ANYTIME ALGORITHMS FOR
LEARNING ANYTIME CLASSIFIERS

SAHER ESMEIR
ANYTIME ALGORITHMS FOR LEARNING
ANYTIME CLASSIFIERS

RESEARCH THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

SAHER ESMEIR

SUBMITTED TO THE SENATE OF THE TECHNION — ISRAEL INSTITUTE OF TECHNOLOGY
ELUL, 5768 HAIFA SEPTEMBER, 2008
ACKNOWLEDGMENTS

Words cannot suffice to describe my deep gratitude to my advisor, Shaul Markovitch, who guided me during my studies and was always available when I needed him. I learned from Shaul how to become an independent researcher and how to think innovatively, yet critically. His encouragement and support were crucial in shaping my research. He taught me not to give up and to believe in what I did. Above all, he perfectly balanced between guiding me professionally and maintaining a respectful and pleasant relationship.

I owe a notable debt to Evgeniy Gabrilovich for his help and advice throughout my studies, as well as to Lev Finkelstein for his contributions to the anytime classification project. I learned a huge deal from the both of them.

I am also grateful to the following individuals, who shared their wisdom, enriched my life with their insights, and were always willing to assist me: Ron Begleiter, Sivan Bercovici, Ofer Egozi, Ran El-Yaniv, Ariel Felner, Yaniv Hamo, Anat Hashavit, Nela Gurevich, Sonya Liberman, Kira Radinsky, Ron Rubinstein, Shadi Saba, Sunil Vadera, Kolman Vornovitsky, Haggai Toledano, Lina Zarivach, and Shlomo Zilberstein.

My sincere thanks and appreciation to Microsoft Research and Development Center at Haifa and Medic Vision Brain for hosting me as summer intern, to the administrative staff at the Computer Science Department for their help and assistance, and to Sharon Kessler, who read significant portions of this manuscript and improved it with her editorial comments, excitement and passion for details.
And there are also those friends who heard me speak about this project, day and night, during the last couple of years. Their immeasurable love, support, and care gave me the power to work and lead me to new paths.

Finally, I am deeply beholden to my loving family, who offered me unconditional love and support throughout the course of this thesis: my sister Samera, my brothers Samer and Esmeir, and my parents, Mona and Nimer, who have raised me to be the person I am today. Mom and dad: without you, nothing of this would have been ever possible.

THE GENEROUS FINANCIAL HELP OF THE TECHNION, THE CHE (COUNCIL FOR HIGHER EDUCATION), AND THE MIRIAM AND AARON GUTWIRTH MEMORIAL FOUNDATION IS GRATEFULLY ACKNOWLEDGED
To my parents,
Mona and Nimer Esmeir
Contents

Abstract 1

1 Introduction 5
1.1 Proposed Solution 8
1.2 Major Contributions 10
1.3 Thesis Outline 11

2 Resource-bounded Learning and Classification 13

3 Contract Anytime Learning of Accurate Trees 19
3.1 Top-down Induction of Decision Trees 19
3.2 Limitation of Greedy Learners 20
3.3 Tree-size as a Preference Bias 22
3.4 Fixed-depth Lookahead 24
3.5 A Contract Algorithm for Learning Decision trees 25
   3.5.1 Evaluating Continuous Attributes by Sampling 30
   3.5.2 Multiway Vs. Binary Splits 32
   3.5.3 Pruning the LSID3 Trees 32
   3.5.4 Mapping Contract Time to Sample Size 33
3.6 Interruptible Induction of Decision Trees 34
   3.6.1 Interruptible Learning by Sequencing 35
   3.6.2 Iterative Improvement of Decision Trees 36
   3.6.3 Incremental Learning: When New Examples Arrive 42
3.7 Empirical Evaluation 45
   3.7.1 Experimental Methodology 45
   3.7.2 Fixed Time Comparison 46
   3.7.3 Anytime Behavior of the Contract Algorithms 54
   3.7.4 Anytime behavior of IIDT 59
   3.7.5 IIDT as Incremental Learner 62
   3.7.6 Comparison with Modern Decision Tree Learners 63
4 Anytime Learning of Cost-sensitive Trees
4.1 Biasing the Sample Towards Low-cost Trees
4.2 Evaluating Sample Trees
4.2.1 Estimating Test Costs
4.2.2 Estimating Misclassification Costs
4.3 Choosing a Split
4.3.1 Choosing a Split: Illustrative Examples
4.3.2 Choosing a Split when Attributes are Numeric
4.4 Cost-sensitive Pruning
4.5 Setting the Parameters of ACT
4.6 Empirical Evaluation
4.6.1 Datasets
4.6.2 Methodology
4.6.3 Fixed-time Comparison
4.6.4 Comparing the Accuracy of the Learned Models
4.6.5 Comparison of Anytime Behavior
4.6.6 Random Costs
4.6.7 Nonuniform Misclassification Costs
5 Anytime Learning of Anycost Classifiers
5.1 Pre-contract: Classification Budget is Known to the Learner
5.1.1 Top-down Induction of Anycost Trees
5.1.2 Greedy TDIDT$ Instantiations
5.1.3 The Pre-Contract-TATA Algorithm
5.1.4 Interruptible Learning of Pre-contract Classifiers
5.2 Contract: When Allocation is Made Right Before Classification
5.2.1 Repertoire of Trees
5.2.2 Learning a repertoire with Nonuniform Cost Gaps
5.3 Interruptible: No Pre-determined Classification Budget
5.4 Empirical Evaluation
5.4.1 Pre-Contract Classification
5.4.2 Contract Classification
5.4.3 Interruptible Classification
6 Related Work
6.1 Non-greedy Cost-insensitive Decision Tree Inducers
6.1.1 Skewing
6.1.2 Other Cost-insensitive Decision-Tree Inducers
6.2 Anytime Induction of Other Cost-insensitive Classifiers
6.3 Cost-sensitive Classification
6.4 Other Learning Components
List of Figures

1.1 Three possible decision trees for diagnosis of heart diseases . . . . . . . 6
2.1 Overview of our proposed framework . . . . . . . . . . . . . . . 15
3.1 Procedure for top-down induction of decision trees . . . . . . . . . . 20
3.2 Learning the 2-XOR concept by ID3 . . . . . . . . . . . . . . . 21
3.3 Procedures for computing entropy $k$ and gain $k$ . . . . . . . . . . 24
3.4 Attribute selection in SID3 . . . . . . . . . . . . . . . . . . . . . 26
3.5 Tree-size frequency curves . . . . . . . . . . . . . . . . . . . . . 27
3.6 The minimum tree size as estimated by SID3 and RTG . . . . . . . . 28
3.7 Attribute evaluation using LSID3 . . . . . . . . . . . . . . . . . 29
3.8 Attribute selection in LSID3 . . . . . . . . . . . . . . . . . . . . . 29
3.9 Monte Carlo evaluation of continuous attributes using SDI3 . . . 31
3.10 The space of decision trees over a set of attributes and examples . . 34
3.11 Converting LSID3 to an interruptible algorithm by sequencing . . . 35
3.12 IIDT: framework for interruptible induction of decision trees . . . . 37
3.13 Iterative improvement of the 2-XOR decision tree . . . . . . . . . . 38
3.14 Choosing a node for reconstruction in IIDT . . . . . . . . . . . . . 40
3.15 IIDT-TS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
3.16 IIDT-EE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
3.17 I-IIDT . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
3.18 Incorporating a new example by I-IIDT-TS . . . . . . . . . . . . . 44
3.19 Incorporating a new example by I-IIDT-EE . . . . . . . . . . . . . 45
3.20 Performance differences between LSID3(5) and ID3 . . . . . . . . . 47
3.21 Performance differences for LSID3(5) and ID3-k . . . . . . . . . . 48
3.22 Performance differences for LSID3-p(5) and C4.5 . . . . . . . . . . 53
3.23 Anytime behavior of ID3-k and LSID3 on the Multiplex-XOR dataset 55
3.24 Anytime behavior of ID3-k and LSID3 on the 10-XOR dataset . . . 56
3.25 Anytime behavior of ID3-k and LSID3 on the Tic-tac-toe dataset . . 57
3.26 Anytime behavior of ID3-k, LSID3 on the Numeric-XOR 4D dataset 58
3.27 Anytime behavior of LSID3-MC on the Numeric-XOR 4D dataset . . 59
3.28 Anytime behavior of IIDT on the Glass dataset . . . . . . . . . . . 60
3.29 Anytime behavior of IIDT on the 10-XOR dataset .............. 61
3.30 Anytime behavior of IIDT on the Tic-tac-toe dataset ........... 62
3.31 Time steps for a single run of IIDT on 10-XOR ................. 63
3.32 Learning the XOR-10 problem incrementally .................. 64
3.33 Anytime behavior of modern learners on the XOR-5 dataset ... 66
3.34 Anytime behavior of modern learners on the Multiplexer-20 dataset 67
3.35 Anytime behavior of modern learners on the Tic-tac-toe dataset .... 68

4.1 Attribute selection in SEG2 ................................. 72
4.2 Procedure for Attribute selection in ACT ..................... 75
4.3 Attribute evaluation in ACT ................................ 75
4.4 Evaluation of tree samples in ACT (uniform costs) ............. 76
4.5 Evaluation of tree samples in ACT (nonuniform test costs) .... 76
4.6 Evaluation of tree samples in ACT (nonuniform error penalties) 77
4.7 Average normalized cost as a function of misclassification cost .... 86
4.8 The differences in performance between ACT and ICET .......... 95
4.9 Average accuracy as a function of misclassification cost ....... 96
4.10 Average normalized cost as a function of time ................ 96
4.11 Average cost when test costs are assigned randomly .......... 97
4.12 Comparison of various algorithms when error costs are nonuniform 98

5.1 Exemplifying the recursive invocation of TDIDT$ ............... 101
5.2 Top-down induction of anycost decision trees .................. 102
5.3 Attribute evaluation in Pre-Contract-TATA .................... 104
5.4 Attribute selection in pre-contract-TATA ...................... 105
5.5 Building a repertoire in contract-TATA with uniform cost gaps ... 106
5.6 Classifying a case using a repertoire in contract-TATA ........ 107
5.7 Building a repertoire in contract-TATA using the hill-climbing approach .............................................. 108
5.8 Using repertoires in interruptible-TATA ....................... 108
5.9 An example of applying cost discounts ......................... 110
5.10 Procedure for applying discounts ............................. 111
5.11 Results for pre-contract classification ......................... 113
5.12 Results for pre-contract classification on Glass, AND-OR, MULTI-XOR, and KRK ........................................... 114
5.13 TATA with different sample sizes on the Multi-XOR dataset ... 115
5.14 Results for contract classification ................................ 116
5.15 Learning repertoires with different time allocations and sample sizes 117
5.16 Results for interruptible classification .......................... 118

6.1 The sensitivity of LSID3 and skewing to irrelevant attributes .... 123
6.2 Exemplifying the difficulties other methods might face .......... 129
B.1 Top-down induction of decision trees .......................... 145
B.2 Attribute selection in SID3 ........................................... 147
B.3 Attribute selection in LSID3 ........................................ 148
B.4 Frequency curves .................................................... 154
B.5 Correlation between size and accuracy for consistent trees .... 155
List of Tables

3.1 Possible training set for learning the 2-XOR concept .......... 21
3.2 Characteristics of the datasets used ......................... 46
3.3 The size of the induced trees on various datasets .......... 49
3.4 The differences in the size of the induced trees on various datasets 50
3.5 The generalization accuracy of the induced trees on various datasets 51
3.6 The differences in generalization accuracy of the induced trees on various datasets 52

4.1 Characteristics of the datasets used to evaluate ACT ........ 82
4.2 Average cost of classification for different \( mc \) values .......... 85
4.3 DTMC vs. ACT and ICET vs. ACT using statistical tests .... 86
4.4 Average cost of classification for \( mc = 100 \) .............. 87
4.5 Average cost of classification for \( mc = 500 \) .............. 88
4.6 Average cost of classification for \( mc = 1000 \) ........... 89
4.7 Average cost of classification for \( mc = 5000 \) ........... 90
4.8 Average cost of classification for \( mc = 10000 \) .......... 91
4.9 Average cost when test costs are assigned randomly .......... 93
4.10 Comparison of various algorithms when error costs are nonuniform 94

5.1 Characteristics of the datasets used to evaluate TATA ........ 112
5.2 Comparing the misclassification cost for different contracts .... 113

B.1 Testing Occam’s empirical principle for consistent trees .... 149
B.2 Testing Occam’s empirical principle when the trees are pruned 151
Abstract

Machine learning techniques are gaining prevalence in the production of a wide range of classifiers for complex real-world applications with nonuniform testing and misclassification costs. The increasing complexity of these applications poses a real challenge to resource management during learning and classification. In this work we introduce a novel framework for operating in such complex environments. The core of our framework is an anytime decision tree learner that allows learning time to be increased in return for lower classification costs. It builds a tree top-down and exploits additional time resources to obtain better estimations for the utility of the different candidate splits. Using sampling techniques, we approximate the cost of the subtree under each candidate split and favor the one with a minimal cost. As a stochastic algorithm, the learner is expected to be able to escape local minima, into which greedy methods may be trapped. Our proposed approach can be applied in two anytime setups: the contract setup, where the allocation of resources is known in advance, and the interruptible setup, where the algorithm might be queried for a solution at any moment. In addition to constraints on learning time, many real-life scenarios involve limits on classification resources. To deal with such situations, we developed a set of learning algorithms that can produce any-cost classifiers, i.e., classifiers that can make accurate decisions within a strict bound on testing costs. The classification budget can be known to the learner, known to the classifier but not to the learner, or not predetermined. Experiments with a variety of datasets, and under different scenarios, were conducted to compare the performance of our proposed methods to that of the state-of-the-art tree learners. The results show that for the majority of domains we produce significantly better trees. In the cost-insensitive setup, where test costs are ignored, we could produce smaller and more accurate trees. When test costs are involved, our learned trees were significantly more efficient for classification. Our learning algorithms are also shown to exhibit good anytime behavior with diminishing returns.
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAPE</td>
<td>Anytime Averaged Probabilistic Estimators</td>
</tr>
<tr>
<td>ACT</td>
<td>Anytime learning of Cost-sensitive Trees</td>
</tr>
<tr>
<td>ANN</td>
<td>Anytime Nearest Neighbor</td>
</tr>
<tr>
<td>DMTI</td>
<td>Direct Metric Tree Induction</td>
</tr>
<tr>
<td>DTMC</td>
<td>Decision Trees with Minimal Cost</td>
</tr>
<tr>
<td>EG2</td>
<td>Economic Generalized 2</td>
</tr>
<tr>
<td>GATree</td>
<td>Genetic Algorithms Tree</td>
</tr>
<tr>
<td>ICET</td>
<td>Inexpensive Classification with Expensive Tests</td>
</tr>
<tr>
<td>ID3</td>
<td>Iterative Dichotomiser 3</td>
</tr>
<tr>
<td>ID3-k</td>
<td>Iterative Dichotomiser 3 with k-depth lookahead</td>
</tr>
<tr>
<td>IDS</td>
<td>Intrusion Detection System</td>
</tr>
<tr>
<td>IIDT</td>
<td>Interruptible Induction of Decision Trees</td>
</tr>
<tr>
<td>IIDT-EE</td>
<td>Interruptible Induction of Decision Trees (Expected Error)</td>
</tr>
<tr>
<td>IIDT-TS</td>
<td>Interruptible Induction of Decision Trees (Tree Size)</td>
</tr>
<tr>
<td>I-IIDT</td>
<td>Incremental Interruptible Induction of Decision Trees</td>
</tr>
<tr>
<td>I-IIDT-EE</td>
<td>Incremental Interruptible Induction of Decision Trees (Expected Error)</td>
</tr>
<tr>
<td>LSID3</td>
<td>Lookahead by Stochastic Iterative Dichotomiser 3</td>
</tr>
<tr>
<td>LSID3-MC</td>
<td>Lookahead by Stochastic Iterative Dichotomiser 3 with Monte-Carlo splits</td>
</tr>
<tr>
<td>LSID3-p</td>
<td>Lookahead by Stochastic Iterative Dichotomiser 3 with Pruning</td>
</tr>
<tr>
<td>MDL</td>
<td>Minimum Description Length</td>
</tr>
<tr>
<td>OC1</td>
<td>Oblique Classifier 1</td>
</tr>
<tr>
<td>PAC</td>
<td>Probably Approximately Correct</td>
</tr>
<tr>
<td>QUEST</td>
<td>Quick, Unbiased and Efficient Statistical Tree</td>
</tr>
<tr>
<td>RTG</td>
<td>Random Tree Generator</td>
</tr>
<tr>
<td>SC4.5</td>
<td>Stochastic C4.5</td>
</tr>
<tr>
<td>SEG2</td>
<td>Stochastic Economic Generalized 2</td>
</tr>
<tr>
<td>SID3</td>
<td>Stochastic Iterative Dichotomiser 3</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>TATA</td>
<td>Tree-classification AT Anycost</td>
</tr>
</tbody>
</table>
TCSC  Time and Cost Sensitive Classification
VIOLA Value of Information Lattice
Chapter 1

Introduction

Machine learning techniques are gaining prevalence in the production of a wide range of classifiers for complex real-world applications with nonuniform testing and misclassification costs. The increasing complexity of these applications poses a real challenge to resource management during learning and classification.

Assume, for example, that a medical center has decided to use machine learning techniques to build a diagnostic tool for heart disease. Figure 1.1 shows three possible trees. The first tree (upper-left) makes diagnoses using only the results of cardiac catheterization (heart cath). This tree is expected to be highly accurate. Nevertheless, the high costs and risks associated with the heart cath procedure make this decision tree impractical. The second tree (lower-left) dispenses with the need for cardiac catheterization and makes a diagnosis based on a single, simple, inexpensive test: whether or not the patient complains of chest pain. Such a tree would be highly accurate: most people do not experience chest pain and are indeed healthy. The tree, however, does not distinguish between the costs of different types of errors. While a false positive prediction might result in extra treatments, a false negative prediction might put a person’s life at risk. Therefore, a third tree (right) is preferred, one that attempts to minimize test costs and misclassification costs simultaneously.

As another example, consider a hardware manufacturer who uses a machine-learning based tool for assuring the quality of produced chips. In realtime, each chip in the pipeline is scanned and several features can be extracted from the image. The features vary in their computation time. The manufacturer trains the component using thousands of chips whose validity is known. Because the training is done offline, the manufacturer can provide the values of all possible features, regardless of their computation time. In realtime, however, the model must make a decision within 2 seconds. Therefore, for each chip, the classifier may use features whose total computation time is at most 2 seconds. Unlike the medical center, the manufacturer is not interested in minimizing the total
**Figure 1.1:** Three possible decision trees for diagnosis of heart disease. The upper-left tree bases its diagnosis solely on heart cath and is therefore accurate but prohibitively expensive. The lower-left tree dispenses with the need for heart cath and makes a diagnosis using a single, simple, and inexpensive test: whether or not the patient complains of chest pain. Such a tree would be highly accurate but does not distinguish between the costs of the different error types. The third (right-hand) tree is preferable: it attempts to minimize test costs and misclassification costs simultaneously.

The predictive models learned by the medical center and by the hardware manufacturer, are going to be used during the coming months, or even years, to make vital decisions. Obviously, in both cases we are willing to invest many resources during the learning phase to reduce future classification costs.

An algorithm that can improve the quality of its output with the increase in allocated resources is called an anytime algorithm (Boddy & Dean, 1994). We use a similar notation and refer to a classifier whose accuracy improves with the increase in allocation of testing resources as an *anycost* classifier. In this work we propose a novel and innovative framework that allows anytime learning of anycost classifiers. Our algorithms are practical for numerous real-life applications that involve offline learning (Esmeir & Markovitch, 2006). They allow for a greater investment of learning resources in return for better predictive models.

Despite the recent progress in advanced induction algorithms such as SVM
(Vapnik, 1995), decision trees are still considered attractive for many real-life applications, mostly due to their interpretability (Hastie, Tibshirani, & Friedman, 2001, chap. 9). Craven (1996) lists several reasons why the understandability of a model by humans is an important criterion for evaluating it. These reasons include, among others, the possibility for human validation of the model and generation of human-readable explanations for the classifier predictions. When classification cost is important, decision trees may be attractive in that they ask only for the values of the features along a single path from the root to a leaf. In terms of accuracy, decision trees have been shown to be competitive with other classifiers for several learning tasks.

For any given set of training examples, many trees can be induced. An important questions is what tree should be preferred? Cost-insensitive decision tree learners attempt to produce small consistent trees, which, by Occam’s razor (Blumer, Ehrenfeucht, Haussler, & Warmuth, 1987), should have better predictive power. The problem of finding the smallest consistent tree is known to be NP-complete (Hyafil & Rivest, 1976; Murphy & McCraw, 1991). Obviously, when costs are involved, the problem of finding the tree with the lowest expected total cost is even more difficult. One way to overcome the unaffordable complexity of finding the optimal tree is to use greedy heuristics.

The majority of existing decision-tree learners build a tree top-down and choose a split based on local heuristics. The well-known, cost-insensitive C4.5 algorithm (Quinlan, 1993) uses the gain ratio as a heuristic for predicting which attribute will yield a smaller tree. Several other alternative local greedy measures have been developed, among which are ID3’s information gain, the Gini index (Breiman, Friedman, Olshen, & Stone, 1984), and the chi-square test (Mingers, 1989). For problems with test costs, a number of works suggest considering both the cost of a split and the immediate gain in information it yields, e.g., EG2 (Nunez, 1991). When nonuniform misclassification costs are involved, the DTMC algorithm evaluates a feature by its immediate reduction in total cost (Ling, Yang, Wang, & Zhang, 2004; Sheng, Ling, Ni, & Zhang, 2006).

The top-down methodology has the advantage of evaluating a potential attribute for a split in the context of the tests associated with the nodes above it. The local greedy measures, however, consider each of the remaining attributes independently, ignoring potential interaction between different attributes (Mingers, 1989; Kononenko, Simec, & Robnik-Sikonja, 1997; Kim & Loh, 2001). In parity-like concepts, for example, the usefulness of an attribute cannot be discovered when the attribute is judged independently. Such concepts naturally arise in real-world data such as the Drosophila survival rate (Page & Ray, 2003). This problem is intensified when test costs are involved. High costs may hide the

---

1A consistent decision tree is a tree that correctly classifies all training examples.
utility of informative features and lead the learner to prefer irrelevant features, which, when evaluated locally, seem better.

The effect of a wrong split decision in the greedy top-down strategy is propagated down to all the nodes below it. The greedy algorithms, nevertheless, cannot utilize extra time resources to avoid the irreversible damage caused by wrong splits. In this work we proposed a novel approach for building decision trees. Our approach allows more resources to be invested in the learning process in return for producing better models. Quinlan (1993, chap. 11) recognized the need for this type of anytime algorithm for decision tree learning: “What is wanted is a resource constrained algorithm that will do the best it can within a specified computational budget and can pick up threads and continue if this budget is increased. This would make a challenging thesis topic!”

Note that in this work we assume the availability of a set of labeled examples, along with their feature values. In many scenarios the budget for obtaining the training examples is limited. Active learning (Lindenbaum, Markovitch, & Rusakov, 2004) and budgeted learning (Lizotte, Madani, & Greiner, 2003) deal with the acquisition of training examples. We view these works as orthogonal to our work.

1.1 Proposed Solution

In order to overcome the inherited shortcomings of greedy split evaluation, a fundamentally new approach for attribute evaluation is needed. One way to exploit additional learning time resources is to perform lookahead. Lookahead search is a well-known technique for improving greedy algorithms (Sarkar, Chakrabarti, Ghose, & DeSarkar, 1994). When applied to decision tree induction, lookahead attempts to predict the profitability of a split at a node by estimating its effect on deeper descendants of the node. The damage caused by a wrong decision in top-down induction is irreversible: once an attribute was chosen to split on, there is no provision for backtracking and choosing another attribute instead. Lookahead search attempts to predict and avoid such non-contributive splits during the process of induction, before the final decision at each node is taken.

Lookahead techniques have been applied to decision tree induction by several researchers. The reported results vary from lookahead produces better trees (Nor- ton, 1989; Ragavan & Rendell, 1993; Dong & Kothari, 2001) to lookahead does not help and can hurt (Murthy & Salzberg, 1995). One problem with these works is their use of a uniform, fixed low-depth lookahead, therefore disqualifying the proposed algorithms from serving as anytime algorithms. Another problem is the datasets on which the lookahead methods were evaluated. For simple learning tasks, such as induction of conjunctive concepts, greedy methods perform quite
well and no lookahead is needed. However, for more difficult concepts such as XOR, the greedy approach is likely to fail. A third problem is their limitedness to a specific objective: they cannot be adapted to different learning setups and other objectives, such as minimizing testing and misclassification costs.

We therefore propose an alternative approach for looking ahead. For each candidate split we sample the spaces of subtrees under it and estimate the utility of the sampled trees. Because we evaluate entire trees, different utility functions can be used, depending on the actual cost scheme. The split with the best tree in its sample is then selected to split on.

In the cost-insensitive setup, our goal is to induce small and accurate trees. Following Occam’s razor, we bias the sample towards small consistent trees and evaluate each sample tree by its size. To avoid overfitting the training examples, we apply a post-pruning phase, similarly to C4.5.

When our objective is to minimize the total cost, we bias the sample towards low cost trees, and evaluate the sampled trees by their expected total cost. The total cost of a tree is estimated using the average costs of classifying the training examples using the tree, and the expected error of the tree. In cost-insensitive environments, the main goal of pruning is to simplify the tree in order to avoid overfitting the training data. A subtree is pruned if the resulting tree is expected to yield a lower error. When test costs are taken into account, pruning has another important role: reducing test costs in a tree. Keeping a subtree is worthwhile only if its expected reduction in misclassification costs is larger than the cost of the tests in that subtree. Therefore, we designed a novel pruning approach based on the expected total cost of a tree.

For the scenarios that constrain the testing costs, we developed a novel top-down approach to exploit the available testing resources. When the bounds are known to the learner, a tree that fits the budget is built. In other cases, a repertoire of trees is formed. If the quota is known before classification, a single tree that best fits the budget is picked. Otherwise, the trees are traversed until resources are exhausted.

Our anytime approach can benefit from extra learning time by creating larger samples. The larger the samples are, the more accurate the attribute evaluation is. There are two main classes of anytime algorithms, namely contract and interruptible (Russell & Zilberstein, 1996). A contract algorithm is one that gets its resource allocation as a parameter. An interruptible algorithm is one whose resource allocation is not given in advance and thus must be prepared to be interrupted at any moment. While the assumption of preallocated resources holds for many induction tasks, in many other real-life applications it is not possible to allocate the resources a priori. Therefore, in our work, we are interested both in contract and interruptible decision tree learners. In the contract setup, the sample size is predetermined according to the available resources. In the interruptible
setup we start with a greedy tree. Then, we repeatedly select a subtree whose reconstruction is expected to yield the highest marginal utility, and rebuild the subtree with a doubled allocation of resources.

1.2 Major Contributions

The major contributions of this thesis are:

1. *First work on anytime induction of anytime classifiers:*
   To the best of our knowledge, this is the first work that deals with anytime learning of anytime classifiers. Our proposed framework allows computational resources during the learning process to be traded for better classifiers, whether these classifiers are cost-insensitive, cost-sensitive, or resource-bounded. Allowing such a tradeoff is useful for many real-world applications where the user is willing to invest more learning resources than required by the greedy methods. The framework can be invoked both in the contract setup where the learning resources are predetermined or in the interruptible setup where the learner may be stopped anytime.

2. *Algorithms for producing both accurate and comprehensible classifiers:*
   In comparison to other recently developed learning algorithms such as SVM, decision trees are known to be easier to use and more humanly comprehensible. LSID3 and IIDT, our proposed algorithms for cost-insensitive learning, make it possible to learn accurate classifiers without compromising on comprehensibility.

3. *Algorithms for producing low error, low cost classifiers:*
   Many real-world problems involve nonuniform testing and misclassification costs. Our novel ACT algorithm can utilize extra learning time to produce cost-efficient trees with lower total costs. ACT evaluates entire trees during the search; thus, it can be adjusted to any cost scheme that is defined over trees.

4. *Algorithms for producing anycost classifiers:*
   When classification resources are limited, we need a classifier that can do its best within the available resources. TATA, our anytime framework for learning anycost classifiers, is designed to produce accurate classifiers that can operate under strict cost constraints, preallocated or not.

5. *Nonparametric method for learning hard concepts:*
   Several hard concepts such as parity functions are known to be a problem
for induction algorithms. In such cases, previous knowledge about the target concept can help in choosing better parameters for the classifier, e.g., the right architecture of a neural network or the right kernel for an SVM. However, such previous knowledge usually does not exist. Our proposed algorithms can learn these hard concepts without the need for an ad hoc setup.

6. **Easy to parallelize learner:**
   Our sampling approach for evaluating candidate splits during top-down tree induction can easily be parallelized: different machines can be used to sample the space of trees simultaneously and independently. Therefore, the method benefits from distributed computer power.

7. **Empirical study of Occam’s razor:**
   Occam’s razor is the principle that, given two hypotheses consistent with the observed data, the simpler one should be preferred. Many machine learning algorithms follow this principle and search for a small hypothesis within the version space. The principle has been the subject of a heated debate with theoretical and empirical arguments both for and against it. Earlier empirical studies lacked sufficient coverage to resolve the debate. In this work we provide convincing empirical evidence for Occam’s razor in the context of decision tree induction, and show that indeed a smaller tree is likely to be more accurate, and that this correlation is statistically significant.

8. **Automatic method for cost-assignment to existing datasets:**
   Typically, machine learning researchers use datasets from the UCI repository (Asuncion & Newman, 2007). Only five UCI datasets, however, have assigned test costs. To gain a wider perspective, we have developed an automatic, parameterized method that assigns costs to existing datasets.

1.3 **Thesis Outline**

The rest of this thesis is organized as follows. In Chapter 2 we provide background on anytime algorithms and resource-bounded classification, and describe the different scenarios under which our proposed framework can operate. Chapter 3 introduces our novel anytime approach for sampling-based attribute evaluation and instantiates this approach for creating accurate decision trees. An interruptible approach for acting when learning resources are not preallocated is described in Chapter 3 as well. In Chapter 4 we present our methodology for constructing low-error, low-cost trees. In Chapter 5 we focus on inducing resource-bounded
classifiers that can utilize strict classification budgets. Each of the algorithms presented in Chapters 3-5 is accompanied by the results of extensive empirical study that compares the proposed methods to previous learners and examines their anytime and anycost behavior. In Chapter 6 we discuss our methodology in the context of prior work and related literature. Finally, Chapter 7 concludes the thesis and outlines directions for future research.
Chapter 2

Resource-bounded Learning and Classification

In real-life applications, inductive concept learning involves several types of cost (Turney, 2000). In the learning phase costs are associated with the acquisition of feature-values (Melville, Saar-Tsechansky, Provost, & Mooney, 2004; Lizotte et al., 2003; Sheng & Ling, 2007a) and with tagging examples (Lindenbaum et al., 2004).

Once the data has been collected, the next stage is model learning. This stage usually requires resources such as memory and computation time. Algorithms whose results improve with the increase in the allocated resources are called anytime algorithms. Anytime algorithms are attractive when optimal solutions require unaffordable resources and quick greedy solutions do not suffice. Similar situations, where the solution can be either of exponential complexity and therefore prohibitively expensive or non-optimal and fast, are common in AI: anytime techniques have been applied to several AI domains, such as heuristic search (Hansen, Zilberstein, & Danilchenko, 1997), constraint satisfaction problems (Fargier, Lang, & Schiex, 1996), and SAT solvers (Wallace & Freuder, 1995).

There are two main classes of anytime algorithms, namely contract and interruptible (Russell & Zilberstein, 1996). A contract algorithm is one that gets its resource allocation as a parameter. If interrupted at any point before the allocation is completed, it might not yield any useful results. An interruptible algorithm is one whose resource allocation is not given in advance and thus must be prepared to be interrupted at any moment. While the assumption of preallocated resources holds for many induction tasks, in many other real-life applications it is not possible to allocate the resources in advance. Therefore, in this work we are interested in both types of anytime algorithms.

During the classification phase, costs are associated with the measurements required by the model and with the prediction errors. Following previous works
on cost-sensitive induction, we adopt the model described by Turney (1995). In a problem with \(|C|\) different classes, a misclassification cost matrix \(M\) is a \(|C| \times |C|\) matrix whose \(M_{i,j}\) entry defines the penalty of assigning the class \(c_i\) to an instance that actually belongs to the class \(c_j\). Let \(\Theta(h,e)\) be the set of tests required by the hypothesis \(h\). We denote by \(\text{cost}(\theta)\) the cost of administering the test \(\theta\). The testing cost of \(e\) in \(h\) is therefore \(\text{tcost}(h,e) = \sum_{\theta \in \Theta} \text{cost}(\theta)\). Note that we use sets notation because tests that appear several times are charged for only once.

In addition, the model described by Turney (1995) handles two special test types, namely grouped and delayed tests.

