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 the cost function is not known in advance and some variant of the problem. Here, we assume that the cost $c(x)$ 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 $f$. We use $\sigma$ to denote the value of $f$ w.r.t. $\sigma$, i.e., $f(\sigma) = f(x_1(\sigma), \ldots, x_n(\sigma))$. An evaluation algorithm $A$ for $f$ under an assignment $\sigma$ is a rule to adaptively read the variables in $V$ until the set of variables read so far is sufficient with respect to $\sigma$. The cost of the algorithm $A$ for an assignment $\sigma$ is the total cost incurred by $A$ to evaluate $f$ under the assignment $\sigma$. Given a cost function $c(\cdot)$, we let $c^*_A(\sigma)$ denote the cost of the algorithm $A$ for an as-

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

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

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 $\sigma$, 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. $\sigma$, 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 $\sigma$ of $V$ if the value of $f$ is determined by the restriction $\sigma|_U$ of $\sigma$ to $U$. A set of variables which is sufficient is also called a proof of the value of $f$ for the given assignment $\sigma$.

An evaluation algorithm $A$ for $f$ under an assignment $\sigma$ is a rule to adaptively read the variables in $V$ until the set of variables read so far is sufficient with respect to $\sigma$. The cost of the algorithm $A$ for an assignment $\sigma$ is the total cost incurred by $A$ to evaluate $f$ under the assignment $\sigma$. Given a cost function $c(\cdot)$, we let $c^*_A(\sigma)$ denote the cost of the algorithm $A$ for an as-
ignment \( \sigma \) and \( c^f(\sigma) \) the cost of the cheapest proof for \( f \) under the assignment \( \sigma \). We say that \( \mathbb{A} \) is \( \rho \)-competitive if \( c^f(\sigma) \leq \rho c^f(\sigma) \), for every possible assignment \( \sigma \). We use \( \gamma^c(f) \) to denote the competitive ratio of \( \mathbb{A} \), that is, the minimum \( \rho \) for which \( \mathbb{A} \) is \( \rho \)-competitive. The best possible competitive ratio for any deterministic algorithm, then, is \( \gamma^c(f) = \min_{\mathbb{A}} \gamma^c(f) \), 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

\[
f = (x_1 \text{ AND } x_2) \text{ OR } (x_2 \text{ AND } x_3) \text{ OR } (x_3 \text{ AND } x_4)
\]

(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 \( \{x_2, x_1\} \). 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 \( c^f(\sigma_R) = 13 \), since \( \mathbb{A} \) reads the variables \( x_1, x_2, x_3, x_4 \). Furthermore, \( c^f(\sigma_S) = 12 \), 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 \( \text{Read}(x, t) \), 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 \( \text{Read} \) operations on \( x \) before executing the present \( \text{Read}(x, t) \). 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 \( \text{Read}(x, t) \) 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 \( \rho \) for which \( c^f(\sigma) \leq \rho c^f(\sigma) \) for every assignment \( \sigma \) 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 \( \gamma^c(f) \) 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 \( \gamma^c(f) \) for threshold tree functions (and \( \text{a fortiori} \) 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 $\gamma^c$ 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 $\gamma^c$-competitive pseudo-polynomial algorithm is provided. A variant of this algorithm is shown to achieve $2\gamma^c$-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 $2\gamma^c$-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 $4\gamma^c$-competitive. Thus, it appears to be no better than the algorithm we propose here for evaluating game trees, which is also a $4\gamma^c$-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 maximal competitive ratio, defined by $\gamma(f) = \min_{\mathcal{A}} \max_{c} \gamma^c(f)$, where the max is computed over all cost functions $c$ and the min over all algorithms $\mathcal{A}$. 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, $\gamma^c$-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 $\gamma^c$ 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 $\sigma_Y$ 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 $\sigma_Y$. 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 \( \text{leaves}(T) \) 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 \( Y \subseteq V \), \( T_Y \) 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 \( T_1, \ldots, T_p \) 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 \( R \subseteq \{1, \ldots, p\} \), with \( |R| = t \), such that: (i) \( C \cap \text{leaves}(T_i) \) is a minterm for \( T_i \), for each \( i \in R \); (ii) \( C \cap \text{leaves}(T_j) = \emptyset \) for each \( j \notin R \).

Analogously, \( C \) is a maxterm for \( T \) if and only if there exists a subset \( S \subseteq \{1, \ldots, p\} \), with \( |S| = p - t + 1 \), such that: (i) \( C \cap \text{leaves}(T_j) \) is a maxterm for \( T_j \), for \( j \in S \); (ii) \( C \cap \text{leaves}(T_j) = \emptyset \) for each \( j \notin S \). 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 \( \{1, 2, \ldots, p\} \).

