T(\sigma) = 0$, then

$$\frac{E[c^T_{EVAL}(\sigma)]}{c^T(\sigma)} \leq \frac{1 + k(T)}{2}$$

Proof: If $h(T) = 0$, then T is trivial and $k(T) = l(T) = 1$. Therefore, the result holds.

Assume that $h(T) = 1$. We only present the proof for the case where $T(\sigma) = 0$, since the proof for the other case is similar.

subcase 1) r is an OR node. In this case, $k(T) = 1$ and the minimum proof consists of all leaves, so as

$$\frac{c^T_{EVAL}(\sigma)}{c^T(\sigma)} = 1 = \frac{k(T) + 1}{2}$$

subcase 2) r is an AND node. In this case, $k(T)$ is the number of leaves in T. Let x_j be the leaf with minimum cost among those with value 0 and let X_{ij} be a random variable defined as follows: $X_{ij} = 1$ if x_i is evaluated by EVAL before x_j and $X_{ij} = 0$, otherwise. If $i = j$, define $X_{ij} = 1$. Then,

$$\frac{E[c^T_{EVAL}(\sigma)]}{c^T(\sigma)} \leq \frac{E[\sum_{i=1}^{k(T)} c_{x_i} X_{ij}]}{c_{x_j}},$$

It follows from Lemma 3 and from the linearity of the expectation that

$$\frac{E[\sum_{i=1}^{k(T)} c_{x_i} X_{ij}]}{c_{x_j}} = \frac{\sum_{i=1}^{k(T)} c_{x_i} Pr[S,i,j]}{c_{x_j}} \leq 1 + (k(T) - 1) \max_{i \neq j} \left\{ \frac{c_{x_i} c_{x_j}}{c^2_{x_i} + c^2_{x_j}} \right\} \leq \frac{1 + k(T)}{2},$$

where the last inequality follows from the arithmetic-geometric inequality. $\square$

We can prove a similar lemma for trees with depth 2.

Lemma 5 *Let T be a AND/OR tree with depth 2 and let σ be a setting for T. If $T(\sigma) = 1$, then*

$$\frac{E[c^T_{EVAL}(\sigma)]}{c^T(\sigma)} \leq \frac{1 + l(T)}{2}$$

On the other hand, if $T(\sigma) = 0$, then

$$\frac{E[c^T_{EVAL}(\sigma)]}{c^T(\sigma)} \leq \frac{1 + k(T)}{2}$$

Proof: See the appendix B. □

We remark that one can construct a tree with height 3 for which no directional algorithm is competitive. In this sense, directional algorithms are only efficient for trees with height at most 2.

4 Evaluating AND/OR Trees of Unrestricted Depth

In this section, we describe the RWB algorithm, which combines the ideas presented at the previous section with some of the ideas introduced in the algorithm WeakBalance [3]. For convenience, we explain the algorithm using a G-AND/OR tree T' obtained through a set of transformations on the given AND/OR tree T that we denote by binarization. This new tree has the following properties

(i) T' is equivalent to T, that is, $T(\sigma) = T'(\sigma)$, for all σ ;

(ii) $k(T) = k(T')$ and $l(T) = l(T')$

(iii) If x is an internal node in T' and $h(T'_x) \geq 3$, then x has exactly two children.

It is easy to obtain such a tree T' starting from T. Basically, while the current tree has a node x that does not satisfy the condition (iii), then the following rule is applied

Binarization Rule : Let $LAB \in \{AND, OR\}$ be the label of x and let $N_1, N_2, \ldots, N_k$, with $k > 2$, be the children of x. Replace x by an internal node x' with two children N_1 and N_0. Assign label LAB for both x' and N_0. Make $N_2, \ldots, N_k$ be the children of N_0.