**Grouped Tests.** Some tests share a common cost, for which we would like to charge only once. Typically, tests also has an extra (possibly different) cost. For example, consider a tree path with tests like cholesterol level and glucose level. For both values to be measured, a blood test is needed. Taking blood samples to measure the cholesterol level clearly lowers the cost of measuring the glucose level. Formally, each test possibly belongs to a group. If it’s the first test from the group to be administered, we charge for the full cost. If another test from the same group has already been administered earlier in the decision path, we charge only for the marginal cost.

**Delayed Tests.** Sometimes the outcome of a test cannot be obtained immediately, e.g., lab test results. Such tests, called delayed tests, force us to wait until the outcome is available. Alternatively, Turney (1995) suggests taking into account all possible outcomes: when a delayed test is encountered, all the tests in the subtree under it are administered and charged for. Once the result of the delayed test is available, the prediction is at hand. One problem with this setup is that it follows all paths in the subtree, regardless of the outcome of non-delayed costs. Moreover, it is not possible to distinguish between the delays different tests impose: for example, one result might be ready after several minutes while another only after a few days. In this work we do not handle delayed tests, but we do explain how our learner can be modified to take them into account.

The total cost of classification is the sum of the miscalculation cost and the testing cost. Cost-sensitive classifiers, which aim to reduce the total cost of classification, require \(M_{i,j}\) and \(\text{cost}(\theta)\) to be given in the same currency. A more general model would require a utility function that combines both types, for example, by eliciting user preferences (Lenert & Soetikno, 1997). An important property of the aforementioned cost-sensitive setup is that maximizing generalization accuracy, which is the goal of most existing learners, can be viewed as a special case: when accuracy is the only objective, test costs are ignored and misclassification cost is uniform.
Figure 2.1: Overview of our proposed framework. The training examples are provided to an anytime learner that can trade computation speed for better classifiers. The learning time, $\rho^l$, can be preallocated (contract learning), or not (interruptible learning). Similarly, the maximal classification budget, $\rho^c$, may be provided to the learner (pre-contract classification), become available right before classification (contract classification), or be unknown, in which case the classifier should utilize additional resources until interrupted (interruptible classification).

While minimizing the total cost of classification is appropriate for many tasks, many environments involve strict constraints on the testing costs that a learner can use. To operate in such environments, the classifier needs to do its best without exceeding the bounds. Therefore, classifiers that focus on reducing the total cost may be inappropriate. Like the bound on the learning process, the bounds on classification costs can be preallocated and given to the learner (pre-contract classification), known to the classifier but not to the learner (contract classification), or determined during classification (interruptible classification). Note that unlike cost-sensitive learners that attempt to minimize the total cost of classification, bounded classification does not require $MC$ and $\rho^c$ to be provided in the same currency. This allows us to operate in environments where it is not possible to convert from one scale to another, for example time to money or money to risk.

Figure 2.1 gives an overview of the different possible scenarios for learning and classification. More formally, let $E$ be the set of training examples, $A$ be the set of attributes, $AC$ the attribute costs, and $M$ be the misclassification cost matrix, and $h$ the learned hypothesis. Further, let $\rho^l$ be the allocation for learning resources and $\rho^c$ denote the limit on classification resources for classifying a case.
e. For the scenarios where $\rho^l$ or $\rho^c$ are unknown, we assume two binary functions $f^l()$ and $f^c()$, which can be queried by the learner and the classifier respectively to find out whether their resources have been exhausted.

In this work we propose a novel framework that can be applied in the following constraint combinations:

- **Contract learning, unlimited classification (CLUC):** the learner is aware of $\rho^l$. The classifier is not limited in the resources it can use ($\rho^c = \infty$). If the environment does not involve test costs ($AC = 0$) and the misclassification costs are uniform ($M = M^U$, a matrix with zeros on the main diagonal and ones elsewhere) the objective of the resulted classifier is to maximize accuracy. Formally, $\text{CLUC}(E, A, 0, M^U, \rho^l, \infty) \rightarrow h(e)$. In the more general case, where test costs exist and misclassification costs may vary, the goal is to learn a classifier that minimizes the total cost of classification. Formally, $\text{CLUC}(E, A, AC, M, \rho^l, \infty) \rightarrow h(e)$.

- **Interruptible learning, unlimited classification (ILUC):** similar to CLUC but the learning time is not preallocated. Formally, $\text{ILUC}(E, A, 0, M^U, f^l(\cdot), \infty) \rightarrow h(e)$ in the cost-insensitive case, and $\text{ILUC}(E, A, AC, M, f^l(\cdot), \infty) \rightarrow h(e)$ in the cost-sensitive case.

- **Contract learning, pre-contract classification (CLPC):** the learner is aware of both $\rho^l$ and $\rho^c$. It produces a classifier that uses at most $\rho^c$ (the classifier does not benefit from any remaining surplus). Formally, $\text{CLPC}(E, A, AC, M, \rho^l, \rho^c) \rightarrow h(e)$.

- **Interruptible learning, pre-contract classification (ILPC):** similar to CLPC but the learning time is not preallocated. Formally, $\text{ILPC}(E, A, AC, M, f^l(\cdot), \rho^c) \rightarrow h(e)$.

- **Contract learning, contract classification (CLCC):** the learner is aware of $\rho^l$ but not of $\rho^c$, which becomes available right before classification. The learner must produce a classifier that can operate under different cost constraints. The produced classifier should be act according to the provided $\rho^c$. Formally, $\text{CLCC}(E, A, AC, M, \rho^l) \rightarrow h(e, \rho^c)$.

- **Interruptible learning, contract classification (ILCC):** similar to CLCC but the learning time is not predetermined. Formally, $\text{ILCC}(E, A, AC, M, f^l(\cdot)) \rightarrow h(e, \rho^c)$.

- **Contract learning, interruptible classification (CLIC):** the learner is aware of $\rho^l$. Neither the learner nor the classifier are provided with $\rho^c$. The learner must produce a classifier that utilizes the available resources until
interrupted and queried for a solution. Formally, CLIC\((E, A, AC, M, \rho^l) \rightarrow h(e, f^c())\).

- Interruptible learning, interruptible classification (ILIC): similar to CLIC but the learning time is unknown. Formally, ILIC\((E, A, AC, M, f^l(), \rho^c) \rightarrow h(e, f^c())\).

In Chapter 3 we present LSID3, a contract algorithm for learning classifiers that aim to maximize accuracy, without bounds on testing costs: LSID3\((E, A, 0, M^U, \rho^l, \infty) \rightarrow h(e)\). Next, we introduce IIDT, a general method to convert our contract learners into interruptible ones, and instantiate it for LSID3. Chapter 4 presents ACT, a contract algorithm for inducing trees that aim to minimize the total cost of classification without bounds on their classification budget: ACT\((E, A, AC, M, \rho^l, \infty) \rightarrow h(e)\). ACT can be converted into an interruptible learner using the IIDT procedure. In Chapter 5 we introduce TATA, a framework for learning resource-bounded classifiers. TATA can operate when the testing budget is known to the learner \((\text{PRE-CONTRACT-TATA}(E, A, AC, M, \rho^l, \rho^c) \rightarrow h(e))\), when it is known to the classifier but not to the learner \((\text{CONTRACT-TATA}(E, A, AC, M, \rho^l) \rightarrow h(e, \rho^c))\), and when it is not predetermined \((\text{INTERRUPTIBLE-TATA}(E, A, AC, M, \rho^l) \rightarrow h(e, f^c())\)). We also explain how to adapt these algorithms for interruptible learning scenarios.
Chapter 3

Contract Anytime Learning of Accurate Trees

Assume that a medical center has decided to use medical records of previous patients in order to build an automatic diagnostic system for a particular disease. The center applies the C4.5 algorithm on thousands of records, and after few seconds receives a decision tree. During the coming months, or even years, the same induced decision tree will be used to predict whether patients have or do not have the disease. Obviously, the medical center is willing to wait much longer to obtain a better tree—either more accurate or more comprehensible.

Consider also a planning agent that has to learn a decision tree from a given set of examples, while the time at which the model will be needed by the agent is not known in advance. In this case, the agent would like the learning procedure to learn the best tree it can until it is interrupted and queried for a solution.

In both of the above scenarios, the learning algorithm is expected to exploit additional time allocation to produce a better tree. In the first case, the additional time is allocated in advance. In the second, it is not. Similar resource-bounded reasoning situations may occur in many real-life applications such as game playing, planning, stock trading and e-mail filtering. In this Chapter, we introduce a framework for exploiting extra time, preallocated or not, in order to learn smaller and more accurate trees (Esmeir & Markovitch, 2007b).

3.1 Top-down Induction of Decision Trees

TDIDT (top-down induction of decision trees) methods start from the entire set of training examples, partition it into subsets by testing the value of an attribute, and then recursively call the induction algorithm for each subset. Figure 3.1 formalizes the basic algorithm for TDIDT. We focus first on consistent trees for
Figure 3.1: Procedure for top-down induction of decision trees. $E$ stands for the set of examples and $A$ stands for the set of attributes.

which the stopping criterion for the top-down recursion is when all the examples have the same class label. Later, we consider pruning, which allows simpler trees at the cost of possible inconsistency with the training data (Breiman et al., 1984; Quinlan, 1993).

In this work we propose investing more time resources for making better split decisions. We first examine the limitations of greedy tree learners. Then, we discuss small size as a desired property of the tree to be learned and describe an anytime algorithm that uses sampling methods to obtain smaller trees.

### 3.2 Limitation of Greedy Learners

The majority of existing methods for decision tree induction build a tree top-down and use local measures in an attempt to produce small trees, which, by Occam’s Razor (Blumer et al., 1987), should have better predictive power. The well-known C4.5 algorithm (Quinlan, 1993) uses the gain ratio as a heuristic for predicting which attribute will yield a smaller tree. Several other alternative local greedy measures have been developed, among which are ID3’s information gain, Gini index (Breiman et al., 1984), and chi-square (Mingers, 1989). Mingers (1989) reports an empirical comparison of several measures, and concludes that the predictive accuracy of the induced trees is not sensitive to the choice of split measure and even random splits do not significantly decrease accuracy. Buntine and Niblett (1992) present additional results on further domains and conclude...
Table 3.1: Possible training set for Learning the 2-XOR concept $a_1 \oplus a_2$, where $a_3$ and $a_4$ are irrelevant

<table>
<thead>
<tr>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>+</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>+</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>+</td>
</tr>
</tbody>
</table>

Figure 3.2: The resulted tree if ID3 is invoked on the training set from Table 3.1

that while random splitting leads to inferior trees, the information gain and Gini index measures are statistically indistinguishable.

The top-down methodology has the advantage of evaluating a potential attribute for a split in the context of the attributes associated with the nodes above it. The local greedy measures, however, consider each of the remaining attributes independently, ignoring potential interaction between different attributes (Mingers, 1989; Kononenko et al., 1997; Kim & Loh, 2001). We refer to the family of learning tasks where the utility of a set of attributes cannot be recognized by examining only subsets of it as tasks with a strong interdependency. Jakulin and Bratko (2003) studied the detection of attribute interactions in machine learning and proposed using the interaction gain to measure attribute interdependencies.

When learning a problem with a strong interdependency, greedy measures can lead to a choice of non-optimal splits. To illustrate the above, let us consider the 2-XOR problem $a_1 \oplus a_2$ with two additional irrelevant attributes, $a_3$ and $a_4$. 

21
Assume that the set of examples is as listed in Table 3.1. We observe that the information gain the irrelevant attribute $a_4$ yields is the highest:

$$0.13 = \text{gain}(a_4) > \text{gain}(a_1) = \text{gain}(a_2) = \text{gain}(a_3) = 0.02,$$

and hence ID3 would choose attribute $a_4$ first. Figure 3.2 gives the decision tree as produced by ID3. Any positive instance with value 0 for $a_4$ would be misclassified by this decision tree. In the general case of parity concepts, the information gain measure is unable to distinguish between the relevant and irrelevant attributes because neither has a positive gain. Consequently, the learner will grow an over-complicated tree with splits on irrelevant variables that come either in addition to or instead of the desired splits.

### 3.3 Tree-size as a Preference Bias

The hypothesis space of TDIDT is huge and a major question is what strategy should be followed to direct the search. In other words, we need to decide what our preference bias Mitchell (1997, chap. 3) will be. This preference bias will be expressed in the \textsc{Choose-Attribute} procedure that determines which tree is explored next.

Ultimately, we would like to follow a policy that maximizes the accuracy of the tree on unseen examples. However, since these examples are not available, a heuristic should be used. Motivated by Occam’s Razor, a widely adopted approach is to prefer smaller trees. The utility of this principle to machine learning algorithms has been the subject of a heated debate. Several studies attempted to justify Occam’s razor with theoretical and empirical arguments (Blumer et al., 1987; Quinlan & Rivest, 1989; Fayyad & Irani, 1990). But a number of recent works have questioned the utility of Occam’s razor, and provided theoretical and experimental evidence against it.

Quinlan and Cameron-Jones (1995) provided empirical evidence that over-searching might result in less accurate rules. Experimental results with several UCI datasets indicate that the complexity of the produced theories does not correlate well with their accuracy, a finding that is inconsistent with Occam’s Razor. Schaffer (1994) proved that no learning bias can outperform another bias over the space of all possible learning tasks. This looks like theoretical evidence against Occam’s razor. Rao, Gordon, and Spears (1995), however, argued against the applicability of this result to real-world problems by questioning the validity of its basic assumption about the uniform distribution of possible learning tasks. Webb (1996) presented C4.5X, an extension to C4.5 that uses similarity considerations to further specialize consistent leaves. Webb reported an empirical evaluation which shows that C4.5X has a slight advantage in a few domains and argued that these results discredit Occam’s thesis.
Murphy and Pazzani (1994) reported a set of experiments in which all the possible consistent decision trees were produced and showed that, for several tasks, the smallest consistent decision tree had higher predictive error than slightly larger trees. However, when the authors compared the likelihood of better generalization for smaller vs. more complex trees, they concluded that simpler hypotheses should be preferred when no further knowledge of the target concept is available. The small number of training examples relative to the size of the tree that perfectly describes the concept might explain why, in these cases, the smallest tree did not generalize best. Another reason could be that only small spaces of decision trees were explored. To verify these explanations, we use a similar experimental setup where the datasets have larger training sets and attribute vectors of higher dimensionality. Because the number of all possible consistent trees is huge, we use a Random Tree Generator (RTG) to sample the space of trees obtainable by TDIDT algorithms. RTG builds a tree top-down and chooses the splitting attribute at random.

We report the results for three datasets: XOR-5, Tic-tac-toe, and Zoo (See Appendix A for detailed descriptions of these datasets). For each dataset, the examples were partitioned into a training set (90%) and a testing set (10%), and RTG was used to generate a sample of 10 million consistent decision trees. The behavior in the three datasets is similar: the accuracy monotonically decreases with the increase in the size of the trees (number of leaves), confirming the utility of Occam’s Razor. In Appendix B we report further experiments with a variety of datasets, which indicate that the inverse correlation between size and accuracy is statistically significant (Esmeir & Markovitch, 2007c).

It is important to note that smaller trees have several advantages aside from their probable greater accuracy, such as greater statistical evidence at the leaves, better comprehensibility, lower storage costs, and faster classification (in terms of total attribute evaluations).

Motivated by the above discussion, our goal is to find the smallest consistent tree. In TDIDT, the learner has to choose a split at each node, given a set of examples $E$ that reach the node and a set of unused attributes $A$. For each attribute $a$, let $T_{\text{min}}(a, A, E)$ be the smallest tree rooted at $a$ that is consistent with $E$ and uses attributes from $A - \{a\}$ for the internal splits. Given two candidate attributes $a_1$ and $a_2$, we obviously prefer the attribute whose associated $T_{\text{min}}(a)$ is the smaller. For a set of attributes $A$, we define $\tilde{A}$ to be the set of attributes whose associated $T_{\text{min}}(a)$ is the smaller. Formally, $\tilde{A} = \arg \min_{a \in A} T_{\text{min}}(a)$. We say that a splitting attribute $a$ is optimal if $a \in \tilde{A}$. Observe that if the learner makes an optimal decision at each node, then the final tree is necessarily globally optimal.
Figure 3.3: Procedures for computing entropy $k$ and gain $k$ for attribute $a$

3.4 Fixed-depth Lookahead

One possible approach for improving greedy TDIDT algorithms is to lookahead in order to examine the effect of a split deeper in the tree. ID3 uses entropy to test the effect of using an attribute one level below the current node. This can be extended to allow measuring the entropy at any depth $k$ below the current node. This approach was the basis for the IDX algorithm (Norton, 1989). The recursive definition minimizes the $k - 1$ entropy for each child and computes their weighted average. Figure 3.3 describes an algorithm for computing entropy $k$ and its associated gain $k$. Note that the gain computed by ID3 is equivalent to gain $k$ for $k = 1$. We refer to this lookahead-based variation of ID3 as ID3-k. At each tree node, ID3-k chooses the attribute that maximizes gain $k$.

ID3-k can be viewed as a contract anytime algorithm parameterized by $k$. However, despite its ability to exploit additional resources when available, the anytime behavior of ID3-k is problematic. The run time of ID3-k grows exponentially as $k$ increases. As a result, the gap between the points of time at which the resulting tree can improve grows wider, limiting the algorithm’s flexibility. Furthermore, it is quite possible that looking ahead to depth $k$ will not be sufficient to find an optimal split. Entropy $k$ measures the weighted average of the entropy

---

1If two attributes yield the same decrease in entropy, we prefer the one whose associated lookahead tree is shallower.

2For example, in the binary case, ID3-k explores $\prod_{i=0}^{k-1} (n - i) 2^i$ combinations of attributes.
in depth \( k \) but does not give a direct estimation of the resulting tree size. Thus, when \( k < |A| \), this heuristic is more informed than entropy, but can still produce misleading results. In most cases we do not know in advance what value of \( k \) would be sufficient for correctly learning the target concept. Invoking exhaustive lookahead, i.e., lookahead to depth \( k = |A| \), will obviously lead to optimal splits, but its computational costs are prohibitively high. In the following section, we propose an alternative approach for evaluating attributes that overcomes the above-mentioned drawbacks of ID3-k.

3.5 A Contract Algorithm for Learning Decision trees

Motivated by the advantages of smaller decision trees, we introduce a novel algorithm that, given an attribute \( a \), evaluates it by estimating the size of the minimal tree under it. The estimation is based on Monte-Carlo sampling of the space of consistent trees rooted at \( a \). We estimate the minimum by the size of the smallest tree in the sample. The number of trees in the sample depends on the available resources, where the quality of the estimation is expected to improve with the increased sample size.

One way to sample the space of trees is to repeatedly produce random consistent trees using the RTG procedure. Since the space of consistent decision trees is large, such a sample might be a poor representative and the resulting estimation inaccurate. We propose an alternative sampling approach that produces trees of smaller expected size. Such a sample is likely to lead to a better estimation of the minimum.

Our approach is based on a new tree generation algorithm that we designed, called Stochastic ID3 (SID3). Using this algorithm repeatedly allows the space of “good” trees to be sampled semi-randomly. In SID3, rather than choose an attribute that maximizes the information gain, we choose the splitting attribute randomly. The likelihood that an attribute will be chosen is proportional to its information gain.\(^3\) However, if there are attributes that decrease the entropy to zero, then one of them is picked randomly. The attribute selection procedure of SID3 is listed in Figure 3.4.

To show that SID3 is a better sampler than RTG, we repeated our sampling experiment (Section 3.3) using SID3 as the sample generator. Figure 3.5 compares the frequency curves of RTG and SID3. Relative to RTG, the graph for SID3 is shifted to the left, indicating that the SID3 trees are clearly smaller.

\(^3\)We make sure that attributes with gain of zero will have a positive probability of being selected.
Next, we compared the average minimum found for samples of different sizes. Figure 3.6 shows the results. For the three datasets, the minimal size found by SID3 is strictly smaller than the value found by RTG. Given the same budget of time, RTG produced, on average, samples that are twice as large as that of SID3. However, even when the results are normalized (dashed line), SID3 is still superior.

Having decided about the sampler, we are ready to describe our proposed contract algorithm, Lookahead-by-Stochastic-ID3 (LSID3). In LSID3, each candidate split is evaluated by the estimated size of the subtree under it. To estimate the size under an attribute \( a \), LSID3 partitions the set of examples according to the values \( a \) can take and repeatedly invokes SID3 to sample the space of trees consistent with each subset. Summing up the minimal tree size for each subset gives an estimation of the minimal total tree size under \( a \).

LSID3 is a contract algorithm parameterized by \( r \), the sample size. LSID3 with \( r = 0 \) is defined to choose the splitting attribute using the standard ID3 selection method. Figure 3.7 illustrates the choice of splitting attributes as made by LSID3. In the given example, SID3 is called twice for each subset and the evaluation of the examined attribute \( a \) is the sum of the two minima: \( \min(4,3) + \min(2,6) = 5 \). The method for choosing a splitting attribute is formalized in Figure 3.8.

To analyze the time complexity of LSID3, let \( m \) be the total number of examples and \( n \) be the total number of attributes. For a given node \( y \), we denote by \( n_y \) the number of candidate attributes at \( y \), and by \( m_y \) the number of examples that reach \( y \). ID3, at each node \( y \), calculates gain for \( n_y \) attributes using \( m_y \) examples, i.e., the complexity of choosing an attribute is \( O(n_y \cdot m_y) \). At level \( i \) of the tree, the total number of examples is bounded by \( m \) and the number of
attributes to consider is $n - i$. Thus, it takes $O(m \cdot (n - i))$ to find the splits for all nodes in level $i$. In the worst case the tree will be of depth $n$ and hence the total runtime complexity of ID3 will be $O(m \cdot n^2)$ (Utgoff, 1989). Shavlik, Mooney, and G. (1991) reported for ID3 an empirically based average-case complexity of $O(m \cdot n)$.

It is easy to see that the complexity of SID3 is similar to that of ID3. LSID3($r$) invokes SID3 $r$ times for each candidate split. Recalling the above analysis for

---

**Figure 3.5:** Tree-size frequency curves for the XOR-5 (left), Tic-tac-toe, and Zoo (right) datasets
Figure 3.6: The minimum tree size as estimated by SID3 and RTG as a function of the sample size. The datasets are XOR5 (left), Tic-tac-toe and Zoo (right). The dashed line describes the results for SID3 normalized by time.

In the average case, we replace the runtime of SID3 by $O(m \cdot (n - i))$, and hence

$$\sum_{i=0}^{n-1} r \cdot (n - i) \cdot O(m \cdot (n - i)^2) = \sum_{i=1}^{n} O(r \cdot m \cdot i^3) = O(r \cdot m \cdot n^4). \quad (3.1)$$
Figure 3.7: Attribute evaluation using LSID3. The estimated subtree size for $a$ is $\min(4, 3) + \min(2, 6) = 5$.

**Procedure** LSID3-Choose-Attribute($E, A, r$)

1. If $r = 0$
   - Return ID3-Choose-Attribute($E, A$)
2. Foreach $a \in A$
   - Foreach $v_i \in \text{domain}(a)$
     - $E_i \leftarrow \{ e \in E \mid a(e) = v_i \}$
     - $\min_i \leftarrow \infty$
   - Repeat $r$ times
     - $\min_i \leftarrow \min(\min_i, |\text{SID3}(E_i, A - \{a\})|)$
     - $\text{total}_a \leftarrow \sum_{i=1}^{\text{domain}(a)} \min_i$
   - Return $a$ for which $\text{total}_a$ is minimal

Figure 3.8: Attribute selection in LSID3

we have

$$\sum_{i=0}^{n-1} r \cdot (n - i) \cdot O(m \cdot (n - i)) = \sum_{i=1}^{n} O(r \cdot m \cdot i^2) = O(r \cdot m \cdot n^3). \quad (3.2)$$

According to the above analysis, the run-time of LSID3 grows at most linearly with $r$ (under the assumption that increasing $r$ does not result in larger trees).
We expect that increasing $r$ will improve the classifier’s quality. To understand why, let us examine the expected behavior of the LSID3 algorithm on the 2-XOR problem used in Figure 3.2. LSID3 with $r = 0$, which is equivalent to ID3, prefers to split on the irrelevant attribute $a_4$. LSID3 with $r \geq 1$ evaluates each attribute $a$ by calling SID3 to estimate the size of the trees rooted at $a$. The attribute with the smallest estimation will be selected. The minimal size for trees rooted at $a_4$ is 6 and for trees rooted at $a_3$ is 7. For $a_1$ and $a_2$, SID3 would necessarily produce trees of size 4.\footnote{Neither $a_3$ nor $a_4$ can be selected at the 2nd level since the remaining relevant attribute reduces the entropy to zero.} Thus, LSID3, even with $r = 1$, will succeed in learning the right tree. For more complicated problems such as 3-XOR, the space of SID3 trees under the relevant attributes includes trees other than the smallest. In that case, the larger the sample is, the higher the likelihood is that the smallest tree will be drawn.

3.5.1 Evaluating Continuous Attributes by Sampling

When attributes have a continuous domain, the decision tree learner needs to discretize their values in order to define splitting tests. One common approach is to partition the range of values an attribute can take into bins, prior to the process of induction. Dougherty, Kohavi, and Sahami (1995) review and compare several different strategies for pre-discretization. Using such a strategy, our lookahead algorithm can operate unchanged. Pre-discretization, however, may be harmful in many domains because the correct partitioning may differ from one context (node) to another.

An alternative approach is to determine dynamically at each node a threshold that splits the examples into two subsets. The number of different possible thresholds is at most $|E| - 1$, and thus the number of candidate tests to consider for each continuous attribute increases to $O(|E|)$. Such an increase in complexity may be insignificant for greedy algorithms, where evaluating each split requires a cheap computation (like information gain in ID3). In LSID3, however, the dynamic method may present a significant problem because evaluating the usefulness of each candidate split is very costly.

The desired method would reduce the number of splits to evaluate while avoiding the disadvantages of pre-discretization. We devised a method for controlling the resources devoted to evaluating a continuous attribute by Monte Carlo sampling of the space of splits. Initially, we evaluate each possible splitting test by the information gain it yields. This can be done in linear time $O(|E|)$. Next, we choose a sample of the candidate splitting tests where the probability that a test will be chosen is proportional to its information gain. Each candidate in the sample is evaluated by a single invocation of SID3. However, since we sample with
Procedure MC-Evaluate-Continuous-Attribute\((E, A, a, p)\)

Foreach \(e \in E\)
\[
\begin{align*}
E_{\leq} & \leftarrow \{e' \mid a(e') \leq a(e)\} \\
E_{> } & \leftarrow \{e' \mid a(e') > a(e)\}
\end{align*}
\]

\[
\begin{align*}
\text{entropy}_e & \leftarrow \frac{|E_{\leq}|}{|E|} \text{entropy}(E_{\leq}) + \frac{|E_{> }|}{|E|} \text{entropy}(E_{> }) \\
gain_e & \leftarrow \text{entropy}(E) - \text{entropy}_e
\end{align*}
\]

\[
sample\text{Size} \leftarrow p \cdot |E|
\]

\[
min \leftarrow \infty
\]

Repeat \(\text{sampleSize}\) times
\[
e \leftarrow \text{Choose an example at random from } E; \text{ for each example } e,
\]
\[
\begin{align*}
E_{\leq} & \leftarrow \{e' \mid a(e') \leq a(e)\} \\
E_{> } & \leftarrow \{e' \mid a(e') > a(e)\}
\end{align*}
\]

\[
total_e \leftarrow |\text{SID3}(E_{\leq}, A)| + |\text{SID3}(E_{> }, A)|
\]

If \(\text{total}_e < min\)
\[
\begin{align*}
min & \leftarrow \text{total}_e \\
\text{bestTest} & \leftarrow a(e)
\end{align*}
\]

Return \(\langle \min, \text{bestTest} \rangle\)

Figure 3.9: Monte Carlo evaluation of continuous attributes using SDI3

repetitions, candidates with high information gain may have multiple instances in the sample, resulting in several SID3 invocations.

The resources allocated for evaluating a continuous attribute using the above method are determined by the sample size. If our goal is to devote a similar amount of resources to all attributes, then we can use \(r\) as the size of the sample. Such an approach, however, does not take into account the size of the population to be sampled. We use a simple alternative approach of taking samples with size \(p \cdot |E|\) where \(p\) is a predetermined parameter, set by the user according to the available resources.\(^5\) Note that \(p\) can be greater than one because we sample with repetition.

We name this variant of the LSID3 algorithm LSID3-MC, and formalize it in Figure 3.9. LSID3-MC can serve as an anytime algorithm that is parameterized by \(p\).

\(^5\)Similarly to the parameter \(r\), a mapping from the available time to \(p\) is needed (see Section 3.5.4).
3.5.2 Multiway Vs. Binary Splits

The TDIDT procedure, as described in Figure 3.1, partitions the current set of examples into subsets according to the different values the splitting attribute can take. LSID3, which is a TDIDT algorithm, inherits this property. While multiway splits can yield more comprehensible trees (Kim & Loh, 2001), they might fragment the data too quickly, decreasing its learnability (Hastie et al., 2001, chap. 9).

To avoid the fragmentation problem, several learners take a different approach and consider only binary splits, e.g., CART and QUEST (Quick, Unbiased and Efficient Statistical Tree, Loh & Shih, 1997). One way to obtain binary splits when an attribute $a$ is categorical is to partition the examples using a single value $v$: all the examples with $a = v$ are bagged into one group and all the other examples are bagged into a second group. Therefore, for each attribute $a$ there are $|\text{domain}(a)|$ candidate splits. While efficient, this strategy does not allow different values to be combined, and therefore might over-fragment the data. CART overcomes this by searching over all non-empty subsets of $\text{domain}(a)$. We denote by BLSID3 a variant of LSID3 that yields binary trees and adopts the CART style exhaustive search. Observe that the complexity of evaluating an attribute in BLSID3 is $O(2^{\text{domain}(a)})$ times that in LSID3.

A second problem with LSID3’s multiway splits is their bias in attribute selection. Consider for example a learning task with four attributes: three binary attributes $a_1, a_2, a_3$, and a categorial attribute $a_4$ whose domain is $\{A, C, G, T\}$. Assume that the concept is $a_1 \oplus a_2$ and that $a_4$ was created from the values of $a_2$ by randomly and uniformly mapping every 0 into $A$ or $C$ and every 1 into $G$ or $T$. Theoretically, $a_2$ and $a_4$ are equally important to the class. LSID3, however, would prefer to split $a_2$ due to its smaller associated total tree size. BLSID3 solves this problem by considering the binary test $\in \{A, C\}$.

Loh and Shih (1997) pointed out that exhaustive search tends to prefer features with larger domains and proposed QUEST, which overcomes this problem by dealing with attribute selection and with test selection separately: first an attribute is chosen and only then is the best test picked. This bias, and its reduction by QUEST, were studied in the context of greedy attribute evaluation, but can potentially be applied to our non-greedy approach.

3.5.3 Pruning the LSID3 Trees

Pruning tackles the problem of how to avoid overfitting the data, mainly in the presence of classification noise. It is not designed, however, to recover from

\footnote{Note that this bias is opposite to that of favoring attributes with a large domain (Quinlan, 1993, chap. 2).}
wrong greedy decisions. When there are sufficient examples, wrong splits do not significantly diminish accuracy, but result in less comprehensible trees and costlier classification. When there are fewer examples the problem is intensified because the accuracy is adversely affected. In our proposed anytime approach, this problem is tackled by allotting additional time for reaching trees unexplored by greedy algorithms with pruning. Therefore, we view pruning as orthogonal to lookahead and suggest considering it as an additional phase in order to avoid overfitting.

Pruning can be applied in two different ways. First, it can be applied as a second phase after each LSID3 final tree has been built. We denote this extension by \( \text{LSID3-p} \) (and \( B\text{LSID3-p} \) for binary splits). Second, it is worthwhile to examine pruning of the lookahead trees. For this purpose, a post-pruning phase will be applied to the trees that were generated by SID3 to form the lookahead sample, and the size of the pruned trees will be considered. The expected contribution of pruning in this case is questionable since it has a similar effect on the entire lookahead sample.

Previous comparative studies did not find a single pruning method that is generally the best and conclude that different pruning techniques behave similarly (Esposito, Malerba, & Semeraro, 1997; Oates & Jensen, 1997). Therefore, in our experiments we adopt the same pruning method as in C4.5, namely error-based pruning (EPB), and examine post-pruning of the final trees as well as pruning of the lookahead trees.

Figure 3.10 describes the spaces of decision trees obtainable by LSID3. Let \( A \) be a set of attributes and \( E \) be a set of examples. The space of decision trees over \( A \) can be partitioned into two: the space of trees consistent with \( E \) and the space of trees inconsistent with \( E \). The space of consistent trees includes as a subspace the trees inducible by TDIDT.\(^7\) The trees that ID3 can produce are a subspace of the TDIDT space. Pruning extends the space of ID3 trees to the inconsistent space. LSID3 can reach trees that are obtainable by TDIDT but not by the greedy methods and their extensions. LSID3-p expands the search space to trees that do not fit the training data perfectly, so that noisy training sets can be handled.