**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 \( \ell \) of \( T \), define \( \theta^0_1(y) = \theta^1_1(y) = 0 \) if \( y < c(\ell) \) and \( \theta^0_1(y) = \theta^1_1(y) = c(\ell) \), otherwise.

For a \( t \)-node \( \nu \) of \( T \) with children \( \nu_1, \ldots, \nu_p \), define 1,

\[
\theta^i_1(y) = \max_{i \in [p] \backslash \{i\}} \left( \max_{\substack{y_1, \ldots, y_p \in p \in I \subseteq \{1, \ldots, p\} \text{ s.t. } \sum_{i \in t} \theta^i_1(y_i) + \sum_{i \in \ell \backslash t} \theta^i_1(y_{\ell}) = \gamma^\ell_1(y) \text{ if } y_{\ell} \in I \text{ and } \theta^i_1(y) \text{ otherwise}} \right).
\]

Define \( \theta^0_0(y) \) to be the function obtained by replacing in the definition of \( \theta^i_1() \) every occurrence of \( t \) with \( p - t + 1 \). Finally, define \( \theta^i_T(y) = \theta^i_1(y) \) and \( \theta^0_T(y) = \theta^0_0(y) \).

Then, \( \theta^i_T(y) \) (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,

\[
\gamma^T_c \geq \max \left\{ \max_{\sigma : f(\sigma) = 0} \frac{\theta^i_T(c^T(\sigma))}{c^T(\sigma)} , \max_{\sigma : f(\sigma) = 1} \frac{\theta^0_T(c^T(\sigma))}{c^T(\sigma)} \right\}
\]

**The Algorithm GREEDY-MIN.** Let \( F \) denote the family of minterms of \( f \). We shall now define a total order \( \chi \) on \( F \) that induces a sorting of the minterms of \( f \) in order of non-decreasing cost.

**Definition 1. (Ranks)** Let \( f = f(x_1, x_2, \ldots, x_n) \) be a boolean function and let \( c(\cdot) \) be a cost function on the variables of \( f \). Let \( \pi \) be the total order on the variables of \( f \) defined by stipulating that, for each \( i = 1, 2, \ldots, n - 1 \), \( x_i \) precedes \( x_{i+1} \) in the order \( \pi \). Therefore, \( c \) and \( \pi \) induce a total order \( \chi \) on the minterms of \( f \) as follows. In \( \chi \) 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 \( \pi \)) precedes the lexicographical order the list of variables in \( D \) (listed according to \( \pi \)).

For each minterm \( C \) of \( f \) we define \( \text{rank}_f(C) \) as the ordinal position of \( C \) in \( \chi \) (i.e., the number of minterms that precede \( C \) in \( \chi \), plus 1). When the function \( f \) is clear from the context we shall write \( \text{rank}(C) \) instead of \( \text{rank}_f(C) \).

The algorithm GREEDY-MIN below examines the minterms of \( F \) 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
  - \( C \leftarrow \text{active minterm of } f \) with minimum rank
  - \( U \leftarrow \text{a strongly active subset of } C \)
  - 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

---

1 In the definition of \( \theta^i() \) the second max operator is taken only over choices \( y_1, \ldots, y_p \) such that there can exist minterms for the trees rooted at \( \nu_1, \ldots, \nu_p \), with costs at most \( y_1, \ldots, y_p \), respectively. If no such \( y_1, \ldots, y_p \) exist for a particular \( y \) then the value of max is 0.
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 $I$ be an implementation for the algorithm GREEDY-MIN. For each function $f$ and for each assignment $\sigma$, the execution $I(f, \sigma)$ of the implementation $I$ of GREEDY-MIN on the function $f$ with assignment $\sigma$ is the sequence of pairs $(x_i, C(x_i)), 1 \leq i \leq q$, where $x_i$ is the $i$-th variable that $I$ 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 $I$ 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 $T_m$ in the order that they appear in $I(T, \sigma)$. Then, there exists an implementation $I_m$ for GREEDY-MIN that satisfies

(i) The first $q$ variables of $I_m(T_m, \sigma_{T_m})$ 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 $I(T, \sigma)$ (resp. $I_m(T_m, \sigma_{T_m})$) 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 $I$ be an arbitrary implementation of GREEDY-MIN. If $f$ can be represented by a threshold tree, then for every assignment $\sigma$ such that $f(\sigma) = 1$, we have $c^f_0(\sigma) \leq \theta^f_0(c^f(\sigma))$.

