On the Competitive Ratio of Evaluating Priced Functions (Extended Abstract)

Ferdinando Cicalese*

Institut für Bioinformatik,
Universität Bielefeld, Germany
e-mail: nando@cebitec.uni-bielefeld.de

Eduardo Sany Laber†

Department of Informatics, PUC
Rio de Janeiro, Brasil
e-mail: laber@inf.puc-rio.br

Abstract

Let f be a function on a set of variables V. For each $x \in V$, let $c(x)$ be the cost of reading the value of x. An algorithm for evaluating f is a strategy for adaptively identifying and reading a set of variables $U \subseteq V$ whose values uniquely determine the value of f. We are interested in finding algorithms which minimize the cost incurred to evaluate f in the above sense. Competitive analysis is employed to measure the performance of the algorithms. We study two variants of the above problem. First we consider the classical setting in which one assumes that the algorithm knows the cost $c(x)$, for each $x \in V$. For the case where f is a monotone boolean function which is representable by a threshold tree, we provide a *polynomial time* algorithm with the best possible competitive ratio γ_c^f for each fixed cost function $c(\cdot)$. Remarkably, the best known result for the same class of functions is a *pseudo-polynomial* algorithm with competitiveness $2\gamma_c^f$. For the class of game tree functions our *polynomial time* algorithm attains, for each fixed cost function, the same competitiveness as the best known algorithm for the same class of functions, which instead runs in *pseudo-polynomial time*.

In the second part of the paper, we study a novel variant of the problem. Here, we assume that the cost function is not known in advance and some *preemption* is allowed in the reading operations. This model has applications, e.g., when reading a variable coincides with obtaining the output of a job on a CPU and the cost is the CPU time. In such a case, it is reasonable to assume that no exact knowledge of the cost is available. We define a new algorithm for this problem based on the solution of a Linear Program. We show the optimality of our algorithm for the class of monotone boolean functions representable by AND-OR-trees. We also show a sub-optimal implementation for general monotone boolean functions.

1 Introduction

In [1], Charikar *et al.* introduced the following basic model of function evaluation in the context of computing with priced information:

Function Evaluation with Priced Information (FEPI). A function $f(x_1, \ldots, x_n)$ has to be evaluated for a fixed but unknown *assignment* σ, i.e., a choice of the values for the set of variables $V = \{x_1, x_2, \ldots, x_n\}$. Each variable x_i has an associated non-negative cost $c(x_i)$ which is the cost incurred to probe x_i, i.e., to read its value $x_i(\sigma)$. For each $i = 1, \ldots, n$, the cost $c(x_i)$ is fixed and known beforehand. The goal is to *adaptively* identify and probe a minimum cost set of variables $U \subseteq V$ whose values uniquely determine the value of f for the given assignment, regardless of the value of the variables not probed. The cost of U is the sum of the costs of the variables it contains, i.e., $c(U) = \sum_{x \in U} c(x)$. We use $f(\sigma)$ to denote the value of f w.r.t. σ, i.e., $f(\sigma) = f(x_1(\sigma), \ldots, x_n(\sigma))$.

A set of variables $U \subseteq V$ is *sufficient* with respect to a given assignment σ of V if the value of f is determined by the restriction $\sigma_{|U}$ of σ to U. A set of variables which is sufficient is also called a *proof* of the value of f for the given assignment σ.

An evaluation algorithm $\mathbb{A}$ for f under an assignment σ is a rule to adaptively read the variables in V until the set of variables read so far is sufficient with respect to σ. The cost of the algorithm $\mathbb{A}$ for an assignment σ is the total cost incurred by $\mathbb{A}$ to evaluate f under the assignment σ. Given a cost function $c(\cdot)$, we let $c_{\mathbb{A}}^f(\sigma)$ denote the cost of the algorithm $\mathbb{A}$ for an as-

*Supported by the Sofja Kovalevskaja Award 2004 of the Alexander von Humboldt Foundation and the Bundesministerium für Bildung und Forschung.

†This research was partially supported by CNPQ under the Grant 300968/2003-5 and the Grant 476323/2004-5

signment σ and $c^f(\sigma)$ the cost of the cheapest proof for f under the assignment σ. We say that $\mathbb{A}$ is ρ-competitive if $c^f_{\mathbb{A}}(\sigma) \leq \rho c^f(\sigma)$, for every possible assignment σ. We use $\gamma_c^{\mathbb{A}}(f)$ to denote the competitive ratio of $\mathbb{A}$, that is, the minimum ρ for which $\mathbb{A}$ is ρ-competitive. The best possible competitive ratio for any deterministic algorithm, then, is $\gamma_c^f = \min_{\mathbb{A}} \gamma_c^{\mathbb{A}}(f)$, where the minimum is computed over all possible deterministic algorithms $\mathbb{A}$.

In order to clarify some of the above definitions, let us consider the boolean function

$$f = (x_1 \text{ AND } x_2) \text{ OR } (x_2 \text{ AND } x_3) \text{ OR } (x_3 \text{ AND } x_4) \quad (1.1)$$

together with the costs $c(x_1) = 3, c(x_2) = 5, c(x_3) = 4$ and $c(x_4) = 1$. For the assignment $\sigma_R = (1, 0, 1, 1)$, we have $f(\sigma_R) = 1$ and $U = \{x_3, x_4\}$ as the only proof of minimum cost. Therefore, $c^f(\sigma_R) = 4 + 1$. On the other hand, for the assignment $\sigma_S = (1, 0, 0, 0)$, we have $f(\sigma_S) = 0$ and the cheapest proof is $\{x_2, x_4\}$. Thus, $c^f(\sigma_S) = 5 + 1$. Let now $\mathbb{A}$ be an algorithm that reads first x_1, then x_2, and so on, just skipping a variable x_i if, due to the values read so far, the value of x_i cannot affect the value of f. Thus, it is not hard to verify that $c^f_{\mathbb{A}}(\sigma_R) = 13$, since $\mathbb{A}$ reads the variables x_1, x_2, x_3, x_4. Furthermore, $c^f_{\mathbb{A}}(\sigma_S) = 12$, since in this case, $\mathbb{A}$ reads x_1, x_2 and x_3.

The model described above has applications in several situations, e.g., gathering information from priced sources on the Internet [5], evaluation of complex predicates in databases [10], learning theory [7], and computational geometry [11]. In general, it covers several situations where the completion of a given task is required, for which information can be collected from many sources at different cost. Different subsets of the information available at the different sources are sufficient to accomplish the desired task, and the problem is how to choose the information sources which together can provide a sufficient amount of data without incurring too high a cost.