### 3.5.4 Mapping Contract Time to Sample Size

LSID3 is parameterized by \( r \), the number of times SID3 is invoked for each candidate. In real-life applications, however, the user supplies the system with

---

\(^7\)TDIDT is a strict subspace because it cannot induce trees that: (1) contain a subtree with all leaves marked with the same class, and (2) include an internal node that has no associated training examples.
the time she is willing to allocate for the learning process. Hence, a mapping from $\rho'$, the contract learning allocation, to $r$ is needed.

One way to obtain such a mapping is to run the algorithm with $r = 1$. Since the runtime grows linearly with $r$, one can now, given the remaining time and the runtime for $r = 1$, easily find the corresponding value of $r$. Then, the whole induction process is restarted with the measured $r$ as the contract parameter. The problem with this method is that, in many cases, running the algorithm with $r = 1$ consumes many of the available resources while not contributing to the final classifier.

Another possibility is mapping time to $r$ on the basis of previous experience on the same domain or a domain with similar properties. When no such knowledge exists, one can derive the runtime from the time complexity analysis. However, in practice, the runtime is highly dependent on the complexity of the domain and the number of splits that in fact take place. Hence it is very difficult to predict.

3.6 Interruptible Induction of Decision Trees

As a contract algorithm, LSID3 assumes that the contract parameter is known in advance. In many real-life cases, however, either the contract time is unknown or mapping the available resources to the appropriate contract parameter is not
**Procedure** SEQUENCED-LSID3($E, A$)
\[
T \leftarrow \text{ID3} (E, A) \\
r \leftarrow 1 \\
\text{While not-interrupted} \\
\quad T \leftarrow \text{LSID3} (E, A, r) \\
\quad r \leftarrow 2 \cdot r \\
\text{Return } T
\]

**Figure 3.11:** Converting LSID3 to an interruptible algorithm by sequencing

possible. To handle such cases, we need to devise an interruptible anytime algorithm.

We start with a method that converts LSID3 to an interruptible algorithm using the general sequencing technique described by Russell and Zilberstein (1996). This conversion, however, uses the contract algorithm as a black box and hence cannot take into account specific aspects of decision tree induction. Next, we present a novel iterative improvement algorithm, *Interruptible Induction of Decision Trees* (IIDT), that repeatedly replaces subtrees of the current hypothesis with subtrees generated with higher resource allocation and thus expected to be better.

### 3.6.1 Interruptible Learning by Sequencing

By definition, every interruptible algorithm can serve as a contract algorithm because one can stop the interruptible algorithm when all the resources have been consumed. Russell and Zilberstein (1996) showed that the other direction works as well: any contract algorithm $A$ can be converted into an interruptible algorithm $B$ with a constant penalty. $B$ is constructed by running $A$ repeatedly with exponentially increasing time limits. This general approach can be used to convert LSID3 into an interruptible algorithm. LSID3 gets its contract time in terms of $r$, the sample size. When $r = 0$, LSID3 is defined to be identical to ID3. Therefore, we first call LSID3 with $r = 0$ and then continue with exponentially increasing values of $r$, starting from $r = 1$. Figure 3.11 formalizes the resulting algorithm. It can be shown that the above sequence of runtimes is optimal when the different runs are scheduled on a single processor (Russell & Zilberstein, 1996).

One problem with the sequencing approach is the exponential growth of the gaps between the times at which an improved result can be obtained. The reason for this problem is the generality of the sequencing approach, which views the contract algorithm as a black box. Thus, in the case of LSID3, at each iteration the whole decision tree is rebuilt. In addition, the minimal value that can be used
for $\tau$ is the runtime of LSID3($r = 1$). In many cases we would like to stop the algorithm earlier. When we do, the sequencing approach will return the initial greedy tree. Our interruptible anytime framework, called IIDT, overcomes these problems by iteratively improving the tree rather than trying to rebuild it.

### 3.6.2 Iterative Improvement of Decision Trees

Iterative improvement approaches that start with an initial solution and repeatedly modify it have been successfully applied to many AI domains, such as CSP, timetabling, and combinatorial problems (Minton, Johnston, Philips, & Laird, 1992; Johnson, Aragon, McGeoch, & Schevon, 1991; Schaerf, 1996). The key idea behind iterative improvement techniques is to choose an element of the current suboptimal solution and improve it by local repairs. IIDT adopts the above approach for decision tree learning and iteratively revises subtrees. It can serve as an interruptible anytime algorithm because, when interrupted, it can immediately return the currently best solution available.

As in LSID3, IIDT exploits additional resources in an attempt to produce better decision trees. The principal difference between the algorithms is that LSID3 uses the available resources to induce a decision tree top-down, where each decision made at a node is final and does not change. IIDT, however, is not allocated resources in advance and uses extra time resources to modify split decisions repeatedly.

IIDT receives a set of examples and a set of attributes. It first performs a quick construction of an initial decision tree by calling ID3. Then it iteratively attempts to improve the current tree by choosing a node and rebuilding its subtree with more resources than those used previously. If the newly induced subtree is better, it will replace the existing one. IIDT is formalized in Figure 3.12.

Figure 3.13 illustrates how IIDT works. The target concept is $a_1 \oplus a_2$, with two additional irrelevant attributes, $a_3$ and $a_4$. The leftmost tree was constructed using ID3. In the first iteration, the subtree rooted at the bolded node is selected for improvement and replaced by a smaller tree (surrounded by a dashed line). Next, the root is selected for improvement and the whole tree is replaced by a tree that perfectly describes the concept. While in this particular example IIDT first chooses to rebuild a subtree at depth 2 and then at depth 1, it considers all subtrees, regardless of their level.

IIDT is designed as a general framework for interruptible learning of decision trees. It can use different approaches for choosing which node to improve, for allocating resources for an improvement iteration, for rebuilding a subtree, and for deciding whether an alternative subtree is better.

After the subtree to be rebuilt is chosen and the resources for a reconstruction iteration allocated, the problem becomes a task for a contract algorithm. A good
Procedure IIDT\((E, A)\)
\[
T \leftarrow \text{GREEDY-BUILD-TREE}(E, A)
\]
While not-interrupted
\[
\text{node} \leftarrow \text{CHOOSE-NODE}(T, E, A)
\]
\[
t \leftarrow \text{subtree of } T \text{ rooted at } \text{node}
\]
\[
r \leftarrow \text{NEXT-R}(\text{node})
\]
\[
E_{\text{node}} \leftarrow \text{EXAMPLES-AT}(\text{node}, T, E)
\]
\[
A_{\text{node}} \leftarrow \text{ATTRIBUTES-AT}(\text{node}, T, A)
\]
\[
t' \leftarrow \text{REBUILD-TREE}(E_{\text{node}}, A_{\text{node}}, r)
\]
If BETTER\(t', t)\) replace \(t\) with \(t'\)

Return \(T\)

Procedure EXAMPLES-AT\((\text{node}, T, E)\)
Return \(\{e \in E \mid e \text{ reaches } \text{node}\}\)

Procedure ATTRIBUTES-AT\((\text{node}, T, A)\)
Return \(\{a \in A \mid a \notin \text{ancestor of } \text{node}\} \cup\)
\[
\{a \in A \mid a \text{ is numeric}\}\]

Procedure NEXT-R\((\text{node})\)
If \(\text{LAST-R}(\text{node}) = 0\)
Return 1
Else
Return \(2 \cdot \text{LAST-R}(\text{node})\)

**Figure 3.12:** IIDT: framework for interruptible induction of decision trees

Candidate for such an algorithm is LSID3, which is expected to produce better subtrees when invoked with a higher resource allocation. In what follows we focus on the different components of IIDT and suggest a possible implementation that uses LSID3 for revising subtrees.

**Choosing a Subtree to Improve**

Intuitively, the next node we would like to improve is the one with the highest expected marginal utility, i.e., the one with the highest ratio between the expected benefit and the expected cost (Horvitz, 1990; Russell & Wefald, 1989). Estimating the expected gain and expected cost of rebuilding a subtree is a difficult problem. There is no apparent way to estimate the expected improvement in terms of either...
Figure 3.13: Iterative improvement of the decision tree produced for the 2-XOR concept $a_1 \oplus a_2$ with two additional irrelevant attributes, $a_3$ and $a_4$. The leftmost tree was constructed using ID3. In the first iteration the subtree rooted at the bolded node is selected for improvement and replaced by a smaller tree (surrounded by a dashed line). Next, the root is selected for improvement and the whole tree is replaced by a tree that perfectly describes the concept.
tree size or generalization accuracy. In addition, the resources to be consumed by LSID3 are difficult to predict precisely. We now show how to approximate these values, and how to incorporate these approximations into the node selection procedure.

**Resource Allocation.** The LSID3 algorithm receives its resource allocation in terms of \( r \), the number of samplings devoted to each attribute. Given a tree node \( y \), we can view the task of rebuilding the subtree below \( y \) as an independent task. Every time \( y \) is selected, we have to allocate resources for the reconstruction process. Following Russell and Zilberstein (1996), the optimal strategy in this case is to double the amount of resources at each iteration. Thus, if the resources allocated for the last attempted improvement of \( y \) were \( \text{Last-R}(y) \), the next allocation will be \( \text{Next-R}(y) = 2 \cdot \text{Last-R}(y) \).

**Expected Cost.** The expected cost can be approximated using the average time complexity of the contract algorithm used to rebuild subtrees. Following Equation 3.2, we estimate \( \text{Next-R}(y) \cdot m \cdot n^3 \) to be the expected runtime of LSID3 when rebuilding a node \( y \), where \( m \) is the number of examples that reach \( y \) and \( n \) is the number of attributes to consider. We observe that subtrees rooted in deeper levels are preferred because they have fewer examples and attributes to consider. Thus, their expected runtime is shorter. Furthermore, because each time allocation for a node doubles the previous one, nodes that have already been selected many times for improvement will have higher associated costs and are less likely to be chosen again.

**Expected benefit.** The whole framework of decision tree induction rests on the assumption that smaller consistent trees are better than larger ones. Therefore, the size of a subtree can serve as a measure for its quality. It is difficult, however, to estimate the size of the reconstructed subtree without actually building it. Therefore, we use instead an upper limit on the possible reduction in size. The minimal size possible for a decision tree is obtained when all examples are labeled with the same class. Such cases are easily recognized by the greedy ID3. Similarly, if a subtree were replaceable by another subtree of depth 1, ID3 (and LSID3) would have chosen the smaller subtree. Thus, the maximal reduction of the size of an existing subtree is to the size of a tree of depth 2. Assuming that the maximal number of values per attribute is \( b \), the maximal size of such a tree is \( b^2 \). Hence, an upper bound on the benefit from reconstructing an existing tree \( t \) is \( \text{SIZE}(t) - b^2 \). Ignoring the expected costs and considering only the expected benefits results in giving the highest score to the root node. This makes sense: assuming that we have infinite resources, we would attempt to improve the entire tree rather than parts of it.
Granularity. Considering the cost and benefit approximations described above, the selection procedure would prefer deep nodes (that are expected to have low costs) with large subtrees (that are expected to yield large benefits). When no such large subtrees exist, our algorithm may repeatedly attempt to improve smaller trees rooted at deep nodes because these trees have low associated costs. In the short term, this behavior would indeed be beneficial but can be harmful in the long term. This is because when the algorithm later improves subtrees in upper levels, the resources spent on deeper nodes will have been wasted. Had the algorithm first selected the upper level trees, this waste would have been avoided, but the time gaps between potential improvements would have increased.

To control the tradeoff between efficient resource use and anytime performance flexibility, we add a granularity parameter $0 \leq g \leq 1$. This parameter serves as a threshold for the minimal time allocation for an improvement phase. A node can be selected for improvement only if its normalized expected cost is above $g$. To compute the normalized expected cost, we divide the expected cost by the expected cost of the root node. Note that it is possible to have nodes with a cost that is higher than the cost of the root node, since the expected cost doubles the cost of the last improvement of the node. Therefore, the normalized expected cost can be higher than 1. Such nodes, however, will never be selected for improvement, because their expected benefit is necessarily lower than the expected benefit of the root node. Hence, when $g = 1$, IIDT is forced to choose the root node and its behavior becomes identical to that of the sequencing algorithm described in Section 3.6.1. Observe that IIDT does not determine $g$ but expects the user to provide this value according to her needs: more frequent small improvements or faster overall progress.

Figure 3.14 formalizes the procedure for choosing a node for reconstruction.
We refer to the above-described instantiation of IIDT that uses the tree size as a quality metric by \textit{IIDT-TS}. Figure 3.15 formalizes IIDT-TS.

\textbf{Evaluating a Subtree}

Although LSID3 is expected to produce better trees when allocated more resources, an improved result is not guaranteed. Thus, to avoid obtaining an induced tree of lower quality, we replace an existing subtree with a newly induced alternative only if the alternative is expected to improve the quality of the complete decision tree. Following Occam’s Razor, we measure the usefulness of a subtree by its size. Only if the reconstructed subtree is smaller does it replace an existing subtree. This guarantees that the size of the complete decision tree will decrease monotonically.

Another possible measure is the accuracy of the decision tree on a set-aside validation set of examples. In this case the training set is split into two subsets: a growing set and a validation set. Only if the accuracy on the validation set increases is the modification applied. This measure suffers from two drawbacks. The first is that putting aside a set of examples for validation results in a smaller set of training examples, making the learning process harder. The second is the bias towards overfitting the validation set, which might reduce the generalization
abilities of the tree. Several of our experiments, which we do not report here, confirmed that relying on the tree size results in better decision trees.

**Pruning the IIDT Trees**

Pruning tackles the problem of how to avoid overfitting the data, mainly in the presence of classification noise. Pruning, however, is not able to overcome wrong decisions made when learning hard concepts. Therefore, our anytime approach and pruning address different problems: whereas pruning attempts to simplify the induced trees to avoid overfitting, IIDT attempts to exploit additional time for better learning of hard concepts. We view pruning as orthogonal to lookahead and suggest considering it to avoid overfitting.

Pruning can be applied to the trees produced by IIDT in two different ways. The first is to post-prune a tree when it is returned to the user. In this case IIDT maintains consistent trees and only when interrupted, it applies a post-pruning phase. When resumed, IIDT continues to improve the unpruned tree. This solution requires the user to take into account the delay imposed by pruning. In the anytime setup, where much time is invested for growing the tree, this delay is usually negligible with respect to the growing time.

The second alternative we propose maintains (possibly) inconsistent trees by building a pruned initial tree (using C4.5), and by post-pruning each of the replacement trees (using LSID3-p). In this case the tree can be returned immediately, without any post-pruning delay.

Using the size of a tree as a measure for its quality is not applicable to inconsistent trees: one can easily create a tree of size 1 when consistency is not forced. One possible solution is to evaluate the quality of a subtree by its expected error. The expected error of a tree is measured by summing up the expected errors of its leaves, exactly as in C4.5’s error-based pruning. We denote this instantiation of IIDT by \textit{IIDT-EE}. In IIDT-EE, the expected benefit of rebuilding a subtree is measured by the maximal possible reduction in the expected error. A trivial lower bound for the expected error is the error if all examples were immediately split into subsets according to their label. Figure 3.16 lists IIDT-EE.

### 3.6.3 Incremental Learning: When New Examples Arrive

As an anytime algorithm, IIDT continuously invests resources in attempt to improve the induced tree. It is not designed, however, to exploit additional data that becomes available with time. One way to handle examples that are presented incrementally is to rerun the inducer when a new example arrives. The restart approach is very reasonable for greedy learners, where the time “wasted” by starting the induction process from scratch is minimal. This straightforward
conversion, however, is much more problematic in the case of anytime learners: it is not reasonable to discard the current hypothesis, which was produced using significant resources, just because a new example arrives. Greedy restructurings, as suggested in ID5R (Utgoff, 1989) and its successive ITI (Utgoff, Berkman, & Clouse, 1997), allow to quickly obtain the same greedy tree as in the batch setup. Applying such corrections to IIDT, however, will nullify its efforts to make better split decisions. To overcome these problems, we propose adapting IIDT to handle incremental learning tasks.

We assume that a stream of examples is being received and stored and that the learner can process the accumulated examples at any time. The most natural point to do so is upon the termination of each improvement iteration. In Figure 3.17 we generalize IIDT for incremental tasks. The incremental version of IIDT, called \textit{I-IIDT}, is identical to the non-incremental IIDT listed in Figure 3.12 with one exception: at the end of each iteration, the algorithm takes the new examples and use them to update the tree. We propose two methods for tree update—one for each instantiation of IIDT.
\[
\text{Procedure TS-Add-Example}(T, E, A, e) \\
\text{\hspace{1em}} l \leftarrow \text{Leaf-Of}(e) \\
\text{\hspace{1em}} T_i \leftarrow \text{ID3}(T, \text{Examples-At}(l, T, E), \text{ATT}(l)) \\
\text{\hspace{1em}} \text{replace } l \text{ with } T_i \\
\text{\hspace{1em}} \Delta \text{size} \leftarrow \text{Tree-Size}(T_i) - 1 \\
\text{\hspace{1em}} \text{Foreach } p \text{ ancestor of } T_i \\
\text{\hspace{2em}} \text{Tree-Size}(p) \leftarrow \text{Tree-Size}(p) + \Delta \text{size}
\]

Figure 3.18: Incorporating a new example by I-IIDT-TS

Our first method for tree update is intended for I-IIDT-TS. Recall that I-IIDT retains for each node \( y \) the set of examples belongs to it, \( E_y \). Given a new example \( e \), and the current tree \( T \), we search for the leaf \( l \) that \( e \) belongs to. If the label of \( l \) differs from that of \( e \), we replace \( l \) by a consistent tree, built from \( E_l \cup \{e\} \). This tree can be quickly obtained by invoking ID3. Such an update, that replaces a leaf \( l \) with a subtree, increases the size of the subtrees under all the nodes along the path from the root to \( l \). Consequently, the expected benefit from rebuilding these subtrees increases. On the other hand, the expected cost, which depends on the number of examples, will increase as well. The selection mechanism of I-IIDT which prefers subtrees with high expected utility will take both changes into consideration. We refer to this variant as I-IIDT-TS. Figure 3.18 lists I-IIDT-TS.

The second method is designed for I-IIDT-EE. With the possibility to maintain inconsistent trees, adding a new example to the existing tree becomes as simple as finding the relevant leaf and updating the expected error of this leaf and of its ancestors. Clearly, a new example that is consistent with the current tree reduces the expected error, while an inconsistent example increases it. This effect is the strongest in the leaf and diminishes for shallower subtrees. We refer to this variant as I-IIDT-EE. Figure 3.19 formalizes I-IIDT-EE.
3.7 Empirical Evaluation

A variety of experiments were conducted to test the performance and behavior of the proposed anytime algorithms. First we describe our experimental methodology and explain its motivation. We then present and discuss our results.

3.7.1 Experimental Methodology

We start our experimental evaluation by comparing our contract algorithm, given a fixed resource allocation, with the basic decision tree induction algorithms. We then compare the anytime behavior of our contract algorithm to that of fixed lookahead. Next we examine the anytime behavior of our interruptible algorithm. Finally, we compare its performance to several modern decision tree induction methods.

Following the recommendations of Bouckaert (2003), 10 runs of a 10-fold cross-validation experiment were conducted for each dataset and the reported results averaged over the 100 individual runs.\(^8\) For the Monks datasets, which were originally partitioned into a training set and a testing set, we report the results on the original partitions. Due to the stochastic nature of LSID3, the reported results in these cases are averaged over 10 different runs.

In order to evaluate the studied algorithms, we used 17 datasets taken from the UCI repository (Asuncion & Newman, 2007).\(^9\) Because greedy learners perform quite well on easy tasks, we looked for problems that hide hard concepts so the advantage of our proposed methods will be emphasized. The UCI repository, nevertheless, contains only few such tasks. Therefore, we added 7 artificial

---

\(^8\)An exception was the Connect-4 dataset, for which only one run of 10-fold CV was conducted because of its enormous size.

\(^9\)Due to the time-intensiveness of the experiments, we limited ourselves to 17 UCI problems.
Table 3.2: Characteristics of the datasets used

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Instances</th>
<th>Attributes</th>
<th>Max</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Nom. (bin.)</td>
<td>Num. domain</td>
<td></td>
</tr>
<tr>
<td>Automobile - Make</td>
<td>160</td>
<td>10 (4)</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>Automobile - Sym.</td>
<td>160</td>
<td>10 (4)</td>
<td>15</td>
<td>22</td>
</tr>
<tr>
<td>Balance Scale</td>
<td>625</td>
<td>4 (0)</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Breast Cancer</td>
<td>277</td>
<td>9 (3)</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>Connect-4</td>
<td>68557</td>
<td>42 (0)</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Corral</td>
<td>32</td>
<td>6 (6)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Glass</td>
<td>214</td>
<td>0 (0)</td>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>Iris</td>
<td>150</td>
<td>0 (0)</td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>Monks-1</td>
<td>124+432</td>
<td>6 (2)</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Monks-2</td>
<td>169+432</td>
<td>6 (2)</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Monks-3</td>
<td>122+432</td>
<td>6 (2)</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Mushroom</td>
<td>8124</td>
<td>22 (4)</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>Solar Flare</td>
<td>323</td>
<td>10 (5)</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Tic-Tac-Toe</td>
<td>958</td>
<td>9 (0)</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Voting</td>
<td>232</td>
<td>16 (16)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Wine</td>
<td>178</td>
<td>0 (0)</td>
<td>13</td>
<td>-</td>
</tr>
<tr>
<td>Zoo</td>
<td>101</td>
<td>16 (15)</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Numeric XOR 3D</td>
<td>200</td>
<td>0 (0)</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>Numeric XOR 4D</td>
<td>500</td>
<td>0 (0)</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td>Multiplexer-20</td>
<td>615</td>
<td>20 (20)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Multiplexer-XOR</td>
<td>200</td>
<td>11 (11)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>XOR-5</td>
<td>200</td>
<td>10 (10)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>XOR-5 10% noise</td>
<td>200</td>
<td>10 (10)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>XOR-10</td>
<td>10000</td>
<td>20 (20)</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Several commonly used UCI datasets were included (among the 17) to allow comparison with results reported in literature. Table 3.2 summarizes the characteristics of these datasets while Appendix A gives more detailed descriptions. To compare the performance of the different algorithms, we will consider two evaluation criteria over decision trees: (1) generalization accuracy, measured by the ratio of the correctly classified examples in the testing set, and (2) tree size, measured by the number of non-empty leaves in the tree.

3.7.2 Fixed Time Comparison

Our first set of experiments compares ID3, C4.5 with its default parameters, ID3-k(k = 2), LSID3(r = 5) and LSID3-p(r = 5). We used our own implementation for all algorithms, where the results of C4.5 were validated with WEKA’s implementation (Witten & Frank, 2005). We first discuss the results for the consistent trees and continue by analyzing the findings when pruning is applied.

---

10 The artificial datasets are available at http://www.cs.technion.ac.il/~esaher/publications/datasets.
Figure 3.20: Performance differences between LSID3(5) and ID3. The upper figure gives the relative size of the trees produced by LSID3 in comparison to ID3. The lower figure plots the accuracy achieved by both algorithms. Each point represents a dataset. The $x$-axis represents the accuracy of ID3 while the $y$-axis represents that of LSID3. The dashed line indicates equality. Points are above it if LSID3 performs better and below it if ID3 is better.

**Consistent Trees**

Figure 3.20 illustrates the differences in tree size and generalization accuracy of LSID3(5) and ID3. Figure 3.21 compares the performance of LSID3 to that of ID3-k. The full results, including significance tests, are available in Tables 3.3-3.6. Table 3.3 shows the size of the trees induced by the above mentioned algorithms, while Table 3.4 lists the differences between the algorithms and states whether
Figure 3.21: Performance differences for LSID3(5) and ID3-k. The upper figure compares the size of trees induced by each algorithm, measured relative to ID3 (in percents). The lower figure plots the absolute differences in terms of accuracy. These differences were found to have t-test significance with $\alpha = 0.05$. Similarly, Tables 3.5 and 3.6 compares the produced trees in terms of their generalization accuracy.

When comparing the algorithms that produce consistent trees, namely ID3, ID3-k and LSID3, the average tree size is the smallest for most datasets when the trees are induced with LSID3. In all cases, as Figure 3.20(a) implies, LSID3 produced smaller trees than ID3 and these improvements were found to be significant. The average reduction in size is 26.5% and for some datasets, such as XOR-5 and Multiplexer-20, it is more than 50%. ID3-k produced smaller trees than ID3 for most but not all of the datasets (see Figure 3.21, a).
Table 3.3: The size of the induced trees on various datasets. The numbers represent the average and standard deviation over the individual runs.

<table>
<thead>
<tr>
<th>DATASET</th>
<th>ID3</th>
<th>C4.5</th>
<th>ID3-k (k = 2)</th>
<th>LSID3 (r = 5)</th>
<th>LSID3-P (r = 5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autos Make</td>
<td>53.6 ±1.2</td>
<td>26.6 ±1.4</td>
<td>56.5 ±2.4</td>
<td>36.6 ±2.0</td>
<td>37.1 ±2.0</td>
</tr>
<tr>
<td>Autos Sym.</td>
<td>46.7 ±1.8</td>
<td>20.1 ±4.1</td>
<td>47.0 ±1.8</td>
<td>26.8 ±1.7</td>
<td>26.5 ±2.1</td>
</tr>
<tr>
<td>Balance</td>
<td>353.9 ±6.9</td>
<td>34.8 ±5.5</td>
<td>351.6 ±7.3</td>
<td>347.7 ±7.7</td>
<td>40.1 ±6.5</td>
</tr>
<tr>
<td>Br. Cancer</td>
<td>129.6 ±5.9</td>
<td>6.1 ±4.0</td>
<td>124.0 ±4.9</td>
<td>100.3 ±4.4</td>
<td>7.7 ±7.1</td>
</tr>
<tr>
<td>Connect-4</td>
<td>18507 ±139</td>
<td>3329 ±64</td>
<td>16143 ±44</td>
<td>14531 ±168</td>
<td>6614 ±173</td>
</tr>
<tr>
<td>Corral</td>
<td>9.2 ±2.1</td>
<td>5.3 ±1.2</td>
<td>7.0 ±0.6</td>
<td>6.7 ±0.5</td>
<td>6.3 ±0.8</td>
</tr>
<tr>
<td>Glass</td>
<td>38.7 ±2.3</td>
<td>23.9 ±2.6</td>
<td>32.3 ±2.3</td>
<td>34.2 ±2.1</td>
<td>35.5 ±2.4</td>
</tr>
<tr>
<td>Iris</td>
<td>8.5 ±1.0</td>
<td>4.7 ±0.5</td>
<td>9.1 ±0.9</td>
<td>7.6 ±0.8</td>
<td>6.6 ±1.6</td>
</tr>
<tr>
<td>Monks-1</td>
<td>62.0 ±0.0</td>
<td>11.0 ±0.0</td>
<td>27.0 ±0.0</td>
<td>27.0 ±0.0</td>
<td>21.0 ±0.0</td>
</tr>
<tr>
<td>Monks-2</td>
<td>109.0 ±0.0</td>
<td>20.0 ±0.0</td>
<td>105.0 ±0.0</td>
<td>92.4 ±3.4</td>
<td>18.3 ±5.0</td>
</tr>
<tr>
<td>Monks-3</td>
<td>31.0 ±0.0</td>
<td>9.0 ±0.0</td>
<td>34.0 ±0.0</td>
<td>26.8 ±1.6</td>
<td>9.9 ±1.4</td>
</tr>
<tr>
<td>Mushroom</td>
<td>24.0 ±0.0</td>
<td>19.0 ±0.2</td>
<td>24.9 ±0.2</td>
<td>16.2 ±0.9</td>
<td>16.2 ±0.9</td>
</tr>
<tr>
<td>Solar-Flare</td>
<td>68.9 ±2.9</td>
<td>2.0 ±1.0</td>
<td>68.4 ±3.2</td>
<td>63.2 ±2.9</td>
<td>1.2 ±0.8</td>
</tr>
<tr>
<td>Tic-tac-toe</td>
<td>189.0 ±13.6</td>
<td>83.4 ±7.8</td>
<td>176.7 ±9.0</td>
<td>151.7 ±5.6</td>
<td>112.6 ±8.9</td>
</tr>
<tr>
<td>Voting</td>
<td>13.6 ±2.2</td>
<td>2.8 ±1.5</td>
<td>12.3 ±1.6</td>
<td>13.0 ±2.0</td>
<td>3.5 ±2.1</td>
</tr>
<tr>
<td>Wine</td>
<td>7.9 ±1.0</td>
<td>5.3 ±0.7</td>
<td>7.3 ±0.9</td>
<td>6.2 ±0.7</td>
<td>7.4 ±1.3</td>
</tr>
<tr>
<td>Zoo</td>
<td>13.8 ±0.4</td>
<td>8.3 ±0.8</td>
<td>13.8 ±0.5</td>
<td>9.9 ±0.8</td>
<td>9.8 ±0.9</td>
</tr>
<tr>
<td>Numeric XOR-3D</td>
<td>43.0 ±5.1</td>
<td>1.0 ±0.1</td>
<td>15.6 ±6.8</td>
<td>9.2 ±1.0</td>
<td>11.7 ±1.7</td>
</tr>
<tr>
<td>Numeric XOR-4D</td>
<td>104.7 ±4.5</td>
<td>2.7 ±1.8</td>
<td>75.5 ±14.0</td>
<td>26.8 ±4.7</td>
<td>30.9 ±5.9</td>
</tr>
<tr>
<td>Multiplexer-20</td>
<td>142.8 ±8.3</td>
<td>66.1 ±6.5</td>
<td>67.1 ±29.0</td>
<td>46.6 ±20.0</td>
<td>41.2 ±16.8</td>
</tr>
<tr>
<td>Multiplex-XOR</td>
<td>84.0 ±5.6</td>
<td>26.0 ±5.2</td>
<td>70.2 ±5.3</td>
<td>43.9 ±5.7</td>
<td>31.6 ±4.8</td>
</tr>
<tr>
<td>XOR-5</td>
<td>92.3 ±7.8</td>
<td>21.9 ±5.3</td>
<td>82.1 ±9.3</td>
<td>32.0 ±0.0</td>
<td>32.0 ±0.0</td>
</tr>
<tr>
<td>XOR-5 Noise</td>
<td>93.6 ±6.0</td>
<td>23.2 ±5.9</td>
<td>82.4 ±7.3</td>
<td>58.2 ±6.1</td>
<td>40.4 ±5.0</td>
</tr>
<tr>
<td>XOR-10</td>
<td>3901 ±34</td>
<td>1367 ±39</td>
<td>3287 ±37</td>
<td>2004 ±585</td>
<td>1641 ±524</td>
</tr>
</tbody>
</table>

In the case of synthetic datasets, the optimal tree size can be found in theory. For instance, the tree that perfectly describes the \( n \) XOR concept is of size \( 2^n \). The results show that in this sense, the trees induced by LSID3 were almost optimal.

Reducing the tree size is usually beneficial only if the associated accuracy is not reduced. Analyzing the accuracy of the produced trees shows that LSID3 significantly outperforms ID3 for most datasets. For the other datasets, the t-test values indicate that the algorithms are not significantly different. The average absolute improvement in the accuracy of LSID3 over ID3 is 11%. The Wilcoxon test (Demsar, 2006), which compares classifiers over multiple datasets, indicates that the advantage of LSID3 over ID3 is significant, with \( \alpha = 0.05 \).