This rule is applied until a tree T' with the desired properties is obtained. Figure 3 shows an example where the Binarization Rule is applied

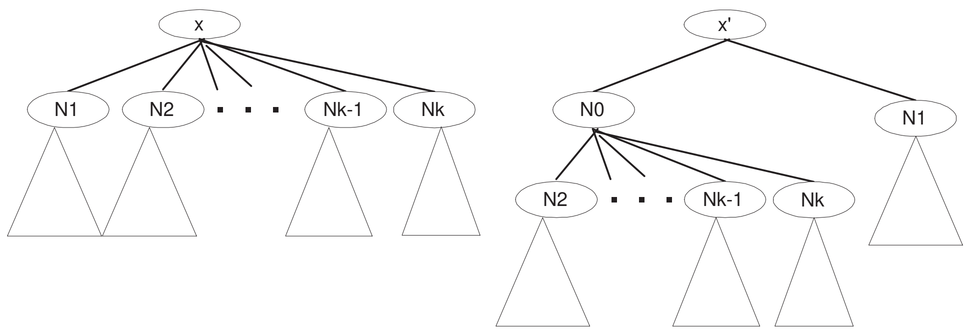

Figure 3: Applying the Transformation Rule on the Node x

It is easy to verify that T' satisfies the desired conditions. Let g be any function of $k(T)$ and $l(T)$. We can prove that $c^T_{\mathcal{A}}(\sigma) \leq g(k(T), l(T))$ by showing that $c^{T'}_{\mathcal{A}}(\sigma) \leq g(k(T'), l(T'))$. We will use this fact in some proofs.

4.1 The RWB Algorithm

4.1.1 A High Level Description

The algorithm gets as input an AND/OR tree T. If $h(T) \leq 2$, then EVAL(T) is executed. Otherwise, T is converted into a G-AND/OR tree T' through the binarization.

If $h(T') \geq 3$, RWB executes a loop, where at each iteration exactly one leaf is read. A pseudo-code is presented in Figure 4.

Every node x stores a recommendation and a variable Cost_x. The recommendation is a pair (L, c_L), where L is a leaf in T'_x of cost c_L. It defines the leaf, among those in T'_x, that will be firstly read by RWB from the current iteration. While the recommendation stored by a leaf L is (L, c_L), the recommendation of an internal node is always updated to the recommendation of one of its unevaluated children. This is detailed in the recommendation scheme presented in the next section. The variable Cost_x keeps track of the cost that RWB has incurred in the subtree T'_x, that is, the sum of the costs of the leaves of T'_x evaluated so far. This information is used in the recommendation updating process. In the pseudo-code, the operation Simplify evaluate as maximum as possible T' after reading the leaf L.

RWB Algorithm
- **For** every $x \in T'$ **do** $Cost_x \leftarrow 0$.
- Initialize the recommendation for every node traversing T' bottom-up.
- Let (L, c_L) be the recommendation stored by the root of T'
- Read L; Simplify T'.
- **While** the value of T' remains unknown **do**
  - **For** every ancestor x of L do $Cost_x \leftarrow Cost_x + c_L$
  - Let $x_1, \ldots, x_p$ be the unevaluated ancestors of L sorted by increasing order of distance to L
  - **For** $i = 1 \ldots p$ **do** update the recommendation of x_i (*)
  - Let (L, c_L) be the recommendation stored by the root of T'
  - Read L; Simplify T'.
- **End While**

Figure 4: The RWB Algorithm

4.1.2 The Recommendation Scheme

The recommendation scheme defines how the recommendation of a node is updated (initialized) during RWB execution. In fact, it provides an order in which the leaves are read by RWB. In particular, when the recommendation of an internal x node is updated, it defines the first leaf among those recommended by the x children that will be read.

In order to get a better intuition of how the recommendation scheme is designed, let us consider a node x with children N_1 and N_2. For illustration purpose, we assume that x is an AND node. We consider two cases. If x evaluates to 1, then the cheapest proof for x consists of leaves from both T'_{N_1} and T'_{N_2}. Thus, if T'_{N_1} and T'_{N_2} are efficiently evaluated, so will be T'_x, no matter how one merges the orders in which the leaves from T'_{N_1} and T'_{N_2} are read. However, if the value of x is 0, then the cheapest proof for x consists of leaves from only one of the subtrees T'_{N_1} and T'_{N_2}. In this case, it is not enough evaluating T'_{N_1} and T'_{N_2} efficiently. In fact, it is necessary to provide a balance between what has been spent in each tree in order to avoid an excessive expense in a tree whose leaves are not part of the cheapest proof for x value. Thus, the recommendation scheme must be devised having this second case in mind.

