Pontifícia Universidade Católica
do Rio de Janeiro

Aline Medeiros Saettler

On the Simultaneous Minimization of Worst Testing Cost and Expected Testing Cost with Decision Trees

DISSERTAÇÃO DE MESTRADO

Dissertation presented to the Programa de Pós-Graduação em Informática of the Departamento de Informática, PUC–Rio as partial fulfillment of the requirements for the degree of Mestre em Informática

Advisor: Prof. Eduardo Sany Laber

Rio de Janeiro
August 2013

Pontifícia Universidade Católica do Rio de Janeiro

Aline Medeiros Saettler

On the Simultaneous Minimization of Worst Testing Cost and Expected Testing Cost with Decision Trees

Dissertation presented to the Programa de Pós-Graduação em Informática of the Departamento de Informática do Centro Técnico Científico da PUC-Rio, as partial fulfillment of the requirements for the degree of Mestre.

Rio de Janeiro, August 23th, 2013

Aline Medeiros Saettler

Aline Medeiros Saettler obtained a bachelor's degree in Computer Science from the Federal University of Espírito Santo (Vitória, Brazil). During her undergraduate years she held a scholarship from the university to study interactive applications for digital television. She also earned a scholarship from CAPES to do her masters at PUC-Rio.

Bibliographic data

Saettler, Aline Medeiros

On the Simultaneous Minimization of Worst Testing Cost and Expected Testing Cost with Decision Trees / Aline Medeiros Saettler ; advisor: Eduardo Sany Laber. — 2013.
51 f. : il. ; 30 cm

Dissertação (Mestrado em Informática)-Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2013.
Inclui bibliografia

1. Informática – Teses. 2. Análise de algoritmos;. 3. otimização combinatória;. 4. algoritmos de aproximação;. 5. árvores de decisão.. I. Laber, Eduardo Sany. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

To my family, who is always with me, even from afar.

To my advisor, Eduardo Laber, for the guidance and support, and for being a source of inspiration to me.

To my friends: those from PUC-Rio, who accompanied me during these two years, and those from Vitória, who helped me to get where I am today.

Abstract

Saettler, Aline Medeiros; Laber, Eduardo Sany (Advisor). **On the Simultaneous Minimization of Worst Testing Cost and Expected Testing Cost with Decision Trees** . Rio de Janeiro, 2013. 51p. MSc Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The problem of minimizing the cost of evaluating a discrete function by sequentially reading its variables is a problem that arises in several applications, among them automatic diagnosis design and active learning. In this problem, each variable of the function is associated with a cost, that we have to pay in order to check its value. In addition, there may exist a probability distribution associated with the points where the function is defined. Most of the work in the area has focussed either on the minimization of the maximum cost or on the minimization of the expected cost spent to evaluate the function. In this dissertation, we show how to obtain an $O(\log n)$ approximation with respect to the worst case minimization (the best possible approximation under the assumption that $\mathcal{P} \neq \mathcal{NP}$). We also show a polynomial time procedure for evaluate a function that simultaneously optimizes both the worst and the expected costs.

Keywords

Analysis of Algorithms; combinatorial optimization; approximation algorithms; decision trees.

Resumo

Saettler, Aline Medeiros; Laber, Eduardo Sany. **Minimização Simultânea do Pior Custo e do Custo Médio em Árvores de Decisão**. Rio de Janeiro, 2013. 51p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

O problema de minimizar o custo de avaliar uma função discreta lendo sequencialmente as suas variáveis é um problema que surge em diversas aplicações, entre elas sistemas de diagnóstico automático e aprendizado ativo. Neste problema, cada variável da função está associada a um custo, que se deve pagar para checar o seu valor. Além disso, pode existir uma distribuição de probabilidades associadas aos pontos onde a função está definida. A maioria dos trabalhos nesta área se concentra ou na minimização do custo máximo ou na minimização do custo esperado gasto para avaliar a função. Nesta dissertação, mostramos como obter uma $O(\log n)$-aproximação em relação à minimização do pior custo (a melhor aproximação possível assumindo que $\mathcal{P} \neq \mathcal{NP}$). Nós também mostramos um procedimento polinomial para avaliar uma função otimizando simultaneamente o pior custo e o custo esperado.

Palavras–chave

Análise de algoritmos; otimização combinatória; algoritmos de aproximação; árvores de decisão.

Contents

1 Introduction

In this thesis, we study the *Discrete Function Evaluation Problem* (DFEP): given a discrete function, we have to determine the value of the function on a given point by sequentially reading the values of its variables. This problem also appears in the literature as the problem of identifying the class (5) (or group (14)) to which an object belongs by asking questions about the object. The questions and answers correspond to the variables and its values, and the classes correspond to function values.

Each variable of the function is associated with a known cost, that we have to pay in order to check its value. In addition, there may exist a probability distribution associated with the points where the function is defined. Typically it's not necessary to check all the variables to evaluate the function (in the same manner that we don't need to answer all possible questions about an unknown object to get some information about it). Consider, for instance, the following boolean function:

$$f(x_1, x_2, x_3) = x_2 \vee (x_1 \wedge \overline{x_2} \wedge \overline{x_3}) \vee (x_1 \wedge \overline{x_2} \wedge x_3)$$

Suppose that the cost to evaluate each variable is equal to 1 and we first choose x_1 to check. If $x_1 = 1$, then $f(x_1, x_2, x_3) = 1$ and we don't need to read any other variable. However, if $x_1 = 0$, we can evaluate x_2 and get the final result: the value of the function is 1 if and only if $x_2 = 1$. In the worst case, we will pay a total cost at most 2. On the other hand, suppose that we choose x_2 to evaluate first and get $x_2 = 0$. Then, we choose x_3 as the second variable to read. In this case, we will pay a cost of 3 to evaluate the function, regardless of the value of x_3.

Thus, we can consider two different problems: in which order shall we choose the variables to achieve our goal either paying the minimum total cost or the minimum expected cost?

Decision versions of these two problems are known to be NP-Complete problems (21). In this dissertation, our goal is to study and find approximation algorithms to solve these questions. The total cost and the expected cost are known as, respectively, *worst testing cost*, and *expected testing cost*, and will be properly defined in the next section.

We first consider the problem of approximating the worst testing cost. Next, we study the issue regarding the existence of a trade-off between the approximation of the worst testing cost and the approximation of the expected testing cost.

The DFEP arises in several applications, such as automatic diagnosis design and active learning. Consider, for example, an automatic trading agent implementing a high frequency trading strategy. In order to decide the next action to be performed, as sending or canceling a buy/sell order, the agent takes into account market variables as well as private variables (28). Among relevant market variables we may have the stock price, traded volume, volatility, order books distributions as well as complex functions involving these variables. Depending on the scenario given by the values of the market variables, the agent does not need to read all the available variables to decide its next action. As an example, if the price of a stock falls below some threshold, the agent buys the stock regardless of the other variables' values. In a competitive market, where market conditions change in a millisecond basis, being able to react very quickly to a new scenario may be the difference between a profitable and a loosing strategy. The problem faced by the agent is how to evaluate as quick as possible a function that maps the set of scenarios into the action to be performed.

Another situation where the DFEP emerges is in active learning. Assume that we are given a set of hypothesis $\{H_1, ..., H_m\}$, a set E of unlabeled examples and we are asked to select the hypothesis that is coherent with all examples. An hypothesis H is a map from the set of examples to the set of labels. To achieve the goal we apply the following steps: (i) pick an example $e \in E$; (ii) label e; (iii) discard all hypothesis that are not coherent with the label of e. These steps are repeated until only one hypothesis remains. Since the process of labeling an example may be expensive either in terms of time or money, we are interested to identify the coherent hypothesis spending as little as possible. In this case, selecting the next example to label corresponds to deciding the next variable to read.

Finally, consider the scenario where a person has an unknown disease and he (she) needs to identify it through a series of medical tests, each of them associated with a price. The result of an specific test can eliminate the possibility of some diseases, and therefore some other tests become unnecessary. However, the person obviously cannot know the result without taking the test. Thus, the doctor has to choose a sequence of tests in order to minimize the total price that can be paid to identify the disease. If all tests have the same price, for example, is desirable to take, in the worst of the cases, as few tests as possible.

1.1 Problem Definition

We define the input of the DFEP as a quintuple $(S, C, T, \mathbf{p}, \mathbf{c})$, where $S = \{s_1, \ldots, s_n\}$ is a set of n objects, $C = \{C_1, C_2, \ldots, C_k\}$ is a partition of S into k classes, $T = \{t_1, \ldots, t_m\}$ is a set of m tests, $\mathbf{p}$ is a probability distribution over the objects of S and $\mathbf{c}$ is a cost function assigning to each test t a cost $c_t \in \mathbb{Q}^+$, the set of non-negative rational numbers. A test $t \in T$, when applied to an object in S, incurs a cost c_t and outputs a number in the set $\{1, \ldots, \ell\}$. When $\ell > 2$, we have a multiway test. One of the objects in S is marked and its identity is hidden; the probability of s being the marked object is p_s. An example is shown in Table 3ftab:objects. In this example, we have $S = \{s_1, s_2, s_3, s_4, s_5\}$, $C = \{C_1, C_2, C_3\}$, $T = \{t_1, t_2, t_3\}$ and $\ell = 2$.

Object	$\mathbf{t_1}$	$\mathbf{t_2}$	$\mathbf{t_3}$	**Class**	**Probability**
s_1	1	1	2	C_1	0.1
s_2	1	2	1	C_1	0.2
s_3	2	2	1	C_2	0.4
s_4	1	2	2	C_3	0.25
s_5	2	2	2	C_3	0.05
Cost	2	1	3		

Table 1.1: An input with 5 objects, 3 tests, 3 classes and $\ell = 2$. Suppose that the costs of the tests are $c_{t_1} = 2$, $c_{t_2} = 1$ and $c_{t_3} = 3$.

We can also view S as a subset of $Z^{|T|}$, where $Z = \{1, \ldots, \ell\}$ and S is mapped to a class by a function $f : S \mapsto C$. The range of the function is a class between C_1 and C_k and the values of the $|T|$ coordinates of an element in S are the answers for the tests. Table 1.2 shows this correspondence for objects of Table 1.1.

Point	**Coordinates**	**Image of f**	**Probability**
s_1	(1, 1, 2)	C_1	0.1
s_2	(1, 2, 1)	C_1	0.2
s_3	(2, 2, 1)	C_2	0.4
s_4	(1, 2, 2)	C_3	0.25
s_5	(2, 2, 2)	C_3	0.05

Table 1.2: Points corresponding to objects in table 1.1.

We assume that the set of tests is complete, that is, for every s_i and s_j belonging to different classes, there is at least one test in T that outputs different values for s_i and s_j. A testing procedure that reveals the class of the marked object can be represented by a *decision tree*, which is a tree where every node is associated with a test and every leaf is associated with a class.

More formally, a decision tree D for $(S, C, T, \mathbf{p}, \mathbf{c})$ is a leaf associated with class i if every object of S belongs to the same class i. Otherwise, the root r of D is associated with some test $t \in T$ and the children of r are decision trees for the sets $\{S_t^1, S_t^2, ..., S_t^\ell\}$, where S_t^i, for $i = 1, \ldots, \ell$, is the subset of S that outputs i for test t.

Consider the example given in Table 1.1. Figure 1.1 shows a decision tree for these objects:

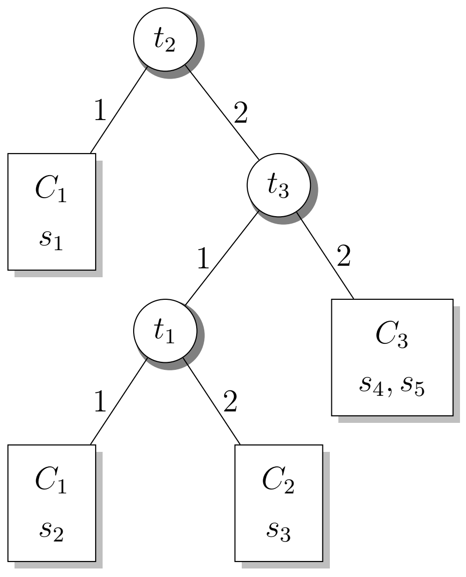


Figure 1.1: Decision Tree for objects of Table 1.1. Squares indicate leaf nodes, that can be associated with more than one object (note that objects s_4 and s_5 are associated with the same leaf).