The accuracy achieved by ID3-k, as shown in Figure 3.21(b), is better than

\[ 11 \text{ Note that a theoretically optimal tree is not necessarily obtainable from a given training set.} \]
Table 3.4: The average differences in tree-size between the different algorithms and their t-test significance, with $\alpha = 0.05$ ($\sqrt{\cdot}$ indicates a significant advantage and $\times$ a significant disadvantage). The t-test is not applicable for the Monk datasets because only 1 train-test partition was used.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LSID3 vs. ID3 Diff</th>
<th>LSID3 vs. C4.5 Diff</th>
<th>LSID3-P vs. C4.5 Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autos Make</td>
<td>-17.1 ±4.8</td>
<td>9.9 ±2.4</td>
<td>10.4 ±2.2</td>
</tr>
<tr>
<td>Autos Sym.</td>
<td>-19.9 ±2.2</td>
<td>6.6 ±4.1</td>
<td>6.3 ±4.3</td>
</tr>
<tr>
<td>Balance</td>
<td>-6.2 ±5.0</td>
<td>312.8 ±9.8</td>
<td>5.2 ±7.7</td>
</tr>
<tr>
<td>Br. Cancer</td>
<td>-29.3 ±5.6</td>
<td>94.2 ±5.4</td>
<td>1.7 ±8.5</td>
</tr>
<tr>
<td>Connect-4</td>
<td>-3976 ±265</td>
<td>11201 ±183</td>
<td>3284 ±186</td>
</tr>
<tr>
<td>Corral</td>
<td>-2.5 ±2.2</td>
<td>1.4 ±1.3</td>
<td>1.0 ±1.3</td>
</tr>
<tr>
<td>Glass</td>
<td>-4.4 ±3.0</td>
<td>10.3 ±3.1</td>
<td>11.6 ±3.7</td>
</tr>
<tr>
<td>Iris</td>
<td>-0.9 ±0.7</td>
<td>2.9 ±1.0</td>
<td>1.9 ±1.6</td>
</tr>
<tr>
<td>Monks-1</td>
<td>-35.0 ±0.0</td>
<td>16.0 ±0.0</td>
<td>10.0 ±0.0</td>
</tr>
<tr>
<td>Monks-2</td>
<td>-16.6 ±3.4</td>
<td>72.4 ±3.4</td>
<td>-1.7 ±5.0</td>
</tr>
<tr>
<td>Monks-3</td>
<td>-4.2 ±1.6</td>
<td>17.8 ±1.6</td>
<td>0.9 ±1.4</td>
</tr>
<tr>
<td>Mushroom</td>
<td>-7.8 ±0.9</td>
<td>-2.8 ±0.9</td>
<td>-2.9 ±0.9</td>
</tr>
<tr>
<td>Solar-Flare</td>
<td>-5.6 ±2.5</td>
<td>61.2 ±3.1</td>
<td>-0.8 ±1.4</td>
</tr>
<tr>
<td>Tic-tac-toe</td>
<td>-37.3 ±15.1</td>
<td>68.3 ±9.3</td>
<td>29.1 ±10.9</td>
</tr>
<tr>
<td>Voting</td>
<td>-0.6 ±2.1</td>
<td>10.2 ±2.5</td>
<td>0.7 ±2.4</td>
</tr>
<tr>
<td>Wine</td>
<td>-1.7 ±1.2</td>
<td>0.9 ±1.0</td>
<td>2.1 ±1.6</td>
</tr>
<tr>
<td>Zoo</td>
<td>-3.9 ±0.9</td>
<td>1.6 ±1.2</td>
<td>1.5 ±1.2</td>
</tr>
<tr>
<td>Numeric XOR-3D</td>
<td>-33.8 ±5.1</td>
<td>8.2 ±1.0</td>
<td>10.7 ±1.7</td>
</tr>
<tr>
<td>Numeric XOR-4D</td>
<td>-77.9 ±6.0</td>
<td>24.1 ±4.8</td>
<td>28.1 ±6.3</td>
</tr>
<tr>
<td>Multiplexer-20</td>
<td>-96.1 ±21.6</td>
<td>-19.4 ±20.4</td>
<td>-24.9 ±17.4</td>
</tr>
<tr>
<td>Multiplex-XOR</td>
<td>-40.1 ±7.3</td>
<td>17.9 ±8.1</td>
<td>5.7 ±7.0</td>
</tr>
<tr>
<td>XOR-5</td>
<td>-60.3 ±7.8</td>
<td>10.1 ±5.3</td>
<td>10.1 ±5.3</td>
</tr>
<tr>
<td>XOR-5 Noise</td>
<td>-35.4 ±8.3</td>
<td>35.0 ±8.9</td>
<td>17.2 ±8.1</td>
</tr>
<tr>
<td>XOR-10</td>
<td>-1897 ±587</td>
<td>637 ±577</td>
<td>273 ±524</td>
</tr>
</tbody>
</table>

that of ID3 on some datasets. ID3-k achieved similar results to LSID3 for some datasets, but performed much worse for others, such as Tic-tac-toe and XOR-10. For most datasets, the decrease in the size of the trees induced by LSID3 is accompanied by an increase in predictive power. This phenomenon is consistent with Occam’s Razor.

Pruned Trees

Pruning techniques help to avoid overfitting. We view pruning as orthogonal to our lookahead approach. Thus, to allow handling noisy datasets, we tested the performance of LSID3-p, which post-prunes the LSID3 trees using error-based pruning.

Figure 3.22 compares the performance of LSID3-p to that of C4.5. Applying
LSID3-p performed as well as LSID3 on most datasets, and significantly better $\alpha$ found to be statistically significant by a Wilcoxon test with into account all 24 datasets, the overall improvement by LSID3-p over C4.5 was than C4.5 on 11, significantly worse on 2, and similar on the remaining 8. Taking 21 tasks for which t-test is applicable, advantage C4.5 had. For some datasets, the trees induced by C4.5 are smaller is known to be noisy. However, LSID3 was improved by pruning, eliminating the data. Before pruning, C4.5 outperformed LSID3 on the Monks-3 problem, which pruning on the trees induced by LSID3 makes it competitive with C4.5 on noisy data. Before pruning, C4.5 outperformed LSID3 on the Monks-3 problem, which is known to be noisy. However, LSID3 was improved by pruning, eliminating the advantage C4.5 had. For some datasets, the trees induced by C4.5 are smaller than those learned by LSID3-p. However, the results indicate that among the 21 tasks for which t-test is applicable,\textsuperscript{12} LSID3-p was significantly more accurate than C4.5 on 11, significantly worse on 2, and similar on the remaining 8. Taking into account all 24 datasets, the overall improvement by LSID3-p over C4.5 was found to be statistically significant by a Wilcoxon test with $\alpha = 0.05$. In general, LSID3-p performed as well as LSID3 on most datasets, and significantly better on the noisy ones.

These results confirm our expectations: the problems addressed by LSID3 and

\textsuperscript{12}The t-test is not applicable for the Monk datasets because only 1 train-test partition was used.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>ID3</th>
<th>C4.5</th>
<th>ID3-k ($r = 2$)</th>
<th>LSID3 ($r = 5$)</th>
<th>LSID3-p ($r = 5$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autos Make</td>
<td>79.1 ±9.8</td>
<td>79.4 ±10.5</td>
<td>78.0 ±9.7</td>
<td>80.3 ±9.9</td>
<td>79.2 ±10.2</td>
</tr>
<tr>
<td>Autos Sym.</td>
<td>81.9 ±9.8</td>
<td>77.4 ±10.6</td>
<td>81.2 ±9.5</td>
<td>81.3 ±10.2</td>
<td>81.1 ±9.5</td>
</tr>
<tr>
<td>Balance</td>
<td>68.8 ±5.4</td>
<td>63.8 ±4.9</td>
<td>69.7 ±5.0</td>
<td>70.3 ±5.7</td>
<td>67.9 ±5.5</td>
</tr>
<tr>
<td>Br. Cancer</td>
<td>67.0 ±8.8</td>
<td>74.7 ±8.1</td>
<td>64.4 ±8.4</td>
<td>66.9 ±9.1</td>
<td>71.1 ±8.2</td>
</tr>
<tr>
<td>Connect-4</td>
<td>75.6 ±0.5</td>
<td>81.3 ±0.4</td>
<td>77.9 ±0.5</td>
<td>78.6 ±0.5</td>
<td>80.4 ±0.5</td>
</tr>
<tr>
<td>Corral</td>
<td>69.7 ±25.4</td>
<td>65.3 ±19.8</td>
<td>89.0 ±19.9</td>
<td>83.5 ±20.9</td>
<td>79.6 ±19.8</td>
</tr>
<tr>
<td>Glass</td>
<td>66.1 ±10.2</td>
<td>67.4 ±10.3</td>
<td>70.6 ±11.0</td>
<td>70.5 ±10.7</td>
<td>69.5 ±10.4</td>
</tr>
<tr>
<td>Iris</td>
<td>93.3 ±5.9</td>
<td>95.3 ±4.8</td>
<td>93.1 ±6.7</td>
<td>94.5 ±6.0</td>
<td>94.3 ±6.2</td>
</tr>
<tr>
<td>Monks-1</td>
<td>82.9 ±0.0</td>
<td>75.7 ±0.0</td>
<td>100.0 ±0.0</td>
<td>100.0 ±0.0</td>
<td>94.4 ±0.0</td>
</tr>
<tr>
<td>Monks-2</td>
<td>69.2 ±0.0</td>
<td>65.0 ±0.0</td>
<td>63.0 ±0.0</td>
<td>67.1 ±0.5</td>
<td>63.6 ±1.1</td>
</tr>
<tr>
<td>Monks-3</td>
<td>94.4 ±0.0</td>
<td>97.2 ±0.0</td>
<td>91.7 ±0.0</td>
<td>91.5 ±1.5</td>
<td>98.1 ±1.3</td>
</tr>
<tr>
<td>Mushroom</td>
<td>100.0 ±0.0</td>
<td>100.0 ±0.0</td>
<td>100.0 ±0.0</td>
<td>100.0 ±0.0</td>
<td>100.0 ±0.0</td>
</tr>
<tr>
<td>Solar Flare</td>
<td>86.6 ±6.1</td>
<td>88.9 ±5.2</td>
<td>86.6 ±6.0</td>
<td>86.5 ±6.4</td>
<td>88.9 ±5.1</td>
</tr>
<tr>
<td>Tic-tac-toe</td>
<td>85.5 ±3.8</td>
<td>85.8 ±3.3</td>
<td>84.2 ±3.4</td>
<td>87.0 ±3.2</td>
<td>87.2 ±3.1</td>
</tr>
<tr>
<td>Voting</td>
<td>95.1 ±4.1</td>
<td>96.4 ±3.8</td>
<td>95.9 ±4.5</td>
<td>95.6 ±4.9</td>
<td>96.5 ±3.5</td>
</tr>
<tr>
<td>Wine</td>
<td>92.6 ±6.9</td>
<td>92.9 ±6.9</td>
<td>91.4 ±6.3</td>
<td>92.5 ±6.2</td>
<td>92.3 ±6.2</td>
</tr>
<tr>
<td>Zoo</td>
<td>95.2 ±7.7</td>
<td>92.2 ±8.0</td>
<td>94.3 ±7.9</td>
<td>94.3 ±6.6</td>
<td>93.5 ±6.9</td>
</tr>
<tr>
<td>Numeric XOR-3D</td>
<td>57.7 ±11.1</td>
<td>43.0 ±5.4</td>
<td>89.5 ±11.7</td>
<td>96.1 ±4.3</td>
<td>93.4 ±5.5</td>
</tr>
<tr>
<td>Numeric XOR-4D</td>
<td>49.8 ±7.0</td>
<td>52.1 ±4.7</td>
<td>62.3 ±11.8</td>
<td>93.2 ±4.4</td>
<td>91.9 ±4.7</td>
</tr>
<tr>
<td>Multiplexer-20</td>
<td>61.3 ±7.2</td>
<td>62.1 ±7.5</td>
<td>87.2 ±14.2</td>
<td>95.5 ±8.5</td>
<td>94.5 ±9.5</td>
</tr>
<tr>
<td>Multiplex-XOR</td>
<td>58.2 ±12.0</td>
<td>55.4 ±11.5</td>
<td>61.2 ±12.2</td>
<td>76.5 ±11.8</td>
<td>80.1 ±9.6</td>
</tr>
<tr>
<td>XOR-5</td>
<td>55.5 ±12.2</td>
<td>54.1 ±12.8</td>
<td>55.8 ±13.2</td>
<td>100.0 ±0.0</td>
<td>100.0 ±0.0</td>
</tr>
<tr>
<td>XOR-5 Noise</td>
<td>54.1 ±12.5</td>
<td>51.7 ±11.9</td>
<td>56.6 ±12.6</td>
<td>74.4 ±11.4</td>
<td>78.2 ±14.0</td>
</tr>
<tr>
<td>XOR-10</td>
<td>49.9 ±1.8</td>
<td>49.8 ±1.8</td>
<td>50.3 ±1.6</td>
<td>77.4 ±14.9</td>
<td>79.4 ±16.5</td>
</tr>
</tbody>
</table>

Table 3.5: The generalization accuracy of the induced trees on various datasets. The numbers represent the average and standard deviation over the individual runs.
Table 3.6: The average differences between the accuracy of the different algorithms and their t-test significance, with $\alpha = 0.05$ ($\sqrt{}$ indicates a significant advantage and $\times$ a significant disadvantage). The t-test is not applicable for the Monk datasets because only 1 train-test partition was used.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LSID3 vs. ID3</th>
<th>LSID3 vs. C4.5</th>
<th>LSID3-P vs. C4.5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Diff</td>
<td>Sig?</td>
<td>Diff</td>
</tr>
<tr>
<td>Autos Make</td>
<td>1.2 ±9.9</td>
<td>~</td>
<td>0.8 ±10.3</td>
</tr>
<tr>
<td>Autos Sym.</td>
<td>-0.6 ±11.4</td>
<td>~</td>
<td>3.9 ±12.3</td>
</tr>
<tr>
<td>Balance</td>
<td>1.5 ±4.5</td>
<td>$\sqrt{}$</td>
<td>6.5 ±5.6</td>
</tr>
<tr>
<td>Br. Cancer</td>
<td>-0.2 ±9.1</td>
<td>~</td>
<td>-7.8 ±9.1</td>
</tr>
<tr>
<td>Connect-4</td>
<td>3.0 ±0.6</td>
<td>$\sqrt{}$</td>
<td>-2.7 ±0.5</td>
</tr>
<tr>
<td>Corral</td>
<td>13.8 ±25.4</td>
<td>$\sqrt{}$</td>
<td>18.2 ±25.1</td>
</tr>
<tr>
<td>Glass</td>
<td>4.3 ±11.6</td>
<td>$\sqrt{}$</td>
<td>3.0 ±12.7</td>
</tr>
<tr>
<td>Iris</td>
<td>1.1 ±4.0</td>
<td>$\sqrt{}$</td>
<td>-0.8 ±7.8</td>
</tr>
<tr>
<td>Monks-1</td>
<td>17.1 ±0.0</td>
<td>$\sim$</td>
<td>24.3 ±0.0</td>
</tr>
<tr>
<td>Monks-2</td>
<td>-2.2 ±0.5</td>
<td>-</td>
<td>2.0 ±0.5</td>
</tr>
<tr>
<td>Monks-3</td>
<td>-3.0 ±1.5</td>
<td>-</td>
<td>-5.7 ±1.5</td>
</tr>
<tr>
<td>Mushroom</td>
<td>0.0 ±0.0</td>
<td>$\sim$</td>
<td>0.0 ±0.0</td>
</tr>
<tr>
<td>Solar Flare</td>
<td>-0.2 ±3.0</td>
<td>$\sim$</td>
<td>-2.4 ±8.0</td>
</tr>
<tr>
<td>Tic-tac-toe</td>
<td>1.5 ±4.5</td>
<td>$\sqrt{}$</td>
<td>1.2 ±4.8</td>
</tr>
<tr>
<td>Voting</td>
<td>0.4 ±4.6</td>
<td>$\sim$</td>
<td>-0.9 ±4.4</td>
</tr>
<tr>
<td>Wine</td>
<td>-0.1 ±7.4</td>
<td>$\sim$</td>
<td>-0.4 ±8.7</td>
</tr>
<tr>
<td>Zoo</td>
<td>-1.0 ±8.0</td>
<td>$\sim$</td>
<td>2.1 ±10.7</td>
</tr>
<tr>
<td>Numeric XOR-3D</td>
<td>38.4 ±12.0</td>
<td>$\sqrt{}$</td>
<td>53.1 ±7.1</td>
</tr>
<tr>
<td>Numeric XOR-4D</td>
<td>43.4 ±7.8</td>
<td>$\sqrt{}$</td>
<td>41.1 ±6.2</td>
</tr>
<tr>
<td>Multiplexer-20</td>
<td>34.1 ±11.8</td>
<td>$\sqrt{}$</td>
<td>33.4 ±11.0</td>
</tr>
<tr>
<td>Multiplex-XOR</td>
<td>18.3 ±16.2</td>
<td>$\sqrt{}$</td>
<td>21.1 ±15.1</td>
</tr>
<tr>
<td>XOR-5</td>
<td>44.5 ±12.2</td>
<td>$\sqrt{}$</td>
<td>45.9 ±12.8</td>
</tr>
<tr>
<td>XOR-5 Noise</td>
<td>20.3 ±17.2</td>
<td>$\sqrt{}$</td>
<td>22.7 ±15.9</td>
</tr>
<tr>
<td>XOR-10</td>
<td>27.5 ±14.7</td>
<td>$\sqrt{}$</td>
<td>27.7 ±14.8</td>
</tr>
</tbody>
</table>

C4.5’s pruning are different. While LSID3 allots more time for better learning of hard concepts, pruning attempts to simplify the induced trees to avoid overfitting the data. The combination of LSID3 and pruning is shown to be worthwhile: it enjoys the benefits associated with lookahead without the need to compromise when the training set is noisy.

We also examined the effect of applying error-based pruning not only to the final tree, but to the lookahead trees as well. The experiments conducted on several noisy datasets showed that the results of this extra pruning phase were very similar to the results without pruning. Although pruning results in samples that better represent the final trees, it affects all samples similarly and hence does not lead to different split decisions.
Figure 3.22: Performance differences for LSID3-p(5) and C4.5. The upper figure compares the size of trees induced by each algorithm, measured relative to ID3 (in percents). The lower figure plots the absolute differences in terms of accuracy.

**Binary Splits**

By default, LSID3 uses multiway splits, i.e., it builds a subtree for each possible value of a nominal attribute. Following the discussion in Section 3.5.2, we also tested how LSID3 performs if binary splits are forced. The tests in this case are found using exhaustive search.

To demonstrate the fragmentation problem, we used two datasets. The first dataset is Tic-tac-toe. When binary splits were forced, the performance of both C4.5 and LSID3-p improved from 85.8 and 87.2 to 94.1 and 94.8 respectively. As in the case of multiway splits, the advantage of LSID3-p over C4.5 is statistically
significant with $\alpha = 0.05$. Note that binary splits come at a price: the number of candidate splits increases and the runtime becomes significantly longer. When allocated the same time budget, LSID3-p can afford larger samples than BLSID3-p. The advantage of the latter, however, is kept.

The second task is a variant on XOR-2, where there are 3 attributes $a_1, a_2, a_3$, each of which can take one of the values $A, C, G, T$. The target concept is $a_1^* \oplus a_2^*$. The values of $a_i^*$ are obtained from $a_i$ by mapping each $A$ and $C$ to 0 and each $G$ and $T$ to 1. The dataset, referred to as Categorial XOR-2, consists of 16 randomly drawn examples. With multiway splits, both LSID3-p and C4.5 could not learn Categorial XOR-2 and their accuracy was about 50%. C4.5 failed also with binary splits. BLSID3-p, on the contrary, was 92% accurate.

We also examined the bias of LSID3 toward binary attributes. For this purpose we used the example described in Section 3.5.2. An artificial dataset with all possible values was created. LSID3 with multiway and with binary splits were run 10000 times. Although $a_4$ is as important to the class as $a_1$ and $a_2$, LSID3 with multiway splits never chose it at the root. LSID3 with binary splits, however, split the root on $a_4$ 35% of the time. These results were similar to $a_1$ (33%) and $a_2$ (32%). They indicate that forcing binary splits removes the LSID3 bias.

### 3.7.3 Anytime Behavior of the Contract Algorithms

Both LSID3 and ID3-k make use of additional resources for generating better trees. However, in order to serve as good anytime algorithms, the quality of their output should improve with the increase in their allocated resources. For a typical anytime algorithm, this improvement is greater at the beginning and diminishes over time. To test the anytime behavior of LSID3 and ID3-k, we invoked them with successive values of $r$ and $k$ respectively. In the first set of experiments we focused on domains with nominal attributes, while in the second set we tested the anytime behavior for domains with continuous attributes.

#### Nominal attributes

Figures 3.23, 3.24 and 3.25 show the average results over 10 runs of 10-fold cross-validation experiments for the Multiplex-XOR, XOR-10 and Tic-tac-toe datasets respectively. The $x$-axis represents the run time in seconds.\(^{13}\) ID3 and C4.5, which are constant time algorithms, terminate quickly and do not improve with time. Since ID3-k with $k = 1$ and LSID3 with $r = 0$ are defined to be identical to ID3, the point at which ID3 yields a result serves also as the starting point of these anytime algorithms.

---

\(^{13}\)The algorithms were implemented in C++, compiled with GCC, and run on Macintosh G5 2.5 GHz.
The graphs indicate that the anytime behavior of LSID3 is better than that of ID3-k. For ID3-k, the gaps between the points (width of the steps) increase exponentially, although successive values of $k$ were used. As a result, any extra time budget that falls into one of these gaps cannot be exploited. For example, when run on the XOR-10 dataset, ID3-k is unable to make use of additional time that is longer than 33 seconds ($k = 3$) but shorter than 350 seconds ($k = 4$). For LSID3, the difference in the time required by the algorithm for any 2 successive values of $r$ is almost the same.

For the Multiplex-XOR dataset, the tree size and generalization accuracy improve with time for both LSID3 and ID3-k, and the improvement decreases with time. Except for a short period of time, LSID3 dominates ID3-k. For the XOR-10 dataset, LSID3 has a great advantage: while ID3-k produced trees whose accuracy was limited to 55%, LSID3 reached an average accuracy of more than 90%.
In the experiment with the Tic-tac-toe dataset, LSID3 dominated ID3-k consistently, both in terms of accuracy and size. ID3-k performs poorly in this case. In addition to the large gaps between successive possible time allocations, a decrease in accuracy and an increase in tree size are observed at $k = 3$. Similar cases of pathology caused by limited-depth lookahead have been reported by Murthy and Salzberg (1995). Starting from $r = 5$, the accuracy of LSID3 does not improve over time and sometimes slightly declines (but still dominates ID3-k). We believe that the multiway splits prevent LSID3 from further improvements. Indeed, our experiments in Section 3.7.2 indicate that LSID3 can perform much better with binary splits.

**Continuous Attributes**

Our next anytime-behavior experiment uses the Numeric-XOR 4D dataset with continuous attributes. Figure 3.26 gives the results for ID3, C4.5, LSID3, and
Figure 3.25: Anytime behavior of ID3-k and LSID3 on the Tic-tac-toe dataset

ID3-k. LSID3 clearly outperforms all the other algorithms and exhibits good anytime behavior. Generalization accuracy and tree size both improve with time. ID3-k behaves poorly in this case. For example, when 200 seconds are allocated, we can run LSID3 with $r = 2$ and achieve accuracy of about 90%. With the same allocation, ID3-k can be run with $k = 2$ and achieve accuracy of about 52%. The next improvement of ID3-k (with $k = 3$) requires 10,000 seconds. But even with such a large allocation (not shown in the graph since it is off the scale), the resulting accuracy is only about 66%.

In Section 3.5.1 we described the LSID3-MC algorithm which, instead of uniformly distributing evaluation resources over all possible splitting points, performs biased sampling towards points with high information gain. Figure 3.27 compares the anytime behavior of LSID3-MC to that of LSID3. The graph of LSID3 shows, as before, the performance for successive values of $r$. The graph of LSID3-MC shows the performance for $p = 10\%, 20\%, \ldots, 150\%$. A few significant conclusions can be drawn from these results:
Figure 3.26: Anytime behavior of ID3-k, LSID3 on the Numeric-XOR 4D dataset

1. The correlation between the parameter $p$ and the runtime is almost linear: the steps in the graph are of almost constant duration.\(^{14}\) We can easily increase the granularity of the anytime graph by smaller gaps between the $p$ values.

2. It looks as if the runtime for LSID3-MC with $p = 100\%$ should be the same as LSID3($r = 1$) without sampling where all candidates are evaluated once. We can see, however, that the runtime of LSID3($r = 1$) is not sufficient for running LSID3-MC with $p = 50\%$. This is due to the overhead associated with the process of ordering the candidates prior to sample selection.

3. LSID3-MC with $p = 100\%$ performs better than LSID3($r = 1$). This difference can be explained by the fact that we performed a biased sample with

\(^{14}\)Some steps look as though they require durations that are twice as long. However, these durations actually represent two $p$ values with identical results.
repetitions, and therefore more resources were devoted to more promising tests rather than one repetition for each point as in LSID3($r = 1$).

4. When the available time is insufficient for running LSDI3($r = 1$) but more than sufficient for running ID3, LSID3-MC is more flexible and allows these intermediate points of time to be exploited. For instance, by using only one-fifth of the time required by LSID3($r = 1$), an absolute accuracy improvement of 20% over ID3 was achieved.

3.7.4 Anytime behavior of IIDT

IIDT was presented as an interruptible decision tree learner that does not require advanced knowledge of its resource allocation: it can be stopped at any moment and return a valid decision tree. We tested two versions of IIDT, the first with granularity threshold $g = 0.1$ and the second with $g = 1$. Figures 3.28, 3.29, and
Figure 3.28: Anytime behavior of IIDT on the Glass dataset

3.30 show the anytime performance of IIDT in terms of tree size and accuracy for the Glass, XOR-10, and Tic-tac-toe datasets. Each graph represents an average of 100 runs (for the $10 \times 10$ cross-validation). Unlike the graphs given in the previous section, these are interruptible anytime graphs, i.e., for each point, the $y$ coordinate reflects the performance if the algorithm was interrupted at the associated $x$ coordinate. In the contract algorithm graphs, however, each point reflects the performance if the algorithm was initially allocated the time represented by the $x$ coordinate.

In all cases, the two anytime versions indeed exploit the additional resources and produce both smaller and more accurate trees. Since our algorithm replaces a subtree only if the new one is smaller, all size graphs decrease monotonically. The most interesting anytime behavior is for the difficult XOR-10 problem. There, the tree size decreases from 4000 leaves to almost the optimal size (1024), and the accuracy increases from 50% (which is the accuracy achieved by ID3 and C4.5) to almost 100%. The shape of the graphs is typical to those of anytime algorithms.
Figure 3.29: Anytime behavior of IIDT on the 10-XOR dataset

with diminishing returns. The improvement in the accuracy of IIDT (at the latest point it was measured) over ID3 and C4.5 was found by t-test ($\alpha = 0.05$) to be significant for the Glass and XOR-10 datasets. The performance of IIDT on Tic-tac-toe slightly degrades over time. We believe that similarly to LSID3, IIDT can perform much better if binary splits are used.

The difference in performance of the two anytime algorithms is interesting. IIDT(0.1), with the lower granularity parameter, indeed produces smoother anytime graphs (with lower volatility), which allows for better control and better predictability of return. Moreover, in large portions of the time axis, the IIDT(0.1) graph shows better performance than that of IIDT(1). This is due to more sophisticated node selection in the former. Recall that $g = 1$ means that the algorithm always selects the entire tree for improvement.

The smoothness of the IIDT(0.1) graphs is somehow misleading because it represents an average of 100 runs, with each step taking place at a different time (this is in contrast to the graph for IIDT(1), where the steps are at roughly the
Figure 3.30: Anytime behavior of IIDT on the Tic-tac-toe dataset

same times). Figure 3.31 illustrates the significance of this smoothing effect on a single anytime graph (out of the 100). We can see that although the IIDT(0.1) graph is less smooth than the average graph, it is still much smoother than the corresponding IIDT(1) graph.

3.7.5 IIDT as Incremental Learner

The two instantiations of I-IIDT, namely I-IIDT-TS(0.1) and I-IIDT-EE(0.1) are compared to ID3<sup>R</sup>, C4.5<sup>R</sup>, and the non-incremental IIDT. The suffix “<sup>R</sup>” means that the algorithm is restarted with the arrival of a new example. In the case of the fixed time algorithms, ID3 and C4.5, we ignore the runtime and assume that each tree is built within 0 seconds. Since the majority of incremental methods proposed in the literature attempt to produce the same tree as ID3 or C4.5 would have produced given all the examples, ID3<sup>R</sup> and C4.5<sup>R</sup> give an upper bound on the performance of these incremental methods. To simulate an incremental setup,
we randomly partition each training set into 25% initial set and 75% incremental set. Note that unlike the other methods, the non-incremental IIDT gets only the initial set of examples.

Figures 3.32 shows the results for the XOR-10 datasets. Both versions of I-IIDT exhibit good anytime behavior. They exploit the additional time and examples. Their advantage over the greedy algorithms ID3 and C4.5, and over the batch anytime algorithm is clear. These results demonstrate the need for combining anytime and incremental learning: the anytime incremental learner succeeds where the greedy incremental inducers and the anytime batch learner fail. Comparing the performance of I-IIDT-TS to that of I-IIDT-EE shows that IIDT-TS is better for this non-noisy setup. When we repeated the same experiment with artificially added noise I-IIDT-EE was superior, due to its ability to avoid overfitting.

3.7.6 Comparison with Modern Decision Tree Learners

During the last decade, several modern decision tree learners were introduced. Although these learners were not presented and studied as anytime algorithms, some of them can be viewed as such. In what follows we compare our proposed anytime framework to three such algorithms: bagging, skewing and GATree. We first give a brief overview of the studied methods and then compare their performance to that of our anytime approach.
Overview of the Compared Methods

Page and Ray (2003) introduced skewing as an alternative to lookahead for addressing problematic concepts such as parity functions. At each node, the algorithm skew the set of examples and produces several versions of it, each with different weights for the instances. The algorithm chooses to split on the attribute that exceeds a pre-set gain threshold for the greatest number of weightings. Skewing was reported to perform well on hard concepts such as XOR-n, mainly when the dataset is large enough relative to the number of attributes. Skewing can be viewed as a contract algorithm parameterized by $w$, the number of weightings.

In order to convert skewing into an interruptible algorithm, we apply the general conversion method described in Section 3.6.1.

Two improvements to the original skewing algorithm were presented, namely sequential skewing (Page & Ray, 2004), which skews one variable at a time instead of all of them simultaneously, and generalized skewing (Ray & Page, 2005),

Figure 3.32: Learning the XOR-10 problem incrementally
which can handle nominal and continuous attributes. Nominal attributes are skewed by randomly reordering the possible values and assigning a weight for a value proportional to its rank. Continuous attributes are handled by altering the input distribution for every possible split point. We test the sequential skewing algorithm on the binary XOR and Multiplexer datasets. This version is not parameterized and hence is not anytime by nature. To convert it into an anytime algorithm, we added a parameter $k$ that controls the number of skewing iterations. Thus, instead of skewing each variable once, we skew it $k$ times. For the Tic-tac-toe dataset, where the attributes are ternary, we used only the generalized skewing algorithm, parameterized by the number of random orderings by which the nominal attributes are reweighed.

Papagelis and Kalles (2001) presented GATree, a learner that uses genetic algorithms to evolve decision trees. The initial population consists of randomly generated 1-level depth decision trees, where both the test and the class labels are drawn randomly. Mutation steps choose a random node and replaces its test with a new random one. If the chosen node is a leaf, the class label is replaced by a random label. Crossover chooses two random nodes, possibly from different trees, and swaps their subtrees. When tested on several UCI datasets, GATree was reported to produce trees as accurate as C4.5 but of significantly smaller size. GATree was also shown to outperform C4.5 on the XOR-2 and XOR-3 problems. GATree can be viewed as an interruptible anytime algorithm that uses additional time to produce more and more generations. In our experiments we used the GATree full original version with the same set of parameters as reported by Papagelis and Kalles, with one exception: we allowed a larger number of generations.

The third algorithm we tested is bagging (Breiman, 1996). Bagging is an ensemble-based method, and as such, it is naturally an interruptible anytime learner. Additional resources can be exploited by bagging to generate larger committees. In our experiments we consider 3 different bagging methods that use ID3, C4.5, and RTG (Random Tree Generator) as base learners. In addition, we tested a committee of trees produced by our LSID3. Since the latter takes significantly more time to run, the LSID3 committees are expected to be smaller than the greedy committees for the same allocated time. Note that bagging is not a direct competitor to our method. We defined our goal as inducing a single “good” decision tree while bagging generates a set of trees. Generating a set of trees rather than a single good tree eliminates one of the greatest advantages of decision trees—their comprehensibility.
Empirical Comparison

We used our own implementation for IIDT, skewing, and bagging, and the commercial version for GATree. The skewing and sequential skewing versions were run with linearly increasing parameters. The generalized skewing algorithm was run with exponentially increasing parameters. The performance of the ensemble method was tested for exponentially increasing committee sizes (1, 2, 4, 8, ...).

Figures 3.33, 3.34, and 3.35 compare IIDT to bagging with ID3 as a base learner, bagging with LSID3 ($r = 1$), and skewing on the XOR-5, Multiplexer-20, and Tic-tac-toe tasks respectively. Note that the results for ID3 are identical to those of bagging-ID3 with a single tree in the committee and hence are not plotted independently. Since GATree was run on a different machine, we report

---

15The experiments with GATree were run on a Pentium IV 2.8 GHz machine with the Windows XP operating system. The reported times are as output by the application itself.
Figure 3.34: Anytime behavior of modern learners on the Multiplexer-20 dataset

its results separately later in this section.

The graphs for the first 2 problems, which are known to be hard, show that IIDT clearly outperforms the other methods both in terms of tree size and accuracy. In both cases IIDT reaches almost perfect accuracy (99%), while bagging-ID3 and skewing topped at 55% for the first problem and 75% for the second.