FEPI with Unknown Costs (FEPI-UC). Assume now that the information sources are jobs in a computer system, i.e., the values of the variables in the above model are the outputs of computer programs. The cost of obtaining such information is the CPU time necessary to run the corresponding job. Then, it is reasonable to assume that the cost for obtaining the value of a variable is unknown beforehand.

These arguments motivate us to extend the FEPI model to consider the case where the costs are not known in advance. Algorithms for this new model are allowed to use preemption: In the original FEPI model, at each step the algorithm chooses an unread variable and pays the cost associated with it to read its value. In the FEPI-UC, the process of reading a variable resembles the execution of a job that can be stopped and resumed several times before producing the desired output. More formally, an algorithm for the FEPI-UC with unknown costs probes the variables by using the operation $\mathit{Read}(x, t)$, where $x \in V$ and t is a real number. Executing such an operation, the algorithm pays an amount of at most t. This is like an installment for covering the unknown cost $c(x)$ of x. Let $\delta(x)$ be the total amount spent by the algorithm in *Read* operations on x before executing the present $\mathit{Read}(x, t)$. If $\delta(x) + t \geq c(x)$, i.e., by paying t the algorithm finishes covering the cost of x, then only $c(x) - \delta(x)$ is charged for the operation $\mathit{Read}(x, t)$ and the value of x is released. Conversely, if $\delta(x) + t < c(x)$, i.e., including the last t paid, the total cost spent on x is still insufficient for the evaluation of x, the algorithm pays t but it does not get the value of x. At any later step the process of reading x can be resumed or the algorithm can decide to ignore x and concentrate only on other variables. Note that when the value of x is finally obtained, the total cost incurred by the algorithm is $c(x)$.

As an obvious adaptation of the notion of competitiveness given in the basic model, here, we define the competitive ratio of an algorithm $\mathbb{A}$ as the minimum ρ for which $c^f_{\mathbb{A}}(\sigma) \leq \rho c^f(\sigma)$ for every assignment σ and for every *feasible* cost function $c(\cdot)$. For sake of definiteness, a cost function is feasible if it satisfies $c(x) \geq M$, for every $x \in V$, where M is a positive constant known to the algorithm. Moreover, we remark that if *preemption* is not allowed, there is no hope of finding efficient strategies. In fact, as opposed to the classical FEPI, here, the algorithm is evaluated against an adversary that can set both the costs and the values of the variables adaptively. Therefore, if the algorithm was not allowed to *read* a variable *one bit at a time*, the adversary could force it to pay an arbitrarily high cost for getting one value, precluding any possibility of being competitive.

Our Contributions. We study the two variants of the problem described above. For the FEPI model in which the costs are known in advance, we study the γ_c^f competitiveness for the class of monotone boolean functions representable by threshold trees and for the class of the game tree functions.

Our main contribution consists of a simple greedy strategy that achieves the best known competitive ratio both for threshold tree and game tree functions. More specifically, we show a polynomial time algorithm with the best possible competitive ratio γ_c^f for threshold tree functions (and *a fortiori* for the particular case of AND/OR tree functions). Then, we show that a variant of this algorithm achieves competitive ratio $4\gamma_c^f$

for game tree functions, in time polynomial only in the size of the tree. It turns out (see Related Work below) that in terms of competitive ratio, this algorithm is not worse than the best known algorithm to date for game trees [1], which, however, has a running time that is polynomial in the size of the tree *and* the magnitude of the costs, i.e., the algorithm is pseudo-polynomial.

It is indeed remarkable that ours are the first algorithms for function evaluation with priced information which are both fully polynomial time and competitive with respect to the γ_c^f metric.

For the FEPI-UC we design a new strategy based on the solution of a Linear Program and show an optimal implementation for the class of AND/OR trees. A suboptimal implementation of our strategy is also given for general monotone boolean functions. We remark that an analogous approach based on the same linear program can be used to design very efficient algorithms for the classical FEPI model.

Related Work. The seminal paper for the study of the effect that priced information has on basic algorithmic problems is due to Charikar *et al.* [1]. Among others, the function evaluation problem for the classes of AND-OR trees, threshold trees, and game trees, is addressed there. For the subclass of the monotone boolean functions that are representable by AND/OR trees, a γ_c^f-competitive pseudo-polynomial algorithm is provided. A variant of this algorithm is shown to achieve $2\gamma_c^f$-competitiveness, in pseudo-polynomial time, for the class of threshold trees. Note that as opposed to the one in [1], the new algorithm we present here for evaluating threshold trees does not lose the factor of 2 and, more importantly, runs in polynomial time.

For the class of functions that are representable by game trees, a pseudo-polynomial algorithm is presented in [1] which is claimed to be $2\gamma_c^f$-competitive. Here, we show that, in actual fact, the lower bound employed in [1] is not sufficient to guarantee this result. We present a "corrected" version of such a lower bound, which, however, allows us only to show that the algorithm of [1] is $4\gamma_c^f$ competitive. Thus, it appears to be no better than the algorithm we propose here for evaluating game trees, which is also a $4\gamma_c^f$-competitive algorithm, but it is a fully polynomial time one.

After [1], a number of papers on this topic have appeared in the literature [4, 8, 6, 11, 9, 3, 7, 2]. In particular, in [3, 2], we studied the *extremal competitive ratio*, defined by $\gamma(f) = \min_{\mathbb{A}} \max_c \gamma_c^{\mathbb{A}}(f)$, where the max is computed over all cost functions c and the min over all algorithms $\mathbb{A}$. We provided polynomial time algorithms with extremal competitive ratio, $\gamma(f)$ (or $K \times \gamma(f)$, for a small constant K), for several classes of functions.

As opposed to our previous work, in the present paper, we achieve optimal competitive ratio rather than optimal extremal competitive ratio. Importantly, we achieve optimality in terms of the stronger measure, at no expense in terms of running time of the algorithms which are still polynomial time. Obviously, γ_c^f-competitiveness implies $\gamma(f)$-competitiveness, whilst $\gamma(f)$-competitive algorithms could perform poorly for some cost functions.

2 The FEPI model with known costs

In this section we shall consider the original FEPI model, in which, in particular, the algorithm has complete access to the costs of the variables. We shall present algorithms that are competitive with respect to γ_c^f for the classes of threshold trees and game trees functions.

We shall start with some basic concepts and notations, introduced for the case when f is a monotone boolean function over the set of variable $V = \{x_1, \ldots, x_n\}$. A fortiori, everything we state here will directly apply to the class of threshold trees, which are in fact monotone boolean functions. Moreover, most basic definitions given here for boolean functions will be extended or generalized to the case of game tree functions, with which we shall deal later.