In order to describe this scheme in detail, we distinguish between three types of nodes. A node x is

- white if $h(T'_x) \leq 2$;

- grey if both $h(T'_x) > 2$ and x has a child y, with $h(T'_y) \leq 1$;
- black if x is neither white nor gray.

The motivation behind this classification is that the evaluation of both white and gray nodes can be optimized using randomization. In fact, we have seen that white nodes can be efficiently evaluated through procedure EVAL.

Now, we present the recommendation scheme for white nodes.

White Nodes. Let x be a white node and let $L_1, L_2, \ldots, L_k$ be the random sequence of leaves that are read when EVAL(T_x) is executed. Then, at the beginning x holds recommendation (L_1, c_{L_1}). When L_i is read, $1 \leq i \leq k-1$, x recommendation is updated to $(L_{i+1}, c_{L_{i+1}})$. This assures that the order in which the leaves from T'_x are read matches with the order defined by EVAL(T'_x).

Now, we explain the recommendations scheme for both black and grey nodes. If x is either a black or grey node in T', then $h(T'_x) \geq 3$ and x has exactly two children that we denote by N_1 and N_2. From now on, we assume w.l.g. that $h(T'_{N_1}) \leq h(T'_{N_2})$. Moreover, we assume that N_1 and N_2 hold recommendations (L_1, c_{L_1}) and (L_2, c_{L_2}), respectively.

Black Nodes. If x has only one unevaluated child, say y, then the recommendation of x is updated to that of y.

Otherwise, the recommendation of x is updated to (L_i, c_{L_i}), with $i \in \{1, 2\}$, such that

$$\frac{Cost_{N_i} + c_{L_i}}{f(T'_{N_i})}$$

is minimized, where $f(T'_{N_i}) = k(T'_{N_i})$ if x is an AND node and $f(T'_{N_i}) = l(T'_{N_i})$, otherwise.

We remark that the recommendation scheme for the black nodes is exactly the same scheme defined by the algorithm WeakBalance [3]. In fact, WeakBalance is equivalent to a version of RWB where every node is handled as a black node.

Gray Nodes. If x has only one unevaluated child, say y, then the recommendation of x is updated to that of y. Otherwise, RWB takes advantage of the following observation:

Observation 6 *if the cheapest proof for the value of x consists of only leaves from T'_{N_1}, then the cost of the cheapest proof for T'_{N_1} is $c_{T'_{N_1}}$*

Roughly speaking, the scheme works as follows. First, it defines a threshold parameter p_x whose value is related to $c_{T'_{N_1}}$. Then, while the cost incurred in $T'_{N_2}(Cost_{N_2})$ is smaller than p_x, the recommendation from N_2 is selected. In some sense, by acting like that, the schme is implictely assuming that the cheapest proof consists only of leaves from T'_{N_2}. Even if such assumption is not correct, it is not a big problem, once the cost spent in the "wrong" tree, T'_{N_2}, is not large at all. However, if $Cost_{N_2}$ becomes comparable to p_x, the scheme reviews its policy by tossing an unbiased coin. Depending on the result, it either keeps selecting recommendations from N_2 or it changes to those from N_1. Finally, if $Cost_{N_2}$ becomes larger than $2p_x$, then the scheme starts accepting the recommendations from T'_{N_1}. This avoids RWB spending too much in T'_{N_2} when the cheapest proof consists only of leaves from T'_{N_1}.

More formally, the scheme is implemented as follows: let p_x be a threshold parameter whose value will be defined later in the analysis and let $b(x)$ be a random bit obtained at the beginning of RWB execution (the value of $b(x)$ does not change throughout the execution). We have the following cases:

1. $Cost_{N_2} + c_{L_2} \leq p_x$. Then, the recommendation of x is updated to (L_2, c_{L_2}).