The inferior performance of bagging-ID3 on the XOR-5 and Multiplexer-20 tasks is not surprising. The trees that form the committee were induced greedily and hence could not discover these difficult concepts, even when they were combined. Similar results were obtained when running bagging over C4.5 and RTG. However, when our LSID3(\( r = 1 \)) was used as a base learner, performance was significantly better than that of the greedy committees. Still, IIDT performed significantly better than bagging-LSID3, indicating that for difficult concepts, it is better to invest more resources for improving a single tree than for adding more trees of lower quality to the committee.
The inferior results of the skewing algorithms are more difficult to interpret, since skewing was shown to handle difficult concepts well. One possible explanation for this is the small number of examples with respect to the difficulty of the problem. To verify that this indeed explains the inferior results, we repeated the experiment with simpler XOR problems such as XOR-2 and XOR-3. In these cases skewing indeed did much better and outperformed ID3, reaching 100% accuracy (as IIDT). When we increased the size of the training set for the XOR-5 domain, skewing also performed better, yet IIDT outperformed it by more than 9%. For a deeper analysis of the difference between IIDT and skewing, see Chapter 6.

The average accuracy of GATree, after 150 generations, was 49.5%. It took more than 3 seconds on average to reach 150 generations. Thus, even when GATree was allocated much more time than IIDT, it could not compete with the latter. We repeated the experiment, allowing GATree to have a larger initial
population and to produce more generations. The accuracy on the testing set, even after thousands of generations, remained very low. Similar results were obtained for the Multiplexers-20 dataset.

The above set of experiments was repeated on the much more difficult XOR-10 dataset. The advantage of IIDT over the other methods was even more evident. While IIDT was able to reach accuracy of 100%, bagging-ID3, skewing, and GATree performed as poorly as a random guesser, with accuracy of only 50%.

The next experiment was with the Tic-tac-toe dataset. In this case, as shown in Figure 3.35, both ensemble-based methods have a significant advantage over the single tree inducers. We speculate that this is because ensemble methods were able to overcome the quick-fragmentation problem associated with multi-way splits by combining several classifiers. We are still looking for ways to verify this hypothesis. Bagging-ID3 outperforms the other methods until the fifth second, where bagging-LSID3 overtakes it slightly. In contrast to the XOR-5 domain, building larger committees is worthwhile in this case, even at the expense of less accurate base classifiers. However, if the time allocation permits, large ensembles of LSID3 trees are shown to be the most accurate. We believe that the general question of tradeoff between the resources allocated for each tree and the number of trees forming the ensemble should be addressed by further research with extensive experiments on various datasets. The performance of generalized skewing and IIDT was similar in this case, with a slight advantage for skewing in terms of accuracy and an advantage for IIDT in terms of tree size. GATree was run on the dataset for 150 generations (30 seconds). The average accuracy was 76.42%, much lower than that of the other methods.
Chapter 4

Anytime Learning of Cost-sensitive Trees

Assume, for example, that a medical center has decided to use machine learning techniques to induce a diagnostic tool from records of previous patients. The center aims to obtain a comprehensible model, with low expected test costs and high expected accuracy. Moreover, in many cases there are costs associated with the predictive errors. In such a scenario, the task of the inducer is to produce a model with low expected test costs and low expected misclassification costs.

In this chapter we build on the LSID3 algorithm, presented in Chapter 3, and introduce ACT (Anytime learning of Cost-sensitive Trees), our proposed anytime framework for induction of cost-sensitive decision trees (Esmeir & Markovitch, 2007a). ACT attempts to minimize the total cost of classification, that is the sum of testing costs and misclassification costs. ACT takes the same sampling approach as LSID3. The three major components of LSID3 that need to be replaced in order to adapt it for cost-sensitive problems are: (1) sampling the space of trees, (2) evaluating a tree, and (3) pruning a tree.

4.1 Biasing the Sample Towards Low-cost Trees

LISD3 uses SID3 to bias the samples towards small trees. In ACT, however, we would like to bias our sample towards low-cost trees. For this purpose, we designed a stochastic version of the EG2 algorithm, which attempts to build low cost trees greedily (Nunez, 1991). In EG2, a tree is built top-down, and the test that maximizes ICF is chosen for splitting a node, where,

$$ICF(\theta) = \frac{2\Delta I(\theta) - 1}{(\text{cost}(\theta) + 1)^w}.$$
Procedure SEG2-Choose-Attribute($E$, $A$)

Foreach $a \in A$

\[
\Delta I(a) \leftarrow \text{Information-Gain}(E, a)
\]

\[
c(a) \leftarrow \text{Cost}(a)
\]

\[
p(a) \leftarrow \frac{2^{\Delta I(a)} - 1}{c(a) + 1}
\]

$a^* \leftarrow$ Choose attribute at random from $A$;
for each attribute $a$, the probability
of selecting it is proportional to $p(a)$

Return $a^*$

**Figure 4.1:** Attribute selection in SEG2

The parameter $w \in [0, 1]$ controls the bias towards lower cost attributes. When $w = 0$, test costs are ignored and ICF relies solely on the information gain. Larger values of $w$ strengthen the effect of test costs on ICF. We discuss setting the value of $w$ in Section 4.5.

In stochastic EG2 (SEG2), we choose splitting attributes semi-randomly, proportionally to their ICF. Because SEG2 is stochastic, we expect to be able to escape local minima for at least some of the trees in the sample. Figure 4.1 formalizes the attribute selection component in SEG2. To obtain a sample of size $r$, ACT uses EG2 once and SEG2 $r - 1$ times. EG2 and SEG2 are given direct access to context-based costs, i.e., if an attribute has already been tested, its cost is zero and if another attribute that belongs to the same group has been tested, a group discount is applied.

### 4.2 Evaluating Sample Trees

LSID3 is a cost-insensitive learning algorithm. As such, its main goal is to maximize the expected accuracy of the learned tree. Occam’s razor states that given two consistent hypotheses, the smaller one is likely to be more accurate. Following Occam’s razor, LSID3 uses the tree size as a preference bias and favors splits that are expected to reduce its final size.

In a cost-sensitive setup, however, our goal is to minimize the expected total cost of classification. Therefore, rather than choosing an attribute that minimizes the size, we would like to choose one that minimizes the total cost. Given a decision tree, we need to come up with a procedure that estimates the expected cost of using the tree to classify a future case. This cost has two components: the test cost and the misclassification cost.
4.2.1 Estimating Test Costs

Assuming that the distribution of future cases would be similar to that of the learning examples, we can estimate the test costs using the training data. Given a tree, we calculate the average test cost of the training examples and use it to estimate the test cost of new cases. For a (sub)tree $T$ built from $E$, a set of $m$ training examples, we denote the average cost of traversing $T$ for an example from $E$ by

$$\overline{t_{\text{cost}}}(T, E) = \frac{1}{m} \sum_{e \in E} t_{\text{cost}}(T, e).$$

The estimated test cost for an unseen example $e^*$ is therefore $\hat{t_{\text{cost}}}(T, e^*) = \overline{t_{\text{cost}}}(T, E)$.

Observe that costs are calculated in the relevant context. If an attribute $a$ has already been tested in upper nodes, we will not charge for testing it again. Similarly, if an attribute from a group $g$ has already been tested, we will apply a group discount to the other attributes from $g$. If a delayed attribute is encountered, we sum the cost of the entire subtree.

4.2.2 Estimating Misclassification Costs

How to go about estimating the cost of misclassification is not obvious. The tree size can no longer be used as a heuristic for predictive errors. Occam’s razor allows the comparison of two consistent trees but provides no means for estimating accuracy. Moreover, tree size is measured in a different currency than accuracy and hence cannot be easily incorporated in the cost function.

Rather than using the tree size, we propose a different estimator: the expected error (Quinlan, 1993). For a leaf with $m$ training examples, of which $s$ are misclassified, the expected error is defined as the upper limit on the probability for error, i.e., $EE(m, s, cf) = U_{\text{bin}}^{\text{cf}}(m, s)$, where $cf$ is the confidence level and $U_{\text{bin}}$ is the upper limit of the confidence interval for binomial distribution. The expected error of a tree is the sum of the expected errors in its leaves.

Originally, the expected error was used by C4.5’s error-based pruning to predict whether a subtree performs better than a leaf. Although lacking a theoretical basis, it was shown experimentally to be a good heuristic. In ACT we use the expected error to approximate the misclassification cost. Assume a problem with $|C|$ classes and a misclassification cost matrix $M$. Let $c$ be the class label in a leaf $l$. Let $m_l$ be the total number of examples in $l$ and $m^c_l$ be the number of examples in $l$ that belong to class $i$. When the penalties for predictive errors are uniform ($M_{i,j} = mc$), the estimated misclassification cost in $l$ is

$$\hat{m_{\text{cost}}}(l) = EE(m_l, m_l - m^c_l, cf) \cdot mc.$$
In a problem with nonuniform misclassification costs, $mc$ should be replaced by the cost of the actual errors the leaf is expected to make. These errors are obviously unknown to the learner. One solution is to estimate each error type separately using confidence intervals for multinomial distribution and multiply it by the associated cost:

$$\hat{mcost}(l) = \sum_{i \neq c} U_{cf}^{|C|}(m_l, m^i_l, |C|) \cdot mc.$$

Such approach, however, would result in an overly pessimistic approximation, mainly when there are many classes. Alternatively, we compute the expected error as in the uniform case and propose replacing $mc$ by the weighted average of the penalty for classifying an instance as $c$ while it belongs to another class. The weights are derived from the proportions $\frac{m^i_l}{m_l - m^c_l}$ using a generalization of Laplace’s law of succession (Good, 1965, chap. 4):

$$\hat{mcost}(l) = \text{EE}(m_l, m_l - m^c_l, cf) \cdot \sum_{i \neq c} \left(\frac{m^i_l + 1}{m_l - m^c_l + |C| - 1} \cdot M_{c,i}\right).$$

Note that in a problem with $C$ classes, the average is over $C - 1$ possible penalties because $M_{c,c} = 0$. Hence, in a problem with two classes $c_1, c_2$ if a leaf is marked as $c_1$, $mc$ would be replaced by $M_{1,2}$. When classifying a new instance, the expected misclassification cost of a tree $T$ built from $m$ examples is the sum of the expected misclassification costs in the leaves divided by $m$:

$$\hat{mcost}(T) = \frac{1}{m} \sum_{l \in L} \hat{mcost}(l),$$

where $L$ is the set of leaves in $T$. Hence, the expected total cost of $T$ when classifying a single instance is:

$$\hat{total}(T, E) = \hat{tcost}(T, E) + \hat{mcost}(T).$$

An alternative approach that we intend to explore in future work is to estimate the cost of the sampled trees using the cost for a set-aside validation set. This approach is attractive mainly when the training set is large and one can afford setting aside a significant part of it.

### 4.3 Choosing a Split

Having decided about the sampler and the tree utility function, we are ready to formalize the tree growing phase in ACT. A tree is built top-down. The procedure for selecting a splitting test at each node is listed in Figure 4.2 and illustrated in Figure 4.3. We give a detailed example of how ACT chooses splits and explain how the split selection procedure is modified for numeric attributes.
Procedure ACT-Choose-Attribute($E, A, r$)

If $r = 0$

Return EG2-Choose-Attribute($E, A$)

Foreach $a \in A$

Foreach $v_i \in \text{domain}(a)$

$E_i \leftarrow \{ e \in E \mid a(e) = v_i \}$

$T \leftarrow \text{EG2}(a, E_i, A - \{a\})$

$min_i \leftarrow \text{Total-Cost}(T, E_i)$

Repeat $r - 1$ times

$T \leftarrow \text{SEG2}(a, E_i, A - \{a\})$

$min_i \leftarrow \min (min_i, \text{Total-Cost}(T, E_i))$

$total_a \leftarrow \text{Cost}(a) + \sum_{i=1}^{\text{domain}(a)} min_i$

Return $a$ for which $total_a$ is minimal

**Figure 4.2:** Procedure for Attribute selection in ACT

**Figure 4.3:** Attribute evaluation in ACT. Assume that the cost of $a$ in the current context is 0.4. The estimated cost of a subtree rooted at $a$ is therefore $0.4 + \min(4.7, 5.1) + \min(8.9, 4.9) = 10$.

### 4.3.1 Choosing a Split: Illustrative Examples

ACT’s evaluation is cost-sensitive both in that it considers test and error costs simultaneously and in that it can take into account different error penalties. To illustrate this let us consider a two-class problem with $mc = 100\$ (uniform) and 6 attributes, $a_1, \ldots, a_6$, whose costs are 10\$. The training data contains 400 examples, out of which 200 are positive and 200 are negative.

Assume that we have to choose between $a_1$ and $a_2$, and that $r = 1$. Let the trees in Figure 4.4, denoted $T1$ and $T2$, be those sampled for $a_1$ and $a_2$
Figure 4.4: Evaluation of tree samples in ACT. The leftmost column defines the costs: 6 attributes with identical cost and uniform error penalties. $T_1$ was sampled for $a_1$ and $T_2$ for $a_2$. $EE$ stands for the expected error. Because the total cost of $T_1$ is lower, ACT would prefer to split on $a_1$.

Figure 4.5: Evaluation of tree samples in ACT. The leftmost column defines the costs: 6 attributes with identical cost (except for the expensive $a_1$) and uniform error penalties. $T_1$ was sampled for $a_1$ and $T_2$ for $a_2$. Because the total cost of $T_2$ is lower, ACT would prefer to split on $a_2$.

respectively. The expected error costs of $T_1$ and $T_2$ are:

$$\widehat{\text{mcost}}(T_1) = \frac{1}{400} \left(4 \cdot \text{EE}(100, 5, 0.25)\right) \cdot 100\$ = \frac{4 \cdot 7.3}{400} \cdot 100\$ = 7.3\$$

$$\widehat{\text{mcost}}(T_2) = \frac{1}{400} \left(2 \cdot \text{EE}(50, 0, 0.25) \cdot 100\$ + 2 \cdot \text{EE}(150, 50, 0.25) \cdot 100\$\right)$$

$$= \frac{2 \cdot 1.4 + 2 \cdot 54.1}{400} \cdot 100\$ = 27.7\$$

When both test and error costs are involved, ACT considers their sum. Since the test cost of both trees is identical (20$), ACT would prefer to split on $a_1$.

In this example we set $c_f$ to 0.25, as in C4.5. In Section 4.5 we discuss how to tune $c_f$.  

---

Technion - Computer Science Department - Ph.D. Thesis PHD-2008-12 - 2008
Figure 4.6: Evaluation of tree samples in ACT. The leftmost column defines the costs: 6 attributes with identical cost and nonuniform error penalties. T1 was sampled for a1 and T2 for a2. Because the total cost of T2 is lower, ACT would prefer to split on a2.

If, however, the cost of a1 were 40$, as in Figure 4.5, tcost(T1) would become 50$ and the total cost of T1 would become 57.3$, while that of T2 would remain 47.7$. Hence, in this case ACT would split on a2.

To illustrate how ACT handles nonuniform error penalties, let us assume that the cost of all attributes is again 10$, while the cost of a false positive (FP) is 1$ and the cost of a false negative (FN) is 199$. Let the trees in Figure 4.6, denoted T1 and T2, be those sampled for a1 and a2 respectively. As in the first example, only misclassification costs play a role because the test costs of both trees is the same. Although on average the misclassification cost is also 100$, ACT now evaluates these trees differently:

\[
\hat{\text{mcost}}(T_1) = \frac{1}{400} \left( 2 \cdot \text{EE}(100, 5, 0.25) \cdot 1$ + 2 \cdot \text{EE}(100, 5, 0.25) \cdot 199$ \right) \\
= \frac{2 \cdot 7.3 \cdot 1$ + 2 \cdot 7.3 \cdot 199$}{400} = 7.3$
\]

\[
\hat{\text{mcost}}(T_2) = \frac{1}{400} \left( 2 \cdot \text{EE}(50, 0, 0.25) \cdot 199$ + 2 \cdot \text{EE}(100, 50, 0.25) \cdot 1$ \right) \\
= \frac{2 \cdot 1.4 \cdot 199$ + 2 \cdot 54.1 \cdot 1$}{400} = 1.7$
\]

Therefore, in the nonuniform setup, ACT would prefer a2. This makes sense because in the given setup we prefer trees that may result in more false positives but reduce the number of expensive false negatives.
4.3.2 Choosing a Split when Attributes are Numeric

The selection procedure as formalized in Figure 4.2 must be modified slightly when an attribute is numeric: rather than iterating over the values the attribute can take, we first pick \( r \) tests (split points) with the highest information gain and then invoke EG2 once for each point. This guarantees that numeric and nominal attributes get the same resources. Chickering, Meek, and Rounthwaite (2001) introduced several techniques for generating a small number of candidate split points dynamically with little overhead. In the future we intend to apply these techniques to select the \( r \) points. Alternatively, we could apply the Monte-carlo method described in Section 3.5.1, which is more computationally expensive due to the need to sort the points.

4.4 Cost-sensitive Pruning

Pruning plays an important role in decision tree induction. In cost-insensitive environments, the main goal of pruning is to simplify the tree in order to avoid overfitting the training data. A subtree is pruned if the resulting tree is expected to yield a lower error.

When test costs are taken into account, pruning has another important role: reducing test costs in a tree. Keeping a subtree is worthwhile only if its expected reduction in misclassification costs is larger than the cost of the tests in that subtree. If the misclassification cost is zero, it makes no sense to keep any split in the tree. If, on the other hand, the misclassification cost is much larger than the test costs, we would expect similar behavior to the cost-insensitive setup.

To handle this challenge, we propose a novel approach for cost-sensitive pruning. As in error-based pruning (Quinlan, 1993), we scan the tree bottom-up. Then we compare the expected total cost of each subtree to that of a leaf. If a leaf is expected to perform better, the subtree is pruned.

The cost of a subtree is estimated as described in Section 4.2. Formally, let \( E \) be the set of training examples that reach a subtree \( T \), and let \( m \) be the size of \( E \). Assume that \( s \) examples in \( E \) do not belong to the default class.\(^2\) Let \( L \) be the set of leaves in \( T \). We prune \( T \) into a leaf if:

\[
\frac{1}{m} \cdot EE(m, s, cf) \cdot mc \leq \hat{t}\text{cost}(T, E) + \sum_{l \in L} \hat{mc}\text{ost}(l).
\]

The above assumes a uniform misclassification cost \( mc \). In the case of nonuniform penalties, we multiply the expected error by the average misclassification cost.

\(^2\)If misclassification costs are uniform, the default class is the majority class. Otherwise, it is the class that minimizes the misclassification cost in the node.
An alternative approach for post-pruning is early stopping of the growing phase. For example, one could limit the depth of the tree, require a minimal number of examples in each child, or prevent splitting nodes when the splitting criterion fails to exceed a predetermined threshold. Obviously, any pre-pruning condition can also be applied as part of the post-pruning procedure. The advantage of post-pruning, however, is its ability to estimate the effect of a split on the entire subtree below it, and not only on its immediate successors (horizon effect).

Consider for example the 2-XOR problem \(a \oplus b\). Splitting on neither \(a\) nor \(b\) would have a positive gain and hence the growing would be stopped. If no pre-pruning is allowed, the optimal tree would be found and would not be post-pruned because the utility of the splits is correctly measured. Frank (2000) reports a comprehensive study about pruning of decision trees, in which he compared pre- to post-pruning empirically in a cost-insensitive setup. His findings show that the advantage of post-pruning on a variety of UCI datasets is not significant. Because pre-pruning is computationally more efficient, Frank concluded that, in practice, it might be a viable alternative to post-pruning. Despite these results, we decided to use post-pruning in ACT, for the following reasons:

1. Several concepts not represented in the UCI repository may appear in real-world problems. For example, parity functions naturally arise in real-world problems, such as the Drosophila survival concept (Page & Ray, 2003).

2. When costs are involved, the horizon effect may appear more frequently because high costs may hide good splits.

3. In our anytime setup the user is willing to wait longer in order to obtain a good tree. Since post-pruning takes even less time than the induction of a single greedy tree, the extra cost of post-pruning is minor.

In the future we plan to add a pre-pruning parameter which will allow early stopping when resources are limited. Another interesting direction for future work would be to post-prune the final tree but pre-prune the lookahead trees that form the samples. This would reduce the runtime at the cost of less accurate estimations for the utility of each candidate split.

### 4.5 Setting the Parameters of ACT

In addition to \(r\), the sample size, ACT is parameterized by \(w\), which controls the weight of the test costs in EG2, and \(c_f\), the confidence factor used both for pruning and for error estimation. In ACT we considered three different alternatives for setting these parameters: (1) keeping EG2’s and C4.5’s default values \(w = 1\)
and $cf = 0.25$, (2) tuning the values using cross-validation, and (3) setting the values a priori, as a function of the problem costs.

While the first solution is the simplest, it does not exploit the potential of adapting the sampling mechanism to the specific problem costs. Although tuning the values using grid search would achieve good results, it may be costly in terms of runtime. For example, if we had 5 values for each parameter and used 5-fold cross-validation, we would need to run ACT 125 times for the sake of tuning alone. In our anytime setup this time could be invested to invoke ACT with a larger $r$ and hence improve the results. Furthermore, the algorithm would not be able to output any valid solution before the tuning stage is finished. Alternatively, we could try to tune the parameters by invoking the much faster EG2, but the results would not be as good because the optimal values for EG2 are not necessarily good for ACT.

The third approach, which we chose for our experiments, is to set $w$ and $cf$ in advance, according to the problem specific costs. $w$ is set inverse proportionally to the misclassification cost: a high misclassification cost results in a smaller $w$, reducing the effect of attribute costs on the split selection measure. The exact formula is:

$$w = 0.5 + e^{-x},$$

where $x$ is the average misclassification cost divided by test costs if all attributes are measured.

In C4.5 the default value of $cf$ is 0.25. Larger $cf$ values result in less pruning. Smaller $cf$ values lead to more aggressive pruning. Therefore, in ACT we set $cf$ to a value in the range $[0.2, 0.3]$; the exact value depends on the problem cost. When test costs are dominant, we prefer aggressive pruning and hence a low value for $cf$. When test costs are negligible, we prefer to prune less. The same value of $cf$ is also used to estimate the expected error. Again, when test costs are dominant, we can afford a pessimistic estimate of the error, but when misclassification costs are dominant, we would prefer that the estimate be closer to the error rate in the training data. The exact formula for setting $cf$ is:

$$cf = 0.2 + 0.05\left(1 + \frac{x - 1}{x + 1}\right).$$

### 4.6 Empirical Evaluation

We conducted a variety of experiments to test the performance and behavior of ACT. First we introduce a novel method for automatic adaptation of existing datasets to the cost-sensitive setup. We then describe our experimental methodology and its motivation. Finally we present and discuss our results.
4.6.1 Datasets

Typically, machine learning researchers use datasets from the UCI repository (Asuncion & Newman, 2007). Only five UCI datasets, however, have assigned test costs. We include these datasets in our experiments. Nevertheless, to gain a wider perspective, we have developed an automatic method that assigns costs to existing datasets. The method is parameterized with:

1. $c_r$, the cost range.

2. $g$, the number of desired groups as a percentage of the number of attributes. In a problem with $|A|$ attributes, there are $g \cdot |A|$ groups. The probability for an attribute to belong to each of these groups is $\frac{1}{g \cdot |A| + 1}$, as is the probability for it not to belong to any of the groups.

3. $d$, the number of delayed tests as a percentage of the number of attributes.

4. $\phi$, the group discount as a percentage of the minimal cost in the group (to ensure positive costs).

5. $\propto$, a binary flag which determines whether costs are drawn randomly, uniformly ($\propto = 0$) or semi-randomly ($\propto = 1$): the cost of a test is drawn proportionally to its information gain, simulating a common case where valuable features tend to have higher costs. In this case we assume that the cost comes from a truncated normal distribution, with the mean being proportional to the gain.

Using this method, we assigned costs to 25 datasets: 20 arbitrarily chosen UCI datasets and 5 datasets that represent hard concepts and have been used in previous research. Table 4.1 summarizes the basic properties of these datasets while Appendix A describes them in more details.

Due to the randomization in the cost assignment process, the same set of parameters defines an infinite space of possible costs. For each of the 25 datasets we sampled this space 4 times with

$$c_r = [1, 100], g = 0.2, d = 0, \phi = 0.8, \propto = 1.$$ 

These parameters were chosen in an attempt to assign costs in a manner similar to that in which real costs are assigned. In total, we have 105 datasets: 5 assigned by human experts and 100 with automatically generated costs.

---

3 Costs for these datasets have been assigned by human experts (Turney, 1995).
4 The chosen UCI datasets vary in size, type of attributes, and dimension.
5 The additional 100 datasets are available at http://www.cs.technion.ac.il/~esaher/publications/cost.
### Table 4.1: Characteristics of the datasets used to evaluate ACT

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Instances</th>
<th>Attributes</th>
<th>Max domain</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast Cancer</td>
<td>277</td>
<td>9 (3)</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>Bupa</td>
<td>345</td>
<td>0 (0)</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>Car</td>
<td>1728</td>
<td>6 (0)</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Flare</td>
<td>323</td>
<td>10 (5)</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Glass</td>
<td>214</td>
<td>0 (0)</td>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>Heart</td>
<td>296</td>
<td>8 (4)</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>154</td>
<td>13 (13)</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Iris</td>
<td>150</td>
<td>0 (0)</td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>KRK</td>
<td>28056</td>
<td>6 (0)</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>Monks-1</td>
<td>124+432</td>
<td>6 (2)</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Monks-2</td>
<td>169+432</td>
<td>6 (2)</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Monks-3</td>
<td>122+432</td>
<td>6 (2)</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Multiplexer-20</td>
<td>615</td>
<td>20 (20)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Multi-XOR</td>
<td>200</td>
<td>11 (11)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Multi-AND-OR</td>
<td>200</td>
<td>11 (11)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Nursery</td>
<td>8703</td>
<td>8 (8)</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Pima</td>
<td>768</td>
<td>0 (0)</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td>TAE</td>
<td>151</td>
<td>4 (1)</td>
<td>1</td>
<td>26</td>
</tr>
<tr>
<td>Tic-Tac-Toe</td>
<td>958</td>
<td>9 (0)</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Titanic</td>
<td>2201</td>
<td>3 (2)</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Thyroid</td>
<td>3772</td>
<td>15 (15)</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Voting</td>
<td>232</td>
<td>16 (16)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Wine</td>
<td>178</td>
<td>0 (0)</td>
<td>13</td>
<td>-</td>
</tr>
<tr>
<td>XOR 3D</td>
<td>200</td>
<td>0 (0)</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>XOR-5</td>
<td>200</td>
<td>10 (10)</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Cost-insensitive learning algorithms focus on accuracy and therefore are expected to perform well when testing costs are negligible relative to misclassification costs. However, when testing costs are significant, ignoring them would result in expensive classifiers. Therefore, evaluating cost-sensitive learners requires a wide spectrum of misclassification costs. For each problem out of the 105, we created 5 instances, with uniform misclassification costs $mc = 100, 500, 1000, 5000, 10000$. Later on, we also consider nonuniform misclassification costs.

#### 4.6.2 Methodology

We start our experimental evaluation by comparing ACT, given a fixed resource allocation, with several other cost-sensitive and cost-insensitive algorithms. Next we examine the anytime behavior of ACT, and finally, we evaluate the algorithms with two modifications on the problem instances: random test cost assignment and nonuniform misclassification costs.
Compared Algorithms

ACT is compared to the following algorithms:

- **C4.5**: A cost-insensitive greedy decision tree learner. The algorithm has been re-implemented following the details in (Quinlan, 1993) and the default parameters have been used.

- **LSID3**: A cost-insensitive anytime decision tree learner. As such it uses extra time to induce trees of higher accuracy. It is not able, however, to exploit additional allotted time to reduce classification costs.

- **IDX**: A greedy top-down learner that prefers splits that maximize \( \Delta I_c \) (Norton, 1989). The algorithm does not take into account misclassification costs. IDX has been implemented on top of C4.5, by modifying the split selection criteria.

- **CSID3**: A greedy top-down learner that prefers splits that maximize \( \Delta I_{c2} \) (Tan & Schlimmer, 1989). The algorithm does not take into account misclassification costs. CSID3 has been implemented on top of C4.5, by modifying the split selection criteria.

- **EG2**: A greedy top-down learner that prefers splits that maximize \( \frac{\Delta I^2}{\theta - 1} \) (Nunez, 1991). The algorithm does not take into account misclassification costs. EG2 has been implemented on top of C4.5, by modifying the split selection criteria.

- **DTMC**: Decision Trees with Minimal Cost (DTMC) is a greedy method that attempts to minimize both types of costs simultaneously (Ling et al., 2004; Sheng et al., 2006). The basic idea is to estimate the immediate reduction in total cost after each split, and to prefer the split with the maximal reduction.\(^6\) DTMC was implemented by following the original pseudo-code (Ling et al., 2004; Sheng et al., 2006). However, the original pseudo-code does not support continuous attributes and multiple class problems. We added support to continuous attributes, as in C4.5’s dynamic binary-cut discretization, with the cost reduction replacing gain ratio for selecting cutting points. The extension to multiple class problems was straightforward. Note that DTMC does not post-prune the trees but only pre-prunes them.

- **ICET**: Inexpensive Classification with Expensive Tests (ICET) (Turney, 1995) uses genetic search to produce a new set of costs that reflects both the

\(^6\)Detailed description of DTMC can be found in Chapter 6.
original costs and the contribution of each attribute in reducing misclassification costs. Then it builds a tree using the EG2 algorithm (Nunez, 1991) but with the evolved costs instead of the original ones. ICET has been reimplemented following the detailed description given by Turney (1995). To verify the results of the reimplemention, we compared them with those reported in the literature. We followed the same experimental setup and used the same 5 datasets. The results are indeed similar: the basic version of ICET achieved an average cost of 50.8 in our reimplementation vs. 50 reported originally. One possible reason for the slight difference may be that the initial population of the genetic algorithm is randomized, as are the genetic operators and the process of partitioning the data into training, validating, and testing sets. In his paper, Turney introduced a seeded version of ICET, which includes the true costs in the initial population, and reported it to perform better than the unseeded version. Therefore, we use the seeded version for our comparison. The other parameters of ICET are the default ones.

**Normalized Cost**

As Turney (1995) points out, using the average cost of classification for each dataset is problematic because: (1) the cost differences of the algorithms become relatively small as the misclassification cost increases, (2) it is difficult to combine the results for multiple datasets in a fair manner (e.g., average), and (3) it is difficult to combine the average of the different misclassification costs. To overcome these problems, Turney suggests normalizing the average cost of classification by dividing it by the standard cost. Let $TC$ be the cost if we take all tests. Let $f_i$ be the frequency of class $i$ in the data. The error if the response is always class $i$ is therefore $(1 - f_i)$. The standard cost is defined as

$$
TC + \min_i (1 - f_i) \cdot \max_{i,j} (M_{i,j}).
$$

The standard cost is an approximation for the maximal cost in a given problem. It consists of two components: the maximal test cost and the misclassification cost if the classifier achieves only the baseline accuracy (e.g., a majority-based classifier when error costs are uniform). Because some classifiers may perform even worse than the baseline accuracy, the standard cost is not strictly an upper bound on real cost. In most of our experiments, however, it has not been exceeded.

---

7Detailed description of ICET can be found in Chapter 6.
Table 4.2: Average cost of classification as a percentage of the standard cost of classification for different \( mc \) values. The numbers represent the average over all 105 datasets and the associated confidence intervals (\( \alpha = 5\% \)).

<table>
<thead>
<tr>
<th>( mc )</th>
<th>C4.5</th>
<th>LSID3</th>
<th>IDX</th>
<th>CSID3</th>
<th>EG2</th>
<th>DTMC</th>
<th>ICET</th>
<th>ACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>50.6 ± 4</td>
<td>45.3 ± 4</td>
<td>34.4 ± 4</td>
<td>41.7 ± 4</td>
<td>35.1 ± 4</td>
<td><strong>14.6 ± 2</strong></td>
<td>24.3 ± 3.1</td>
<td>15.2 ± 2</td>
</tr>
<tr>
<td>500</td>
<td>49.9 ± 4</td>
<td>43.0 ± 4</td>
<td>42.4 ± 4</td>
<td>45.2 ± 4</td>
<td>42.5 ± 4</td>
<td>37.7 ± 3</td>
<td>36.3 ± 3.1</td>
<td><strong>34.5 ± 3</strong></td>
</tr>
<tr>
<td>1000</td>
<td>50.4 ± 5</td>
<td>42.4 ± 5</td>
<td>47.5 ± 4</td>
<td>47.8 ± 4</td>
<td>47.3 ± 4</td>
<td>47.1 ± 4</td>
<td>40.6 ± 3.9</td>
<td><strong>39.1 ± 4</strong></td>
</tr>
<tr>
<td>5000</td>
<td>53.3 ± 6</td>
<td>43.6 ± 6</td>
<td>58.1 ± 6</td>
<td>54.3 ± 6</td>
<td>57.3 ± 6</td>
<td>57.6 ± 5</td>
<td>45.7 ± 5.6</td>
<td><strong>41.5 ± 6</strong></td>
</tr>
<tr>
<td>10000</td>
<td>54.5 ± 6</td>
<td>44.5 ± 7</td>
<td>60.8 ± 6</td>
<td>56.2 ± 6</td>
<td>59.9 ± 6</td>
<td>59.5 ± 6</td>
<td>47.1 ± 6.0</td>
<td><strong>41.4 ± 6</strong></td>
</tr>
</tbody>
</table>

Statistical Significance

For each problem, a single 10-fold cross-validation experiment was conducted. The same partition to train-test sets was used for all compared algorithms. To determine statistical significance of the performance differences between ACT, ICET, and DTMC we used two tests:

- A t-test with \( \alpha = 5\% \) confidence. For each method we counted how many times it was a significant winner.