Let $Y \subseteq V$ and let σ_Y be an assignment for the variables of Y. We use f_Y to denote the restriction of f obtained by fixing the values of the variables in Y as given by σ_Y. Consider, e.g., the function f in (1.1). Let $Y = \{x_2, x_4\}$ and $\sigma_Y = (x_2 = 1, x_4 = 1)$. Then, we have $f_Y = x_1$ OR x_3.

A minterm for f is a minimal set of variables $C^- \subseteq V$ such that if $x(\sigma) = 1$ for every $x \in C^-$, then f evaluates to 1, no matter how are assigned the values for the remaining variables. A maxterm for f is a minimal set of variable $C^+ \subseteq V$ such that if $x(\sigma) = 0$ for every $x \in C^+$, then f evaluates to 0. As an example, in the function presented in (1.1), $\{x_1, x_2\}$ is a minterm and $\{x_2, x_4\}$ is a maxterm. We use the term *certificate* to either refer to a minterm or to a maxterm. Obviously, every proof for f contains a certificate.

An immediate property of the certificates of a function f is that for each minterm C^- and each maxterm C^+ of f it holds that $C^- \cap C^+ \neq \emptyset$.

2.1 Threshold Trees

A threshold tree over a set of boolean variables V is a rooted tree T, where each internal node is associated with an integer number and each leaf is associated with a distinct variable of V. The value of a leaf is the value of its associated variable. The value of a node whose associated integer is t (a t-

node) is 1 if at least t of its children have value 1 and it is 0, otherwise. The boolean function computed by a threshold tree T is the one mapping the values of the leaves of T to the value of the root of T.

Given a threshold tree T, we use $leaves(T)$ to denote its set of leaves. Abusing notation, we use T to denote also the function, say f, computed by the tree T. Accordingly, for every $Y \subset V$, T_Y will denote both the threshold tree computing f_Y and the function f_Y itself.

The certificates of a threshold tree. Let T be a threshold tree rooted on a t-node r and let $T_1, \dots, T_p$ be the subtrees of T rooted at the children of r. Then, C is a minterm for T if and only if there exists a subset $R \subseteq \{1, \dots, p\}$, with $|R| = t$, such that: (i) $C \cap leaves(T_i)$ is a minterm for T_i, for each $i \in R$; (ii) $C \cap leaves(T_i) = \emptyset$ for each $i \notin R$.

Analogously, C is a maxterm for T if and only if there exists a subset $S \subseteq \{1, \dots, p\}$, with $|S| = p - t + 1$, such that: (i) $C \cap leaves(T_j)$ is a maxterm for T_j, for $j \in S$; (ii) $C \cap leaves(T_j) = \emptyset$ for $j \notin S$. This characterization allows us to easily compute the cheapest minterm (maxterm) recursively in polynomial time.

The Lower Bound. We shall now recall a lower bound on the competitive ratio of any deterministic algorithm which evaluates threshold trees proved in [1]. In the proposition below $[p]$ denotes the set $\{1, 2, \dots, p\}$.

PROPOSITION 2.1. *[1] Let T be a threshold tree and r denote the root of T. Let $c(\cdot)$ be the cost function on the leaves of T. For each leaf ℓ of T, define $\theta_0^\ell(y) = \theta_1^\ell(y) = 0$ if $y < c(\ell)$ and $\theta_0^\ell(y) = \theta_1^\ell(y) = c(\ell)$, otherwise.*
For a t-node ν of T with children $\nu_1, \dots, \nu_p$ define [1],

$$\theta_1^\nu(y) = \max_{\substack{I \subseteq [p] \\ |I| = t}} \left(\max_{\substack{\{y_1, \dots, y_t\} \\ \sum_{i=1}^t y_i = y}} \left(\sum_{i \in I} \theta_1^{\nu_i}(y_i) + \sum_{i \notin I} \theta_1^{\nu_i}(\max_{j=1}^{t} y_j) \right) \right).$$

Define $\theta_0^\nu(y)$ to be the function obtained by replacing in the definition of $\theta_1^\nu()$ every occurrence of t with $p - t + 1$. Finally, define $\theta_1^T(y) = \theta_1^r(y)$ and $\theta_0^T(y) = \theta_0^r(y)$.
Then, $\theta_1^T(y)$ (resp. $\theta_0^T(y)$) is a lower bound on the cost that any algorithm must incur in the worst case in order to determine the value of a 1-witness (resp. a 0-witness) of cost at most y. Hence,

$$\gamma_c^T \geq \max \left\{ \max_{\sigma : f(\sigma)=0} \frac{\theta_0^T(c^T(\sigma))}{c^T(\sigma)}, \max_{\sigma : f(\sigma)=1} \frac{\theta_1^T(c^T(\sigma))}{c^T(\sigma)} \right\}$$

[1] In the definition of $\theta_1^\nu()$ the second max operator is taken only over choices $y_1, \dots, y_t$ such that there can exist minterms for the trees rooted at $\nu_{i_1}, \dots, \nu_{i_t}$ with costs at most $y_1, \dots, y_t$, respectively. If no such $y_1, \dots, y_p$ exist for a particular y then the value of max is 0.

The Algorithm GREEDY-MIN**.** Let $\mathcal{F}$ denote the family of minterms of f. We shall now define a total order χ on $\mathcal{F}$ that induces a sorting of the minterms of f in order of non-decreasing cost.

DEFINITION 1. (RANKS) *Let $f = f(x_1, x_2, \dots, x_n)$ be a boolean function and let $c(\cdot)$ be a cost function on the variables of f. Let π be the total order on the variables of f defined by stipulating that, for each $i = 1, 2, \dots, n-1$, x_i precedes x_{i+1} in the order π. Therefore, c and π induce a total order χ on the minterms of f as follows. In χ a minterm C precedes a minterm D if and only if one of the following conditions holds: (a) $c(C) < c(D)$; (b) $c(C) = c(D)$ and the list of variables in C (listed according to π) precedes in the lexicographical order the list of variables in D (listed according to π).*
For each minterm C of f we define $rank_f(C)$ as the ordinal position of C in χ (i.e., the number of minterms that precede C in χ, plus 1). When the function f is clear from the context we shall write $rank(C)$ instead of $rank_f(C)$.

The algorithm GREEDY-MIN below examines the minterms of $\mathcal{F}$ in order of increasing rank.

By the value of a minterm we shall mean the AND of the values of its variables. Therefore, the value of a minterm C is determined by GREEDY-MIN as soon as either one of the variables in C is found to have value 0, or all the variables in C are found to have value 1.