**Proof.** Recall the definition of the functions $\theta^f_0$ and $\theta^f_1$ given in Proposition 2.1. We shall prove, by induction on the height of the tree, that for every assignment $\sigma'$, the cost incurred by $I$ 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 $\sigma'$ is an assignment for which $f$ evaluates to 1 and $C$ is the cheapest proof for $f$ under the assignment $\sigma'$.

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 $\theta^f_1(c(l)) = c(l)$.

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.o.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 $\text{rank}(C') < \text{rank}(C)$. 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 $\text{rank}(C^*) < \text{rank}(C')$.

Let $D$ be the last minterm evaluated in the execution $I$ of $f$. If $\text{rank}(D) \leq \text{rank}(C^*)$ we have that $C'$ is not evaluated. On the other hand, if $\text{rank}(D) \geq \text{rank}(C^*) + 1$, then at the time when $I$ evaluates for the first time a minterm of rank $\geq \text{rank}(C^*) + 1$, 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 $c_{\text{max}} = \max_{i=1,\ldots,t}\{c(C \cap T_i)\}$. For the sake of contradiction, we assume that $c(C' \cap T_j) > c_{\text{max}}$ for some $j > t$. Let $T_j$, with $i \leq t$, be a subtree such that $C' \cap T_i = \emptyset$ and define $C^* = (C' \setminus T_i) \cup (C \cap T_i)$. 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 $I(T, \sigma)$ before determining the value of $C$, listed in the order in which they are read by $I$. 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 $I_i$ such that the sequence of variables in the execution $I_i(T_i, \sigma_{T_i})$ 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 $i > t$), $I_i$ only evaluates minterms of cost not larger than $c(C \cap T_i)$ (respectively $c_{\text{max}}$) 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 $\theta^T_i(c(C \cap T_i))$ (respectively $\theta^T_i(c_{\text{max}})$).

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

$$\sum_{i=1}^{t} \theta^T_i(c(C \cap T_i)) + \sum_{i=t+1}^{p} \theta^T_i(c_{\text{max}}) \leq \theta^T_0(c(C)),$$

where the inequality directly follows from the definition of $\theta^T_0(y)$. 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 \( \theta_0 \) and \( \theta_1 \), one can easily prove the following dual result.

**Theorem 2.2.** Let \( T \) be an arbitrary implementation of Greedy-MAX. If \( f \) can be represented by a threshold tree, then for every assignment \( \sigma \) such that \( f(\sigma) = 0 \), we have \( c^f_\ell(\sigma) \leq \theta_0^\ell(c^f_\ell(\sigma)) \).

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 \( \gamma^p \)-competitiveness, as it is proved in the next theorem.

**Algorithm Greedy*\(^g\)(f, V, 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 \( \mathcal{A} \) be an implementation of Greedy*\(^g\). 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 \( \gamma^\mathcal{A}_c(f) = \gamma^f_c \).

**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 \( f \), hence their intersection cannot be empty.

Therefore, Greedy*\(^g\) 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

\[
\gamma^\mathcal{A}_c(f) = \max_{\sigma} \frac{c^f_\ell(\sigma)}{c^f_\ell(\sigma)} \leq \max \left\{ \max_{\sigma: f(\sigma) = 0} \frac{\theta_0^\ell(c^f_\ell(\sigma))}{c^f_\ell(\sigma)}, \max_{\sigma: f(\sigma) = 1} \frac{\theta_1^\ell(c^f_\ell(\sigma))}{c^f_\ell(\sigma)} \right\} \leq \gamma^f_c
\]

\( \blacksquare \)

For the polynomial implementation of \( \mathcal{A} \) 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 minterm 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 minterms is \( \{\{x_1, x_2\}, \{x_1, x_4, x_5\}, \{x_2, x_3\}, \{x_2, x_4, x_5\}\} \) and the family of minterms is \( \{\{x_1, x_2\}, \{x_2, x_4, x_5\}, \{x_3, x_5\}\} \).

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

**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 [1] has competitive ratio \( 4 \gamma \) in contrast to the \( 2 \gamma \)-competitiveness claimed in [1]. Note that rather than stating that BALANCE is not \( 2 \gamma \)-competitive, we 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 \( \tau_c^L(\cdot) \) and \( \tau_c^U(\cdot) \) presented below. In [1], it is shown that \( \tau_c^L(y)(\tau_c^L(y)) \) 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 \( \ell \) of \( T \) define \( \tau_c^L(y) = \tau_c^U(y) = 0 \), if \( y < c(\ell) \) and \( \tau_c^L(y) = \tau_c^U(y) = c(\ell) \), otherwise.