- Wilcoxon test (Demsar, 2006), which compares classifiers over multiple datasets and states whether one method is significantly better than the other (\( \alpha = 5\% \)).

4.6.3 Fixed-time Comparison

For each of the 105 \( \times \) 5 problem instances, we ran the different algorithms, including ACT with \( r = 5 \). We chose \( r = 5 \) so the average runtime of ACT would be shorter than ICET over all problems. The other methods have much shorter runtime due to their greedy nature.

Table 4.2 summarizes the results and Tables 4.4-4.8 list the detailed results for the different datasets.\(^8\) Each table corresponds to a different \( mc \) value. Each pair of numbers represents the average normalized cost and its associated confidence interval (\( \alpha = 5\% \)). Figure 4.7 illustrates the average results and plots the normalized costs for the different algorithms and misclassification costs.

Statistical significance test results for ACT, ICET, and DTMC are given in Table 4.3. The algorithms are compared using both the t-test and the Wilcoxon

---

\(^8\)For the 25 datasets with automatic cost assignment, the average over the 4 seeds is listed. The full results are available at http://www.cs.technion.ac.il/~esaher/publications/cost.
Table 4.3: DTMC vs. ACT and ICET vs. ACT using statistical tests. For each \( mc \), the first column lists the number of t-test significant wins while the second column gives the winner, if any, as implied by a Wilcoxon test over all datasets with \( \alpha = 5\% \).

<table>
<thead>
<tr>
<th>( mc )</th>
<th>( t - test ) WINS</th>
<th>( Wilcoxon ) WINNER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DTMC vs. ACT</td>
<td>ICET vs. ACT</td>
</tr>
<tr>
<td>100</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>500</td>
<td>9</td>
<td>29</td>
</tr>
<tr>
<td>1000</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>5000</td>
<td>7</td>
<td>50</td>
</tr>
<tr>
<td>10000</td>
<td>6</td>
<td>56</td>
</tr>
</tbody>
</table>

Figure 4.7: Average normalized cost as a function of misclassification cost

test. The table lists the number of t-test wins for each algorithm out of the 105 datasets, as well as the winner, if any, when the Wilcoxon test was applied.

When misclassification cost is relatively small (\( mc = 100 \)), ACT clearly outperforms ICET, with 54 significant wins as opposed to ICET’s 4 significant wins. No significant difference was found in the remaining runs. In this setup ACT was able to produce very small trees, sometimes consisting of one node; the accuracy of the learned model was ignored in this setup. ICET, on the contrary, produced, for some of the datasets, larger and more costly trees. DTMC achieved the best results, and outperformed ACT 14 times. The Wilcoxon test also indicates that DTMC is better than ACT and that ACT is better than ICET. Further investigation showed that for a few datasets ACT produced unnecessarily larger trees.
Table 4.4: Average cost of classification as a percentage of the standard cost of classification for \( mc = 100 \). The first 25 rows list for each dataset the average cost over the different 4 cost-assignments, while the last 5 rows give the results for the datasets with costs from (Turney, 1995).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>C4.5</th>
<th>LSID3</th>
<th>IDX</th>
<th>CSID3</th>
<th>EG2</th>
<th>DTMC</th>
<th>ICET</th>
<th>ACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast-cancer</td>
<td>28.9</td>
<td>22.7</td>
<td>14.3</td>
<td>25.9</td>
<td>14.6</td>
<td>7.1</td>
<td>9.6</td>
<td>7.1</td>
</tr>
<tr>
<td>Bupa</td>
<td>88.7</td>
<td>74.3</td>
<td>55.5</td>
<td>75.6</td>
<td>56.6</td>
<td>15.7</td>
<td>20.9</td>
<td>15.7</td>
</tr>
<tr>
<td>Car</td>
<td>62.2</td>
<td>64.4</td>
<td>57.6</td>
<td>60.5</td>
<td>57.6</td>
<td>11.8</td>
<td>43.0</td>
<td>11.8</td>
</tr>
<tr>
<td>Flare</td>
<td>3.4</td>
<td>3.4</td>
<td>6.1</td>
<td>4.1</td>
<td>6.1</td>
<td>3.4</td>
<td>3.8</td>
<td>3.4</td>
</tr>
<tr>
<td>Glass</td>
<td>54.3</td>
<td>47.1</td>
<td>17.4</td>
<td>25.5</td>
<td>16.8</td>
<td>15.8</td>
<td>16.0</td>
<td>16.1</td>
</tr>
<tr>
<td>Heart</td>
<td>38.5</td>
<td>38.1</td>
<td>22.7</td>
<td>32.3</td>
<td>24.0</td>
<td>8.7</td>
<td>14.3</td>
<td>8.6</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>26.8</td>
<td>20.9</td>
<td>9.2</td>
<td>17.5</td>
<td>10.2</td>
<td>3.1</td>
<td>5.4</td>
<td>3.1</td>
</tr>
<tr>
<td>Iris</td>
<td>50.8</td>
<td>46.3</td>
<td>40.4</td>
<td>40.5</td>
<td>39.6</td>
<td>19.1</td>
<td>29.1</td>
<td>22.5</td>
</tr>
<tr>
<td>KRK</td>
<td>73.6</td>
<td>67.9</td>
<td>59.7</td>
<td>63.1</td>
<td>60.4</td>
<td>23.4</td>
<td>50.4</td>
<td>23.4</td>
</tr>
<tr>
<td>Multi-ANDOR</td>
<td>46.2</td>
<td>25.7</td>
<td>25.8</td>
<td>29.6</td>
<td>26.5</td>
<td>11.6</td>
<td>13.1</td>
<td>12.5</td>
</tr>
<tr>
<td>Monks1</td>
<td>52.1</td>
<td>40.8</td>
<td>50.4</td>
<td>52.1</td>
<td>52.0</td>
<td>26.2</td>
<td>45.2</td>
<td>27.6</td>
</tr>
<tr>
<td>Monks2</td>
<td>21.6</td>
<td>49.6</td>
<td>34.0</td>
<td>28.0</td>
<td>31.2</td>
<td>11.9</td>
<td>11.9</td>
<td>11.9</td>
</tr>
<tr>
<td>Monks3</td>
<td>58.1</td>
<td>55.6</td>
<td>53.8</td>
<td>54.0</td>
<td>53.8</td>
<td>17.7</td>
<td>51.3</td>
<td>17.7</td>
</tr>
<tr>
<td>Multiplexer</td>
<td>44.1</td>
<td>24.2</td>
<td>21.3</td>
<td>31.5</td>
<td>22.1</td>
<td>10.9</td>
<td>10.9</td>
<td>10.7</td>
</tr>
<tr>
<td>MultiXOR</td>
<td>56.8</td>
<td>39.3</td>
<td>29.5</td>
<td>42.7</td>
<td>30.7</td>
<td>13.2</td>
<td>13.1</td>
<td>13.3</td>
</tr>
<tr>
<td>Nursery</td>
<td>51.8</td>
<td>55.2</td>
<td>51.6</td>
<td>50.6</td>
<td>50.4</td>
<td>23.8</td>
<td>49.2</td>
<td>23.8</td>
</tr>
<tr>
<td>Pima</td>
<td>70.5</td>
<td>72.1</td>
<td>35.7</td>
<td>63.2</td>
<td>40.7</td>
<td>12.4</td>
<td>16.2</td>
<td>12.5</td>
</tr>
<tr>
<td>Tae</td>
<td>64.3</td>
<td>52.0</td>
<td>47.4</td>
<td>54.2</td>
<td>51.0</td>
<td>28.1</td>
<td>34.4</td>
<td>27.1</td>
</tr>
<tr>
<td>Tic-tac-toe</td>
<td>60.4</td>
<td>56.1</td>
<td>30.7</td>
<td>51.4</td>
<td>32.1</td>
<td>13.5</td>
<td>25.3</td>
<td>13.5</td>
</tr>
<tr>
<td>Titanic</td>
<td>85.3</td>
<td>53.4</td>
<td>67.6</td>
<td>72.2</td>
<td>68.2</td>
<td>18.4</td>
<td>48.2</td>
<td>18.4</td>
</tr>
<tr>
<td>Thyroid</td>
<td>26.6</td>
<td>25.6</td>
<td>27.6</td>
<td>26.3</td>
<td>26.3</td>
<td>2.1</td>
<td>20.8</td>
<td>2.1</td>
</tr>
<tr>
<td>Voting</td>
<td>16.8</td>
<td>17.8</td>
<td>13.3</td>
<td>16.5</td>
<td>13.9</td>
<td>5.6</td>
<td>13.4</td>
<td>7.0</td>
</tr>
<tr>
<td>Wine</td>
<td>31.7</td>
<td>31.4</td>
<td>11.0</td>
<td>18.2</td>
<td>12.2</td>
<td>7.8</td>
<td>10.1</td>
<td>8.6</td>
</tr>
<tr>
<td>XOR3d</td>
<td>82.2</td>
<td>66.1</td>
<td>29.4</td>
<td>57.8</td>
<td>30.9</td>
<td>22.0</td>
<td>22.5</td>
<td>24.1</td>
</tr>
<tr>
<td>XOR5</td>
<td>56.0</td>
<td>64.0</td>
<td>38.9</td>
<td>45.4</td>
<td>39.9</td>
<td>12.4</td>
<td>12.4</td>
<td>12.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>C4.5</th>
<th>LSID3</th>
<th>IDX</th>
<th>CSID3</th>
<th>EG2</th>
<th>DTMC</th>
<th>ICET</th>
<th>ACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bupa</td>
<td>89.8</td>
<td>85.2</td>
<td>88.7</td>
<td>86.7</td>
<td>88.7</td>
<td>65.1</td>
<td>66.5</td>
<td>76.2</td>
</tr>
<tr>
<td>Heart</td>
<td>58.6</td>
<td>57.0</td>
<td>8.5</td>
<td>11.7</td>
<td>8.8</td>
<td>8.2</td>
<td>8.0</td>
<td>8.4</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>53.8</td>
<td>48.6</td>
<td>31.3</td>
<td>35.3</td>
<td>34.0</td>
<td>26.8</td>
<td>36.4</td>
<td>34.8</td>
</tr>
<tr>
<td>Pima</td>
<td>65.8</td>
<td>71.4</td>
<td>52.0</td>
<td>54.4</td>
<td>52.0</td>
<td>44.7</td>
<td>45.7</td>
<td>46.6</td>
</tr>
<tr>
<td>Thyroid</td>
<td>33.6</td>
<td>33.5</td>
<td>32.5</td>
<td>32.9</td>
<td>32.3</td>
<td>9.7</td>
<td>28.3</td>
<td>9.7</td>
</tr>
</tbody>
</table>

We believe that a better tuning of \( cf \) would improve ACT in this scenario by making the pruning more aggressive.

At the other extreme, when misclassification costs dominate (\( mc = 10000 \)), the performance of DTMC is worse than ACT and ICET. The t-test indicates that ACT was significantly better than ICET 24 times and significantly worse only 7 times. According to the Wilcoxon test with \( \alpha = 5\% \), the difference between ACT and ICET is not significant. Taking \( \alpha > 5.05\% \), however, would turn the
Table 4.5: Average cost of classification as a percentage of the standard cost of classification for $mc = 500$. The first 25 rows list for each dataset the average cost over the different 4 cost-assignments, while the last 5 rows give the results for the datasets with costs from (Turney, 1995).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>C4.5</th>
<th>LSID3</th>
<th>IDX</th>
<th>CSID3</th>
<th>EG2</th>
<th>DTMC</th>
<th>ICET</th>
<th>ACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast-cancer</td>
<td>43.3</td>
<td>42.0</td>
<td>33.4</td>
<td>42.7</td>
<td>33.9</td>
<td>27.8</td>
<td>29.1</td>
<td>28.0</td>
</tr>
<tr>
<td>Bupa</td>
<td>90.5</td>
<td>81.2</td>
<td>70.6</td>
<td>82.1</td>
<td>70.9</td>
<td>49.6</td>
<td>51.6</td>
<td>53.4</td>
</tr>
<tr>
<td>Car</td>
<td>49.3</td>
<td>51.9</td>
<td>55.2</td>
<td>49.5</td>
<td>53.7</td>
<td>40.1</td>
<td>48.5</td>
<td>40.1</td>
</tr>
<tr>
<td>Flare</td>
<td>15.0</td>
<td>15.0</td>
<td>16.8</td>
<td>15.4</td>
<td>16.8</td>
<td>15.0</td>
<td>16.5</td>
<td>15.0</td>
</tr>
<tr>
<td>Glass</td>
<td>54.1</td>
<td>47.2</td>
<td>42.3</td>
<td>42.3</td>
<td>38.9</td>
<td>37.8</td>
<td>37.5</td>
<td>36.6</td>
</tr>
<tr>
<td>Heart</td>
<td>42.3</td>
<td>44.8</td>
<td>34.5</td>
<td>39.2</td>
<td>35.0</td>
<td>30.2</td>
<td>29.9</td>
<td>31.0</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>35.6</td>
<td>29.6</td>
<td>20.7</td>
<td>27.9</td>
<td>21.3</td>
<td>13.9</td>
<td>14.8</td>
<td>14.8</td>
</tr>
<tr>
<td>Iris</td>
<td>30.6</td>
<td>28.2</td>
<td>29.0</td>
<td>27.0</td>
<td>27.8</td>
<td>21.7</td>
<td>22.1</td>
<td>20.7</td>
</tr>
<tr>
<td>KRK</td>
<td>62.4</td>
<td>64.9</td>
<td>63.9</td>
<td>63.5</td>
<td>63.6</td>
<td>62.2</td>
<td>61.9</td>
<td>59.7</td>
</tr>
<tr>
<td>Multi-ANDOR</td>
<td>47.3</td>
<td>18.6</td>
<td>29.3</td>
<td>31.2</td>
<td>30.1</td>
<td>35.7</td>
<td>19.8</td>
<td>17.2</td>
</tr>
<tr>
<td>Monks1</td>
<td>26.0</td>
<td>20.0</td>
<td>25.2</td>
<td>25.9</td>
<td>26.0</td>
<td>53.2</td>
<td>23.5</td>
<td>22.8</td>
</tr>
<tr>
<td>Monks2</td>
<td>48.9</td>
<td>66.9</td>
<td>57.3</td>
<td>53.8</td>
<td>55.5</td>
<td>40.3</td>
<td>40.3</td>
<td>40.3</td>
</tr>
<tr>
<td>Monks3</td>
<td>35.0</td>
<td>33.5</td>
<td>32.7</td>
<td>32.6</td>
<td>32.6</td>
<td>34.6</td>
<td>32.2</td>
<td>32.5</td>
</tr>
<tr>
<td>Multiplexer</td>
<td>53.3</td>
<td>19.3</td>
<td>39.5</td>
<td>45.6</td>
<td>40.2</td>
<td>39.0</td>
<td>22.3</td>
<td>25.3</td>
</tr>
<tr>
<td>MultiXOR</td>
<td>60.4</td>
<td>26.1</td>
<td>43.2</td>
<td>51.7</td>
<td>44.4</td>
<td>42.9</td>
<td>43.8</td>
<td>32.5</td>
</tr>
<tr>
<td>Nursery</td>
<td>29.3</td>
<td>31.1</td>
<td>29.9</td>
<td>29.0</td>
<td>29.2</td>
<td>31.5</td>
<td>28.1</td>
<td>28.8</td>
</tr>
<tr>
<td>Pima</td>
<td>73.1</td>
<td>77.3</td>
<td>54.5</td>
<td>68.3</td>
<td>58.1</td>
<td>42.0</td>
<td>45.9</td>
<td>48.3</td>
</tr>
<tr>
<td>Tae</td>
<td>77.2</td>
<td>67.6</td>
<td>65.5</td>
<td>68.2</td>
<td>67.9</td>
<td>59.3</td>
<td>59.7</td>
<td>58.2</td>
</tr>
<tr>
<td>Tic-Tac-Toe</td>
<td>55.8</td>
<td>52.3</td>
<td>40.2</td>
<td>49.6</td>
<td>40.9</td>
<td>43.9</td>
<td>42.4</td>
<td>36.4</td>
</tr>
<tr>
<td>Titanic</td>
<td>76.6</td>
<td>59.4</td>
<td>66.5</td>
<td>69.2</td>
<td>66.8</td>
<td>51.4</td>
<td>56.5</td>
<td>51.4</td>
</tr>
<tr>
<td>Thyroid</td>
<td>25.5</td>
<td>26.5</td>
<td>26.5</td>
<td>25.3</td>
<td>25.2</td>
<td>9.7</td>
<td>24.5</td>
<td>9.7</td>
</tr>
<tr>
<td>Voting</td>
<td>14.4</td>
<td>15.6</td>
<td>14.2</td>
<td>14.4</td>
<td>14.2</td>
<td>13.9</td>
<td>13.9</td>
<td>13.6</td>
</tr>
<tr>
<td>Wine</td>
<td>24.8</td>
<td>25.7</td>
<td>22.4</td>
<td>19.7</td>
<td>21.0</td>
<td>19.8</td>
<td>18.4</td>
<td>17.6</td>
</tr>
<tr>
<td>XOR3d</td>
<td>86.9</td>
<td>54.7</td>
<td>61.1</td>
<td>71.5</td>
<td>61.5</td>
<td>64.5</td>
<td>58.6</td>
<td>60.1</td>
</tr>
<tr>
<td>XOR5</td>
<td>69.7</td>
<td>43.7</td>
<td>64.5</td>
<td>67.6</td>
<td>65.4</td>
<td>41.3</td>
<td>41.3</td>
<td>41.3</td>
</tr>
</tbody>
</table>

| Bupa        | 92.4 | 90.2  | 92.2 | 88.2  | 92.2 | 80.3 | 86.0 | 86.6 |
| Heart       | 57.0 | 59.6  | 26.7 | 28.8  | 27.3 | 26.3 | 26.3 | 28.9 |
| Hepatitis   | 83.7 | 77.5  | 59.5 | 64.9  | 64.3 | 59.7 | 72.9 | 72.6 |
| Pima        | 73.8 | 81.8  | 87.2 | 79.2  | 87.2 | 70.7 | 67.5 | 70.6 |
| Thyroid     | 27.7 | 34.3  | 27.8 | 27.3  | 26.5 | 35.0 | 27.8 | 25.0 |

result in favor of ACT. Observe that DTMC, the winner when $mc = 100$, becomes the worst algorithm when $mc = 10000$. One reason for this phenomenon is that DTMC, as introduced by Ling et al. (2004), does not perform post-pruning, although doing so might improve accuracy in some domains.

The above two extremes are less interesting: for the first we could use an algorithm that always outputs a tree of size 1 while for the second we could use cost-insensitive learners. The middle range, where $mc \in \{500, 1000, 5000\}$,
Table 4.6: Average cost of classification as a percentage of the standard cost of classification for \( mc = 1000 \). The first 25 rows list for each dataset the average cost over the different 4 cost-assignments, while the last 5 rows give the results for the datasets with costs from (Turney, 1995).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>C4.5</th>
<th>LSID3</th>
<th>IDX</th>
<th>CSID3</th>
<th>EG2</th>
<th>DTMC</th>
<th>ICET</th>
<th>ACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast-cancer</td>
<td>54.3</td>
<td>56.7</td>
<td>48.0</td>
<td>55.4</td>
<td>48.6</td>
<td>43.6</td>
<td>45.2</td>
<td>45.5</td>
</tr>
<tr>
<td>Bupa</td>
<td>91.5</td>
<td>84.8</td>
<td>78.4</td>
<td>85.5</td>
<td>78.2</td>
<td>72.2</td>
<td>69.8</td>
<td>72.8</td>
</tr>
<tr>
<td>Car</td>
<td>41.4</td>
<td>44.3</td>
<td>53.8</td>
<td>42.8</td>
<td>51.4</td>
<td>57.2</td>
<td>41.2</td>
<td>42.1</td>
</tr>
<tr>
<td>Flare</td>
<td>26.0</td>
<td>26.0</td>
<td>27.1</td>
<td>26.3</td>
<td>27.1</td>
<td>26.0</td>
<td>25.9</td>
<td>26.0</td>
</tr>
<tr>
<td>Glass</td>
<td>53.9</td>
<td>47.2</td>
<td>55.0</td>
<td>50.9</td>
<td>50.2</td>
<td>49.2</td>
<td>46.1</td>
<td>39.7</td>
</tr>
<tr>
<td>Heart</td>
<td>45.0</td>
<td>49.5</td>
<td>43.0</td>
<td>44.2</td>
<td>42.8</td>
<td>40.3</td>
<td>41.2</td>
<td>41.4</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>44.2</td>
<td>38.1</td>
<td>31.9</td>
<td>38.1</td>
<td>32.2</td>
<td>28.6</td>
<td>29.3</td>
<td>29.7</td>
</tr>
<tr>
<td>Iris</td>
<td>22.3</td>
<td>20.8</td>
<td>24.1</td>
<td>21.3</td>
<td>22.8</td>
<td>18.9</td>
<td>16.8</td>
<td>16.3</td>
</tr>
<tr>
<td>KRK</td>
<td>57.8</td>
<td>63.7</td>
<td>65.6</td>
<td>63.6</td>
<td>64.8</td>
<td>65.2</td>
<td>55.1</td>
<td>55.6</td>
</tr>
<tr>
<td>Multi-ANDOR</td>
<td>47.9</td>
<td>14.1</td>
<td>31.5</td>
<td>32.2</td>
<td>32.4</td>
<td>49.0</td>
<td>16.2</td>
<td>11.8</td>
</tr>
<tr>
<td>Monks1</td>
<td>16.3</td>
<td>12.2</td>
<td>15.8</td>
<td>16.2</td>
<td>16.3</td>
<td>52.3</td>
<td>15.6</td>
<td>12.2</td>
</tr>
<tr>
<td>Monks2</td>
<td>65.4</td>
<td>77.4</td>
<td>71.3</td>
<td>69.4</td>
<td>70.1</td>
<td>57.5</td>
<td>57.5</td>
<td>57.5</td>
</tr>
<tr>
<td>Monks3</td>
<td>23.8</td>
<td>22.9</td>
<td>22.5</td>
<td>22.3</td>
<td>22.3</td>
<td>25.5</td>
<td>22.1</td>
<td>22.0</td>
</tr>
<tr>
<td>Multiplexer</td>
<td>59.3</td>
<td>16.2</td>
<td>51.4</td>
<td>54.7</td>
<td>52.0</td>
<td>57.9</td>
<td>18.5</td>
<td>19.9</td>
</tr>
<tr>
<td>MultiXOR</td>
<td>62.5</td>
<td>18.5</td>
<td>51.0</td>
<td>56.9</td>
<td>52.3</td>
<td>56.7</td>
<td>49.5</td>
<td>21.3</td>
</tr>
<tr>
<td>Nursery</td>
<td>20.3</td>
<td>21.6</td>
<td>21.4</td>
<td>20.5</td>
<td>20.8</td>
<td>24.7</td>
<td>18.6</td>
<td>20.2</td>
</tr>
<tr>
<td>Pima</td>
<td>74.6</td>
<td>80.3</td>
<td>65.6</td>
<td>71.3</td>
<td>68.4</td>
<td>62.3</td>
<td>60.8</td>
<td>64.0</td>
</tr>
<tr>
<td>Tae</td>
<td>82.1</td>
<td>73.5</td>
<td>72.3</td>
<td>73.6</td>
<td>74.3</td>
<td>64.2</td>
<td>76.8</td>
<td>67.4</td>
</tr>
<tr>
<td>Tic-tac-toe</td>
<td>53.3</td>
<td>50.2</td>
<td>45.6</td>
<td>48.6</td>
<td>45.9</td>
<td>60.9</td>
<td>44.8</td>
<td>40.2</td>
</tr>
<tr>
<td>Titanic</td>
<td>72.5</td>
<td>62.3</td>
<td>65.9</td>
<td>67.8</td>
<td>66.2</td>
<td>64.3</td>
<td>60.5</td>
<td>61.2</td>
</tr>
<tr>
<td>Thyroid</td>
<td>24.3</td>
<td>27.3</td>
<td>25.4</td>
<td>24.1</td>
<td>24.1</td>
<td>17.7</td>
<td>24.1</td>
<td>17.7</td>
</tr>
<tr>
<td>Voting</td>
<td>12.7</td>
<td>14.1</td>
<td>14.8</td>
<td>13.0</td>
<td>14.3</td>
<td>11.8</td>
<td>12.0</td>
<td>11.8</td>
</tr>
<tr>
<td>Wine</td>
<td>20.3</td>
<td>21.9</td>
<td>30.0</td>
<td>20.7</td>
<td>26.8</td>
<td>20.6</td>
<td>18.8</td>
<td>19.1</td>
</tr>
<tr>
<td>XOR3d</td>
<td>89.0</td>
<td>49.6</td>
<td>75.4</td>
<td>77.8</td>
<td>75.3</td>
<td>84.4</td>
<td>70.1</td>
<td>66.7</td>
</tr>
<tr>
<td>XOR5</td>
<td>77.9</td>
<td>31.6</td>
<td>79.7</td>
<td>80.7</td>
<td>80.6</td>
<td>58.4</td>
<td>58.4</td>
<td>67.4</td>
</tr>
<tr>
<td>Bupa</td>
<td>93.0</td>
<td>91.2</td>
<td>92.8</td>
<td>88.5</td>
<td>92.8</td>
<td>80.8</td>
<td>84.3</td>
<td>91.2</td>
</tr>
<tr>
<td>Heart</td>
<td>56.0</td>
<td>61.1</td>
<td>37.4</td>
<td>38.9</td>
<td>38.3</td>
<td>38.6</td>
<td>35.4</td>
<td>42.3</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>93.1</td>
<td>86.5</td>
<td>68.4</td>
<td>74.2</td>
<td>73.8</td>
<td>77.3</td>
<td>68.1</td>
<td>76.8</td>
</tr>
<tr>
<td>Pima</td>
<td>75.8</td>
<td>84.4</td>
<td>96.0</td>
<td>85.4</td>
<td>96.0</td>
<td>73.4</td>
<td>70.5</td>
<td>72.1</td>
</tr>
<tr>
<td>Thyroid</td>
<td>23.8</td>
<td>34.8</td>
<td>24.7</td>
<td>23.6</td>
<td>22.7</td>
<td>22.4</td>
<td>24.6</td>
<td>22.9</td>
</tr>
</tbody>
</table>

requires that the learner carefully balance the two types of cost. In these cases ACT has the lowest average cost and the largest number of t-test wins. Moreover, the Wilcoxon test indicates that it is superior. ICET is the second best method. As reported by Turney (1995), ICET is clearly better than the greedy methods EG2, IDX, and CSID3.

Note that EG2, IDX, and CSID3, which are insensitive to misclassification cost, produced the same trees for all values of \( mc \). These trees, however, are
Table 4.7: Average cost of classification as a percentage of the standard cost of classification for $mc = 5000$. The first 25 rows list for each dataset the average cost over the different 4 cost-assignments, while the last 5 rows give the results for the datasets with costs from (Turney, 1995).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>C4.5</th>
<th>LSID3</th>
<th>IDX</th>
<th>CSID3</th>
<th>EG2</th>
<th>DTMC</th>
<th>ICET</th>
<th>ACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast-cancer</td>
<td>79.4</td>
<td>90.3</td>
<td>81.4</td>
<td>84.4</td>
<td>82.2</td>
<td>83.2</td>
<td>85.1</td>
<td>73.4</td>
</tr>
<tr>
<td>Bupa</td>
<td>92.9</td>
<td>90.2</td>
<td>90.0</td>
<td>90.8</td>
<td>89.2</td>
<td>82.3</td>
<td>85.4</td>
<td>83.9</td>
</tr>
<tr>
<td>Car</td>
<td>27.8</td>
<td>31.1</td>
<td>51.5</td>
<td>31.2</td>
<td>47.4</td>
<td>86.9</td>
<td>27.3</td>
<td>27.6</td>
</tr>
<tr>
<td>flare</td>
<td>63.6</td>
<td>63.6</td>
<td>62.0</td>
<td>63.4</td>
<td>62.0</td>
<td>63.8</td>
<td>63.7</td>
<td>61.4</td>
</tr>
<tr>
<td>glass</td>
<td>53.8</td>
<td>47.3</td>
<td>74.1</td>
<td>63.7</td>
<td>67.1</td>
<td>50.7</td>
<td>50.4</td>
<td>49.6</td>
</tr>
<tr>
<td>heart</td>
<td>50.7</td>
<td>59.5</td>
<td>60.9</td>
<td>54.6</td>
<td>59.4</td>
<td>51.1</td>
<td>51.8</td>
<td>57.9</td>
</tr>
<tr>
<td>hepatitis</td>
<td>74.8</td>
<td>68.2</td>
<td>71.9</td>
<td>74.5</td>
<td>70.8</td>
<td>60.6</td>
<td>56.2</td>
<td>57.2</td>
</tr>
<tr>
<td>Iris</td>
<td>11.6</td>
<td>11.2</td>
<td>17.9</td>
<td>14.0</td>
<td>16.4</td>
<td>10.8</td>
<td>10.0</td>
<td>11.1</td>
</tr>
<tr>
<td>KRK</td>
<td>52.2</td>
<td>62.3</td>
<td>67.6</td>
<td>63.7</td>
<td>66.3</td>
<td>51.9</td>
<td>47.8</td>
<td>47.3</td>
</tr>
<tr>
<td>multi-ANDOR</td>
<td>49.1</td>
<td>6.1</td>
<td>35.4</td>
<td>34.0</td>
<td>36.6</td>
<td>75.0</td>
<td>9.4</td>
<td>3.7</td>
</tr>
<tr>
<td>monks1</td>
<td>4.8</td>
<td>3.0</td>
<td>4.7</td>
<td>4.7</td>
<td>4.9</td>
<td>51.1</td>
<td>4.8</td>
<td>3.0</td>
</tr>
<tr>
<td>monks2</td>
<td>93.8</td>
<td>95.5</td>
<td>95.7</td>
<td>96.3</td>
<td>95.5</td>
<td>87.1</td>
<td>81.0</td>
<td>89.4</td>
</tr>
<tr>
<td>monks3</td>
<td>8.1</td>
<td>7.8</td>
<td>8.1</td>
<td>7.6</td>
<td>7.8</td>
<td>12.3</td>
<td>7.7</td>
<td>7.7</td>
</tr>
<tr>
<td>multiplexer</td>
<td>70.6</td>
<td>10.3</td>
<td>74.2</td>
<td>72.1</td>
<td>74.5</td>
<td>74.7</td>
<td>24.6</td>
<td>5.4</td>
</tr>
<tr>
<td>multiXOR</td>
<td>66.0</td>
<td>6.0</td>
<td>64.1</td>
<td>65.4</td>
<td>65.5</td>
<td>73.0</td>
<td>53.2</td>
<td>6.7</td>
</tr>
<tr>
<td>nursery</td>
<td>9.3</td>
<td>9.7</td>
<td>10.8</td>
<td>9.8</td>
<td>10.4</td>
<td>16.8</td>
<td>6.8</td>
<td>8.6</td>
</tr>
<tr>
<td>Pima</td>
<td>77.2</td>
<td>85.5</td>
<td>84.4</td>
<td>76.4</td>
<td>85.9</td>
<td>73.2</td>
<td>69.2</td>
<td>78.0</td>
</tr>
<tr>
<td>Tae</td>
<td>88.0</td>
<td>80.5</td>
<td>80.5</td>
<td>80.0</td>
<td>82.0</td>
<td>66.7</td>
<td>86.4</td>
<td>78.7</td>
</tr>
<tr>
<td>Tic-tac-toe</td>
<td>49.0</td>
<td>46.6</td>
<td>54.2</td>
<td>47.0</td>
<td>53.9</td>
<td>57.6</td>
<td>37.6</td>
<td>47.3</td>
</tr>
<tr>
<td>Titanic</td>
<td>66.9</td>
<td>66.1</td>
<td>65.1</td>
<td>65.6</td>
<td>65.2</td>
<td>67.2</td>
<td>65.7</td>
<td>66.5</td>
</tr>
<tr>
<td>thyroid</td>
<td>19.5</td>
<td>31.0</td>
<td>20.8</td>
<td>19.3</td>
<td>19.3</td>
<td>19.3</td>
<td>19.6</td>
<td>19.8</td>
</tr>
<tr>
<td>Voting</td>
<td>9.2</td>
<td>11.0</td>
<td>16.0</td>
<td>10.1</td>
<td>14.6</td>
<td>8.3</td>
<td>8.4</td>
<td>8.6</td>
</tr>
<tr>
<td>wine</td>
<td>11.7</td>
<td>14.6</td>
<td>44.4</td>
<td>22.7</td>
<td>37.9</td>
<td>29.6</td>
<td>14.0</td>
<td>15.4</td>
</tr>
<tr>
<td>xor3d</td>
<td>91.9</td>
<td>42.8</td>
<td>94.9</td>
<td>86.4</td>
<td>94.1</td>
<td>100.3</td>
<td>62.9</td>
<td>48.2</td>
</tr>
<tr>
<td>xor5</td>
<td>91.8</td>
<td>88.4</td>
<td>105.8</td>
<td>103.1</td>
<td>106.5</td>
<td>87.5</td>
<td>87.5</td>
<td>51.9</td>
</tr>
</tbody>
</table>

| Bupa          | 93.4 | 92.1  | 93.4| 88.7  | 93.4| 81.0 | 94.2 | 90.2|
| heart         | 54.4 | 63.7  | 55.6| 56.0  | 56.9| 63.7 | 49.5 | 58.9|
| hepatitis     | 103.2| 96.3  | 77.9| 84.2  | 84.1| 85.1 | 76.1 | 85.2|
| Pima          | 77.8 | 87.0  | 104.7|91.5  |104.7| 76.6 | 70.6 | 75.8|
| thyroid       | 16.3 | 35.8  | 18.7| 16.5  | 15.2| 15.4 | 15.4 | 16.5|

judged differently with the change in misclassification cost.