We shall say that a minterm C is *active* for GREEDY-MIN as long as the value of C is not determined. Given an active minterm C, we shall say that $U \subseteq C$ is *strongly active* iff : (i) U is the set of unread variables of some active minterm of f and (ii) U is minimal, i.e., no proper subset of U satisfies condition (i). Note that such a strongly active set U is always a minterm for f_Y, where Y is the set of variables read so far.

```
Algorithm GREEDY-MIN(f, V, c)
    While the value of f is unknown
        C ← active minterm of f with minimum rank
        U ← a strongly active subset of C
        Read a variable of U
    End While
```

We shall say that a minterm C is *evaluated* by GREEDY-MIN if and only if C is one of the minterms selected by GREEDY-MIN during the main loop. Note that, according to this definition, it may happen that C is not evaluated although some of its variables are read. The algorithm GREEDY-MIN does not specify which strongly active set U is selected nor the variable of U that is read.

An *implementation* for the algorithm GREEDY-MIN is a rule that defines both the strongly active set U contained in C and the variable of U to be selected.

Let $\mathbb{I}$ be an implementation for the algorithm GREEDY-MIN. For each function f and for each assignment σ, the *execution* $\mathbb{I}(f, \sigma)$ of the implementation $\mathbb{I}$ of GREEDY-MIN on the function f with assignment σ is the sequence of pairs $(x_i, C(x_i))$, i = 1,2,..., where x_i is the i-th variable that $\mathbb{I}$ reads and $C(x_i)$ is the minterm of f that is being evaluated when x_i is probed.

The following lemma originally presented in [2] will be useful in the recursive analysis of the cost incurred by GREEDY-MIN on a threshold tree.

LEMMA 2.1. *[2] Let $\mathbb{I}$ be an arbitrary implementation of* GREEDY-MIN*. Let T_m be a subtree of T, rooted at one of the children of r. Let $x_1, x_2, \ldots, x_q$ be the leaves of T_m listed in the order that they appear in $\mathbb{I}(T, \sigma)$. Then, there exists an implementation $\mathbb{I}_m$ for* GREEDY-MIN *that satisfies*

(i) The q first variables of $\mathbb{I}_m(T_m, \sigma_{|T_m})$ are $x_1, x_2, \ldots, x_q$

For $i = 1, 2, \ldots, q$ let $C(x_i)$ (respectively $C_m(x_i)$) denote the minterm of T (resp. T_m) that is evaluated in $\mathbb{I}(T, \sigma)$ (resp. $\mathbb{I}_m(T_m, \sigma_{|T_m})$) when x_i is probed.

(ii) Then, for $i = 1, 2, \ldots, q$, we have $C_m(x_i) = C(x_i) \cap T_m$.

THEOREM 2.1. *Let $\mathbb{I}$ be an arbitrary implementation of* GREEDY-MIN*. If f can be represented by a threshold tree, then for every assignment σ such that $f(\sigma) = 1$, we have $c^f_{\mathbb{I}}(\sigma) \leq \theta^f_1(c^f(\sigma))$.*

Proof. Recall the definition of the functions θ^f_0 and θ^f_1 given in Proposition 2.1. We shall prove, by induction on the height of the tree, that for every assignment σ', the cost incurred by $\mathbb{I}$ before determining the value of a minterm C is at most $\theta^f_1(c(C))$. This will suffice to establish the theorem since we can consider the particular case where σ' is an assignment for which f evaluates to 1 and C is the cheapest proof for f under the assignment σ'.

For the basis we assume that T has height 0, that is T is a single leaf l. In this case, the result trivially holds since the unique minterm is l and $\theta^\ell_1(c(l)) = c(l)$.

Let us assume that the claim holds for every threshold tree of height at most h. Let T be a tree with height $h + 1$ rooted at a t-node r. In addition, let $T_1, \ldots, T_p$ be the subtrees rooted at the children of r. We assume w.l.g. that $C \cap T_i \neq \emptyset$ for $i = 1, \ldots, t$. This implies that $C \cap T_i = \emptyset$ for $i = t + 1, \ldots, p$.

Claim Let C' be a minterm of T such that $rank(C') < rank(C)$. If C' is evaluated before the value of C is determined we must have

(i) $c(C' \cap T_i) \leq c(C \cap T_i)$ for $i = 1, \ldots, t$

(ii) $c(C' \cap T_j) \leq \max_{i=1,\ldots,t}\{c(C \cap T_i)\}$, for $j > t$.

Proof of the Claim. (i) For the sake of contradiction, we assume that $c(C' \cap T_i) > c(C \cap T_i)$ for some $i \in \{1, \ldots, t\}$. Let $C^* = (C' \setminus T_i) \cup (C \cap T_i)$. Since $c(C^*) < c(C')$ then $rank(C^*) < rank(C')$.

Let D be the last minterm evaluated in the execution $\mathbb{I}$. If $rank(D) \leq rank(C^*)$ we have that C' is not evaluated. On the other hand, if $rank(D) \geq rank(C*) + 1$, then at the time when $\mathbb{I}$ evaluates for the first time a minterm of rank $\geq rank(C^*) + 1$, some variable $x \in C^*$ with value 0 must have been read. If $x \in T_i$ then C evaluates to 0. Otherwise, if $x \notin T_i$, then C' is not evaluated because after reading x also the minterm C' becomes non-active. In both cases, C' is not evaluated before the value of C is determined.

(ii) Let $c_{max} = \max^t_{i=1}\{c(C \cap T_i)\}$. For the sake of contradiction, we assume that $c(C' \cap T_j) > c_{max}$ for some $j > t$. Let T_i, with $i \leq t$, be a subtree such that $C' \cap T_i = \emptyset$ and define $C^* = (C' \setminus T_j) \cup (C \cap T_i)$. The same arguments employed in the case (i) allow us to obtain a contradiction.

Let X_i be the sequence of leaves of T_i that are read in the execution $\mathbb{I}(T, \sigma)$ before determining the value of C, listed in the order in which they are read by $\mathbb{I}$. In order to bound the sum of the costs of these variables, we use the fact, assured by Lemma 2.1, that there is an implementation $\mathbb{I}_i$ such that the sequence of variables in the execution $\mathbb{I}_i(T_i, \sigma_{|T_i})$ coincides exactly with X_i.

In fact, the second statement in Lemma 2.1 together with the previous claim guarantees that, for each $i \leq t$ (respectively $i > t$), $\mathbb{I}_i$ only evaluates minterms of cost not larger than $c(C \cap T_i)$ (respectively c_{max}) while reading the variables in X_i. Thus, by induction hypothesis we have that the cost incurred due to the variables of T_i is at most $\theta^{T_i}_1(c(C \cap T_i))$ (respectively $\theta^{T_i}_1(c_{max})$).