Note that we can have more than one leaf associated with an specific class. In Figure 1.1 there are two leaves associated with class C_1.

The cost of a root-to-leaf path in a decision tree is defined as the sum of the costs of the tests associated with the nodes in the path. We use $Cost(D, s)$ to denote the cost of a path from the root of D to the leaf where object s is located. Then, we can define the *worst testing cost* of D as:

$$Cost_W(D) = \max_{s \in S}\{Cost(D, s)\} \tag{1-1}$$

And the *expected testing cost* of D is given by:

$$Cost_E(D) = \sum_{s \in S} Cost(D, s) p_s \tag{1-2}$$

The worst testing cost of the decision tree in Figure 1.1 is $\max\{1, (1 + 3 + 2), (1 + 3 + 2), (1 + 3)\} = 6$, and the expected testing cost is $(1 \times 0.1) + (6 \times 0.2) + (6 \times 0.4) + (4 \times 0.3) = 4.9$.

Now we can formulate the problems aforementioned more precisely:

1. *What is the decision tree with minimum worst testing cost for a given quintuple* $(S, C, T, \mathbf{p}, \mathbf{c})$*?*

2. *What is the decision tree with minimum expected testing cost for a given quintuple* $(S, C, T, \mathbf{p}, \mathbf{c})$*?*

Even in the simplest case where all testing costs and probabilities are uniform, with $\ell = 2$, these two problems are known to be NP-Complete problems (21). In fact, they are also hard to approximate: an $o(log(n))$-approximation implies that $\mathcal{P} = \mathcal{NP}$ (7, 23).

1.2 Our Results

Our first result is an algorithm for the DFEP that approximates the worst testing cost in the general case, where the testing costs can be uniform or not. Other approximations for the worst testing cost are not known in the literature, even for uniform costs. The algorithm, called DividePairs, produces a tree whose worst testing cost is at most $O(ln(n))$ times the worst testing cost of the optimal decision tree. This is the best possible result under the assumption that $\mathcal{NP}$ does not admit a polynomial time algorithm.

Our second contribution is a polynomial time procedure called CombineTrees, that given a parameter $\rho > 0$ and two decision trees D_W and D_E, the former with worst testing cost W and the latter with expected testing cost E, it produces a decision tree D with worst testing cost at most $(1 + \rho)W$ and expected testing cost at most $(1 + 1/\rho)E$.

Finally, some experiments were done in order to analyse the performance of the proposed algorithms in practice. We compared the DividePairs to other algorithms available in literature, and used these algorithms to create decision trees which were used to measure the performance of CombineTrees. We used both real and synthetic data to run the experiments. The final results show that DividePairs builds trees with a low worst cost. Our second algorithm (CombineTrees), with the proper modifications, also presented good results.

1.3 Related Work

The DFEP has been recently studied under the names of class equivalence problem (14) and group identification problem (5). We shall mention, however, that this problem had been defined much before as described in the excellent survey by Moret (6). Both (14) and (5) give $O(\log(1/p_{min}))$ approximation algorithms for the version of the DFEP, where the expected testing cost has to be minimized and both the probabilities and the testing costs are non-uniform. In addition, when the testing costs are uniform both algorithms can be converted into a $O(\log n)$ approximation algorithm via Kosaraju approach (22). The algorithm in (14) is more general because it addresses multiway tests rather than binary ones.

The particular version of the DFEP where each object belongs to a different class is known as the *identification problem* and it has a vast literature. This problem was first studied by Garey (12) who proposed an exponential algorithm based on dynamic programming. Both the worst testing cost and the expected testing cost minimization are known to be NP-complete (21), even when the tests are binary and the testing costs are uniform. In fact, both minimization goals do not admit a sublogarithmic approximation unless $\mathcal{P} = \mathcal{NP}$ as proved by (23) and (7).

For the expected testing cost, Kosaraju et. al. (22) presents a $O(\log n)$ approximation for binary tests and uniform costs. Adler et. al. (2) presents a $O(\log n)$ approximation for binary tests, uniform probabilities and non-uniform costs. In (7), Chakaravarthy et. al. started the study of multiway tests. They present an $O(\log^2 n)$ approximation for uniform testing costs. In (8), a $\log n$ factor is cut for instances with uniform probabilities. For the most general case with multiway tests, non uniform probabilities and non uniform testing costs, an $O(\log(1/p_{min}))$ approximation is given in (15).

Most of the strategies mentioned so far fit into a general framework commonly referred to as Generalized Binary Search (GBS). Algorithms based on this approach greedily select a test t that minimizes the ratio between the testing cost and the balance of the partition induced by t on the set of objects. However, the best known result to minimize the expected testing cost for the identification problem, due to Gupta et al. (18), cannot be seen, at least directly, as a GBS. It achieves a $O(\log n)$ approximation for the most general case, with multiway tests and non-uniform testing costs and probabilities.

The minimization of the worst testing cost for the identification problem has received less attention. In (3), Arkin et. al. presents a $\log n$ approximation algorithm for binary tests and uniform costs. A logarithmic approximation for the general problem with mutliway test and non-uniform testing costs is given by Hanneke (19). Results regarding the minimization of the worst testing cost are also presented in (16, 17). We are aware of only a few results that discuss trade-offs between the minimization of the expected testing cost and the worst testing cost for DFEP. In (9), Cicalese et. al. observed that a GBS based algorithm guarantees an $O(\log n)$ approximation for both criteria for the restricted version of the identification problem where the probabilities are uniform.

There are also results of this flavor for the problem of constructing an optimal prefix code, a problem that can be viewed as a very restricted version of the identification problem. In this problem, all tests have unitary costs and the set of tests is in one to one correspondence with the set of all binary strings of length n. The test corresponding to a binary string $\mathbf{b}$ outputs 0 (1) for object s_i if and only if the i^{th} bit of $\mathbf{b}$ is 0 (1). The goal is to construct a tree with optimal expected testing cost, which can be solved by Huffman's algorithm (11). The references (13), (20), (25) and (24) present algorithms for a more restricted case of this problem where the worst cost of the tree cannot exceed a given constant L.

Another special case of the identification problem occurs when the elements of S form a partially ordered set. A Hasse diagram is a graph where an edge (A, B) from a node A to a node B indicates that $A > B$ and there is no C such that $A > C > B$. Hasse diagrams can be represented by several decision trees, whose leaves are its nodes. This version of the problem was defined by (26) and studied by (23).

1.4 Thesis Organization

This work is organized as follows: Chapter 2 presents basic concepts about approximation algorithms; Chapter 3 shows a new algorithm that approximates the worst testing cost and the procedure to minimize both the worst and the expected costs of a tree. In Chapter 4 some experimental results related to chapter 3 are presented; finally, Chapter 5, discusses conclusions and future directions of this work.

2 Basic Concepts

In this Chapter, we introduce some basic concepts needed to the understanding of this dissertation. We present the definition of an algorithm to build a decision tree with optimal expected testing cost and the definition of approximation algorithms.

2.1 Optimal Algorithms to build Decision Trees

A dynamic programming method can be used to compute optimal decision trees and was described by (4), (29) and (27). Consider, for example, the algorithm to construct an optimal decision tree with respect to the expected testing cost (the optimal algorithm w. r. t. the worst cost is similar). The idea is to build successively larger optimal subtrees from smaller optimal subtrees. The algorithm can be represented by the following recurrence relation:

$$OPT(f) = \begin{cases} 0 & \text{if f is constant} \\ \min\limits_{t \in T} \left(c_t \pi_f + \sum\limits_{i=1}^{\ell} OPT(f|_{\mathcal{A}(t)=i}) \right) & \text{otherwise} \end{cases} \tag{2-1}$$

where $f|_{\mathcal{A}(t)=i}$ is the same function, but taking into account only the points for which t outputs i, and π_f is the sum of the probabilities associated with the points where f is defined. We can use memoization (11) to avoid repeated calculations, but the number of restrictions can be as large as $(\ell+1)^m$, where $m = |T|$. The optimal algorithms (for both the worst and the expected costs) were implemented for comparison purposes.

2.2 Approximation Algorithms

Since the DFEP is NP-Complete, no polynomial algorithm to find optimal decision trees with respect to the worst and expected costs is known. However, in many situations we can be interested in a solution that is not necessarily optimal, but is “good enough”. An *approximation algorithm* is an algorithm that finds a solution which is always within a factor of the optimal one. As defined in (10), an algorithm for a minimization problem with input of size n has a *ratio bound* of $\rho(n)$ if:

$$\frac{C(I)}{C^*(I)} \leq \rho(n),$$

for every instance I of the problem, where $C(I)$ is the cost of the solution produced by the algorithm and $C^*(I)$ is the cost of the optimal solution. We say that an algorithm with this property has a *$\rho(n)$-approximation*.

2.2.1 Example: an Approximation Algorithm for Vertex Cover

As an example, consider the following definition: given an undirected graph $G = (V, E)$, a subset $V' \subseteq V$ is a *cover* for G if for every edge $(u, v) \in E$ we have that $u \in V'$ or $v \in V'$. We say that each vertex of V' *covers* its incident edges. The problem of finding the smallest subset V' that covers G is called the *minimum vertex cover problem* and it's NP-Complete. Consider the solution provided by algorithm 1, presented in (10):

```
1 APPROX-VERTEX-COVER($G$)
2 $C = \emptyset$
3 $E' = E$
4 while $E' \neq \emptyset$ do
5     Let $(u, v)$ be an arbitrary edge of $G$;
6     $C = C \cup \{u, v\}$
7     remove from $E'$ every edge incident on either $u$ or $v$
8 end
9 return $C$
```

Algorithm 1: APPROX-VERTEX-COVER

Note that the solution C returned by Algorithm 1 is not only a cover, but is also a 2-approximation, since no two edges selected by APPROX-VERTEX-COVER in line 5 share an endpoint, and all selected edges have to be covered by an optimal solution.

That is:

$$\frac{1}{2}C(I) \leq C^*(I)$$

and so:

$$\frac{C(I)}{C^*(I)} \leq 2$$

3
An Approximation for the Worst Case and a Bicriteria Approximation

In this Chapter, we first show an algorithm for the DFEP that attains an approximation of $O(\log n)$ w. r. t. the worst cost. We consider the most general case of non-uniform costs and multiway tests. We also show a polynomial time procedure that simultaneously approximates both the worst and the expected testing costs.

3.1
A Logarithmic Approximation for the Worst Testing Cost

Let $(S, C, T, \mathbf{p}, \mathbf{c})$ be an instance of DFEP and let S' be a subset of S. In addition, let C' and $\mathbf{p}'$ be, respectively, the restrictions of C and $\mathbf{p}$ to the set S'. For example, for the subset $S' = \{s_1, s_2, s_3\}$ in Table 3.1 we have $C' = \{C_1, C_2\}$, and objects s_1, s_2 and s_3 are mapped to the same classes for which they were mapped in C.

Our first observation is that every decision tree D for $(S, C, T, \mathbf{p}, \mathbf{c})$ is also a decision tree for $(S', C', T, \mathbf{p}', \mathbf{c})$. The following proposition is a direct consequence of this observation.

Proposition 1. *Let $(S, C, T, \mathbf{p}, \mathbf{c})$ be an instance of the DFEP and let S' be a subset of S. Then, $OPT_E(S') \leq OPT_E(S)$ and $OPT_W(S') \leq OPT_W(S)$.*

We say that a pair of objects (s_i, s_j) from a set S is *separable* if s_i and s_j belong to different classes. For a set of objects G we use $P(G)$ to denote the number of separable pairs in G. In formulae,

$$P(G) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} n_i n_j, \tag{3-1}$$

where n_i is the number of objects in G that belong to class i. The set given in Table 3.1 has $(2 \times 1) + (2 \times 2) + (1 \times 2) = 8$ separable pairs.