2. $p_x < Cost_{N_2} + c_{L_2} \leq 2p_x$. If $b(x) = 0$, then the recommendation of x is updated to (L_1, c_{L_1}). Otherwise, it is updated to (L_2, c_{L_2}).

3. $Cost_{N_2} + c_{L_2} > 2p_x$. Then, the recommendation of x is updated to (L_1, c_{L_1}).

4.2 RWB Analysis

In order to establish our main result,

$$\frac{E[c^T_{RWB}(\sigma)]}{c^T(\sigma)} \leq 0.792 \max\{k(T), l(T)\},$$

we first prove by induction that for every node x of T', the tree obtained from T by binarization, we have

$$\frac{E[c^{T'_x}_{RWB}(\sigma)]}{c^{T'_x}(\sigma)} \leq \max\{\alpha_1(T'_x)l(T'_x), \alpha_0(T'_x)k(T'_x)\},$$

where α_0 and α_1 are functions defined at appendix A that associate a G-AND/OR tree to a real number. Then, our main result is established by proving upper bounds on both α_1 and α_0.

Lemma 7 *Let T' be the G-AND/OR tree obtained from the input AND/OR tree T. Furthermore, let x be a node in T' and let σ be a setting for T'. If $T'_x(\sigma) = 1$, then*

$$\frac{E[c^{T'_x}_{RWB}(\sigma)]}{c^{T'_x}(\sigma)} \leq \alpha_1(T'_x)l(T'_x)$$

On the other hand, if $T'_x(\sigma) = 0$, then

$$\frac{E[c^{T'_x}_{RWB}(\sigma)]}{c^{T'_x}(\sigma)} \leq \alpha_0(T'_x)k(T'_x)$$

Proof: We only consider the case where $T'_x(\sigma) = 1$, since the proof for the other case is similar. The proof is by induction on the height of T'_x. The basis are the white nodes. If $h(T'_x) \leq 2$, it follows from lemmas 4 and 5 and from the definitions of α for white nodes (equations (11) and (12)) that the result holds. Now, let x be a node of T' such that $h(T'_x) \geq 3$.

subcase 1) x is either a gray or black internal node with label AND. In this case, the cost of the minimum proof for T'_x is the sum of the costs of the minimum proofs for its children, that is, $c^{T'_x}(\sigma) = c^{T'_{N_1}}(\sigma) + c^{T'_{N_2}}(\sigma)$. Moreover, the value of T'_x is determined right after the value of the last of its children is determined. Hence,

$$E\left[c^{T'_x}_{RWB}(\sigma)\right] = E\left[c^{T'_{N_1}}_{RWB}(\sigma) + c^{T'_{N_2}}_{RWB}(\sigma)\right] = E\left[c^{T'_{N_1}}_{RWB}(\sigma)\right] + E\left[c^{T'_{N_2}}_{RWB}(\sigma)\right]$$

It follows from the inductive hypothesis that

$$\frac{E[c^{T'_{N_i}}_{RWB}(\sigma)]}{c^{T'_{N_i}}(\sigma)} \leq \alpha_1(T'_{N_i})l(T'_{N_i}),$$

for $i = 1, 2$. Thus, we have that

$$\frac{E[c^{T'_x}_{RWB}(\sigma)]}{c^{T'}(\sigma)} = \frac{E[c^{T'_{N_1}}_{RWB}(\sigma)] + E[c^{T'_{N_2}}_{RWB}(\sigma)]}{c^{T'_{N_1}}(\sigma) + c^{T'_{N_2}}(\sigma)} \leq \max_{i=1\ldots 2}\{\alpha_1(T'_{N_i})l(T'_{N_i})\} = \alpha_1(T'_x)l(T'),$$

where the second inequality from left to right follows from the fact that $(a+b)/(c+d) \le \max\{a/c, b/d\}$ if a,b,c,d are positive real numbers and the rightmost expression is a consequence of the definition of α^1, equations (13) and (20).