Therefore, the cost spent by $\mathbb{I}$ before determining the value of C is at most

$$\sum_{i=1}^{t} \theta^{T_i}_1(c(C \cap T_i)) + \sum_{i=t+1}^{p} \theta^{T_i}_1(c_{max}) \leq \theta^T_1(c(C)),$$

where the inequality directly follows from the definition of $\theta^\nu_0(y)$ Thus, our induction is completed. ■

The Algorithm GREEDY-MAX**.** Let GREEDY-MAX be the variant of GREEDY-MIN that evaluates the maxterms of f instead of the minterms. Proceeding as before and using the dualities between minterms and maxterms, and between θ_0 and θ_1, one can easily prove the following dual result.

THEOREM 2.2. *Let $\mathbb{I}$ be an arbitrary implementation of* GREEDY-MAX*. If f can be represented by a threshold tree, then for every assignment σ such that $f(\sigma) = 0$, we have $c^f_{\mathbb{I}}(\sigma) \leq \theta^f_0(c^f(\sigma))$.*

The Optimal Algorithm for Threshold Tree. We now present an algorithm that combines the "optimal" features of GREEDY-MAX and GREEDY-MIN. It exploits the structure of the strongly active subsets of minterms and maxterms to attain γ^f_c-competitiveness, as it is proved in the next theorem.

Algorithm GREEDY*$(f, V, \mathbf{c})$
Fix an arbitrary order on the variables of V.
While the value of f is unknown
    Let U^1 be a strong active subset of the active minterm of minimum rank
    Let U^0 be a strong active subset of the active maxterm of minimum rank
    Read a variable of $U^1 \cap U^0$
End While

THEOREM 2.3. *Let $\mathbb{A}$ be an implementation of* GREEDY**. Then, for every monotone boolean function f represented by a threshold tree and for every cost function $c(\cdot)$ on the leaves of f, we have that $\gamma^{\mathbb{A}}_c(f) = \gamma^f_c$.*

Proof. First we notice that $U^0 \cap U^1 \neq \emptyset$. In fact, by the definition of strongly active sets, U^0 and U^1 are, respectively, a maxterm and a minterm of f_Y, whence their intersection cannot be empty.

Therefore, GREEDY* is *simultaneously* an implementation of GREEDY-MIN and GREEDY-MAX. whence, putting together Theorems 2.1 and 2.2 and Proposition 2.1 we have

$$\gamma^{\mathbb{A}}_c(f) = \max_{\sigma} \frac{c^f_{\mathbb{A}}(\sigma)}{c^f(\sigma)} \leq$$

$$\max\left\{ \max_{\sigma\,:\,f(\sigma)=0} \frac{\theta^f_0(c^f(\sigma))}{c^f(\sigma)}, \max_{\sigma\,:\,f(\sigma)=1} \frac{\theta^f_1(c^f(\sigma))}{c^f(\sigma)} \right\} \leq \gamma^f_c$$

■

For the polynomial implementation of $\mathbb{A}$ for a threshold tree functions f, the only point that must be clarified is how to select the active minterm and maxterm of f with minimum rank.

We shall limit ourselves to describe the procedure for the case of a minterm, since the case of a maxterm can be treated analogously.

First, we note that the minterm with minimum rank in a threshold tree can be easily found by using the given recursive characterization of minterms for threshold trees. Then, we observe that the active minterm of f of minimum rank is exactly the one with minimum rank in the tree T' obtained from T through the removal of all the leaves that have already been read and found to have value 0 assigned.

2.2 Game Trees

A game tree T is a rooted tree such that every internal node has either a MIN or a MAX label and the parent of every MIN (MAX) node is a MAX (MIN) node. Let V be the set of leaves of T. Every leaf of V is associated with a real number, its value. The value of a MIN (MAX) node is the minimum (maximum) of the values of its children. The function computed by T (the value of T) is the value of its root. Like in the previous section we shall identify T with the function it computes. Thus, if f is the function computed by the game tree T, we shall also write T for f and T_Y for f_Y.

By a minterm (maxterm) of a game tree we shall understand a minimal set of leaves whose values allow to state a lower (upper) bound on the value of the game tree. More precisely, a minterm (maxterm) for a game tree T rooted at r is a minimal set C of leaves of T such that if $x(\sigma) \geq \ell$ ($x(\sigma) \leq \ell$,) for each $x \in C$ then $r(\sigma) \geq \ell$ ($r(\sigma) \leq \ell$) regardless of the values of the leaves $y \notin C$. As with monotone boolean function, we shall use the more general term *certificate* to either refer to a minterm or to a maxterm.

The certificates of a game tree. Let $T_1, \ldots, T_p$ be the subtrees of T rooted at the children of r. If r is a MAX node then C^L is a minterm for T if and only if C^L is also a minterm for some subtree T_i, with $i \in \{1, \ldots, p\}$. Furthermore, C^U is a maxterm for T if and only if $C^U \cap T_i$ is a maxterm of T_i, for $i = 1, \ldots, p$.

If r is a MIN node, then C^L is a minterm for T if and only $C^L \cap T_i$ is a minterm of T_i, for $i = 1, \ldots, p$; and C^U is a maxterm for T if and only if C^U is also a maxterm for some subtree T_i, with $i \in \{1, \ldots, p\}$.

For the game tree function

$$T = \max\{\min\{x_1, x_2\}, \min\{x_3, \max\{x_4, x_5\}\}\},$$

the family of maxterms is $\{\{x_1, x_3\}, \{x_1, x_4, x_5\}, \{x_2, x_3\}, \{x_2, x_4, x_5\}\}$ and the family of minterms is $\{\{x_1, x_2\}, \{x_3, x_4\}, \{x_3, x_5\}\}$.

We define the value $C(\sigma)$ of a minterm (maxterm)

C w.r.t. assignment σ as the minimum (maximum) of the values of its leaves. We note that a proof for T, under an assignment σ, contains a minterm C^L and a maxterm C^U such that $C^U(\sigma) = C^L(\sigma) = T(\sigma)$.

Lower Bound. We start by presenting a lower bound on the competitive ratio of any deterministic algorithm for evaluating game trees. Our lower bound resembles the one proposed in [1]. However, we believe that such a lower bound includes a technical problem. Due to space constraints, the detailed discussion of this point is deferred to the extended version of this paper. The new lower bound presented here only allows to prove that the algorithm *BALANCE* from [1] has competitive ratio $4\gamma_c^f$ in contrast to the $2\gamma_c^f$-competitiveness claimed in [1]. Note that rather than stating that *BALANCE* is not $2\gamma_c^f$-competitive, we are here only claiming that the proof of this fact, as given in [1] is arguable, and our attempt to adjust it results in an additional 2 factor on the competitiveness.