We also have the following proposition:

Proposition 2. *For any group of objects G, we have that $P(G) < n^2$, where $n = |G|$.*

In the worst case, where the number of objects is equal to the number of classes, we have $P(G) = (n-1) + (n-2) + \ldots + 1 = n(n-1)/2 < n^2$.

Object	$\mathbf{t_1}$	$\mathbf{t_2}$	$\mathbf{t_3}$	**Class**	**Probability**
s_1	1	1	2	C_1	0.1
s_2	1	2	1	C_1	0.2
s_3	2	2	1	C_2	0.4
s_4	1	2	2	C_3	0.25
s_5	2	2	2	C_3	0.05

Table 3.1: $c_{t_1} = 2$, $c_{t_2} = 2$ and $c_{t_3} = 3$

We say that a test t *separates* a separable pair (s_i, s_j) if t outputs different values when applied to s_i and s_j. We shall observe that if D is a decision tree for S then the set of tests associated with the nodes of D separate all separable pairs in S. A good test shall be able to split the set of objects into a partition such that none of the sets in the partition has a large number of objects to be separated. The ability of a test to achieve such a goal has been used in (14) for the minimization of the expected testing cost and it will also be used in the algorithm described below.

The DividePairs algorithm. When a test t is applied to every object in a set S it splits S into ℓ different subsets $\{S_t^1, S_t^2, ..., S_t^\ell\}$. We use S_t^* to denote the subset with the largest number of separable pairs, that is, $S_t^* = S_t^j$ such that $P(S_t^j) = max\{P(S_t^1), \ldots, P(S_t^\ell)\}$. Our algorithm, called DividePairs, chooses the test $\in T$ that minimizes $c_t/(P(S) - P(S_t^*))$. Then the objects in S are splitted according to the values of t for each object, and DividePairs is recursively called for each new set of objects. When all objects in a set are from the same class, a leaf is created. This procedure is shown in Algorithm 2.

```
1  DividePairs($S$)
2  if all objects in $S$ belong to the same class $k$ then
3      return a leaf node with label $k$
4  else
5      choose the test $t$ that minimizes $c_t/(P(S) - P(S_t^*))$
6      create a node $v$ associated with $t$
7      for $i$ = 1 to $\ell$ do
8          create a node $v_i$ as a child of $v$
9          $v_i \leftarrow$ DividePairs($S_t^i$)
10     end
11     return $v$
12 end
```

Algorithm 2: DividePairs Algorithm

As an example, consider what happens when DividePairs is executed over the instance in Table 3.1. Consider $c_{t_1} = 2$, $c_{t_2} = 2$ and $c_{t_3} = 3$. Applying test t_1 will split S in $S_{t_1}^1 = \{s_1, s_2, s_4\}$ and $S_{t_1}^2 = \{s_3, s_5\}$. In the new group $S_{t_1}^1$, we have two separable pairs: (s_1,s_4) and (s_2, s_4). On the other hand, (s_1, s_2) is not a separable pair because both are from class C_1. Thus, $P(S_{t_1}^1) = 2$. In group $S_{t_1}^2$, we know that s_3 and s_5 belong to distinct classes, constituting a separable pair. Therefore, $P(S_{t_1}^2) = 1$ and $P(S_{t_1}^*) = \max\{P(S_{t_1}^1), P(S_{t_1}^2)\} = \max\{2, 1\} = 2$. Test t_2 splits the objects in $S_{t_2}^1 = \{s_1\}$ (a group without pairs) and $S_{t_2}^2 = \{s_2, s_3, s_4, s_5\}$ (a group with 5 pairs), and test t_3 divides S in $S_{t_3}^1 = \{s_2, s_3\}$ (1 pair) and $S_{t_3}^2 = \{s_1, s_4, s_5\}$ (2 pairs). Thus, we have that $P(S_{t_1}^*) = 2$, $P(S_{t_2}^*) = 5$ and $P(S_{t_3}^*) = 2$. We have that:

$$\frac{c_{t_1}}{(P(S) - P(S_{t_1}^*))} = \frac{2}{(8-2)} = \frac{1}{3} \tag{3-2}$$

$$\frac{c_{t_2}}{(P(S) - P(S_{t_2}^*))} = \frac{2}{(8-5)} = \frac{2}{3} \tag{3-3}$$

$$\frac{c_{t_3}}{(P(S) - P(S_{t_3}^*))} = \frac{3}{(8-2)} = \frac{1}{2} \tag{3-4}$$

Thus, DividePairs chooses test t_1 as the root of the tree. Then, we have the new groups $S_{t_1}^1 = \{s_1, s_2, s_4\}$ and $S_{t_1}^2 = \{s_3, s_5\}$, presented in Tables 3.2 and 3.3. DividePairs is recursively called for these groups and we only have to choose between t_2 and t_3.

To simplify the notation, let $A = S_{t_1}^1$ and $B = S_{t_1}^2$. We have that $P(A) = 2$, $P(A_{t_2}^1) = 0$ and $P(A_{t_2}^2) = 1$. Test t_3 produces $P(A_{t_3}^1) = 0$ and $P(A_{t_3}^2) = 1$. Therefore:

Object	$\mathbf{t_2}$	$\mathbf{t_3}$	**Class**	**Probability**
s_1	1	2	C_1	0.1
s_2	2	1	C_1	0.2
s_4	2	2	C_3	0.25

Table 3.2: Objects in group $S^1_{t_1}$ ($c_{t_2} = 2$ and $c_{t_3} = 3$)

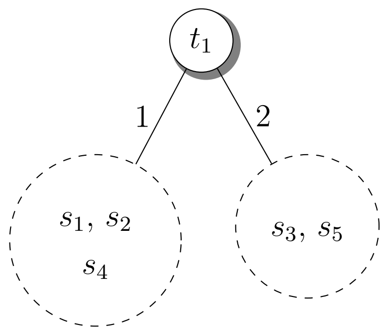
Object	$\mathbf{t_2}$	$\mathbf{t_3}$	**Class**	**Probability**
s_3	2	1	C_2	0.4
s_5	2	2	C_3	0.05

Table 3.3: Objects in group $S^2_{t_1}$ ($c_{t_2} = 2$ and $c_{t_3} = 3$)

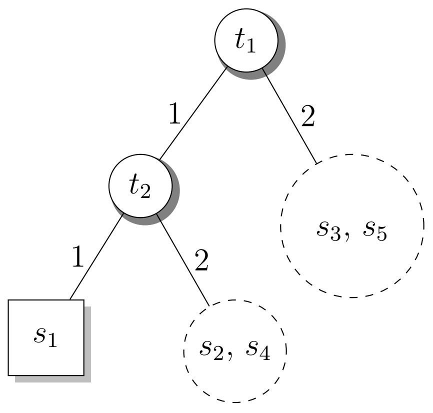
$$\frac{c_{t_2}}{(P(A) - P(A^*_{t_2}))} = \frac{2}{(2 - \max\{0, 1\})} = 2 \tag{3-5}$$

$$\frac{c_{t_3}}{(P(A) - P(A^*_{t_3}))} = \frac{3}{(2 - \max\{0, 1\})} = 3 \tag{3-6}$$

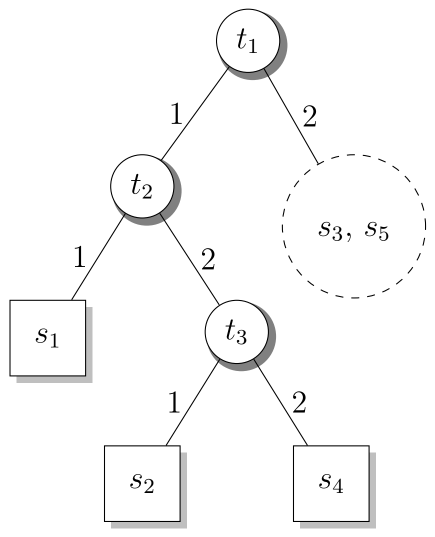
and the algorithm chooses test t_2 at this step. Figure 3.1 shows its complete execution.



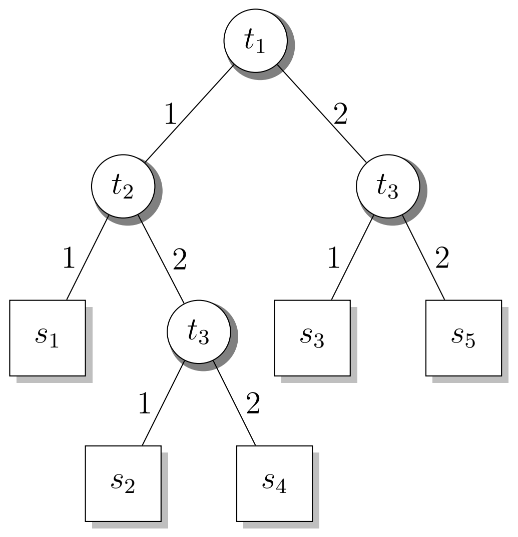
(a) Test t_1 is chosen and splits S in $S^1_{t_1} = \{s_1, s_2, s_4\}$ and $S^2_{t_1} = \{s_3, s_5\}$



(b) The algorithm is called for group $S^1_{t_1}$ and chooses test t_2



(c) The algorithm chooses the only available test, t_3, which separates s_2 and s_4



(d) At this point, note that test t_2 doesn't separate any pair in Table 3.3 (i. e., $B = B^*_{t_2}$). In this case the algorithm doesn't need to check t_2 and chooses test t_3, which separates objects s_4 and s_5.

Figure 3.1: Execution of DividePairs for objects in table 3.1.

We note that it's possible to calculate the value of $P(S) - P(S^*_t)$ for any test t in $O(k\ell)$ time. We have that:

$$P(S) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} n_{C_i} n_{C_j} = \frac{(n_{C_1} + n_{C_2} + \ldots + n_{C_k})^2 - ({n_{C_1}}^2 + {n_{C_2}}^2 + \ldots + {n_{C_k}}^2)}{2}. \tag{3-7}$$

We can use the same idea to compute the value of $P(S_t^*)$ for a given test t. A matrix with k rows and ℓ columns can store in a_{ij} the number of objects of class i which have answer j for test t. Then, we can traverse a column c as in 3-7 and compute $P(S_t^c)$. The value of $P(S_t^*)$ is obtained in $k\ell$ time.

3.1.1 Approximation Analysis.

In order to analyze the algorithm, we use $Cost_W(S)$ to denote the cost of the decision tree that DividePairs constructs for a set of objects S. Let τ be the first test selected by DividePairs. We can bound the ratio between the worst testing cost of the decision tree generated by DividePairs and the cost of the decision tree with minimum worst testing cost as

$$\frac{Cost_W(S)}{OPT_W(S)} = \frac{c_\tau + \max_{1 \le i \le \ell}\{Cost_W(S_\tau^i)\}}{OPT_W(S)} \le \frac{c_\tau}{OPT_W(S)} + \max_{1 \le i \le \ell} \left\{ \frac{Cost_W(S_\tau^i)}{OPT_W(S_\tau^i)} \right\}, \tag{3-8}$$

where the last inequality follows from Proposition 1.

The following lemma shows that $OPT_W(S)$ is at least $c_\tau P(S)/(P(S) - P(S_\tau^*))$.