subcase 2) x is a gray internal node with label OR. Since $T'_x(\sigma)=1$, we have two possibilities: either the cheapest proof consists of leaves from T'_{N_1} or from T'_{N_2}. First, we consider the case where the cheapest proof consists of all the leaves in T'_{N_1}. Analyzing the cases 1-3 of the recommendation scheme for grey nodes, we can conclude that

$$E[c_{RWB}^{T'_x}(\sigma)] \le c_{T'_{N_1}} + 1.5p_x \tag{6}$$

Let us consider the case where the cheapest proof consists of some leaves in T'_{N_2}. Then, let

$$P_1 = Pr\left[c_{RWB}^{T'_{N_2}}(\sigma) \ge 2p_x\right]$$

and let

$$P_2 = Pr\left[c_{RWB}^{T'_{N_2}}(\sigma) \ge p_x\right] - P_1.$$

Since, by inductive hypothesis, the expected cost incurred at N_2 when its value is determined is at most $\alpha_1(T'_{N_2})l(T'_{N_2})c^{T'_{N_2}}(\sigma)$, we have that

$$P_1 2p_x + P_2 p_x \le \alpha_1(T'_{N_2})l(T'_{N_2})c^{T'_{N_2}}(\sigma) \tag{7}$$

Now, we give an upper bound on $E[c_{RWB}^{T'_x}(\sigma)]$. Assume that RWB spends z to determine the value of T'_{N_2}. If $z > 2p_x$, then the item 3 of the recommendation schem for OR grey nodes assures that RWB spends $z + c_{T'_{N_1}}$ to determine $T'_x(\sigma)$. Otherwise, RWB pays at most $z + c_{T'_{N_1}}$ with probability 1/2 and pays z with probability 1/2. Taking the expectation of z we get that

$$E[c_{RWB}^{T'_x}(\sigma)] \le P_1 c_{T'_{N_1}} + 0.5P_2 c_{T'_{N_1}} + \alpha_1(T'_{N_2})l(T'_{N_2})c^{T'_{N_2}}(\sigma)$$

It follows from the equation above and from inequality (7) that

$$E[c_{RWB}^{T'_x}(\sigma)] \le \frac{c^{T'_{N_2}}(\sigma)l(T'_{N_2})\alpha_1(T'_{N_2})c_{T'_{N_1}}}{2p_x} + \alpha_1(T'_{N_2})l(T'_{N_2})c^{T'_{N_2}}(\sigma) \tag{8}$$

Hence, it follows from inequalities (6) and (8) that

$$\frac{E[c_{RWB}^{T'_x}(\sigma)]}{c^{T'_x}(\sigma)} \le \max\left\{1 + \frac{3p_x}{2c_{T'_{N_1}}}, \frac{\alpha_1(T'_{N_2})l(T'_{N_2})c_{T'_{N_1}}}{2p_x} + \alpha_1(T'_{N_2})l(T'_{N_2})\right\} \tag{9}$$

At this point, we can finally define a suitable value for p_x by setting it as the value that equalizes the arguments of the max expression above. One can verify, thaat this value is exactly

$$\frac{\alpha_1(T'_x)(l(T'_{N_2})+1)2c_{T'_{N_1}} - 2c_{T'_{N_1}}}{3},$$

where $\alpha_1(T'_x)$ is given by equation (18) [2] . Thus, we have that

$$\frac{E[c_{RWB}^{T'_x}(\sigma)]}{c^{T'_x}(\sigma)} \le 1 + \frac{3p_x}{2c_{T'_{N_1}}} = \alpha_1(T'_x)(l(T'_{N_2})+1) = \alpha_1(T'_x)l(T'_x)$$

[2]This shows that the function α must be calculated in order to implement the recommendation scheme for grey nodes

subcase 3) x is a black internal node with label OR. In this case, the cost of the minimum proof for T'_x is equal to the cost of the minimum proof for one of its children that outputs 1. We assume w.l.g. that N_1 is such a child. Then, $c^{T'_x}(\sigma) = c^{T'_{N_1}}(\sigma)$.