Our lower bound makes use of the functions $\tau_L^T(\cdot)$ and $\tau_U^T(\cdot)$ presented below. In [1], it is shown that $\tau_L^T(y)$ ($\tau_U^T(y)$) is a lower bound on the minimum cost incurred by any deterministic algorithm, in the worst case, to determine a lower (upper) bound on the value of T when the cheapest lower (upper) bound witness costs at most y.

DEFINITION 2. *[1] Let T be a game tree and $c(\cdot)$ be the cost function on the leaves of T. For each leaf ℓ of T define $\tau_U^\ell(y) = \tau_L^\ell(y) = 0$, if $y < c(\ell)$ and $\tau_U^\ell(y) = \tau_L^\ell(y) = c(l)$, otherwise.*

Let ν be an internal node of T. Let $p = p_\nu$ be the number of children of ν and let us denote them by $\nu_1, \ldots, \nu_p$. If ν is a MIN node [2], then

$$\tau_U^\nu(y) = \sum_{i=1}^{p} \tau_U^{\nu_i}(y)$$

and

$$\tau_L^\nu(y) = \max_{\{y_1,\ldots,y_p\}:\sum_i y_i = y} \left(\sum_{i=1}^{p} \tau_L^{\nu_i}(y_i) \right) \tag{2.2}$$

If ν is a MAX node the definition of $\tau_U^\nu(y)$ and $\tau_L^\nu(y)$ can be obtained replacing in the two equations above every occurrence of U by L and every occurrence of L by U.

Finally, define $\tau_L^T(y) = \tau_L^r(y)$ and $\tau_U^T(y) = \tau_U^r(y)$, where r denote the root of T.

[2] In (2.2) the max operator is taken only over those y_i such that there can exist a minterm for the tree rooted at ν_i with cost at most y_i. If no such $y_1, \ldots, y_p$ exist then $\tau_U^\nu(y) = 0$.

The next theorem, whose proof is deferred for the extended version of this paper, gives a lower bound on the competitive ratio of any deterministic algorithm for evaluating game trees.

THEOREM 2.4. *Let T be a game tree T. Then, for each minterm C^L and maxterm C^U of T, we have that $\gamma_c^T \geq \frac{\max\{\tau_L^T(c(C^L)), \tau_U^T(c(C^U))\}}{c(C^L)+c(C^U)}$*

The Algorithm GREEDY-MIN **for Game Trees.** Consider a run of an algorithm for evaluating a game tree. Let Y be the set of leaves that have already been read. Let $UB(LB)$ be the minimum (maximum) among the values of the maxterms (minterms) that have been fully read. Otherwise, if none of the maxterms (minterms) has been completely read then $UB = \infty$ ($LB = -\infty$). Then, we know the value of T is in the interval $[LB, UB]$, with $LB < UB$. We say that a maxterm (minterm) C is *active* if for each leaf $x \in C \cap Y$, we have $x(\sigma) < UB$ ($x(\sigma) > LB$). In words, a maxterm (minterm) C is active if the evaluation of its unevaluated leaves can still lead to an improvement of the upper bound UB (lower bound LB), i.e., can provide additional information on the value of the game tree.

The algorithm GREEDY-MIN (GREEDY-MAX) can be easily applied to Game Trees since, with the generalized notion of minterms (maxterms) and active minterms (maxterms) stated above, the definitions of *ranks* and strongly active sets naturally extend to Game Trees.

Lemma 2.2 gives an upper bound on the cost spent by GREEDY-MIN (GREEDY-MAX) to prove that $f(\sigma) \geq B$ ($f(\sigma) \leq B$), for each assignment σ and for each lower (upper) bound B. Its proof has the same flavor of that of Theorem 2.1. Due to space constraints we omit it.

LEMMA 2.2. *Let f be a function that can be represented by a game tree.*

Let $\mathbb{I}$ be an arbitrary implementation of GREEDY-MIN. *Then, for every assignment σ and for every B such that $f(\sigma) \geq B$, we have*

$$c_{\mathbb{I}}^{f,B}(\sigma) \leq \tau_L(c_B^f(\sigma))$$

where $c_{\mathbb{I}}^{f,B}(\sigma)$ denotes the cost spent by $\mathbb{I}$ to find a certificate that $f(\sigma) \geq B$ and $c_B^f(\sigma)$ denotes the cost of the cheapest certificate that allows to proving that $f(\sigma) \geq B$.

Similarly, let $\mathbb{I}^+$ be an arbitrary implementation of GREEDY-MAX. *Then for every assignment σ and for every B such that $f(\sigma) \leq B$, we have*

$$c_{\mathbb{I}^+}^{f,B}(\sigma) \leq \tau_U(c_B^f(\sigma))$$

where $c^{f,B}_{\mathbb{I}^+}(\sigma)$ denotes the cost spent by $\mathbb{I}^+$ to find a certificate that $f(\sigma) \leq B$ and $c^f_B(\sigma)$ denotes the cost of the cheapest certificate that allows to proving that $f(\sigma) \leq B$.

To evaluate a game tree we can run Greedy-MIN and Greedy-MAX in 'parallel', that is, at each step the next variable to be read is either the next variable to be read in Greedy-MIN's execution or the next variable to be read in Greedy-MAX's execution. The decision will be to read the variable picked up by Greedy-MIN (Greedy-MAX) if, including the last variables chosen by the two algorithm the cost incurred by Greedy-MIN (Greedy-MAX) is smaller than the cost incurred by Greedy-MAX (Greedy-MIN) . The algorithm stops when the lower bound found by Greedy-MIN and the upper bound found by Greedy-MAX match. We use $\mathbb{PAR}$ to denote this *parallel* algorithm.

Let σ be the assignment for $\mathbb{PAR}$ which maximizes $c^T_{\mathbb{PAR}}(\sigma)/c^T(\sigma)$ and let C^L and C^U be respectively the minterm and the maxterm contained in the cheapest proof for the value of the game tree T w.r.t. σ. Assume, without loss of generality, that $\tau^T_L(c(C^L)) \leq \tau^T_U(c(C^U))$. Therefore, the competitive ratio of $\mathbb{PAR}$ is at most

$$\frac{2\tau^T_U(c(C^U))}{c(C^L \cup C^U)} \leq \frac{4\tau^T_U(c(C^U))}{c(C^L) + c(C^U)}, \tag{2.3}$$

where the last inequality follows from $c(C^L \cup C^U) \geq (c(C^L) + c(C^U))/2$.

Putting together Theorem 2.4 and (2.3), we can state the following theorem.

Theorem 2.5. *If the function f can be represented by a game tree, then $\gamma^{\mathbb{PAR}}_c(f) \leq 4\gamma^f_c$.*

We conclude this section by noticing that there exists a polynomial implementation of Greedy-MIN (Greedy-MAX) for game trees. In order to efficiently determine the active minterm (maxterm) C of minimum rank, such implementation relies on the removal of all leaves of the tree whose value is known to be not greater than the best lower bound so far LB (not smaller than the best upper bound so far, UB). A more detailed explanation is deferred to the extended version of this paper.