Lemma 1. *$c_\tau P(S)/(P(S) - P(S_\tau^*))$ is a lower bound on the worst testing cost of the optimal tree.*

Proof: Let v be an arbitrarily chosen node in a decision tree with minimum worst testing cost, let γ be the test associated with v and let R be the set of objects associated with the leaves of the subtree rooted at v. In order to establish the result, it will be useful to show that

$$\frac{c_\tau}{P(S) - P(S_\tau^*)} \le \frac{c_\gamma}{P(S) - P(S_\gamma^*)} \le \frac{c_\gamma}{P(R) - P(R_\gamma^*)}. \tag{3-9}$$

The leftmost inequality holds due to the greedy choice. Then, it suffices to show that the rightmost inequality holds or, equivalently, $P(S) - P(S_\gamma^*) \ge P(R) - P(R_\gamma^*)$. Recall that a test t splits a set $G \subseteq S$ of objects into $\{G_t^1, G_t^2, ..., G_t^\ell\}$. Let S_γ^i and R_γ^j be such that $S_\gamma^i = S_\gamma^*$ and $R_\gamma^j = R_\gamma^*$. Let r_γ^R (resp. r_γ^S) be the the number of separable pairs separated by test γ when it is applied on set R (resp. S). Since $R \subseteq S$ we have that $r_\gamma^R \le r_\gamma^S$ and $P(R_\gamma^k) \le P(S_\gamma^k)$ for $k = 1, \ldots, \ell$. Hence,

$$P(S) - P(S^*_\gamma) = r^S_\gamma + \sum_{k \neq i} P(S^k_\gamma) \geq r^R_\gamma + \sum_{k \neq i} P(R^k_\gamma) \geq r^R_\gamma + \sum_{k \neq j} P(R^k_\gamma) \quad (3\text{-}10)$$

$$= P(R) - P(R^*_\gamma), \quad (3\text{-}11)$$

where the first inequality follows because $r^R_\gamma \leq r^S_\gamma$ and $P(R^k_\gamma) \leq P(S^k_\gamma)$, for $k = 1, \ldots, \ell$, and the second one holds because $P(R^j_\gamma) \geq P(R^i_\gamma)$. Thus, we have the following lemma:

Lemma 2. $P(S) - P(S^*_\gamma) \geq P(R) - P(R^*_\gamma)$

and can conclude that inequality (3-9) holds.

Let $v_1, v_2, \ldots, v_p$ be a root-to-leaf path on the optimal tree defined as follows: v_1 is the root of the tree, and for each $i = 1, \ldots, p-2$ the node v_{i+1} is a child of v_i with the largest number of separable pairs. The last node of the path, v_p, is any of the children of v_{p-1}, which are all leaves. Let t_i be the test associated with v_i and let S_i be the set of objects that are associated with the leaves in the subtree rooted at v_i. Note that, by definition, $S_1 = S$ and $S_{i+1} = G^*_{t_i}$, where $G = S_i$. It follows from inequality 3-9 that

$$\frac{(P(S_i) - P(S_{i+1}))c_\tau}{P(S) - P(S^*_\tau)} \leq c_{t_i} \quad (3\text{-}12)$$

for $i = 1, \ldots, p-1$. Since the cost of the path from v_1 to v_p is not larger than the worst testing cost of the optimal decision tree, we have that

$$OPT_W(S) \geq \sum_{i=1}^{p-1} c_{t_i} \geq \frac{c_\tau}{P(S) - P(S^*_\tau)} \sum_{i=1}^{p-1} (P(S_i) - P(S_{i+1})) = \frac{c_\tau P(S)}{P(S) - P(S^*_\tau)},$$

where the second inequality follows from (3-12) and the last identity holds because $S_1 = S$ and $P(S_p) = 0$. □

As a result, the ratio given by equation (3-8) is:

$$\frac{Cost_W(S)}{OPT_W(S)} \leq \frac{P(S) - P(S^*_\tau)}{P(S)} + \max_{1 \leq i \leq \ell} \left\{ \frac{Cost_W(S^i_\tau)}{OPT_W(S^i_\tau)} \right\}. \quad (3\text{-}13)$$

Note that:

$$\frac{P(S) - P(S^*_\tau)}{P(S)} = \sum_{i=1}^{P(S)-P(S^*_\tau)} \left(\frac{1}{P(S)} \right) \leq \sum_{i=1}^{P(S)-P(S^*_\tau)} \left(\frac{1}{P(S^*_\tau) + i} \right). \quad (3\text{-}14)$$

By induction in the number of separable pairs, we assume that for each $G \subset S$, $Cost_W(G)/OPT_W(G) \leq H(P(G))$, where $H(n) = \sum_{i=1}^{n} 1/i$. Since $P(S^i_\tau) \leq P(S^*_\tau)$ it follows that $H(P(S^i_\tau)) \leq H(P(S^*_\tau))$. From (3-13) and (3-14) we have that

$$\frac{Cost_W(S)}{OPT_W(S)} \leq \sum_{i=1}^{P(S)-P(S^*_\tau)} \left(\frac{1}{P(S^*_\tau)+i} \right) + H(P(S^*_\tau)) = H(P(S)) \leq 2\ln(n).$$

Thus, we have the following theorem.

Theorem 1. *The* DividePairs *algorithm achieves an* $O(\ln(n))$*-approximation for minimizing the worst testing cost of the DFEP.*

3.2 A Bicriteria Approximation

In this section, we present an algorithm that provides a simultaneous approximation for the minimization of expected testing cost and worst testing cost. Before presenting the algorithm, let us consider an instance of the DFEP that motivates the search for this simultaneous optimization.

The instance is indeed an instance of the problem of constructing a binary tree corresponding to an optimal prefix code, described in Chapter 1. Recall that in this problem all tests have unitary costs and the set of tests is in one to one correspondence with the set of all binary strings of length n. The test corresponding to a binary string $\mathbf{b}$ outputs 0 (1) for object s_i if and only if the i^{th} bit of $\mathbf{b}$ is 0 (1).

The probability distribution is given by $p_i = 2^{-i}$ for each object s_i, for $i = 1, \ldots, n-1$ and $p_n = 2^{-(n-1)}$. Table 3.4 show an example for $n = 4$, and Figure 3.2 shows an optimal decision tree for this example.

Object	$\mathbf{t_1}$	$\mathbf{t_2}$	$\mathbf{t_3}$	$\cdots$	$\mathbf{t_{14}}$	$\mathbf{t_{15}}$	$\mathbf{t_{16}}$	**Class**	**Probability**
s_1	0	0	0		1	1	1	C_1	0.5
s_2	0	0	0	$\cdots$	1	1	1	C_2	0.25
s_3	0	0	1		0	1	1	C_3	0.125
s_4	0	1	0		1	0	1	C_4	0.125

Table 3.4: Example for $n = 4$.

Let D^*_E and D^*_W be, respectively, the decision trees with minimum expected cost and minimum worst testing cost for the instance. By following the execution of Huffman's algorithm, it is not difficult to verify that $Cost_E(D^*_E) \leq 3$ and $Cost_W(D^*_E) = n - 1$. In addition, we have that

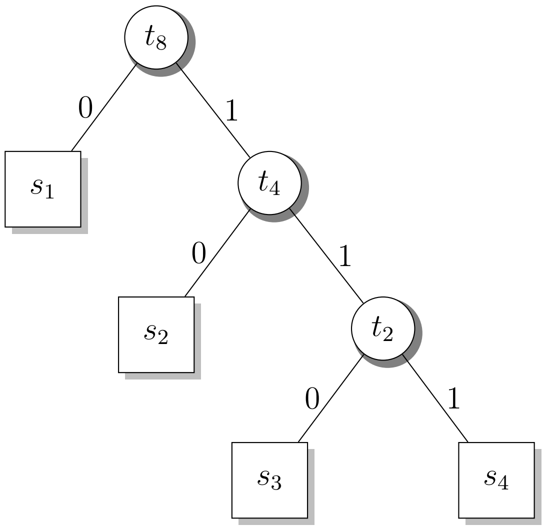


Figure 3.2: An optimal decision tree for objects of Table 3.4, produced by Huffman's algorithm.

$Cost_E(D^*_W) = Cost_W(D^*_W) = \log n$. This example shows that the minimization of the expected testing cost may produce a decision tree with high worst testing cost as well as the minimization of the worst testing cost may produce a decision tree with high expected testing cost. Therefore, it makes sense to look for a trade-off between minimizing the expected testing cost and the worst testing cost.

Given a positive number ρ, two decision trees D_E and D_W for the instance $(S, C, T, \mathbf{p}, \mathbf{c})$ of the DFEP, the former with expected testing cost E and the latter with worst testing cost W, we devise a polynomial time procedure to construct a new decision tree D, from D_E and D_W, with expected cost at most $(1 + 1/\rho)E$ and worst testing cost at most $(1 + \rho)W$. The procedure is very simple:

```
1 CombineTrees($D_E$,$D_W$,$\rho$)
2 $D = D_E$
3 Define a node $v$ from $D$ as replaceable if the cost of the path from
  the root of $D$ to $v$ (including $v$) is at least $\rho W$ and the cost of the
  path from the root of $D$ to the parent of $v$ is smaller than $\rho W$. At
  this step we traverse $D$ to find the set $R$ of the replaceable nodes.
4 for every node $v \in R$ do
5     Let $S(v)$ be the set of objects associated with leaves located at
      the subtree rooted at $v$ in $D$. In addition, let $D_W^{S(v)}$ be a decision
      tree for $S(v)$ obtained by disassociating every object in $S - S(v)$
      from $D_W$.
6     Replace the subtree of $D$ rooted at $v$ with the decision tree $D_W^{S(v)}$.
7 end
8 Return $D$
```

Algorithm 3: The CombineTrees procedure

Figures from 3.3 to 3.6 show the execution of CombineTrees.

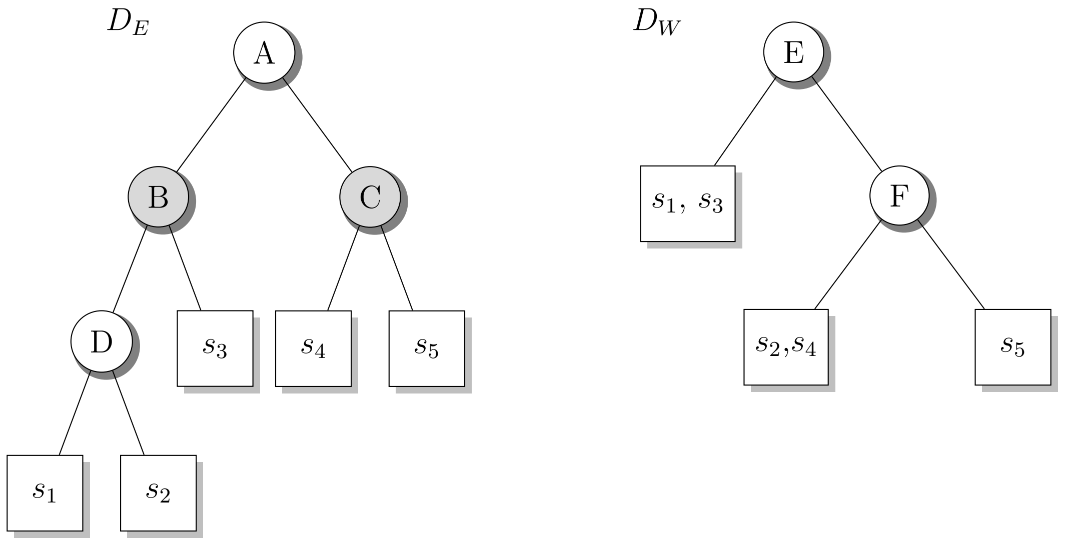


Figure 3.3: First, CombineTrees receives two trees: D_E, (with replaceable nodes marked in gray), and D_W. Suppose that s_1 and s_3 belong to class C_1, s_2 and s_4 to class C_2 and s_5 to class C_3. The replaceable nodes are B and C, which means that the cost associated with node A is less than ρW, but the costs associated with the paths from A to B and from A to C are greater or equal than ρW.

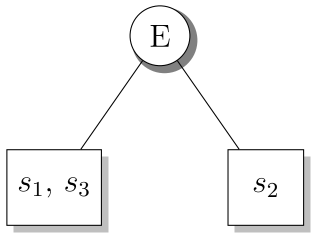


Figure 3.4: We have to replace the subtree rooted at B with the tree D_W without objects s_4 and s_5, which is shown above. Note that if we discard s_4 and s_5 from D_W, the only object below F is s_2. Therefore, we can replace this node with a leaf node with object s_2.