Figure 4.8 illustrates the differences between ICET and ACT for $mc = 100, 1000, 10000$. Each point represents one of the 105 datasets. The $x$-axis represents the cost of ICET while the $y$-axis represents that of ACT. The dashed line indicates equality. As we can see, the majority of points are below the equality line, indicating that ACT performs better. For $mc = 10000$ we can see that there are points located close to the $x$-axis but with large $x$ value. These points
Table 4.8: Average cost of classification as a percentage of the standard cost of classification for $mc = 10000$. The first 25 rows list for each dataset the average cost over the different 4 cost-assignments, while the last 5 rows give the results for the datasets with costs from (Turney, 1995).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>C4.5</th>
<th>LSID3</th>
<th>IDX</th>
<th>CSID3</th>
<th>EG2</th>
<th>DTMC</th>
<th>ICET</th>
<th>ACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast-cancer</td>
<td>85.8</td>
<td>98.9</td>
<td>90.0</td>
<td>91.8</td>
<td>90.8</td>
<td>84.4</td>
<td>81.7</td>
<td>84.0</td>
</tr>
<tr>
<td>Bupa</td>
<td>93.2</td>
<td>91.2</td>
<td>92.1</td>
<td>91.8</td>
<td>91.2</td>
<td>81.9</td>
<td>88.1</td>
<td>86.2</td>
</tr>
<tr>
<td>Car</td>
<td>25.0</td>
<td>28.4</td>
<td>51.1</td>
<td>28.9</td>
<td>46.7</td>
<td>93.0</td>
<td>24.7</td>
<td>24.2</td>
</tr>
<tr>
<td>Flare</td>
<td>77.7</td>
<td>77.7</td>
<td>75.1</td>
<td>77.3</td>
<td>75.1</td>
<td>77.9</td>
<td>78.1</td>
<td>72.8</td>
</tr>
<tr>
<td>Glass</td>
<td>53.8</td>
<td>47.3</td>
<td>77.5</td>
<td>66.0</td>
<td>70.2</td>
<td>50.6</td>
<td>51.2</td>
<td>48.3</td>
</tr>
<tr>
<td>Heart</td>
<td>52.0</td>
<td>61.9</td>
<td>65.2</td>
<td>57.1</td>
<td>63.4</td>
<td>52.8</td>
<td>53.2</td>
<td>59.8</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>86.8</td>
<td>80.1</td>
<td>87.5</td>
<td>88.8</td>
<td>85.9</td>
<td>72.9</td>
<td>64.8</td>
<td>70.7</td>
</tr>
<tr>
<td>Iris</td>
<td>9.9</td>
<td>9.7</td>
<td>16.9</td>
<td>12.8</td>
<td>15.4</td>
<td>9.1</td>
<td>8.6</td>
<td>8.1</td>
</tr>
<tr>
<td>KRK</td>
<td>51.3</td>
<td>62.0</td>
<td>67.9</td>
<td>63.7</td>
<td>66.5</td>
<td>49.0</td>
<td>46.4</td>
<td>45.8</td>
</tr>
<tr>
<td>Multi-ANDOR</td>
<td>49.3</td>
<td>4.4</td>
<td>36.2</td>
<td>34.4</td>
<td>37.5</td>
<td>75.4</td>
<td>10.4</td>
<td>2.0</td>
</tr>
<tr>
<td>Monks1</td>
<td>3.0</td>
<td>1.5</td>
<td>3.0</td>
<td>2.9</td>
<td>3.1</td>
<td>50.9</td>
<td>4.6</td>
<td>1.5</td>
</tr>
<tr>
<td>Monks2</td>
<td>99.6</td>
<td>99.2</td>
<td>100.6</td>
<td>101.8</td>
<td>100.6</td>
<td>93.1</td>
<td>93.1</td>
<td>86.0</td>
</tr>
<tr>
<td>Monks3</td>
<td>5.3</td>
<td>5.1</td>
<td>5.6</td>
<td>5.1</td>
<td>5.2</td>
<td>10.0</td>
<td>5.1</td>
<td>5.1</td>
</tr>
<tr>
<td>Multiplexer</td>
<td>73.1</td>
<td>9.0</td>
<td>79.2</td>
<td>75.9</td>
<td>79.4</td>
<td>79.2</td>
<td>31.6</td>
<td>4.7</td>
</tr>
<tr>
<td>MultiXOR</td>
<td>66.7</td>
<td>3.6</td>
<td>66.7</td>
<td>67.1</td>
<td>68.1</td>
<td>79.8</td>
<td>52.1</td>
<td>4.0</td>
</tr>
<tr>
<td>Nursery</td>
<td>7.5</td>
<td>7.8</td>
<td>9.1</td>
<td>8.1</td>
<td>8.8</td>
<td>15.5</td>
<td>5.1</td>
<td>6.7</td>
</tr>
<tr>
<td>Pima</td>
<td>77.8</td>
<td>86.5</td>
<td>88.2</td>
<td>77.4</td>
<td>89.5</td>
<td>75.8</td>
<td>72.9</td>
<td>78.3</td>
</tr>
<tr>
<td>Tae</td>
<td>88.9</td>
<td>81.6</td>
<td>81.8</td>
<td>81.0</td>
<td>83.2</td>
<td>66.8</td>
<td>83.4</td>
<td>78.2</td>
</tr>
<tr>
<td>Tic-tac-toe</td>
<td>48.2</td>
<td>46.0</td>
<td>55.9</td>
<td>46.7</td>
<td>55.5</td>
<td>58.2</td>
<td>41.5</td>
<td>45.5</td>
</tr>
<tr>
<td>Titanic</td>
<td>65.9</td>
<td>66.8</td>
<td>65.0</td>
<td>65.2</td>
<td>65.0</td>
<td>67.1</td>
<td>65.3</td>
<td>65.3</td>
</tr>
<tr>
<td>Thyroid</td>
<td>17.2</td>
<td>32.8</td>
<td>18.6</td>
<td>17.0</td>
<td>16.9</td>
<td>17.0</td>
<td>16.7</td>
<td>24.2</td>
</tr>
<tr>
<td>Voting</td>
<td>8.4</td>
<td>10.2</td>
<td>16.2</td>
<td>9.4</td>
<td>14.6</td>
<td>7.5</td>
<td>8.0</td>
<td>8.9</td>
</tr>
<tr>
<td>Wine</td>
<td>9.8</td>
<td>13.1</td>
<td>47.6</td>
<td>23.1</td>
<td>40.3</td>
<td>18.9</td>
<td>13.6</td>
<td>14.2</td>
</tr>
<tr>
<td>XOR3d</td>
<td>92.4</td>
<td>41.7</td>
<td>98.2</td>
<td>87.8</td>
<td>97.3</td>
<td>101.7</td>
<td>63.1</td>
<td>45.1</td>
</tr>
<tr>
<td>XOR5</td>
<td>94.5</td>
<td>7.0</td>
<td>111.0</td>
<td>107.6</td>
<td>111.7</td>
<td>93.3</td>
<td>92.9</td>
<td>33.6</td>
</tr>
<tr>
<td>Bupa</td>
<td>93.5</td>
<td>92.2</td>
<td>93.5</td>
<td>88.8</td>
<td>93.5</td>
<td>81.1</td>
<td>85.6</td>
<td>90.4</td>
</tr>
<tr>
<td>Heart</td>
<td>54.1</td>
<td>64.2</td>
<td>59.2</td>
<td>59.4</td>
<td>60.6</td>
<td>62.0</td>
<td>55.9</td>
<td>61.1</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>104.7</td>
<td>97.8</td>
<td>79.3</td>
<td>85.7</td>
<td>85.7</td>
<td>86.2</td>
<td>91.8</td>
<td>86.4</td>
</tr>
<tr>
<td>Pima</td>
<td>78.1</td>
<td>87.3</td>
<td>105.9</td>
<td>92.4</td>
<td>105.9</td>
<td>77.0</td>
<td>72.4</td>
<td>81.0</td>
</tr>
<tr>
<td>Thyroid</td>
<td>14.6</td>
<td>36.0</td>
<td>17.4</td>
<td>14.9</td>
<td>13.6</td>
<td>13.7</td>
<td>14.0</td>
<td>17.9</td>
</tr>
</tbody>
</table>

represent the difficult domains, such as XOR, which ICET could not learn but ACT could.

### 4.6.4 Comparing the Accuracy of the Learned Models

When misclassification costs are low, an optimal algorithm would produce a very shallow tree. When misclassification costs are dominant, an optimal algorithm
would produce a highly accurate tree. As we can see, ACT’s normalized cost increases with the increase in misclassification cost. While it is relatively easy to produce shallow trees, some concepts are not easily learnable and even cost-insensitive algorithms fail to achieve perfect accuracy on them. Hence, as the importance of accuracy increases, the normalized cost increases too because the predictive errors affect it more dramatically.

To learn more about the effect of misclassification costs on accuracy, we compare the accuracy of the built trees for different misclassification costs. Figure 4.9 shows the results. An important property of DTMC, ICET, and ACT is their ability to compromise on accuracy when needed. They produce inaccurate trees when accuracy is insignificant and much more accurate trees when the penalty for error is high. ACT’s flexibility, however, is more noteworthy: from the second least accurate method it becomes the most accurate one.

Interestingly, when accuracy is extremely important, both ICET and ACT achieve even better accuracy than C4.5. The reason is their non-greedy nature. ICET performs an implicit lookahead by reweighting attributes according to their importance. ACT performs lookahead by sampling the space of subtrees before every split. Of the two, the results indicate that ACT’s lookahead is more efficient in terms of accuracy. DTMC is less accurate than C4.5. The reason is the different split selection criterion and the different pruning mechanism.

In comparison to our anytime cost insensitive algorithm LSID3, ACT produced less accurate trees when \( mc \) was relatively low. When \( mc \) was set to 5000, however, ACT achieved comparable accuracy to LSID3 and slightly outperformed it for \( mc = 10000 \). Statistical tests found the differences between the accuracy of the two algorithms in this case to be insignificant. ACT’s small advantage on some of the datasets indicates that, for some problems, expected error is a better heuristic than tree size for maximizing accuracy.

4.6.5 Comparison of Anytime Behavior

Both ICET and ACT, like other typical anytime algorithms, perform better with increased resource allocation. ICET is expected to exploit the extra time by producing more generations and hence better tuning the parameters for the final invocation of EG2. ACT can use the extra time to acquire larger samples and hence achieve better cost estimations.

To examine the anytime behavior of ICET and ACT, we ran them on 4 problems, namely Breast-cancer-20, Monks-1, Multi-XOR, and XOR5, with exponentially increasing time allocation. \( mc \) was set to 5000. ICET was run with 2, 4, 8, \ldots generations and ACT with a sample size of 1, 2, 4, \ldots. As in the fixed-time comparison, we used 4 instances for each problem. Figure 4.10 plots the results averaged over the 4 instances. We also included the results for the greedy
methods EG2 and DTMC.

The results show good anytime behavior of both ICET and ACT: generally it is worthwhile to allocate more time. ACT dominates ICET for the four domains and is able to produce less costly trees in shorter time.

One advantage of ACT over ICET is that it is able to consider the context in which an attribute is judged. ICET, on the contrary, reassigned the cost of the attributes globally: an attribute cannot be assigned a high cost in one subtree and a low cost in another. The Multi-XOR dataset exemplifies a concept whose attributes are important only in one sub-concept. The concept is composed of four sub-concepts, each of which relies on different attributes (see Appendix A for further details). As we expected, ACT outperforms ICET significantly because the latter cannot assign context-based costs. Allowing ICET to produce more and more generations (up to 128) does not result in trees comparable to those obtained by ACT.

### 4.6.6 Random Costs

The costs of 100 out of the 105 datasets were assigned using a semi-random mechanism that gives higher costs to informative attributes. To ensure that ACT’s success is not due to this particular cost assignment scheme, we repeated the experiments with the costs drawn randomly uniformly from the given cost range $cr$, i.e., $\propto$ was set to 0. Figure 4.11 and Table 4.9 show the results. As we can see, ACT maintains its advantage over the other methods: it dominates them along the scale of $mc$ values.

### 4.6.7 Nonuniform Misclassification Costs

So far, we have only used uniform misclassification cost matrices, i.e., the cost of each error type was identical. As explained in Section 4.2, the ACT algorithm can also handle complex misclassification cost matrices where the penalty for one type
Table 4.10: Comparison of C4.5, EG2, DTMC, ACT, and ICET when misclassification costs are nonuniform. $FP$ denotes the penalty for a false positive and $FN$ the penalty for a false negative. $\gamma$ denotes the basic mc unit.

<table>
<thead>
<tr>
<th>$FP$</th>
<th>$\gamma$</th>
<th>$4\gamma$</th>
<th>$2\gamma$</th>
<th>$4\gamma$</th>
<th>$8\gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FN$</td>
<td>$8\gamma$</td>
<td>$4\gamma$</td>
<td>$2\gamma$</td>
<td>$4\gamma$</td>
<td>$8\gamma$</td>
</tr>
<tr>
<td>500</td>
<td>C4.5</td>
<td>29.2</td>
<td>34.2</td>
<td>41.3</td>
<td>49.9</td>
</tr>
<tr>
<td></td>
<td>EG2</td>
<td>30.1</td>
<td>33.0</td>
<td>37.2</td>
<td>42.5</td>
</tr>
<tr>
<td></td>
<td>DTMC</td>
<td>12.4</td>
<td>20.3</td>
<td>29.8</td>
<td>37.7</td>
</tr>
<tr>
<td></td>
<td>ICET</td>
<td>23.3</td>
<td>27.0</td>
<td>31.5</td>
<td>36.3</td>
</tr>
<tr>
<td></td>
<td>ACT</td>
<td>11.9</td>
<td>18.5</td>
<td>27.2</td>
<td>34.5</td>
</tr>
<tr>
<td>5000</td>
<td>C4.5</td>
<td>27.0</td>
<td>31.3</td>
<td>39.2</td>
<td>53.3</td>
</tr>
<tr>
<td></td>
<td>EG2</td>
<td>30.9</td>
<td>35.2</td>
<td>43.1</td>
<td>57.3</td>
</tr>
<tr>
<td></td>
<td>DTMC</td>
<td>13.8</td>
<td>23.6</td>
<td>38.0</td>
<td>57.6</td>
</tr>
<tr>
<td></td>
<td>ICET</td>
<td>21.4</td>
<td>25.6</td>
<td>32.7</td>
<td>45.7</td>
</tr>
<tr>
<td></td>
<td>ACT</td>
<td>12.9</td>
<td>19.1</td>
<td>28.8</td>
<td>41.5</td>
</tr>
</tbody>
</table>

of error might be higher than the penalty for another type. Our next experiment examines ACT in the nonuniform scenario. Let $FP$ denote the penalty for a false positive and $FN$ the penalty for false negative. When there are more than 2 classes, we split the classes into 2 equal groups according to their order (or randomly if no order exists). We then assign a penalty $FP$ for misclassifying an instance that belongs to the first group and $FN$ for one that belongs to the second group.

To obtain a wide view, we vary the ratio between $FP$ and $FN$ and also examine different absolute values. Table 4.10 and Figure 4.12 give the average results. It is easy to see that ACT consistently outperforms the other methods.

Interestingly, the graphs are slightly asymmetric. The reason could be that for some datasets, for example medical ones, it is more difficult to reduce negative errors than positive ones, or vice versa. A similar phenomenon is reported by Turney (1995).

The highest cost for all algorithms is observed when $FP = FN$ because, when the ratio between $FP$ and $FN$ is extremely large or extremely small, the learner can easily build a small tree whose leaves are labeled with the class that minimizes costs. When misclassification costs are more balanced, however, the learning process becomes much more complicated.
Figure 4.8: Illustration of the differences in performance between ACT and ICET for $mc = 100, 1000, 10000$ (from left to right). Each point represents a dataset. The $x$-axis represents the cost of ICET while the $y$-axis represents that of ACT. The dashed line indicates equality. Points are below it if ACT performs better and above it if ICET is better.
Figure 4.9: Average accuracy as a function of misclassification cost

Figure 4.10: Average normalized cost as a function of time for (from top-left to bottom-right) Breast-cancer-20, Monks1, Multi-XOR, and XOR5
Figure 4.11: Average cost when test costs are assigned randomly
Figure 4.12: Comparison of C4.5, EG2, DTMC, ACT, and ICET when misclassification costs are nonuniform. The misclassification costs are represented as a pair \((FP/FN)\). \(FP\) denotes the penalty for a false positive and \(FM\) the penalty for a false negative. \(\gamma\) denotes the basic cost unit. The figures plot the average cost as a function of the ratio between \(FP\) and \(FN\) costs, for \(\gamma = 500\) (top) and \(\gamma = 5000\) (bottom).
Chapter 5

Anytime Learning of Anycost Classifiers

Assume that a hardware manufacturer who uses a machine-learning based tool for assuring the quality of produced chips. In realtime, each chip in the pipeline is scanned and several features can be extracted from the image. The features vary in their computation time. The manufacturer trains the component using thousands of chips whose validity is known. Because the training is done offline, the manufacturer can provide the values of all possible features, regardless of their computation time. In realtime, however, the model must make a decision within 2 seconds. Therefore, for each chip, the classifier may use features whose total computation time is at most 2 seconds. Alternatively, the manufacturer might want to provide the classifier with a different maximal time, depending on the case, or even configuring the classifier to utilize time until the next item arrives and then querying it for a decision.

To act under these different resource-bounded classification scenarios, our framework should produce predictive models that can control testing costs efficiently, and should also be able to exploit additional learning resources in order to improve the produced models.

For the first requirement, i.e., resource-bounded classification, a decision-tree based classifier would make an ideal candidate. When classifying a new case, decision trees ask only for the values of the tests on a single path from the root to one of the leaves. Tests that do not appear on the actual path need not be administered. Decision tree models are also considered attractive due to their interpretability (Hastie et al., 2001), an important criterion for evaluating a classifier (Craven, 1996), their simplicity of use, and their accuracy, which has been shown to be competitive with other classifiers for several learning tasks.

Decision trees, however, cannot be used as is: when the classification budget does not allow exploring the entire path, the tree cannot make a decision. This
problem can be overcome by slightly modifying the learners and storing a default class at each internal node. If classification is stopped early, the prediction is made according to the last explored node. This modification makes decision trees valid active classifiers, i.e., classifiers that given a partially specified instance, return either a class label or a strategy that specifies which test should be performed next (Greiner, Grove, & Roth, 2002).

Formally, let $\rho^c$ be the bound on testing costs for each case. When classifying an example $e$ using a tree $T$, we propagate $e$ down the tree along a single path from the root of $T$ until a leaf is reached or classification resources are exhausted. Let $\Theta(T, e)$ be the set of tests along this path. We denote by $\text{cost}(\theta)$ the cost of administering the test $\theta$. The testing cost of $e$ in $T$ is therefore $t\text{cost}(T, e) = \sum_{\theta \in \Theta(T, e)} \text{cost}(\theta)$. Note that tests that appear several times are charged for only once. The cost of a tree is defined as the maximal cost of classifying an example using $T$. Formally,

$$\text{Test-Cost}(T) = \begin{cases} 0 & T \text{ is a leaf} \\ \text{cost}(\theta_{\text{root}(T)}) + \max_{T \in \text{Children}(T)} \text{Test-Cost}(\hat{T}) & \text{otherwise} \end{cases}$$

Having decided about the model, we are interested in a learning algorithm that can exploit additional time resources, preallocated or not, in order to improve the induced anycost trees. In the previous Chapters we presented two anytime algorithms for learning decision trees. The first, called LSID3 (Chapter 3), can trade computation speed for smaller and more accurate trees, and the second, ACT (Chapter 4), can exploit extra learning time to produce more cost-efficient trees. Neither algorithm, however, can produce trees that operate under given classification budgets; therefore, they are inappropriate for our task. In what follows we introduce Tree-classification AT Anycost (TATA), our proposed anytime algorithm for learning anycost trees. TATA has three learning components, for the three different testing-cost scenarios described in Chapter 2: pre-contract-TATA, contract-TATA, and interruptible-TATA.

### 5.1 Pre-contract: Classification Budget is Known to the Learner

The most common method for learning decision trees is TDIDT (top-down induction of decision trees). TDIDT algorithms start with the entire set of training examples, partition it into subsets by testing the value of an attribute, and then recursively build subtrees. The TDIDT scheme, however, does not take into account testing cost budgets. Therefore, we first describe TDIDT$, a modification for adapting any TDIDT algorithm to the pre-contract scenario, and then
5.1.1 Top-down Induction of Anycost Trees

Any decision tree can serve as an anycost classifier by simply storing a default classification at each node and predicting according to the last explored node. Expensive tests, in this case, may block their subtrees and make them useless. In the pre-contract setup, \( \rho^c \), the maximal testing cost, is known in advance and thus the tree learner can avoid exceeding \( \rho^c \) by considering only tests whose costs are lower. When a choice is made to split on a test \( \theta \), the cost bound for building the subtree under it is decreased by \( \text{cost}(\theta) \). Figure 5.1 exemplifies the recursive invocation of TDIDT$. The initial \( \rho^c \) was $60 (left-hand figure). Since two tests \( (a_2 \text{ and } a_5) \) with a total cost of $30 have been used, their current cost is zero and the remaining budget for the TDIDT$ process is $30. \( a_1 \), whose cost is $40, is filtered out. Assume that, among the remaining candidates, \( a_6 \) is chosen. The subtrees of the new split (right-hand figure) are built recursively with a budget of $20.

TDIDT$ is designed to search in the space of trees whose test-cost is at most
Procedure TDIDT$(E, A, \rho^c)$

If $E = \emptyset$
    Return Leaf(nil)

If $\exists c$ such that $\forall e \in E$ class$(e) = c$
    Return Leaf$(c)$

$\theta \leftarrow$ CHOOSE-TEST$(A, E, \rho^c)$
$V \leftarrow$ OUTCOMES$(\theta)$

Foreach $v_i \in V$
    $E_i \leftarrow \{e \in E \mid \theta(e) = v_i\}$
    $S_i \leftarrow$ TDIDT$(E_i, A, \rho^c - \text{cost}(\theta))$

Return Node$(\theta, \{\langle v_i, S_i \rangle \mid i = 1 \ldots |V|\})$

Figure 5.2: Top-down induction of anycost decision trees. $E$ is the set of examples and $A$ is the set of attributes.

$\rho^c$. It can be instantiated by means of an attribute selection procedure.

### 5.1.2 Greedy TDIDT$ Instantiations

Our first proposed anycost learner, called C4.5$, instantiates TDIDT$ by taking information gain as the split selection criteria, as in C4.5. Unlike the original C4.5, C4.5$ filters out unaffordable tests. It does not, however, consider the costs of the remaining candidates. Assume, for example, that the cost of the attribute with the highest information gain is exactly $\rho^c$, the maximal classification budget. C4.5$ will choose to split on it and will not be able to further develop the tree.

The EG2 algorithm (Nunez, 1991) chooses attributes based on their Information Cost Function, ICF$(\theta) = (2^{\Delta I(\theta)} - 1)/(\text{cost}(\theta) + 1)$, where $\Delta I$ is the information gain. We denote by EG2$ an instantiation of TDIDT$ that uses ICF. EG2$ does not use $\rho^c$ when evaluating splits and therefore might over-penalize informative yet costly attributes, even when $\rho^c$ is large enough. This problem is intensified when the immediate information gain of relevant attributes is low, for example due to inter-correlation as in XOR-like concepts. Both C4.5$ and EG2$ require a fixed short runtime and cannot exploit additional resources to improve the resulting classifier.

### 5.1.3 The Pre-Contract-TATA Algorithm

Our ACT algorithm (Chapter 4) overcomes the problems greedy cost-sensitive learners encounter by allowing learning time to be traded for lower total classification costs. ACT is an anytime TDIDT algorithm that invests more time resources for making better split decisions. ACT attempts to minimize the sum
of the testing and misclassification costs. However, it does not consider the maximal classification budget and may violate testing cost limits. Therefore, it is inappropriate for building pre-contract anycost classifiers. TATA, our proposed anytime learner, has a different goal: minimizing the misclassification costs given a bound on the testing costs. Hence, TATA must take a different approach for (1) top-down induction, (2) pruning, (3) biasing the sample, and (4) evaluating trees in the sample.

While ACT adopts the TDIDT approach, in TATA we use TDIDT$. This carries 2 benefits: first, it guarantees that the produced tree will not violate the maximal cost, and second, it filters out some of the attributes and saves their evaluation time, which can be costly in the anytime setup. Once the tree is built, ACT post-prunes it in an attempt to reduce testing and misclassification costs. In TATA reducing testing costs is not beneficial because the built tree already fits the budget. Therefore, the objective of TATA’s post-pruning is to reduce misclassification costs, regardless of testing costs. When misclassification costs are uniform, this problem reduces to maximizing accuracy and thus we adopt C4.5’s error-based pruning (EBP). When misclassification cost is not uniform, we slightly modify EPB, and prune if the expected misclassification costs decrease (rather than the expected error).

Like ACT, TATA samples the spaces of subtrees under each split to estimate its utility. ACT uses SEG2 to bias the sample towards low-cost trees. In TATA, however, we would like to bias our sample towards accurate trees that fit our testing costs budget. Therefore, instead of using SEG2, we designed a stochastic version of C4.5$, called SC4.5$.$^1$ In SC4.5$ attributes are chosen semi-randomly, proportionally to their information gain. Note that the cost of the attributes affects the split decisions twice: first, TDIDT$ filters out the unaffordable tests, and second, the sampled trees themselves must fit the remaining budget, which penalizes expensive tests when the budget is relatively low. We also tried to sample using SEG2 and a stochastic version of DTMC. However, these cost-sensitive samplers did not yield any benefit and in some cases over-punished features with high costs.

Having decided about the sampler, we need to decide how to evaluate the sampled trees. ACT favors trees whose expected total cost is lowest. In TATA all sampled trees fit the budget and therefore we choose to split on the attribute whose subtree is expected to have the lowest misclassification cost, as exemplified in Figure 5.3.

Figure 5.4 formalizes the split selection component in pre-contract-TATA. Pre-contract-TATA is an anytime algorithm parameterized by $r$, the sample size. For a given set of examples and attributes, the runtime complexity of TATA grows

$^1$Unlike the SID3 algorithm used by LSID3 (Chapter 3), SC4.5 and SC4.5$ post-prune the induced trees.
Figure 5.3: Attribute evaluation in Pre-Contract-TATA. E-MC stands for the expected misclassification cost. For each candidate split, we sample the space of trees under it that fit the remaining budget ($60 in the example) and evaluate the split by the minimal expected misclassification cost in the sample ($110 in the example).

linearly with $r$, just as it does in ACT (Esmeir & Markovitch, 2007a). When we cannot afford sampling ($r = 0$), TATA builds the tree using C4.5$.

5.1.4 Interruptible Learning of Pre-contract Classifiers

The algorithm presented in Section 5.1.3 requires $r$, the sample size, as a parameter. When the learning resources are not predetermined, we would like the learner to utilize extra time until interrupted. In Chapter 3 we presented IIDT, a general framework for Interruptible Induction of Decision Trees, that need not be allocated resources ahead of time. IIDT starts with building a greedy tree. Then, it repeatedly selects a subtree whose reconstruction is expected to yield the highest marginal utility, and rebuilds the subtree with a doubled allocation of resources.

The same iterative improvement approach can be applied to convert pre-contract-TATA into an interruptible algorithm. The initial greedy tree would be built with C4.5$, and subtree reconstructions would be made using pre-contract-TATA. The marginal utility of constructing a tree would take into account both the expected misclassification cost of the tree and the expected resources required
Figure 5.4: Attribute selection in pre-contract-TATA. \textsc{ExpectedMC}(T) returns the expected misclassification cost of $T$.

5.2 Contract: When Allocation is Made Right

Before Classification

The pre-contract classification scenario assumes that $\rho^c$, the bound on testing costs, is known to the learner. In many real-life scenarios, however, we do not know $\rho^c$ before building the model and therefore we need classifiers that either get $\rho^c$ as a parameter before proceeding with classification (contract classification) or can do their best until stopped and queried for a decision (interruptible classification). Note that TDIDT$\$\$-based algorithms cannot be used as is because $\rho^c$ is unavailable at the time of learning. Obviously, C4.5, EG2, and ACT, can be slightly modified, by storing default classifications at each internal node, to produce contract and interruptible trees because they do not need the value of $\rho^c$. However, they are not designed to exploit a given testing budget. Therefore, we are looking for a learner that has the advantages of pre-contract-TATA without getting the value of $\rho^c$ as parameter.
Procedure CONTRACT-TATA-UNIFORM-LEARN($E, A, r, k$)

$\Pi \leftarrow \langle \rangle$

$\rho_{\text{c min}} \leftarrow \text{MINIMAL-ATTRIBUTE-COST}(A)$

$\rho_{\text{c max}} \leftarrow \text{TOTAL-ATTRIBUTE-COSTS}(A)$

\textbf{foreach } $\rho^c \in \bigcup_{i=0}^{k-1} \{ \rho_{\text{c min}} + \frac{i}{k-1} (\rho_{\text{c max}} - \rho_{\text{c min}}) \}$

$T \leftarrow \text{PRE-CONTRACT-TATA}(E, A, r, \rho^c)$

$\hat{\rho}^c \leftarrow \text{TEST-COST}(T)$

\textbf{insert-sorted}($\Pi, \langle \hat{\rho}^c, T \rangle$)  \hspace{1em} // insert pair to $\Pi$, sorted by $\hat{\rho}^c$

\textbf{return } $\Pi$

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5_5.png}
\caption{Building a repertoire in contract-TATA with uniform cost gaps}
\end{figure}

5.2.1 Repertoire of Trees

A good contract classifier must be prepared to utilize any $\rho^c$ it gets. For this purpose we suggest using pre-contract-TATA and build a collection of $k$ trees, each with a different value of $\rho^c$. We call such a collection a repertoire. In pre-contract-TATA we exploit extra learning resources in order to increase $r$, the sample size, and thus improve attribute-utility estimations. When composing a repertoire we can benefit from additional learning time either for building trees of higher quality or for increasing the granularity of the repertoire. The values of $k$ and $r$ impose a tradeoff between tree quality and granularity. Larger values of $r$ mean better trees. Larger values of $k$ mean less time to invest in each tree and higher memory requirements but increase the chances of finding a tree that fits the allocation. These chances depend also on the selection of $\rho^c$ values. Let $\rho_{\text{c max}}$ be the cost of taking all tests and let $\rho_{\text{c min}}$ be the cost of the least expensive attribute. Obviously, there is no need to build trees with $\rho^c > \rho_{\text{c max}}$ or $\rho^c < \rho_{\text{c min}}$. Therefore, we uniformly distribute the $k$ values in the range $[\rho_{\text{c min}} - \rho_{\text{c max}}]$, where the difference in the cost contract of each two successive trees is \( \delta = \frac{2 \rho_{\text{c max}} - \rho_{\text{c min}}}{k-1} \).

Unlike ensembles that combine predictions from several trees, the final decision of a repertoire classifier is based on a single tree. In the contract setup, classification resources are pre-allocated and therefore we pick a tree from the repertoire that best fits the cost bound. Given a repertoire $B$ and a classification budget $\rho^c$, we would use the $(\frac{\rho - \rho_{\text{c min}}}{\delta} + 1)^{th}$ tree in the repertoire. However, a tree does not necessarily use the maximal cost it is allowed to use. Therefore when a tree is formed, we store it along with its actual maximal cost $\hat{\rho}^c$. In the classification phase, we choose the tree associated with the largest $\hat{\rho}^c$ that does not exceed $\rho^c$.