Let c_1 and c_2 be, respectively, the costs incurred at T'_{N_1} and T'_{N_2} when the value of T'_{N_1} is determined. The recommendation rule assures that $c_2 \leq (l(T'_{N_2})c_1)/l(T'_{N_1})$. Thus, the cost incurred at T'_x when the value of T'_{N_1} is determined is bounded above by

$$c_1 + \frac{l(T'_{N_2})c_1}{l(T'_{N_1})} = \frac{l(T'_x)c_1}{l(T'_{N_1})}$$

Therefore,

$$\frac{E[c^{T'_x}_{RWB}(\sigma)]}{c^{T'_x}(\sigma)} \leq \left(\frac{l(T'_x)}{l(T'_{N_1})}\right) \frac{E[c^{T'_{N_1}}_{RWB}(\sigma)]}{c^{T'_{N_1}}(\sigma)} \leq \alpha_1(T'_{N_1})l(T'_x) \leq \alpha_1(T'_x)l(T'_x),$$

where the second inequality from left to right follows from the application of the inductive hypothesis on T'_{N_1}. □

Now, we prove our main theorem by describing a method that provides upper bounds on both α_0 and α_1.

Theorem 8 *Let T be a non-trivial AND/OR tree. Then, for every setting σ,*

$$\frac{E[c^T_{RWB}(\sigma)]}{c^T(\sigma)} \leq 0.792 \max\{k(T), l(T)\},$$

Proof: If either $h(T) = 1$ or $h(T) = 2$, we have that $\max\{k(T), l(T)\} \geq 2$. By inspection, one can verify that

$$\frac{E[c^T_{RWB}(\sigma)]}{c^T(\sigma)} = \frac{E[c^T_{EVAL}(\sigma)]}{c^T(\sigma)} = \frac{1 + \max\{k(T), l(T)\}}{2} \leq 0.75 \max\{k(T), l(T)\}$$

Let us consider the case where $h(T) \geq 3$. Let T' be the tree obtained from T through the binarization process. We only consider the case where $T'(\sigma) = 1$, since the proof for the other case is similar. If we prove that $\alpha_1(T') \leq 0.792$, then the Lemma 7 assures the correctness of the theorem.

Let x be a node in T', with $h(T'_x) \geq 2$. We define a function g such that $g(x) \geq \alpha_1(T'_x)$ as follows

$$g(x) = \begin{cases} 0.75, \text{ if } x \text{ is a white node} \\ \max\{g(N_1), g(N_2)\}, \text{ if } x \text{ is a black node} \\ \max\{0.75, g(N_2)\}, \text{ if } x \text{ is a grey AND node} \\ \frac{g(N_2)l(T'_{N_2})+1+\sqrt{(g(N_2)l(T'_{N_2}))^2+g(N_2)l(T'_{N_2})+1}}{2(l(T'_{N_2})+1)}, \text{ if } x \text{ is a grey OR node} \end{cases}$$

We observe that the expression for grey OR nodes decreases with $l(T'_{N_2})$ (the proof for that demands some algebraic manipulation) and increases with $g(N_2)$.

By carefully considering the definitions of g and α_1, one can verify that $\alpha_1(T'_x) \leq g(x)$ for every x such that $h(T'_x) \geq 2$. Furthermore, it is easy to verify that only an OR grey node may have g value larger than the maximum g value of its children. Hence, $g(r)$, where r is the root of T', is either equal to 0.75 or to the g value of the gray OR node with maximum g value in T'. Another careful analysis shows that we can obtain an upper bound on $g(r)$ by computing the maximum of the sequence $(q_1, q_2, \ldots, q_{h(T')})$, where $q_1 = 0.75$ and

$$q_{i+1} = \frac{q_i(i+1) + 1 + \sqrt{(q_i(i+1))^2 + q_i(i+1) + 1}}{2i+4} \tag{10}$$

By using a numerical procedure, we can observe that $q_i \leq 0.792$, for all i $\square$

5 Conclusion

In this paper, we have presented a $0.792 \max\{k(T), l(T)\}$-competitive randomized polynomial time algorithm for evaluating AND/OR trees with non uniform costs on its leaves. This result is interesting once one cannot design a randomized algorithm with competitiveness smaller than $0.5 \max\{k(T), l(T)\}$ nor a deterministic algorithm with competitiveness smaller than $\max\{k(T), l(T)\}$.