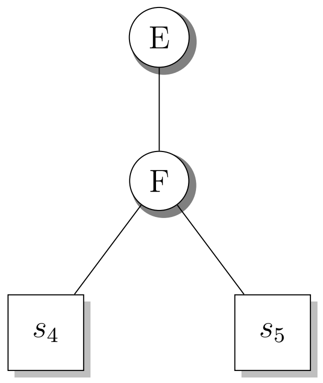


Figure 3.5: The next step is to replace the subtree rooted at C with the tree D_W associated with s_4 and s_5. This step involves removing the leaf node associated with s_1 and s_3. Note that since s_4 and s_5 belong to different classes, we can't replace E or F in D_W with a leaf node at first.

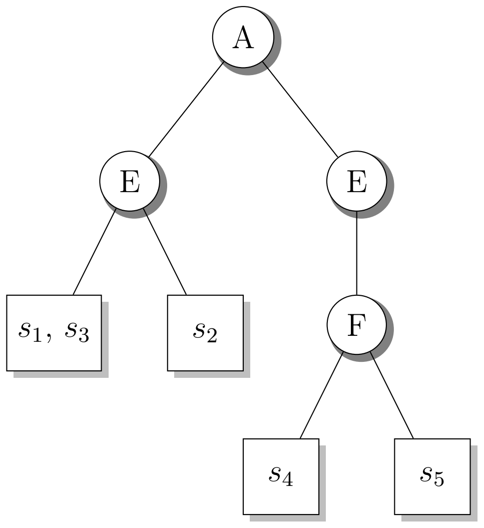


Figure 3.6: The final tree.

Theorem 2. *The decision tree D has expected testing cost at most* $(1+1/\rho)E$ *and worst testing cost at most* $(1+\rho)W$.

Proof. First we argue that the worst testing cost of D is at most $(1+\rho)W$. Let s be an object in S. If s is not a descendant of a replaceable node in D_E then the cost of the path from the root of D_E to s is at most ρW. Since this path remains the same in D, we have that the cost to reach s in D is at most ρW. On the other hand, if s is a descendant of a replaceable node v in D_E, then the cost to reach s in D is at most $(1+\rho)W$ because the cost of the path from the root of D to the parent of v is at most ρW and the cost to reach s from the root of the tree $D_W^{S(v)}$ is at most W.

Now, we prove that the expected testing cost of D is at most $(1+1/\rho)E$. For that it is enough to show that for every object $s \in S$, the cost to reach s in D is at most $(1+1/\rho)$ times the cost of reaching s in D_E. We split the analysis into two cases:

Case 1. s is not a descendant of a replaceable node in D_E. In this case, the cost to reach s in D_E is equal to the cost of reaching s in D.

Case 2. s is a descendant of a replaceable node v in D_E. Let K be the cost of the path from the root of D_E to v. Then, the cost to reach s in D_E is at least K. In addition, since v is replaceable we have that $K \geq \rho W$. On the other hand, the cost to reach s in D is at most $\rho W + W$. Since $K \geq \rho W$ we have that the cost to reach s in D is at most $(1+1/\rho)$ times the cost of reaching s in D_E. □

4 Experimental Results

In this chapter, the performance of algorithms proposed in Chapters 3.1 and 3.2 is analysed through a set of experiments.

The experiments were done in a computer with the following configuration:

- Ubuntu Release 12.04 64-bit
- Kernel Linux 3.2.0-23-generic
- Memory: 7.7 GB
- Processor: Intel®Core^{TM}i7 CPU 870 @ 2.93GHz $\times$ 8

The following algorithms were implemented:

- **DIVIDEPAIRS:** described in Chapter 3.1
- **EC^{2}**: an algorithm that approximates the expected testing cost proposed by (14)
- **OPT$_{WC}$**: an exponential time algorithm that produces the optimal decision tree with respect to the worst testing cost for a given instance
- **OPT$_{EC}$**: an exponential time algorithm that produces the optimal decision tree with respect to the expected testing cost for a given instance
- **COMBINETREES**: procedure described in Chapter 3.2
- **COMBINETREESDP**: the COMBINETREES procedure with some modifications

Algorithms EC^2, OPT_{WC} and OPT_{EC} were used in order to analyse the results obtained by algorithm DIVIDEPAIRS. The algorithm by Golovin (EC^2) was also used to generate trees for COMBINETREES and COMBINETREESDP procedures. This dissertation will focus in algorithm DIVIDEPAIRS and in procedure COMBINETREES, which were our contributions, as well as in procedure COMBINETREESDP, a modification of COMBINETREES which will be described in next sections.

```
3 8 2 2
0 0 0 0 1
0 0 1 0 1
0 1 0 1 3
0 1 1 1 2
1 0 0 0 5
1 0 1 0 1
1 1 0 0 2
1 1 1 1 3
```

Figure 4.1: Example of input data

4.1 Input Data

We created an experimental environment where the user defines the maximum number of classes, the maximum number of outcomes of a test, the number of objects and the number of tests of the input data, as well as whether the probabilities are uniform or not. A source file written in C programming language generates the instance based on this information. Another C source file reads the data generated and run the algorithms.

Figure 4.1 shows the structure of an instance generated.

The first line of each instance consists of four numbers: the number of tests, the number of objects, the maximum number of classes and the maximum number of outcomes for a test, respectively. If the number of classes is equal to X, then the classes can vary from 0 to $X - 1$, and the same occurs with the number of outcomes for a test. In Figure 4.1, there are 3 tests, 8 objects, 2 classes and 2 possible outcomes for each test. The other lines represent the objects: the first numbers in a line are the outcomes for the tests, and the last two numbers represent the class and the frequency associated with the object. In the figure above, there are 5 objects belonging to class 0 and 3 objects belonging to class 1.

The answers for each test and the classes of the objects are generated in the same way: a random number between 0 and the number of outcomes is chosen, as well as a random number between 0 and the number of classes, by calling $(rand() \mod number\_of\_outcomes)$ and $(rand() \mod number\_of\_classes)$. The frequency is generated by calling $(rand() \mod 10007 + 1)$, (10007 is a large prime number that gives an upper bound for the frequency), generating an uniform probability distribution.

The file is processed to eliminate objects from different classes that have the same answer for all tests and to normalize the frequencies (a probability distribution is generated according to the frequencies). Table 4.1 shows the number of objects, the number of tests, the number of classes and the number of outcomes for a test for each one of the postprocessed instances used in the experiments. The last column of the table is in the form $aCbP$, where $a, b \in \{u, n\}$ indicates whether the costs and probabilities are uniform or not (u for *uniform* and n for *non-uniform*).

Instance	**Objects**	**Tests**	**Classes**	**Outcomes**	**Type**
Instance1	100	12	3	2	nCnP
Instance2	200	10	3	3	nCnP
Instance3	299	10	3	3	nCnP
Instance4	5000	100	5	3	nCnP
Instance5	20000	50	5	2	nCnP
Instance6	20000	40	8	2	nCnP
Instance7	20000	90	30	2	nCnP
Instance8	20000	40	5	2	uCuP
Instance9	39086	48	7	2	uCnP
Instance10	39086	48	7	2	nCnP

Table 4.1: Instances used in the experiments

Instance INSTANCE10 is the same instance as INSTANCE09, except by the costs of the tests. These instances were obtained from a High Frequency Trading application. The estimation of the probabilities was obtained by normalizing the frequency of appearance of each scenario (object) during the execution of the trading strategy in a simulation on real market data.

All instances of types $nCnP$ and $nCuP$ have their testing costs as random numbers in the range $[1, 10]$. Instances with uniform costs have all costs equal to 1.

4.2 DividePairs Algorithm

The first algorithm implemented was DIVIDEPAIRS.

In order to choose which test to perform next, first an array with positions corresponding to classes was created. The position i of the array stores the number of objects of class i. Then, in a single pass, we can compute the total number of pairs of objects. Recall that $P(S_t^*) = max\{P(S_t^1), \ldots, P(S_t^\ell)\}$, where S_t^i is the subset of S that outputs i for test t. If there are k classes, a matrix with k rows and ℓ columns can be used to compute the value of $P(S_t^*)$ in $O(n)$ time (since $n >> k\ell$ for each instance in Table 4.1), for a given test t.

Table 4.2 shows some results for the first 3 instances. The first column shows the ratio between the worst cost generated by DividePairs and the optimal worst cost (computed by OPT_{WC}) and the second column shows the ratio between the expected testing cost and the optimal expected testing cost (computed by OPT_{EC}). Table 4.3 shows the same ratios for algorithm EC^2.

Instance	**Worst Cost**	**Expected Cost**
Instance1	1.2	1.04
Instance2	1.24	1.06
Instance3	1.15	1.05

Table 4.2: DividePairs in contrast with the optimal algorithms.

Instance	**Worst Cost**	**Expected Cost**
Instance1	1.2	1.03
Instance2	1.37	1.05
Instance3	1.26	1.04

Table 4.3: EC^2 in contrast with the optimal algorithms.

Table 4.4 shows the costs of the trees generated by the execution of DividePairs and of EC^2. Recall that DividePairs doesn't take into account the probability distribution of the objects to build the tree. As expected, in general DividePairs achieved a smaller worst cost than EC^2. The algorithm by Golovin, on the other hand, attained a better expected testing cost — which was also expected. For all instances, there were no significant differences between the execution times of the two algorithms. Although the worst costs achieved by EC^2 were greater or equal than the worst costs achieved by DividePairs for these instances, this is not always the case. During the experiments we found some instances for which the worst cost achieved by EC^2 was smaller than the worst cost achieved by DividePairs.

4.3 The CombineTrees and the CombineTreesDP Procedures

The procedure CombineTrees, described in Chapter 3.2, was implemented with the first parameter being a tree D_E produced by EC^2, the second parameter being a tree D_W produced by DividePairs and different values of ρ.

Note that the final tree produced by CombineTrees can have paths with repeated tests, since a test t can be associated with a node in D_W and can also be associated with a node that is an ancestor of a replaceable node r

Instance	$\mathbf{WC_{DP}}$	$\mathbf{EC_{DP}}$	$\mathbf{WC_{EC^2}}$	$\mathbf{EC_{EC^2}}$	Time DP	Time $\mathbf{EC^2}$
Instance1	18	11.62	18	11.55	0	0
Instance2	36	22.97	40	22.83	0	0
Instance3	22	14.01	24	13.90	0	0
Instance4	12	8.20	12	8.08	0.06	0.05
Instance5	36	28.23	39	27.82	0.2	0.19
Instance6	42	31.91	51	31.15	0.18	0.18
Instance7	29	22.71	33	22.40	0.65	0.49
Instance8	15	13.52	15	13.25	0.15	0.14
Instance9	18	11.41	19	9.51	0.31	0.3
Instance10	103	41.58	104	38.22	0.34	0.34

Table 4.4: Execution of DividePairs and EC^2. Columns WC_{DP} and EC_{DP} present the worst and the expected testing costs achieved by DividePairs, and columns WC_{EC^2} and EC_{EC^2} present the worst and the expected testing costs achieved by EC^2 algorithm. Finally, columns *Time DP* and *Time* EC^2 show the running time (in seconds) of DividePairs and EC^2.

in D_E. Therefore, all nodes with only one child were removed from the final tree (the running time of CombineTrees includes the time spent in these modifications). In almost all executions of the procedure, for small values of ρ both worst and expected costs of the modified tree were equal to the costs of D_W (because the root was replaceable), and for large values of ρ the costs of the modified tree were similar to the costs of D_E.

The second column of Table 4.5 presents the result of the execution of CombineTrees over instance 6 for different values of ρ. It shows the first of the two interesting situations that were observed in the experiments running CombineTrees. Note that as ρ increases the worst cost also increases, until it reaches the worst cost of D_E for large values of ρ. However, the values of the expected testing cost of the modified tree for intermediate values of ρ were greater than the values of the expected testing cost of D_W and of D_E, which is not desirable.