3 FEPI with Unknown Costs

In this section, we study the new model of Function Evaluation with Priced Information and Unknown Costs (FEPI-UC). Recall that since the costs are unknown, in the FEPI-UC we allow the evaluation strategies to be preemptive, i.e., the process of paying for reading the value of a variable can be stopped before the full cost has been paid. We also assume that the cost of reading a variable is lower bounded by some known value M. The formal definition of the allowed operations and their semantics are given in the introduction.

A Liner-Programming based algorithm. Let f be the function to evaluate and V its set of variables. Let $\mathcal{P}$ denote the set of all minimal proofs for f. We define the following linear program $\mathbf{LP_f}$ where we have one non-negative real variable $s(x)$ for each variable $x \in V$ and one constraint for each minimal proof $P \in \mathcal{P}$.

$$\mathbf{LP_f} : \text{Minimize } \sum_{x \in V} s(x)$$

$$\sum_{x \in P} s(x) \geq 1, \text{ for every } P \in \mathcal{P}$$

$$s(x) \geq 0, \text{ for every } x \in V$$

The procedure below uses the set Y to keep track of the variables whose values have already been determined. At each iteration of the most external loop, the algorithm finds a feasible solution $\mathbf{s}$ of LP_{f_Y}. This solution is then used to fix the relative *speed* at which each variable is read, i.e., for each variable $x \in V \setminus Y$, the algorithm iteratively increases the amount spent on x by $t \times s(x)$ (for some suitable t) until for some variable y the cost $c(y)$ has been completely paid. Then, the value of y is read, the set Y is updated and, if necessary, the algorithm starts a new loop by solving the linear program for f_Y.

We call the internal **For** loop a *phase* of the algorithm $\mathbb{LINPR}$. In each phase, for each variable x whose value is unknown, the amount $t \times s(x)$ is read. A phase can be interrupted if the value of some variable, say y, becomes determined during its execution. The instruction Break in the pseudo-code below forces the end of a phase.

Since the algorithm terminates when it determines the value of f, and each time a phase is interrupted a new variable is read, we can have at most n interrupted phases.

Algorithm $\mathbb{LINPR}(f, V, t)$
$Y \leftarrow \emptyset$;
While f is unknown
  Let s_Y be a feasible solution for LP_{f_Y}.
  Repeat
    For every variable $x \in V \setminus Y$ do
      Read(x, $t \times s_Y(x)$)
      If the value of x becomes known
        $Y \leftarrow Y \cup x$;
        Break
  Until the value of a new variable is found

We now provide an upper bound on the competitive ratio of the algorithm $\mathbb{LINPR}$.

Theorem 3.1. *Let* $s^* = \max_{Y \subset V} \sum_{x \in V-Y} s_Y(x)$. *Then,* $\mathbb{LINPR}$ *is* $s^* + \frac{n \times t \times s^*}{M}$*-competitive. In particular, when* t *tends to* 0, $\mathbb{LINPR}$ *becomes* s^**-competitive.*

Proof. In each phase, the algorithm spends at most $t \times s^*$. Thus, the cost spent due to the interrupted phases is at most $n \times t \times s^*$. The cost spent due to the non-interrupted phases is at most $p \times t \times s^*$, where p is the total number of non-interrupted phases. In total $\mathbb{LINPR}$ spends at most $(n + p) \times t \times s^*$

Now, let P be the cheapest proof for the value of f. We have that for every subset of variables Y read by $\mathbb{LINPR}$, $P \setminus Y$ is a minimal proof for f_Y. Since s_Y is a feasible solution for LP_{f_Y}, then $\sum_{x \in P \setminus Y} s_Y(x) \geq 1$. As a consequence, we can charge to the cheapest proof, t units from every non-interrupted phase. Since the minimum amount spent for reading one variable is M, we have that, the cheapest proof costs at least $c(P) \geq \max\{M, p \times t\}$.

Dividing the upper bound on the cost spent by $\mathbb{LINPR}$ by the lower bound on $c(P)$ we have the result ■

Note that, in this model, we neglect the cost of starting a *Read* operation. Alternatively, one could also try to keep the number of such operations small. If this is the case, the choice of a very small value for t is not acceptable. On the other hand, standard techniques can be employed to tune the parameter t to reduce the number of operation, e.g., doubling. In the absence of any additional modification, this would imply an additional factor 2 in the estimate of the competitiveness. We shall discuss this issue at greater length in the extended version of the paper.

Now we shall show that the $\mathbb{LINPR}$ is a useful tool for designing algorithms in the FEPI-UC model for monotone boolean functions. In fact, it can be adapted to deal with the classical FEPI model as we shall discuss in the extended version of this paper.

Implementations for Monotone Boolean Functions. For a monotone boolean function f over the set of variables $V = \{x_1, x_2, \ldots, x_n\}$, we shall use $k(f)$ and $l(f)$ to denote the size of the largest minterm and the largest maxterm of f, respectively. The next theorem presents a lower bound on the competitiveness of any algorithm for the FEPI-UC model, when the function to evaluate is a monotone boolean function.

Theorem 3.2. *For every monotone boolean function* f *and for every* $\epsilon > 0$, $\max\{k(f), l(f)\} - \epsilon$ *is a lower bound on the competitive ratio of any deterministic algorithm* $\mathbb{A}$ *that computes* f *in the FEPI-UC model.*

Proof. The adversary constructs an assignment σ_A and a vector of costs c as follows. Let C be the largest minterm of f. If $x \notin C$, then $x(\sigma)$ is set to 0 and $c(x) = M$. On the other hand, for every variable $x \in C$, the adversary sets $c(x) = h$. Finally, all the variables in C are assigned value 1 but the last one read by A. Let P be the cheapest proof for f, w.r.t. σ_A. We have that $|P| \leq l(f)$ and that P contains a variable of cost h. Thus, $c(P) \geq h + (l(f) - 1) \times M$. Therefore, $\frac{k(f)h}{h+(l(f)-1)\times M}$ is a lower bound on the competitive ratio of any deterministic algorithm. Since this expression goes to $k(f)$ as h goes to ∞, we have that $k(f) - \epsilon$ is a lower bound on the competitive ratio of any deterministic algorithm, for every $\epsilon > 0$.

A similar argument shows that $l(f) - \epsilon$ is also a lower bound. The proof is complete. ■

In general, a reasonable implementation of $\mathbb{LINPR}$ must find a good feasible solution for LP_f in polynomial time. By good we mean that the objective value associated with such a solution should not be far from that associated with the optimal one. Although there are polynomial time algorithms for solving the linear program problem, their application is limited since the number of equations of the linear program (number of certificates) may be exponential on the size of the function representation and, even worse, the separation problem may be NP-Complete.