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

\[
\tau_c^U(y) = \sum_{i=1}^{p} \tau_c^U(y_i)
\]

and

\[
\tau_c^L(y) = \max_{\{y_1, \ldots, y_p\}} \sum_{y_i = y} \left( \sum_{i=1}^{p} \tau_c^L(y_i) \right)
\]

(2.2)

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

Finally, define \( \tau_c^L(y) = \tau_c^U(y) \) and \( \tau_c^L(y) = \tau_c^U(y) \), 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 \( \nu_i \) with cost at most \( y \). If no such \( y_1, \ldots, y_p \) exist then \( \tau_c^L(y) = 0 \).

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

\[
\gamma_c^T \geq \frac{\max(\tau_c^L(c(C^L)), \tau_c^U(c(C^U)))}{c(c(C^L)) + c(c(C^U))}
\]

**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.** 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 \( \sigma \) 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.
where \( c_1^{f,B}(\sigma) \) denotes the cost spent by \( \Gamma^+ \) to find a certificate that \( f(\sigma) \leq B \) and \( c_0^{f}(\sigma) \) 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 value 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 \( \text{PAR} \) to denote this parallel algorithm.

Let \( \sigma \) be the assignment for \( \text{PAR} \) which maximizes \( c_{\text{PAR}}^f(\sigma)/c^f(\sigma) \) 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. \( \sigma \). Assume, without loss of generality, that \( \gamma^L_{\text{PAR}}(c(C^L)) \leq \gamma^U_{\text{PAR}}(c(C^U)) \). Therefore, the competitive ratio of \( \text{PAR} \) is at most

\[
\gamma^U_{\text{PAR}}(f) \leq 4\gamma^L_{\text{PAR}}(f)
\]

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 \( \gamma^U_{\text{PAR}}(f) \leq 4\gamma^L_{\text{PAR}}(f) \).

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 Linear-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 \( \mathcal{LP}_f \) where we have one non-negative real variable \( s(x) \) for each variable \( x \in V \) and one constraint for each minimal proof \( P \in \mathcal{P} \).

\[
\mathcal{LP}_f : \begin{align*}
\text{Minimize} \quad & \sum_{x \in V} s(x) \\
\text{subject to} \quad & \sum_{x \in P} s(x) \geq 1, \text{ for every } P \in \mathcal{P} \\
& s(x) \geq 0, \text{ for every } x \in V
\end{align*}
\]

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 \( s \) of \( \mathcal{LP}_f \). 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 \( \text{LINPR} \). 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** \( \text{LINPR}(f, V, t) \)

\begin{verbatim}
Y ← Ø;
While f is unknown
    Let s_Y be a feasible solution for \( \mathcal{LP}_{f_Y} \).
    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 ← Y \cup x;
            Break
    Until the value of a new variable is found
\end{verbatim}
We now provide an upper bound on the competitive ratio of the algorithm LINPR.

**Theorem 3.1.** Let $s^* = \max_{Y \subseteq V} \sum_{x \in V \setminus Y} s_Y(x)$. Then, LINPR is $s^* + \frac{nM}{M}$-competitive. In particular, when $t$ tends to 0, 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 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 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 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 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 $A$ that computes $f$ in the FEPI-UC model.

**Proof.** The adversary constructs an assignment $\sigma$ 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. $\sigma$. 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, $\frac{k(f) + l(f)}{h + (l(f) - 1) \times M}$ is a lower bound on the competitive ratio of any deterministic algorithm. Since this expression goes to $k(f)$ as $h$ goes to $\infty$, 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 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 $x \in \{0, 1\}^n$ 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 $s$ 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 LINPR.

**Proof.** Let $CL$ and $CU$ be, respectively, arbitrary minterms and maxterms for $f$. We construct the speeds $s(x)$ by setting $s(x) = 1$ if $x \in CL \cup CU$ and $s(x) = 0$, otherwise. Recall that every certificate has a non-empty intersection with $CL \cup CU$. 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 $\sigma_S$ 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 $S \setminus \{v\}$ is a 1-witness  

$S \leftarrow S \setminus \{v\}$  

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 $s$ 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 $\text{LINPR}$.  

**Proof.** In [1], Charikar et. al. show how to construct in $O(n^2)$ time a vector of positive reals $\mathbf{p} = \langle p(x_1), \ldots, p(x_n) \rangle$, called ultra-uniform price vector, that has the following properties: there exists positive numbers $c_1$ and $c_2$ such that $\sum_{x \in C_1} p(x) = c_1$ for every minterm $C_1$ of $f$ and $\sum_{x \in C_0} p(x) = c_0$ for every maxterm $C_0$ 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, $s(x) = \frac{p(x)}{c_1}$. 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**