The second situation, that was observed in other instances, was that for intermediate values of ρ this problem happened with the worst cost too (i. e., the worst cost of the modified tree was greater than the worst costs of D_W and of D_E). This situation is presented in the second column of Table 4.6. Actually, except for instances 2, 3, 7 and 10, we observed that if the costs of the trees produced by DividePairs and EC^2 are given by, respectively, (WC_{DP}, EC_{DP}) and (WC_{EC^2}, EC_{EC^2}), where (a, b) indicates that a is the worst cost and b is the expected cost, the worst and expected costs (WC_{CT}, EC_{CT}) obtained by CombineTrees satisfied, for all values of ρ,

– $WC_{CT} \geq WC_{DP}$ and $EC_{CT} \geq EC_{DP}$

ρ	**CombineTrees**	**CombineTreesDP**
0.05	(42, 31.91)	(42, 31.91)
0.1	(42, 31.91)	(42, 31.91)
0.15	(42, 31.91)	(42, 31.91)
0.2	(42, 31.91)	(42, 31.91)
0.25	(42, 31.91)	(42, 31.91)
0.3	(42, 31.91)	(42, 31.91)
0.35	(42, 31.91)	(42, 31.91)
0.4	(45, 31.92)	(42, 31.91)
0.45	(45, 31.92)	(42, 31.90)
0.5	(46, 31.98)	(42, 31.89)
0.55	(46, 31.99)	(43, 31.80)
0.6	(47, 31.99)	(46, 31.59)
0.65	(49, 31.92)	(46, 31.52)
0.7	(49, 31.92)	(46, 31.52)
0.75	(51, 31.40)	(47, 31.23)
0.8	(51, 31.38)	(47, 31.23)
0.85	(51, 31.21)	(47, 31.17)
0.9	(51, 31.17)	(47, 31.15)
0.95	(51, 31.17)	(47, 31.15)
1	(51, 31.15)	(51, 31.15)
∞	(42, 31.91)	(51, 31.15)

Table 4.5: Results of CombineTreesDP procedure and CombineTrees procedure for Instance6. The row with $\rho = \infty$, shows the costs obtained by DividePairs (second column) and EC^2. (third column). The costs are in the form (a, b), where a is the worst cost and b is the expected cost

or

– $WC_{CT} \geq WC_{EC^2}$ and $EC_{CT} \geq EC_{EC^2}$.

The results obtained by running the CombineTrees procedure suggest that, despite its theoretical guarantees, it can give a poor result in practice. A possible explanation for this situation is that the tree D_W is a tree that approximates the worst testing cost when we consider the initial set S. Recall that $S(v)$ is the set of objects associated with leaves located at the subtree rooted at v in D_E, and $D_W^{S(v)}$ is the tree obtained by disassociating every object in $S - S(v)$ from D_W. For a subset $S(v) \in S$, $D_W^{S(v)}$ can be very unbalanced, even with the modification done to remove repeated tests.

Given the results obtained by CombineTrees, the following modification was implemented: for each replaceable v, instead of replacing the subtree rooted at v with $D_W^{S(v)}$ in CombineTrees, we replace this subtree with the tree produced by DividePairs for the set $S(v)$. We call this procedure CombineTreesDP. Note that we can maintain our theoretical guarantees by running

ρ	**CombineTrees**	**CombineTreesDP**
0.05	(36, 28.23)	(36, 28.23)
0.1	(36, 28.23)	(36, 28.23)
0.15	(36, 28.24)	(36, 28.23)
0.2	(36, 28.25)	(36, 28.24)
0.25	(37, 28.26)	(36, 28.23)
0.3	(37, 28.28)	(36, 28.23)
0.35	(37, 28.30)	(36, 28.23)
0.4	(38, 28.31)	(38, 28.23)
0.45	(38, 28.32)	(38, 28.23)
0.5	(38, 28.32)	(38, 28.23)
0.55	(38, 28.33)	(36, 28.23)
0.6	(38, 28.34)	(36, 28.22)
0.65	(38, 28.33)	(38, 28.15)
0.7	(38, 28.27)	(38, 27.99)
0.75	(41, 28.16)	(38, 27.95)
0.8	(41, 27.93)	(38, 27.85)
0.85	(40, 27.88)	(38, 27.83)
0.9	(39, 27.83)	(38, 27.82)
0.95	(39, 27.82)	(38, 27.82)
1	(39, 27.82)	(39, 27.82)
∞	(36, 28.23)	(39, 27.82)

Table 4.6: Results of CombineTrees procedure and CombineTreesDP procedure for Instance5. The row with $\rho = \infty$, shows the costs obtained by DividePairs (second column) and EC^2. (third column). The costs are in the form (a, b), where a is the worst cost and b is the expected cost

the two procedures and choosing the tree with the lower worst testing cost. The third columns of Tables 4.5 and 4.6 show that the two situations observed above for instances 5 and 6 disappeared with this modification Note that the costs for low values of ρ were similar to the cost of D_W, and the costs for high values of ρ were similar to the cost of D_E. But for intermediate values the costs (both the worst and the expected cost) of the tree produced by CombineTreesDP were values between the costs of D_W and D_E. For instance 6 we can note this when $0.45 \leq \rho \leq 0.95$ in Table 4.5, and for instance 5 when $0.4 \leq \rho \leq 0.95$ in Table 4.6.

The performance of the two procedures is presented in tables from A.1 to A.10, in Appendix A, and Table 4.7 shows their performance for $\rho = 0.5$. The lower of the two costs obtained is presented in bold font and the running time of the procedures is presented in seconds (we use CT to denote CombineTrees and $CTDP$ to denote CombineTreesDP).

As in CombineTrees procedure, when $\rho = 1$ the costs of the tree were the same costs of D_E. The difference was for intermediate values: as ρ increased,

Instance	**CombineTrees**	**CombineTreesDP**	**Time CT**	**Time CTDP**
Instance1	(18, **11.59**)	(18, 11.60)	0	0
Instance2	(39, 23.20)	(**36**, **22.85**)	0	0
Instance3	(22, **13.92**)	(22, 13.94)	0	0
Instance4	(13, 8.35)	(**12**, **8.18**)	0.18	0.04
Instance5	(38, 28.32)	(38, **28.23**)	2.97	0.14
Instance6	(46, 31.98)	(**42**, **31.89**)	16.59	0.12
Instance7	(31, 22.87)	(**29**, **22.69**)	7.58	0.52
Instance8	(19, 14.39)	(**15**, **13.52**)	0.44	0.11
Instance9	(21, 10.46)	(**19**, **9.81**)	0.43	0.14
Instance10	(108, 38.52)	(108, **38.42**)	3.6	0.14

Table 4.7: Execution of CombineTrees and CombineTreesDP. The costs are in the form (a, b), where a is the worst cost and b is the expected cost

the worst cost of the tree increased, but the expected cost decreased. As a result, both the two costs were values between the costs of D_W and D_E. In most cases, CombineTreesDP produced a tree with both the worst and the expected costs better than the costs obtained by CombineTrees.

As already expected, the running time of the CombinetreesDP procedure was smaller than the running time of the Combinetrees procedure, since it's faster to run DividePairs than to rebuild the whole tree D_W for a given subset $S(v)$ of S. Consider, for example, two objects of different classes that are separated in the last node x of a path in D_W. Then, rebuild the tree for these objects implies that at each node of the path in D_W, from the root until x, we have to remove a considerable amount of subtrees. The final tree is a tree with only a single path from the root to x — which is removed because each node has only one child.

Finally, a last experiment was made with Instance10. CombineTreesDP was executed for this instance 100 times with $\rho = 0.5$, and in each time the costs were computed as random numbers in the range $[1, 10]$. The results are presented in Figure 4.2. The graph on the left shows the worst testing cost achieved by each algorithm in each execution. As somehow expected, for most of the cases (80 %) DividePairs outperforms EC^2 in terms of minimizing the worst testing cost. The graph on the right side shows the expected testing cost for the 100 instances. In all cases, EC^2 outperforms DividePairs in terms of minimizing the expected testing cost. The worst testing cost of DividePairs is in average 4% smaller than the worst testing cost of EC^2 while the expected testing cost achieved for EC^2 is in average 10% smaller than that achieved by DividePairs.

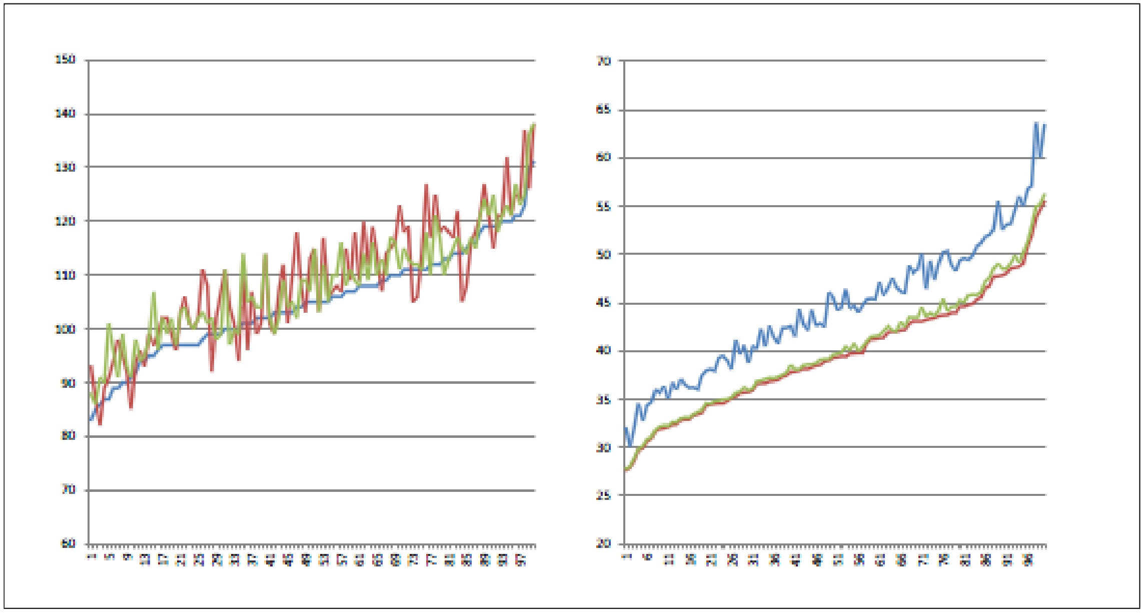


Figure 4.2: The graph on the lefthand side shows the worst testing cost achieved by DividePairs (blue), CombineTrees (green) and EC^2 (red) for 100 instances. The graph on the righthand side shows the expected testing cost achieved by the same algorithms over the same instances.

5
Conclusions and Future Works

In this thesis, we introduced and discussed the DFEP and presented two main contributions.

Our first contribution was the DividePairs, an approximation algorithm for the worst case of the DFEP. We showed, through a simple analysis, how it attains an $O(\log n)$-approximation — the best possible approximation assuming that $\mathcal{P} \neq \mathcal{NP}$. Note that we studied here the most general case of the problem, with non-uniform costs and multiway tests. To the best of our knowledge, this is the first algorithm to approximate the worst case when the number of classes is not equal to the number of objects.

Finally, our second contribution was the CombineTrees procedure, which receives as input a tree D_E with expected cost at most E, a tree D_W with worst cost at most W and a parameter $\rho > 0$, and returns a decision tree with worst cost at most $(1+\rho)W$ and expected cost at most $(1+1/\rho)E$. This implies that there exists a decision tree that provides simultaneously an $O(\log n)$-approximation for both the worst testing cost and the expected cost versions of the DFEP, since our algorithm for the worst case can be used with the algorithms presented in (1) and (18). These two algorithms produce decision trees with an $O(\log n)$-approximation w. r. t. the expected testing cost (the former for the $DFEP$, and the latter for the identification problem). This is the best possible approximation for the two goals (unless $\mathcal{P} = \mathcal{NP}$).