In the following we show that we can obtain good solutions for LP_f without solving it. Let us assume that the access to our monotone boolean function f is given by an oracle that for every input $\mathbf{x} \in \{0,1\}^n$ responds whether $f(x) = 0$ or $f(x) = 1$. The next theorem shows that one can find in polynomial time (on the number of calls to the oracle) a feasible solution $\mathbf{s}$ for LP_f such that $\sum_{x \in V} s(x) \leq k(f) + l(f) - 1 \leq 2\max\{k(f), l(f)\}$.

Theorem 3.3. *If* f *is a monotone boolean function, then there is a polynomial time* $(k(f) + l(f) - 1)$*-competitive implementation for* $\mathbb{LINPR}$.

Proof. Let C^L and C^U be, respectively, arbitrary minterms and maxterms for f. We construct the speeds $s(x)$ by setting $s(x) = 1$ if $x \in C^L \cup C^U$ and $s(x) = 0$, otherwise. Recall that every certificate has a non-empty intersection with $C^L \cup C^U$. Thus, the above $s(x)$'s define a feasible solution for LP_f. Moreover, $\sum_{x \in V} s(x) \leq k(f) + l(f) - 1$.

For a polynomial time implementation it suffices to show how to find a minterm (maxterm) in polytime.

We only present the procedure for finding a minterm. An analogous procedure can be easily constructed for finding a maxterm.

Note that it is possible to verify in polynomial time whether a given set S is a 1-witness by evaluating f on the assignment σ_S where every variable of S is set to 1 and the remaining ones are set to 0. If $f(\sigma_S) = 0$ then S is not a 1-witness and, as a consequence, it does not contain a minterm. If $f(\sigma_S) = 1$ then, due to the monotonicity of f, S is 1-witness and it contains a minterm. This observation leads to the procedure below which finds a minterm in f by calling the oracle at most $|V|^2$ times.

$S \leftarrow V$.
While there is $v \in S$ such that $S \backslash \{v\}$ is a 1-witness
$\quad S \leftarrow S \setminus \{v\}$
Return S

This completes our proof. ■

Finally we show that if our monotone boolean function f can be represented by an AND/OR tree (a restriction of game trees where every leaf is associated with a boolean value), the situation is even better since we can always find in polynomial time a feasible solution $\mathbf{s}$ for LP_f such that $\sum_{x \in V} s(x) \leq \max\{k(f), l(f)\}$.

THEOREM 3.4. *If f can be represented by an AND/OR tree then there is a $\max\{k(f), l(f)\}$-competitive polynomial time implementation for* $\mathbb{LINPR}$

Proof. In [1], Charikar et. al. show how to construct in $O(n^2)$ time a vector of positive reals $\mathbf{p} =< p(x_1), \ldots, p(x_n) >$, called ultra-uniform price vector, that has the following properties: there exists positive numbers c_1 and c_2 such that $\sum_{x \in C_L} p(x) = c_1$ for every minterm C_L of f and $\sum_{x \in C_U} p(x) = c_0$ for every maxterm C_U of f.

Now, we give an indirect argument to show that $\sum_{x \in V} p(x) \leq \min\{c_1, c_2\} \max\{k(f), l(f)\}$. Let us consider an instance of FEPI with known costs, where the input is given by the function f and the costs of the variables are given by $\mathbf{p}$. Since every function represented by an AND/OR tree is evasive then every deterministic algorithm for evaluating f is forced to spend $\sum_{x \in V} p(x)$ before determining the value of f in the worst case. Hence, $\sum_{x \in V} p(x) / \min\{c_1, c_2\}$ is a lower bound on the competitive ratio of any deterministic algorithm.

On the other hand, there is a $\max\{k(f), l(f)\}$-competitive algorithm for evaluating AND/OR trees in the FEPI model with known costs [1, 2]. Thus, we have that $\sum_{x \in V} p(x) \leq \min\{c_1, c_2\} \max\{k(f), l(f)\}$.

Now, let us assume w.l.g that $c_1 \leq c_2$. For each $x \in V$, we define the *speed* $s(x)$ as follows, $s(x) = p(x)/c_1$. It is straightforward to check that this constitutes a feasible solution for LP_f and that $\sum_{x \in V} s(x) \leq \max\{k(f), l(f)\}$. ■

References

[1] M. Charikar, R. Fagin, V. Guruswami, J. Kleinberg, P. Raghavan, and A. Sahai. Query strategies for priced information. *JCSS: Journal of Computer and System Sciences*, 64:785–819, 2002.

[2] F. Cicalese and E. Laber. An optimal algorithm for querying priced information: Monotone boolean functions and game trees. In *ESA: Annual European Symposium on Algorithms*, 2005. To appear.

[3] F. Cicalese and E. S. Laber. A new strategy for querying priced information. In *Proceedings of the 37th ACM Symposium on Theory of Computing*, pages 674–683, Baltimore, 2005. ACM Press.

[4] A. Gupta and A. Kumar. Sorting and selection with structured costs. In IEEE, editor, *42nd IEEE Symposium on Foundations of Computer Science*, pages 416–425, 2001.

[5] A. Gupta, D. O. Stahl, and A. B. Whinston. Pricing of services on the internet. In *IMPACT: How IC2 Research Affects Public Policy and Business Markets.* Quorum Books, forthcoming.

[6] S. Kannan and S. Khanna. Selection with monotone comparison costs. In *Proceedings of the fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-03)*, pages 10–17, 2003.

[7] H. Kaplan, E. Kushilevitz, and Y. Mansour. Learning with attribute costs. In *Proceedings of the 37th ACM Symposium on Theory of Computing*, pages 356–365, Baltimore, 2005. ACM Press.

[8] S. Khanna and W. Tan. On computing functions with uncertainty. In *Symposium on Principles of Database Systems*, pages 171–182, 2001.

[9] E. Laber. A randomized competitive algorithm for evaluating priced AND/OR trees. In *STACS: Annual Symposium on Theoretical Aspects of Computer Science*, pages 501–512, 2004.

[10] E. Laber, R. Carmo, and Y. Kohayakawa. Querying priced information in databases: The conjunctive case: Extended abstract. In *LATIN: Latin American Symposium on Theoretical Informatics*, pages 6–15, 2004.

[11] Maheshwari and Smid. A dynamic dictionary for priced information with application. In *14th International Symposium on Algorithms and Computation (ISAAC 2003)*, volume 2906 of *Lecture Notes in Computer Science*, pages 16–25. Springer, 2003.