For a particular cost vector c, randomization may help a lot, in the sense that the gap between δ_c^T and γ_c^T can be much larger than that between $\delta(T)$ and $\gamma(T)$ (recall equations (1), (2), (4) and (5)). As an example, for a complete binary AND/OR tree T, with depth $2d$ and $n = 2^{2d}$ leaves, Snir proposed a randomized algorithm [11], say $\mathcal{A}$, which reads at most $n^{0.753}$ leaves, in the average, to determine the value of T. Moreover, $\mathcal{A}$ is an optimal algorithm [10]. It is easy to verify that the cost of the cheapest proof for T is $n^{0.5}$, for every setting σ. Thus, $\delta_c^T = n^{0.253}$, where c is the all-ones cost vector. On the other hand, $\gamma_c^T = n^{0.5}$, since any deterministic algorithm, in the worst case, must read all leaves before determining the value of T. Designing an efficient δ_c^T-competitive randomized algorithm for general c and T seems to be a challenge problem which deserves further research.

References

[1] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, 1998.

[2] L. Bouganim, F. Fabret, F. Porto, and P. Valduriez, *Processing queries with expensive functions and large objects in distributed mediator systems*, in 17th International Conference on Data Engineering (ICDE' 01), Washington - Brussels - Tokyo, Apr. 2001, IEEE, pp. 91–98.

[3] Charikar, Fagin, Guruswami, Kleinberg, Raghavan, and Sahai, *Query strategies for priced information*, JCSS: Journal of Computer and System Sciences, 64 (2002). pp. 785–819.

[4] S. Chaudhuri and K. Shim, *Query optimization in the presence of foreign functions*, in Very large data bases, VLDB '93: proceedings of the 19th International Conference on Very Large Data Bases, August 24–27, 1993, Dublin, Ireland, R. Agrawal, S. Baker, and D. Bell, eds., Los Altos, CA 94022, USA, 1993, Morgan Kaufmann Publishers, pp. 529–542.

[5] J. M. Hellerstein, *Optimization techniques for queries with expensive methods*, ACM Transactions on Database Systems, 23 (1998), pp. 113–157.

[6] D. E. Knuth and R. W. Moore, *An analysis of alpha-beta pruning*, Artificial Intelligence, 6 (1975), pp. 293–326.

[7] E. Laber, O. Parekh, and R. Ravi, *Randomized approximation algorithms for query optimization problems on two processors*, Lecture Notes in Computer Science, 2461 (2002), pp. 649–661.

[8] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, England, June 1995.

[9] J. Pearl, *The solution for the branching factor of the alpha-beta pruning algorithm and its optimality*, Communications of the ACM, 25 (1982), pp. 559–564.

[10] M. Saks and A. Wigderson, *Probabilistic Boolean decision trees and the complexity of evaluating game trees*, in 27th Annual Symposium on Foundations of Computer Science, Toronto, Ontario, Canada, 27–29 Oct. 1986, IEEE, pp. 29–38.

[11] M. Snir, *Lower bounds on probabilistic linear decision trees*, Theoretical Computer Science, 38 (1985), pp. 69–82.

[12] M. Tarsi, *Optimal search on some game trees*, Journal of the ACM, 30 (1983), pp. 389–396.

[13] Y. Zhang, *On the optimality of randomized α-β search*, SIAM Journal on Computing, 24 (1995), pp. 138–147.

A The function α

Here, we give recursive definitions for $\alpha_0(T')$ and $\alpha_1(T')$. At a first view, these definitions (equations (11)-(20)) seem to be rather non-intuitive. However, they become much more natural when the reader examines the proof of lemma 7. Thus, we strongly suggest the reader to skip the definitions below and come back to them whenever they are refered in the proof of such a lemma.