As a future work, we can further investigate the proposed algorithms, since the experiments reported in this thesis can be expanded. We can see how these algorithms behave with other probability distributions, for example, among other modifications.

A
Performance of CombineTrees and CombineTreesDP

Tables from A.1 to A.10 show the results obtained by the algorithms. Again, the lower of the two costs obtained is presented in bold font and the running time of the procedures is presented in seconds. The row with $\rho = \infty$, shows the costs obtained by DIVIDEPAIRS and EC^2. The second and fourth columns of this row present the performance of DIVIDEPAIRS, and the third and last columns present the performance of EC^2.

ρ	CombineTrees	CombineTreesDP	Time CT	Time CTDP
0.05	(18, 11.62)	(18, 11.62)	0	0
0.1	(18, 11.62)	(18, 11.62)	0	0
0.15	(18, 11.62)	(18, 11.62)	0	0
0.2	(18, 11.62)	(18, 11.62)	0	0
0.25	(18, 11.62)	(18, 11.62)	0	0
0.3	(18, 11.62)	(18, 11.62)	0	0
0.35	(18, 11.64)	(18, 11.64)	0	0
0.4	(18, **11.59**)	(18, 11.60)	0	0
0.45	(18, **11.59**)	(18, 11.60)	0	0
0.5	(18, **11.59**)	(18, 11.60)	0	0
0.55	(18, **11.60**)	(18, 11.62)	0	0
0.6	(18, **11.57**)	(18, 11.59)	0	0
0.65	(18, **11.57**)	(18, 11.59)	0	0
0.7	(18, **11.57**)	(18, 11.59)	0	0
0.75	(18, **11.55**)	(18, 11.57)	0	0
0.8	(18, **11.55**)	(18, 11.57)	0	0
0.85	(18, **11.55**)	(18, 11.57)	0	0
0.9	(18, 11.55)	(18, 11.55)	0	0
0.95	(18, 11.55)	(18, 11.55)	0	0
1	(18, 11.55)	(18, 11.55)	0	0
∞	(18, 11.62)	(18, **11.55**)	0	0

Table A.1: Results of COMBINETREESDP procedure and COMBINETREES procedure for INSTANCE1.

ρ	CombineTrees	CombineTreesDP	Time CT	Time CTDP
0.05	(36, 22.97)	(36, 22.97)	0	0
0.1	(36, 22.97)	(36, 22.97)	0	0
0.15	(36, 22.97)	(36, 22.97)	0	0
0.2	(36, 22.97)	(36, 22.97)	0	0
0.25	(36, 22.97)	(36, 22.97)	0	0
0.3	(36, 22.97)	(36, 22.97)	0	0
0.35	(36, 22.97)	(36, 22.97)	0	0
0.4	(36, 22.97)	(36, 22.97)	0	0
0.45	(36, 22.97)	(36, 22.97)	0	0
0.5	(39, 23.20)	(**36**, **22.85**)	0	0
0.55	(39, 23.20)	(**36**, **22.85**)	0	0
0.6	(39, 23.20)	(**36**, **22.85**)	0	0
0.65	(42, 23.13)	(**33**, **22.87**)	0	0
0.7	(42, 23.08)	(**33**, **22.87**)	0	0
0.75	(42, 22.82)	(**33**, **22.73**)	0	0
0.8	(42, 22.82)	(**33**, **22.73**)	0	0
0.85	(40, 22.85)	(40, **22.83**)	0.01	0
0.9	(40, 22.85)	(40, **22.83**)	0	0
0.95	(40, 22.83)	(40, 22.83)	0	0
1	(40, 22.83)	(40, 22.83)	0	0
∞	(**36**, 22.97)	(40, **22.83**)	0	0

Table A.2: Results of CombineTreesDP procedure and CombineTrees procedure for Instance2

ρ	CombineTrees	CombineTreesDP	Time CT	Time CTDP
0.05	(22, 14.01)	(22, 14.01)	0	0
0.1	(22, 14.01)	(22, 14.01)	0	0
0.15	(22, 14.01)	(22, 14.01)	0	0
0.2	(22, 14.01)	(22, 14.01)	0	0
0.25	(22, 14.01)	(22, 14.01)	0	0
0.3	(22, 14.01)	(22, 14.01)	0	0
0.35	(22, 14.01)	(22, 14.01)	0.01	0
0.4	(22, 14.01)	(22, 14.01)	0	0
0.45	(22, **13.92**)	(22, 13.94)	0	0
0.5	(22, **13.92**)	(22, 13.94)	0	0
0.55	(22, **13.95**)	(22, 13.96)	0.01	0
0.6	(22, **13.92**)	(22, 13.93)	0	0.01
0.65	(22, **13.92**)	(22, 13.92)	0	0
0.7	(22, 13.92)	(22, 13.92)	0	0
0.75	(22, 13.92)	(22, 13.92)	0	0
0.8	(26, 13.91)	(**24**, **13.90**)	0	0
0.85	(26, 13.91)	(**24**, **13.90**)	0	0
0.9	(26, 13.91)	(**24**, **13.90**)	0	0
0.95	(26, 13.91)	(**24**, **13.90**)	0	0
1	(26, 13.91)	(**24**, **13.90**)	0	0
∞	(**22**, 14.01)	(24, **13.90**)	0	0

Table A.3: Results of CombineTreesDP procedure and CombineTrees procedure for Instance3

ρ	CombineTrees	CombineTreesDP	Time CT	Time CTDP
0.05	(12, 8.20)	(12, 8.20)	0.01	0.07
0.1	(13, 8.25)	(**12**, **8.19**)	0.02	0.06
0.15	(13, 8.25)	(**12**, **8.19**)	0.03	0.06
0.2	(13, 8.28)	(**12**, **8.20**)	0.03	0.05
0.25	(13, 8.28)	(**12**, **8.20**)	0.02	0.06
0.3	(13, 8.31)	(**12**, **8.19**)	0.04	0.05
0.35	(13, 8.32)	(**12**, **8.18**)	0.09	0.05
0.4	(13, 8.32)	(**12**, **8.18**)	0.08	0.04
0.45	(13, 8.35)	(**12**, **8.18**)	0.18	0.04
0.5	(13, 8.35)	(**12**, **8.18**)	0.18	0.04
0.55	(13, 8.34)	(**12**, **8.15**)	0.46	0.04
0.6	(12, 8.18)	(12, **8.11**)	0.84	0.03
0.65	(12, 8.18)	(12, **8.11**)	0.84	0.03
0.7	(12, 8.12)	(12, **8.10**)	0.42	0.02
0.75	(12, 8.12)	(12, **8.10**)	0.44	0.02
0.8	(12, 8.09)	(12, **8.08**)	0.2	0.02
0.85	(12, 8.08)	(12, 8.08)	0.04	0.01
0.9	(12, 8.08)	(12, 8.08)	0.04	0.01
0.95	(12, 8.08)	(12, 8.08)	0.03	0.01
1	(12, 8.08)	(12, 8.08)	0.02	0.02
∞	(12, 8.20)	(12, **8.08**)	0.06	0.05

Table A.4: Results of CombineTreesDP procedure and CombineTrees procedure for Instance4

ρ	CombineTrees	CombineTreesDP	Time CT	Time CTDP
0.05	(36, 28.23)	(36, 28.23)	0.08	0.24
0.1	(36, 28.23)	(36, 28.23)	0.1	0.21
0.15	(36, 28.24)	(36, **28.23**)	0.12	0.2
0.2	(36, 28.25)	(36, **28.24**)	0.17	0.18
0.25	(37, 28.26)	(**36**, **28.23**)	0.25	0.16
0.3	(37, 28.28)	(**36**, **28.23**)	0.41	0.16
0.35	(37, 28.30)	(**36**, **28.23**)	0.79	0.16
0.4	(38, 28.31)	(38, **28.23**)	1.66	0.14
0.45	(38, 28.32)	(38, **28.23**)	2.93	0.14
0.5	(38, 28.32)	(38, **28.23**)	2.97	0.14
0.55	(38, 28.33)	(**36**, **28.23**)	5.31	0.13
0.6	(38, 28.34)	(**36**, **28.22**)	9.96	0.13
0.65	(38, 28.33)	(38, **28.15**)	18.84	0.12
0.7	(38, 28.27)	(38, **27.99**)	23.39	0.11
0.75	(41, 28.16)	(**38**, **27.95**)	23.89	0.1
0.8	(41, 27.93)	(**38**, **27.85**)	12.83	0.09
0.85	(40, 27.88)	(**38**, **27.83**)	10.85	0.09
0.9	(39, 27.83)	(**38**, **27.82**)	2.21	0.08
0.95	(39, 27.82)	(**38**, 27.82)	0.55	0.08
1	(39, 27.82)	(39, 27.82)	0.24	0.08
∞	(**36**, 28.23)	(39, **27.82**)	0.2	0.19

Table A.5: Results of COMBINETREESDP procedure and COMBINETREES procedure for INSTANCE5

ρ	CombineTrees	CombineTreesDP	Time CT	Time CTDP
0.05	(42, 31.91)	(42, 31.91)	0.1	0.2
0.1	(42, 31.91)	(42, 31.91)	0.15	0.18
0.15	(42, 31.91)	(42, 31.91)	0.34	0.15
0.2	(42, 31.91)	(42, 31.91)	0.6	0.16
0.25	(42, 31.91)	(42, 31.91)	1.22	0.14
0.3	(42, 31.91)	(42, 31.91)	2.63	0.14
0.35	(42, 31.91)	(42, 31.91)	5.13	0.13
0.4	(45, 31.92)	(**42**, **31.91**)	10.08	0.13
0.45	(45, 31.92)	(**42**, **31.90**)	10.73	0.13
0.5	(46, 31.98)	(**42**, **31.89**)	16.59	0.12
0.55	(46, 31.99)	(**43**, **31.80**)	18.44	0.12
0.6	(47, 31.99)	(**46**, **31.59**)	25.5	0.11
0.65	(49, 31.92)	(**46**, **31.52**)	24.83	0.11
0.7	(49, 31.92)	(**46**, **31.52**)	24.72	0.11
0.75	(51, 31.40)	(**47**, **31.23**)	20.39	0.09
0.8	(51, 31.38)	(**47**, **31.23**)	20.44	0.09
0.85	(51, 31.21)	(**47**, **31.17**)	7.42	0.08
0.9	(51, 31.17)	(**47**, **31.15**)	5.68	0.07
0.95	(51, 31.17)	(**47**, **31.15**)	5.75	0.08
1	(51, 31.15)	(51, 31.15)	0.79	0.07
∞	(**42**, 31.91)	(51, **31.15**)	0.18	0.18

Table A.6: Results of CombineTreesDP procedure and CombineTrees procedure for Instance6

ρ	CombineTrees	CombineTreesDP	Time CT	Time CTDP
0.05	(29, 22.71)	(29, 22.71)	0.13	0.72
0.1	(29, 22.71)	(29, 22.71)	0.16	0.67
0.15	(29, 22.71)	(29, 22.71)	0.24	0.61
0.2	(29, 22.71)	(29, 22.71)	0.31	0.59
0.25	(29, 22.71)	(29, 22.71)	0.8	0.57
0.3	(29, 22.71)	(29, 22.71)	0.88	0.57
0.35	(30, 22.76)	(**29**, **22.70**)	1.76	0.56
0.4	(31, 22.81)	(**29**, **22.70**)	3.58	0.54
0.45	(31, 22.87)	(**29**, **22.69**)	7.07	0.53
0.5	(31, 22.87)	(**29**, **22.69**)	7.58	0.52
0.55	(31, 22.89)	(**29**, **22.68**)	13.65	0.52
0.6	(31, 22.81)	(**30**, **22.65**)	27.24	0.46
0.65	(31, 22.81)	(**30**, **22.65**)	27.07	0.45
0.7	(31, 22.63)	(**30**, **22.57**)	45.27	0.36
0.75	(31, 22.63)	(**30**, **22.57**)	38.94	0.31
0.8	(31, 22.45)	(**30**, **22.44**)	26.26	0.22
0.85	(34, 22.42)	(**31**, 22.42)	13.87	0.17
0.9	(34, 22.41)	(**31**, 22.41)	5.04	0.13
0.95	(34, 22.41)	(**33**, 22.41)	1.62	0.12
1	(34, 22.41)	(**33**, **22.40**)	0.51	0.12
∞	(**29**, 22.71)	(33, **22.40**)	0.65	0.49

Table A.7: Results of CombineTreesDP procedure and CombineTrees procedure for Instance7

ρ	CombineTrees	CombineTreesDP	Time CT	Time CTDP
0.05	(15, 13.52)	(15, 13.52)	0.01	0
0.1	(15, 13.52)	(15, 13.52)	0.02	0.17
0.15	(16, 13.70)	(**15**, **13.52**)	0.03	0.15
0.2	(16, 13.70)	(**15**, **13.52**)	0.04	0.15
0.25	(16, 13.98)	(**15**, **13.51**)	0.05	0.12
0.3	(17, 14.16)	(**15**, **13.51**)	0.08	0.1
0.35	(18, 14.28)	(**15**, **13.50**)	0.13	0.1
0.4	(18, 14.28)	(**15**, **13.50**)	0.14	0.09
0.45	(18, 14.34)	(**15**, **13.51**)	0.26	0.1
0.5	(19, 14.39)	(**15**, **13.52**)	0.44	0.11
0.55	(18, 14.41)	(**15**, **13.51**)	0.74	0.13
0.6	(18, 14.41)	(**15**, **13.51**)	0.75	0.13
0.65	(18, 14.40)	(**15**, **13.49**)	1.37	0.18
0.7	(19, 14.35)	(**15**, **13.48**)	2.59	0.29
0.75	(18, 14.22)	(**15**, **13.43**)	5.06	0.52
0.8	(18, 14.22)	(**15**, **13.43**)	5.06	0.51
0.85	(17, 13.88)	(**15**, **13.32**)	9.38	1.03
0.9	(16, 13.32)	(**15**, **13.25**)	9.51	2.8
0.95	(15, 13.25)	(15, 13.25)	5.09	4.73
1	(15, 13.25)	(15, 13.25)	5.05	4.71
∞	(15, 13.52)	(15 ,**13.25**)	0.15	0.14

Table A.8: Results of COMBINETREESDP procedure and COMBINETREES procedure for INSTANCE8

ρ	CombineTrees	CombineTreesDP	Time CT	Time CTDP
0.05	(18, 11.41)	(18, 11.41)	0.01	0
0.1	(18, **11.73**)	(**17**, 11.75)	0.04	0.38
0.15	(19, 12.10)	(19, **10.87**)	0.05	0.33
0.2	(20, 11.96)	(**18**, **10.55**)	0.06	0.28
0.25	(21, 11.99)	(**19**, **10.38**)	0.06	0.22
0.3	(21, 11.97)	(**19**, **10.42**)	0.1	0.19
0.35	(21, 11.50)	(**18**, **9.98**)	0.16	0.16
0.4	(21, 11.09)	(**20**, **10.04**)	0.27	0.14
0.45	(21, 10.46)	(**19**, **9.81**)	0.43	0.15
0.5	(21, 10.46)	(**19**, **9.81**)	0.43	0.14
0.55	(21, 10.13)	(**19**, **9.67**)	0.65	0.17
0.6	(21, 9.86)	(**19**, **9.60**)	0.97	0.24
0.65	(21, 9.71)	(**19**, **9.55**)	1.38	0.36
0.7	(21, 9.61)	(**19**, **9.52**)	1.78	0.53
0.75	(22, 9.55)	(**19**, **9.52**)	2.13	0.79
0.8	(21, 9.52)	(**19**, **9.51**)	2.22	1.1
0.85	(20, 9.51)	(**19**, 9.51)	2.13	1.39
0.9	(20, 9.51)	(**19**, 9.51)	1.97	1.6
0.95	(20, 9.51)	(**19**, 9.51)	1.82	1.71
1	(20, 9.51)	(**19**, 9.51)	1.83	1.7
∞	(**18**, 11.41)	(19, **9.51**)	0.31	0.3

Table A.9: Results of CombineTreesDP procedure and CombineTrees procedure for Instance9

ρ	CombineTrees	CombineTreesDP	Time CT	Time CTDP
0.05	(103, **41.36**)	(103, 41.42)	0.18	0.28
0.1	(103, **41.39**)	(103, 41.47)	0.24	0.24
0.15	(103, 41.37)	(103, **41.32**)	0.43	0.2
0.2	(103, 41.25)	(103, **41.03**)	0.88	0.19
0.25	(103, 40.91)	(103, **40.74**)	1.6	0.18
0.3	(103, 40.45)	(103, **40.24**)	2.49	0.17
0.35	(103, 39.54)	(103, **39.29**)	3.39	0.16
0.4	(103, 39.06)	(103, **38.85**)	3.96	0.15
0.45	(108, 38.75)	(108, **38.55**)	3.79	0.14
0.5	(108, 38.52)	(108, **38.42**)	3.6	0.14
0.55	(108, 38.40)	(108, **38.31**)	3.11	0.13
0.6	(108, 38.31)	(108, **38.26**)	2.19	0.13
0.65	(108, 38.26)	(108, **38.24**)	1.54	0.13
0.7	(108, 38.23)	(108, **38.22**)	0.98	0.13
0.75	(108, 38.23)	(108, **38.22**)	0.62	0.13
0.8	(108, 38.22)	(**104**, 38.22)	0.39	0.12
0.85	(108, 38.22)	(**104**, 38.22)	0.24	0.13
0.9	(104, 38.22)	(104, 38.22)	0.19	0.13
0.95	(104, 38.22)	(104, 38.22)	0.14	0.13
1	(104, 38.22)	(104, 38.22)	0.15	0.12
∞	(**103**, 41.58)	(104, **38.22**)	0.34	0.34

Table A.10: Results of CombineTreesDP procedure and CombineTrees procedure for Instance10

Bibliography

[1] A logarithmic approximation algorithm for evaluating discrete functions. submitted, journal.

[2] ADLER, M.; HEERINGA, B. **Approximating optimal binary decision trees**. APPROX '08 / RANDOM '08, p. 1–9, 2008.

[3] ARKIN, E. M.; MEIJER, H.; MITCHELL, J. S. B.; RAPPAPORT, D. ; SKIENA, S. S. **Decision trees for geometric models**. In: PROCEEDINGS OF THE NINTH ANNUAL SYMPOSIUM ON COMPUTATIONAL GEOMETRY, SCG '93, p. 369–378, 1993.

[4] BAYES, A. J. **Australian Computer Journal**. A dynamic programming algorithm to optimise decision table code, journal, v.5, n.2, p. 77–79, 1973.

[5] BELLALA, G.; BHAVNANI, S. K. ; SCOTT, C. **IEEE Trans. Inf. Theor.** Group-based active query selection for rapid diagnosis in time-critical situations, journal, v.58, n.1, p. 459–478, 2012.

[6] B.M.E. MORET. **ACM Computing Surveys**. Decision Trees and Diagrams, journal, p. 593–623, 1982.

[7] CHAKARAVARTHY, V.; PANDIT, V.; ROY, S.; AWASTHI, P. ; MOHANIA, M. **Decision trees for entity identification: Approximation algorithms and hardness results**. In: PODS, p. 53–62, 2007.

[8] CHAKARAVARTHY, V. T.; PANDIT, V.; ROY, S. ; SABHARWAL, Y. **Approximating decision trees with multiway branches**. In: PROCEEDINGS OF THE 36TH INTERNATIONAL COLLOQUIUM ON AUTOMATA, LANGUAGES AND PROGRAMMING: PART I, ICALP '09, p. 210–221, 2009.

[9] CICALESE, F.; JACOBS, T.; LABER, E. ; MOLINARO, M. **On greedy algorithms for decision trees**. In: ISAAC, 2010.

[10] CORMEN, T. H.; LEISERSON, C. E. ; RIVEST, R. L. **Introduction to Algorithms**. 1st. ed., The MIT Press, 1990.

[11] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. ; STEIN, C. **Introduction to Algorithms**. Cambridge, MA: MIT Press, 2001.

[12] GAREY, M. R. **SIAM Journal on Applied Mathematics**. Optimal binary identification procedures, journal, v.23, n.2, p. 173–186, Sept. 1972.

[13] GAREY, M. R. **SIAM Journal on Computing**. Optimal binary search trees with restricted maximal depth, journal, v.3, n.2, p. 101–110, June 1974.

[14] GOLOVIN, D.; KRAUSE, A. ; RAY, D. **Near-optimal bayesian active learning with noisy observations**. In: Lafferty, J. D.; Williams, C. K. I.; Shawe-Taylor, J.; Zemel, R. S. ; Culotta, A., editors, NIPS, p. 766–774. Curran Associates, Inc, 2010.

[15] GUILLORY, A.; BILMES, J. **Average-case active learning with costs**. In: PROCEEDINGS OF THE 20TH INTERNATIONAL CONFERENCE ON ALGORITHMIC LEARNING THEORY, ALT'09, p. 141–155, 2009.

[16] GUILLORY, A.; BILMES, J. **Interactive submodular set cover**. In: Fürnkranz, J.; Joachims, T., editors, ICML, p. 415–422. Omnipress, 2010.

[17] GUILLORY, A.; BILMES, J. **Simultaneous learning and covering with adversarial noise**. In: Getoor, L.; Scheffer, T., editors, ICML, p. 369–376. Omnipress, 2011.

[18] GUPTA, A.; NAGARAJAN, V. ; RAVI, R. **Approximation algorithms for optimal decision trees and adaptive tsp problems**. In: PROCEEDINGS OF THE 37TH INTERNATIONAL COLLOQUIUM CONFERENCE ON AUTOMATA, LANGUAGES AND PROGRAMMING, ICALP'10, p. 690–701, 2010.

[19] HANNEKE, S. **unpublished**. The cost complexity of interactive learning, journal, v.http://www.stat.cmu.edu/ shanneke/docs/2006/cost-complexity-working-notes.pdf, 2006.

[20] HU, T. C.; TAN, K. C. **SIAM Journal on Applied Mathematics**. Path length of binary search trees, journal, v.22, n.2, p. 225–234, Mar. 1972.

[21] HYAFIL, L.; RIVEST, R. L. **Inf. Process. Lett.** Constructing optimal binary decision trees is np-complete, journal, v.5, n.1, p. 15–17, 1976.

[22] KOSARAJU; PRZYTYCKA ; BORGSTROM. **On an optimal split tree problem**. In: WADS: 6TH WORKSHOP ON ALGORITHMS AND DATA STRUCTURES, 1999.

[23] LABER, E. S.; NOGUEIRA, L. T. **Discrete Appl. Math.** On the hardness of the minimum height decision tree problem, journal, v.144, p. 209–212, 2004.

[24] LARMORE; HIRSCHBERG. **JACM: Journal of the ACM**. A fast algorithm for optimal length-limited huffman codes, journal, v.37, 1990.

[25] LARMORE, L. L. **SIAM Journal on Computing**. Height restricted optimal binary trees, journal, v.16, n.6, p. 1115–1123, Dec. 1987.

[26] LIPMAN, M. J.; ABRAHAMS, J. **IEEE Transactions on Information Theory**. Minimum average cost testing for partially ordered components, journal, v.41, n.1, p. 287–291, 1995.

[27] MARTELLI, A.; MONTANARI, U. **Communications of the Association for Computing Machinery**. Optimizing decision trees through heuristically guided search, journal, v.21, p. 1025–1039, 1978.

[28] NEVMYVAKA, Y.; FENG, Y. ; KEARNS, M. **Reinforcement learning for optimized trade execution**. Technical report, In ICML '06: Proceedings of the 23rd international conference on Machine learning, 2006.

[29] SCHUMACHER, H.; SEVCIK, K. C. **Commun. ACM**. The synthetic approach to decision table conversion, journal, v.19, n.6, p. 343–351, 1976.