White Nodes If x is a white node then define

$$\alpha_0(T'_x) = (k(T'_x) + 1)/2k(T'_x) \tag{11}$$

and

$$\alpha_1(T'_x) = (l(T'_x) + 1)/2l(T'_x). \tag{12}$$

Black Nodes If x is an AND node, then define

$$\alpha_1(T'_x) = \frac{\max\{\alpha_1(T'_{N_1})l(T'_{N_1}), \alpha_1(T'_{N_2})l(T'_{N_2})\}}{l(T'_x)} \tag{13}$$

and

$$\alpha_0(T'_x) = \max\{\alpha_0(T'_{N_1}), \alpha_0(T'_{N_2})\}. \tag{14}$$

If x is an OR node, then define

$$\alpha_0(T'_x) = \frac{\max\{\alpha_0(T'_{N_1})k(T'_{N_1}), \alpha_0(T'_{N_2})k(T'_{N_2})\}}{k(T'_x)} \tag{15}$$

and

$$\alpha_1(T'_x) = \max\{\alpha_1(T'_{N_1}), \alpha_1(T'_{N_2})\} \tag{16}$$

Gray Nodes If x is an OR node, then define

$$\alpha_0(T'_x) = \frac{\max\{\alpha_0(T'_{N_1})k(T'_{N_1}), \alpha_0(T'_{N_2})k(T'_{N_2})\}}{k(T'_x)} \tag{17}$$

and

$$\alpha_1(T'_x) = \frac{\alpha_1(T'_{N_2})l(T'_{N_2}) + 1 + \sqrt{(\alpha_1(T'_{N_2})l(T'_{N_2}))^2 + \alpha_1(T'_{N_2})l(T'_{N_2}) + 1}}{2(l(T'_{N_2}) + 1)} \tag{18}$$

If x is an AND node, then define

$$\alpha_0(T'_x) = \frac{\alpha_0(T'_{N_2})k(T'_{N_2}) + 1 + \sqrt{(\alpha_0(T'_{N_2})k(T'_{N_2}))^2 + \alpha_0(T'_{N_2})k(T'_{N_2}) + 1}}{2(k(T'_{N_2}) + 1)} \tag{19}$$

and

$$\alpha_1(T'_x) = \frac{\max\{\alpha_1(T'_{N_1})l(T'_{N_1}), \alpha_1(T'_{N_2})l(T'_{N_2})\}}{l(T'_x)} \tag{20}$$

B Proof of Lemma 5

We only present the proof for the case where $T(\sigma) = 1$, since the proof for the other case is similar.

subcase 1) r is an OR node. Since $T(\sigma) = 1$, then $T_i(\sigma) = 1$ for some subtree T_i. Hence, the cost of the cheapest proof for T under σ is the sum of the costs of the leaves of the subtree T_j which minimizes c_{T_j} among those that output 1. Hence, by replacing each subtree T_i by a leaf with cost c_{T_i}, one can apply the same analysis employed in the proof of lemma 4 to show that

$$\frac{E[c^T_{EVAL}(\sigma)]}{c(\sigma)} \leq \frac{k(T)+1}{2}.$$

subcase 2) r is an AND node. For $i = 1, \ldots, k$, let t_i be the number of leaves in T_i. We have that $k(T) = k$ and $l(T) = \max_{i=1\ldots k}\{t_i\}$.

Since $T(\sigma) = 1$, then $T_i(\sigma) = 1$, for $i = 1, \ldots, k$. Then, the minimum proof consists of one leaf from each of the subtrees, and so the cost of the minimum proof is given by $c^T(\sigma) = \sum_{i=1}^k c^{T_i}(\sigma)$. Hence,

$$\frac{E[c^T_{EVAL}(\sigma)]}{c^T(\sigma)} \leq \frac{\sum_{i=1}^k E[c^{T_i}_{EVAL}(\sigma)]}{\sum_{i=1}^k c^{T_i}(\sigma)} \leq \max_{i=1\ldots k} \frac{E[c^{T_i}_{EVAL}(\sigma)]}{c^{T_i}(\sigma)}$$

Therefore, it follows from Lemma 4 that

$$\frac{E[c^T_{EVAL}(\sigma)]}{c^T(\sigma)} \leq \frac{1+\max_{i=1\ldots k}\{t_i\}}{2} = \frac{1+l(T)}{2}$$