Figure 5.5 formalizes the procedures for forming a repertoire with uniform cost gaps. Figure 5.6 formalizes the procedures for using a repertoire to classify
Procedure CONTRACT-TATA-CLASSIFY($\Pi, e, \rho^c$)

$\rho^{c*} \leftarrow \max\{\rho^{c<} \mid \rho^{c<} \leq \rho^c \land \exists T, (\rho^{c<}, T) \in \Pi\}$

$T^* \leftarrow T \mid (\rho^{c*}, T) \in \Pi$

Return $\text{CLASSIFY}(T^*, e)$

Figure 5.6: Classifying a case using a repertoire in contract-TATA

a case under the contract setup.

5.2.2 Learning a repertoire with Nonuniform Cost Gaps

In repertoires with uniform cost distribution, every two successive trees are built with maximal cost allocations that differ by $\delta$. In many domains, however, increasing the contract by $\delta$ results in the same tree or a tree that performs similarly. In such a case, the efforts to build the second tree are wasted. These efforts could have been invested in more interesting cost regions where increasing the contract is useful. This problem is intensified when $k$, the repertoire size, is small. Consider, for example, a problem with 4 attributes, 3 of which cost $\$1$ and 1 of which costs $\$13$. In this case, $\rho^{c\min} = 1$ and $\rho^{c\max} = 16$. Assume that we would like to build a repertoire of size $k = 4$. Because $\delta = 5$, the four trees will be built with $\rho^c = 1, 6, 11, 16$ respectively. As a result, the 2nd and the 3rd trees would be identical because they can test the same set of attributes (all but the expensive one).

Another problem with the uniform approach is the requirement that $k$, the repertoire size, be decided in advance. This is affordable when learning resources are predetermined (contract learning) but imposes a problem if the learning process itself needs to be interruptible. For example, in many real-world problems learning resources are not preallocated; rather, the learning algorithm is expected to exploit any additional time until interrupted.

To overcome the aforementioned shortcomings, we propose a different approach for choosing the sequence of $\rho^c$ values in a repertoire. Rather than uniform gaps, we propose an iterative improvement solution. We start with a repertoire that consists of two trees: one built with $\rho^{c\min}$ and the other with $\rho^{c\max}$. We then repeatedly choose two successive trees, $T_1$ and $T_2$, built with $\rho^c_1$ and $\rho^c_2$ respectively, and add another tree with $\rho^c = 0.5(\rho^c_1 + \rho^c_2)$.

Choosing $\rho^c_1$ and $\rho^c_2$ is not trivial. We are interested in maximizing the benefit from adding a new tree. Therefore, $\rho^c_2 - \rho^c_1$ should be large enough to cover a wide range of potential $\rho^c$ values. At the same time, the expected reduction in misclassification costs, if a tree is built in between, should be significant. To estimate the latter we use the expected errors of the built trees. Two successive
Procedure **CONTRACT-TATA-HILL-LEARN**\((E, A, r, k)\)
\[
\begin{align*}
\rho_{\text{min}} & \leftarrow \text{MINIMAL-ATTRIBUTE-COST}(A) \\
\rho_{\text{max}} & \leftarrow \text{TOTAL-ATTRIBUTE-COSTS}(A) \\
T_{\text{min}} & \leftarrow \text{PRE-CONTRACT-TATA}(E, A, r, \rho_{\text{min}}) \\
T_{\text{max}} & \leftarrow \text{PRE-CONTRACT-TATA}(E, A, r, \rho_{\text{max}}) \\
\Pi & \leftarrow \langle \langle \rho_{\text{min}}, T_{\text{min}} \rangle, \langle \rho_{\text{max}}, T_{\text{max}} \rangle \rangle \\
\text{While } |\Pi| < k & \\
& \qquad i \leftarrow \arg \max_{i=1, \ldots, |\Pi|-1} (\text{EE}(T_i) - \text{EE}(T_{i+1})) \cdot (\rho_{i+1} - \rho_i) \\
& \qquad \text{INSERT-SORTED}(B, \langle \frac{\rho_i + \rho_{i+1}}{2}, \text{PRE-CONTRACT-TATA}(E, A, r, \frac{\rho_i + \rho_{i+1}}{2}) \rangle) \\
\text{Return } \Pi
\end{align*}
\]

**Figure 5.7:** Building a repertoire in contract-TATA using the hill-climbing approach

Procedure **INTERRUPTIBLE-CLASSIFY**\(\Pi, e\)
\[
\begin{align*}
\text{While } \text{NOT-INTERRUPTED} & \\
& \qquad T \leftarrow \text{NEXT-TREE}(\Pi) \\
& \qquad l \leftarrow \text{CLASSIFY}(T, e) \\
\text{Return } l
\end{align*}
\]

**Figure 5.8:** Using repertoires in interruptible-TATA

trees whose expected errors vary significantly indicate that it may be worthwhile to build a tree in between them. Combining these two criteria, we choose \(T_1\) and \(T_2\) that maximize \((\rho_2 - \rho_1) \cdot (\text{EE}(T_2) - \text{EE}(T_1))\). In this case, setting \(\rho^*\) of the new tree to 0.5(\(\rho_1 + \rho_2\)) is optimal. We refer to this method as a hill-climbing repertoire. This process can be repeated until the learner is interrupted, or until building a predetermined number of trees. Figure 5.7 formalizes the procedure for building hill-climbing repertoires.

Similarly to the uniform-gaps case, each tree is stored along with its actual maximal classification cost. When classifying an instance, we choose the tree that best fits the given classification limit.

### 5.3 Interruptible: No Pre-determined Classification Budget

In the interruptible setup, not only is the classification budget not provided to the learner but it is not provided to the classifier. The classifier is expected to utilize resources until it is interrupted and queried for class label. To operate
in such a setup we again form repertoires. Unlike the contract setup, however, $\rho^c$ is available to the classifier. Therefore, we cannot choose a single tree for classification.

One solution is to form a repertoire with uniform gaps, similarly to the contract setup (see Figure 5.5). When we need to classify an instance, we start by using the lowest-cost tree, which is built with $\rho^c = \rho_{\text{min}}^c$. We then move on to the next tree, as long as resources allow. When interrupted, we use the prediction of the last fully explored tree as the final decision. Figure 5.8 formalizes the classification procedure under the interruptible setup.

While simple, this approach raises two important questions. First, because we are charged for the cumulative cost of using the trees, it is no longer ideal to have as many trees in the repertoire as possible. Second, when forming the repertoire, the learner should take into account tests that appear in previous trees in the repertoire because their outcome might have been already known when using the tree for classification.

Deciding about the repertoire size is not trivial. Even if we are not limited in our learning resources, we would like to limit the number of trees in the repertoire to avoid wasting classification resources on trees that will not affect the final decision. On the other hand, too small repertoires, may be insufficient to optimally exploit the available resources. Obviously, the optimal size depends much on the characteristics of the problem. In our experimental study we will consider using a fixed number of trees. In the future, we intend to analyze this problem theoretically and find the optimal value. This task is even more challenging because of the second issue (retaking tests).

Once a tree is used for classification and the relevant tests in the decision path are administered, we do not charge for the same test in the subsequent trees because its outcome is already at hand. The learner could use this information to improve the structure of the repertoire. Assume, for example, that the value of an attribute $a_1$ is measured at the root node of the first tree in the repertoire. Subsequent trees that ask for $a_1$’s value are not charged. Therefore, when building the subsequent trees, the cost of $a_1$ should be treated as zero. Nevertheless, given a case to classify, not all tests that appear in earlier trees are necessarily administered, but only those that appear in decision paths that have been followed. Obviously, one cannot know in advance which tree paths will be followed because this depends on the feature values of the actual case.

To handle this situation we take a probabilistic approach. We assume that future test cases will come from the same distribution as the training examples. Let $T_1, \ldots, T_i$ be the trees that have already been built. When building $T_{i+1}$, we discount the cost of each attribute by the probability of examining its value at least once in $T_1, \ldots, T_i$. For an attribute $a$, we measure this probability by the percentage of training examples whose $a$ value would have been queried by
Figure 5.9: An example of applying cost discounts when forming a repertoire for interruptible classification. The numbers represent the number of examples that follow each edge. Because the value of $a_1$ is required at the root of $T_1$, any subsequent tree can obtain this value at no cost. The probability for testing $a_2$ is 50% in $T_2$. When inducing $T_3$, the attribute $a_2$ has already been measured with a probability of 50%. Hence, we discount the cost of $a_2$ by 50% ($5 instead of $10). Similarly, the cost of $a_3$ is discounted by 80% ($2 instead of $10). The numbers represent the number of examples that follow each edge. Because the value of $a_1$ is required at the root of $T_1$, any subsequent tree can obtain this value at no cost. The probability for testing $a_2$ is 50% in $T_2$. When inducing $T_3$, the attribute $a_2$ has already been measured with a probability of 50%. Hence, we discount the cost of $a_2$ by 50% ($5 instead of $10). Similarly, the cost of $a_3$ is discounted by 80% ($2 instead of $10). Because the trees may be strongly correlated, we cannot simply calculate this probability independently for each tree. For example, if $T_3$ in the aforementioned example tests $a_2$ for 70% of the examples, we would like to know for how many of these examples $a_2$ has been tested also in $T_2$. Therefore, we traverse the previous trees with each of the training examples and mark the attributes that are tested at least once. For efficiency, the matrix that represents which tests were administered for which case is built incrementally and updated after building each new tree.

We refer to this method as discount repertoire. The repertoire is formed using the same method in Figure 5.5 with a single change: before building each tree, cost discounts are applied; the discounts are based on the trees already in the repertoire. Figure 5.10 formalizes the procedure for updating test costs. During classification we iterate over the trees until interrupted, as described in Figure 5.8.

5.4 Empirical Evaluation

A variety of experiments were conducted to test the performance and behavior of TATA in 3 different setups: pre-contract, contract, and interruptible. In Chapter
Figure 5.10: Procedure for applying discounts when forming discount repertoires for interruptible classification

4, we presented an automatic method for assigning testing costs to attributes in existing datasets. We applied this method 4 times on 20 UCI (Asuncion & Newman, 2007) problems\textsuperscript{2} and another 5 datasets that hide hard concepts and have been used in previous machine learning literature. Table 5.1 summarizes the basic properties of these datasets while Appendix A describes them in more details.\textsuperscript{3}

Following the recommendations of Bouckaert (2003), 10 runs of a 10-fold cross-validation experiment were conducted for each dataset and the reported results are averaged over the 100 individual runs.

5.4.1 Pre-Contract Classification

Our first set of experiments compares C4.5, EG2, EG2$, TATA(\(r = 0\)), which is equivalent to C4.5$, and TATA(\(r = 5\)) in the pre-contract setup. Misclassification has been set uniformly to 100.\textsuperscript{4} For each dataset we invoked the algorithms 30 times, each with a different \(\rho^c\) value taken from the range \([0, 120\% \rho^c_{\text{max}}]\), with uniform steps. Figure 5.11 describes the misclassification cost of the different algorithms, as a function of \(\rho^c\). For each point (\(\rho^c\) value), the results are averaged over the 100 datasets.\textsuperscript{5}

Clearly, TATA(\(r = 5\)) is dominant. When \(\rho^c \leq \rho^c_{\text{min}}\), the algorithms cannot

\begin{procedure}
\begin{algorithmic}
\Procedure{Apply-Discount}(E, A, \Pi)\EndProcedure
\State \(M_{i,j} \leftarrow 0\)
\ForAll{\(e \in E\)}
\ForAll{T \in \Pi}
\State \(\hat{A} \leftarrow \text{attributes whose values tested by } T \text{ when classifying } e\)
\EndFor
\ForAll{a \in \hat{A}}
\State \(M_{e,a} \leftarrow 1\)
\EndFor
\EndFor
\EndProcedure
\end{algorithmic}

\textbf{Procedure Apply-Discount}(E, A, \Pi)
\State \(M_{i,j} \leftarrow 0\)
\ForAll{\(e \in E\)}
\ForAll{T \in \Pi}
\State \(\hat{A} \leftarrow \text{attributes whose values tested by } T \text{ when classifying } e\)
\EndFor
\ForAll{a \in \hat{A}}
\State \(M_{e,a} \leftarrow 1\)
\EndFor
\EndFor
\end{procedure}

\begin{equation}
p_a \leftarrow \frac{\sum_{e=1}^{\mid E \mid} M_{e,a}}{\mid E \mid}
\end{equation}

\begin{equation}
cost(a) \leftarrow \text{cost}(a) \cdot (1 - p_a)
\end{equation}

\textsuperscript{2}The datasets vary in size, type of attributes, and dimension.
\textsuperscript{3}The 4X25 datasets are available at http://www.cs.technion.ac.il/~esaher/publications/cost.
\textsuperscript{4}Note that the absolute value of the misclassification cost does not matter because we do not assume same-scale.
\textsuperscript{5}The full results are available at http://www.cs.technion.ac.il/~esaher/publications/rbc.
Table 5.1: Characteristics of the datasets used to evaluate TATA

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Instances</th>
<th>Attributes</th>
<th>Max domain</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast Cancer</td>
<td>277</td>
<td>9 (3)</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>Bupa</td>
<td>345</td>
<td>0 (0)</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>Car</td>
<td>1728</td>
<td>6 (0)</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Flare</td>
<td>323</td>
<td>10 (5)</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Glass</td>
<td>214</td>
<td>0 (0)</td>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>Heart</td>
<td>296</td>
<td>8 (4)</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>154</td>
<td>13(13)</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Iris</td>
<td>150</td>
<td>0 (0)</td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>KRK</td>
<td>28056</td>
<td>6(0)</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>Monks-1</td>
<td>124+432</td>
<td>6 (2)</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Monks-2</td>
<td>169+432</td>
<td>6 (2)</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Monks-3</td>
<td>122+432</td>
<td>6 (2)</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Multiplexer-20</td>
<td>615</td>
<td>20(20)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Multi-XOR</td>
<td>200</td>
<td>11 (11)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Multi-AND-OR</td>
<td>200</td>
<td>11 (11)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Nursery</td>
<td>8703</td>
<td>8(8)</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Pima</td>
<td>768</td>
<td>0(0)</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td>TAE</td>
<td>151</td>
<td>4(1)</td>
<td>1</td>
<td>26</td>
</tr>
<tr>
<td>Tic-Tac-Toe</td>
<td>958</td>
<td>9 (0)</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Titanic</td>
<td>2201</td>
<td>3(2)</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Thyroid</td>
<td>3772</td>
<td>15(15)</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Voting</td>
<td>232</td>
<td>16 (16)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Wine</td>
<td>178</td>
<td>0 (0)</td>
<td>13</td>
<td>-</td>
</tr>
<tr>
<td>XOR 3D</td>
<td>200</td>
<td>0 (0)</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>XOR-5</td>
<td>200</td>
<td>10 (10)</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

administer any test and thus their performance is identical. At the other end, when \( \rho \geq \rho_{\text{max}} \), the attribute costs are actually not a constraint. In this case TATA\((r = 5)\) performed best, confirming the results reported in Chapter 4 when misclassification costs were dominant. The more interesting \( \rho \) values are those in between. Table 5.2 lists the normalized area under the misclassification cost curve over the range \([33\% - 99\%] \rho_{\text{max}}\).Confirming the curves, the results indicate that TATA\((r = 5)\) has the best overall performance. The Wilcoxon test (Demsar, 2006), which compares classifiers over multiple datasets, finds TATA\((r = 5)\) to be significantly better than all the other algorithms.

As expected, all five algorithms improve with the increase in \( \rho \) because they can use more features. For \( \rho \) values slightly larger than \( \rho_{\text{min}} \), we can see that EG2, which is cost-sensitive, performs better than C4.5. The reason is that EG2 takes into account attribute costs and hence will prefer lower cost attributes. With the increase in \( \rho \) and the relaxation of cost constraints, C4.5 becomes better than EG2.
Figure 5.11: Results for pre-contract classification: the misclassification cost for different preallocated testing costs, as percentage of the total cost. The results are averaged over all 100 datasets.

Table 5.2: Comparing the misclassification cost for different testing cost contracts. The numbers represent the average over 100 datasets. The last column indicates whether the advantage of TATA($r = 5$) is statistically significant according to the Wilcoxon test ($\alpha = 5\%$).

<table>
<thead>
<tr>
<th>Learner</th>
<th>$\int_{0.3\rho_{c,\text{max}}}^{\rho_{c,\text{max}}} \text{MC}(\rho) , d\rho$</th>
<th>Wilcoxon</th>
</tr>
</thead>
<tbody>
<tr>
<td>TATA($r = 5$)</td>
<td>21.12</td>
<td>✔</td>
</tr>
<tr>
<td>C4.5</td>
<td>28.84</td>
<td></td>
</tr>
<tr>
<td>TATA($r = 0$)</td>
<td>26.93</td>
<td>✔</td>
</tr>
<tr>
<td>EG2</td>
<td>31.21</td>
<td>✔</td>
</tr>
<tr>
<td>EG2$</td>
<td>30.48</td>
<td>✔</td>
</tr>
</tbody>
</table>

It is interesting to compare the TDIDT$\$ variants of C4.5 and EG2 to their TDIDT counterparts. It is easy to see that both TDIDT$\$ variants exhibit better anycost behavior, until the point where all relevant attributes can be used ($\rho^c \sim \rho_{c,\text{max}}^c$), where the performance of each couple becomes identical. The advantage of the TDIDT$\$ variants is because they will not choose tests that violate the cost limits and therefore will not be forced to stop the induction process earlier.

A comparison of TATA($r = 0$) to TATA($r = 5$) indicates that the latter is clearly better: while TATA($r = 0$) chooses split attributes greedily, TATA($r = 5$)
Figure 5.12: Results for pre-contract classification: the misclassification cost as a function of the preallocated testing costs contract for one instance of Glass (upper-left), AND-OR (upper-right), MULTI-XOR (lower-left) and KRK (lower-right).

samples the possible subtrees of each candidate and bases its decision on the quality of the sampled trees. Of course, the advantage of TATA\( (r=5) \) comes at a price: it uses far more learning resources.

Figure 5.12 gives the results for 4 individual datasets, namely Glass, AND-OR, MULTI-XOR and KRK. In all 4 cases TATA is dominant with its \( r=5 \) version being the best method. We can see that the misclassification cost decreases almost monotonically. The curves, however, are less smooth than the average result from Figure 5.11 and slight degradations are observed. The reason could be that irrelevant features become available and mislead the learner. The KRK curve looks much like steps. The algorithms improve at certain points. These points represent the budgets when the use of another relevant attribute becomes possible. The graphs of AND-OR and MULTI-XOR do not share this phenomenon because these concepts hide interdependency between the attributes. As a result, an attribute may be useless if other attributes cannot be considered. The performance of the greedy C4.5 and TATA\( (r=0) \) on these problems is very poor.
Figure 5.13: TATA with different sample sizes on the Multi-XOR dataset

Besides being able to produce good anycost trees, TATA itself is an anytime algorithm that can trade learning resources for producing better trees. Our next experiment examines the anytime behavior of TATA by invoking it with different values of $r$. Figure 5.13 shows the results. As we can see, all TATA versions are better than C4.5. With the increase in $r$, the advantage of TATA increases. The most significant improvement is from $r = 0$ to $r = 1$. TATA with $r = 0$ is a greedy algorithm that uses local heuristics. When $r > 0$, TATA uses sampling techniques to evaluate attributes and therefore becomes significantly better. For more difficult concepts, where only a combination of a large number of attributes yields information, a larger value of $r$ would be needed.

5.4.2 Contract Classification

TATA uses repertoires to operate in the contract and interruptible classification setups. To examine the anycost behavior of TATA in the contract setup we built 3 repertoires, each of size 16. The first repertoire uses \textit{pre-contract-TATA}(r = 0) to induce the trees, with uniform contract gaps. The second repertoire uses \textit{pre-contract-TATA}(r = 3) to induce the trees, with uniform contract gaps. The trees in the third repertoire are also formed using \textit{pre-contract-TATA}(r = 3), but with the hill-climbing approach instead of uniform gaps.

The repertoires were used to classify examples in the contract setup. 120 uniformly distributed $\rho$ values in the range $[0 – 120\%\rho_{max}]$ were used as contract parameters. Figure 5.14 describes the performance of these three repertoires on 4 datasets (Glass, AND-OR, MULTI-XOR, and KRK), averaged over the 100 runs of 10 times 10-fold cross-validation. In addition, we report the results of the
It is easy to see that across all 4 domains Uni- and Hill-TATA($r = 3$) are dominant. Uniform-TATA($r = 0$) is better than C4.5 when the provided contracts are low. When the contracts can afford using all the attributes, both algorithms perform similarly. In comparison to Uniform-TATA($r = 0$), the anycost behavior of Uniform-TATA($r = 3$) is better: it is monotonic and utilizes testing resources better.

The differences in performance between Uniform- and Hill-TATA($r = 3$) are interesting. While both algorithms exhibit similar trends, Hill-TATA reaches better results slightly earlier than Uniform-TATA on 3 out of the 4 domains (with the exception of KRK). The reason is that Hill-TATA selects the series of $\rho$’s heuristically, rather than by means of blind uniform gaps. As a result, it can focus on cost ranges where it is worthwhile to build more trees. These differences are expected to diminish when the repertoires are larger, which enables Uniform-TATA to cover more contracts. To verify this hypothesis, we repeated the experiments with $k = 32$ and indeed the performance differences between the
methods disappeared. It is important to note, however, that while Uniform-TATA is a contract learner that requires \( k \) in advance, Hill-TATA is an interruptible learner and is therefore appropriate also for cases where the learning resources are not preallocated.

In Section 5.4.1 we examined the anytime behavior of the learner in the pre-contract setup. The results indicate that the misclassification costs decrease with the increase in the sample size (and hence learning resources). In the contract setup, given a fixed learning time, increasing the sample size comes at a price: reducing the number of trees in the repertoire. An interesting question is whether one should invest more resources in building better single trees or in forming larger repertoires? To investigate this, we learned several repertoires using the hill-climbing approach. The trees of each repertoire were induced with a different \( r \) parameter for pre-contract-TATA, from 0 up to 7. When \( r = 0 \), pre-contract-TATA behaves like the greedy C4.5$. In this case we assumed that an infinite number of trees can be built (in practice a tree was built for every tested value of \( \rho \)).

Because we used the hill-climbing approach, we could stop the learning process at any time. We chose three different stopping points: 1 seconds, 3 seconds, and 5 seconds. We tested the performance of these \( 8 \times 3 \) repertoires. Figure 5.15 gives the results. Each curve stands for a different time allocation. The first plot gives the normalized AUC in the range \( \rho = 33\% - 99\% \rho_{\text{max}} \).

It is easy to see that in all three graphs, increasing the learning time allows the production of more trees: the curve for \( T = 5 \) is lower than that of \( T = 3 \)
Figure 5.16: Results for interruptible classification: the misclassification cost as a function of the interruption costs for Glass (upper-left), AND-OR (upper-right), MULTI-XOR (lower-left) and KRK (lower-right).

and $T = 1$. Interestingly, for each of the fixed-time curves, a U shape is observed. On the one hand, too small samples may not be sufficient to learn the concept perfectly. On the other hand, increasing $r$ means more time per tree and thus results in smaller repertoires, without covering significant ranges of $\rho^c$. In the extreme case, for example $T = 1, r > 3$, it is impossible to build even a single tree and the error-rate is 50%, as is the base-line error. We can see that the optimal value of $r$ differs from one time allocation to another.

5.4.3 Interruptible Classification

In the interruptible classification setup, TATA forms a repertoire of trees and traverses it until interrupted. Unlike the contract setup, here we would like to limit the number of trees in the repertoire to avoid wasting resources on tests that will not be needed by the tree that makes the final decision. This would be the case even if we had infinite learning resources. However, too small repertoires may lead to poor performance in significant $\rho^c$ ranges. To examine the anycost
behavior of TATA in the interruptible setup, we built three different repertoires:

- Repertoire of 3 TATA\( (r = 0)\) trees with uniform distribution learned with attribute discounts.
- Repertoire of 3 TATA\( (r = 3)\) trees with uniform distribution learned with attribute discounts.
- Repertoire of 16 TATA\( (r = 3)\) trees with uniform distribution learned with attribute discounts.

The repertoires were used to classify examples in the interruptible setup. In addition we report the results of C4.5. 120 uniformly distributed \( \rho_c \) values in the range \([0−120\%\rho_{max}]\) were used as interruption triggers. Figure 5.16 describes the performance of these repertoires on 4 datasets (Glass, AND-OR, MULTI-XOR, and KRK), averaged over the 100 runs of 10 times 10-fold cross-validation.

As we can see, TATA\( (r = 3)\) variants achieve the best performance along a wide range of \( \rho_c \) values. Interestingly, for some cases (e.g., MULTI-ANDOR) it is better to form smaller repertoires while for other cases (e.g., Glass) larger repertoires yield better results. Furthermore, in the MULTI-XOR problem \( k = 16 \) is better at the beginning, then at some point \( k = 3 \) becomes better, and finally they become almost equal. The reason for this phenomenon is that the \( k = 3 \) repertoire does not contain trees suitable for lower \( \rho_c \) values. The \( k = 16 \) repertoire, on the other hand, spends too many resources on cheap trees, and hence it’s performance when \( \rho_c \) increases is weaker than \( k = 3 \). At some point, however, both reach the same performance because they have administered all the relevant tests.
Chapter 6

Related Work

While to the best of our knowledge no other work has tried to design an anytime algorithm for learning anytime decision trees in particular, there are several related works that warrant discussion here. We consider first works that deal with non-greedy, cost-insensitive decision tree inducers. We then discuss algorithms for anytime induction of other cost-insensitive models. Next we discuss works on cost-sensitive and resource-bounded classification, and finally we consider methods that focus on aspects of the learning process other than the model-induction phase.

6.1 Non-greedy Cost-insensitive Decision Tree Inducers

Several other algorithms for induction of single, cost-insensitive decision trees can either be considered anytime algorithms or can be converted into them with relative ease. We view the recently introduced skewing approach as the most relevant to our LSID3. Therefore, we focus first on the differences between skewing and our anytime approach, and then we consider other cost-insensitive tree learners.

6.1.1 Skewing

The skewing approach was presented as an efficient alternative to lookahead (Page & Ray, 2003). In our discussion, we analyze the original skewing algorithm, which is applicable only to binary attributes. This analysis, however, holds also for the other versions of skewing (see Section 3.7.6), since they are based on the same ideas.

Skewing relies on the hypothesis that, when learning a hard concept, it may be easier for a greedy decision tree inducer to pick a relevant attribute if the
distribution over the data is significantly different from the uniform distribution. Therefore, at each node, skewing repeatedly attempts to produce, from the existing set of examples, skewed versions that represent different distributions. Several different weight vectors are repeatedly assigned to the examples for this purpose. Then, the information gain of each attribute is calculated on the basis of each weight vector. The attribute that exceeds a pre-determined gain threshold for the greatest number of times is chosen. In order to generate the weight vectors, a favored value \( v_i \) is randomly drawn for each candidate attribute \( a_i \). Then, the weight of each example in which \( a_i \) takes the value \( v_i \) is increased.

This process can be viewed as a stochastic clustering of the examples according to the values they take for a subset of the attributes: examples that most agree with the favored value for a large number of attributes are assigned the highest weight. Therefore, the calculation of the information gain will be most affected by sets of examples that share the same values for different subsets of the attributes.

In regular \( k \)-steps lookahead, for each tuple of \( k \) attributes we divide the set of examples into \( 2^k \) subsets, each of which is associated with a different \( k \)-tuple of binary values, i.e., a different sub-path of length \( k \) in the lookahead tree. Then, we calculate the information gain for each subset and compute the average gain weighted by subset sizes.

In skewing, each iteration assigns a high weight to a few subsets of the examples that have common values for some of the attributes. These subsets have the greatest influence on the gain calculation at that skewing iteration. Hence, skewing iterations can be seen as sampling the space of lookahead sub-paths and considering a few of them at a time.

The great advantage of skewing over our proposed framework is efficiency. While LSID3 samples full decision trees, skewing samples only single sub-paths. When there are many examples but relatively few attributes, skewing improves greatly over the greedy learners at a low cost. The effectiveness of skewing, however, noticeably degrades when the concept hides a mutual dependency between a large number of attributes. When the number of attributes increases, it becomes harder to create large clusters with common values for some of the relevant ones, and hence the resulting lookahead is shallower.

Consider, for example, the \( n \)-XOR problem with \( n \) additional irrelevant attributes. For \( n = 2 \), a reweighting that assigns a high weight to examples that agree only on 1 of the 2 relevant attributes results in a high gain for the second attribute because it decreases the entropy of the cluster to 0. Nevertheless, in order to have a positive gain for a relevant attribute when \( n = 10 \), we should cluster together examples that agree on 9 of the 10 relevant attributes. Obviously, the probability for a good cluster in the first case is high while in the second case it is almost 0. The experiments reported in Section 3.7.6 provide an empirical backup for this argument.
Figure 6.1: The sensitivity of LSID3 and skewing to irrelevant attributes. The concept is XOR-4 while all the other attributes are irrelevant. The $x$-axis represents the total number of attributes.

A similar problem occurs if we fix the number of relevant attributes but increase the number of irrelevant ones. In that case, the space of possible sub-paths becomes larger and the probability of creating a good cluster decreases. To test this hypothesis, we compared the sensitivity of LSID3, skewing, and Sequential skewing to the number of irrelevant attributes. LSID3 was run with $r = 5$ while the skewing algorithms were run with their default parameters. The target concept (XOR-4) and the number of examples (512) were fixed, while the number of irrelevant attributes ranged from 6 to 56 (thus, the total number of attributes ranged from 10 to 60). Figure 6.1 displays the results.

The graphs indicate that LSID3 continues to attain high accuracy and produce almost optimal trees even when the number of irrelevant attributes increases. The degradation in LSID3 performance is noticeable only when the number of
attributes exceeds 35, but even with 60 attributes the algorithm achieves an accuracy of about 82%. The skewing algorithms, on the other hand, are much more sensitive to irrelevant attributes and their performance decreases drastically with the increase in the number of irrelevant attributes. When the number of attributes is more than 35, the skewing algorithms become no better than a random guesser. The consistent advantage of LSID3 is clear also in terms of tree size, where the trees produced by ID3 and skewing are significantly larger.

To be fair, it is important to note that LSID3 had a much longer runtime than skewing with its default parameters. However, our previous experiments with parity concepts showed that the performance of skewing does not improve with time and hence the results are expected to be the same, even if skewing were to be allocated the same amount of time. To verify this, we repeated the experiment for 35 and 60 attributes and allocated skewing the same time as LSID3($r = 5$). The results were similar to those reported in Figure 6.1 and no improvement in the performance of skewing was observed.

### 6.1.2 Other Cost-insensitive Decision-Tree Inducers

Papagelis and Kalles (2001) studied GATree, a learner that uses genetic algorithms for building decision trees. GATree does not adopt the top-down scheme. Instead, it starts with a population of random trees and uses a mutation operation of randomly changing a splitting test and a crossover operation of exchanging subtrees. Unlike our approach, GATree is not designed to generate consistent decision trees and searches the space of all possible trees over a given set of attributes. Thus, it is not appropriate for applications where a consistent tree is required. Like most genetic algorithms, GATree requires cautious parameter tuning and its performance depends greatly on the chosen setting. Comparing GATree to our algorithm (see Section 3.7.6) shows that, especially for hard concepts, it is much better to invest the resources in careful tuning of a single tree than to perform genetic search over the large population of decision trees.

Utgoff et al. (1997) presented DMTI (Direct Metric Tree Induction), an induction algorithm that chooses an attribute by building a single decision tree under each candidate attribute and evaluates it using various measures. Several possible tree measures were examined and the MDL (Minimum Description Length) measure performed best. DMTI is similar to LSID3($r = 1$) but, unlike LSID3, it can only use a fixed amount of additional resources and hence cannot serve as an anytime algorithm. When the user can afford using more resources than required by DMTI, the latter does not provide means to improve the learned model further. Furthermore, DMTI uses a single greedy lookahead tree for attribute evaluation, while we use a biased sample of the possible lookahead trees. Our experiments with DMTI (as available online) show that while it can solve
simpler XOR and multiplexer problems, its limited lookahead is not sufficient for learning complex concepts such as XOR-10: DMTI achieved an accuracy of 50%. IIDT and LSID3, by producing larger samples, overcame this problem and reached high accuracies.

Kim and Loh (2001) introduced CRUISE, a bias-free decision tree learner that attempts to produce more compact trees by (1) using multiway splits—one subnode for each class, and (2) examining pair-wise interactions among the variables. CRUISE is able to learn XOR-2 and Chess-board (numeric XOR-2) concepts. Much like ID3-k with $k = 2$, it cannot recognize more complex interactions.

Bennett (1994) presented GTO, a non-greedy approach for repairing multivariate decision trees. GTO requires as input an initial tree. The algorithm retains the structure of the tree but attempts to simultaneously improve all the multivariate decisions of the tree using iterative linear programming. GTO and IIDT both use a non-greedy approach to improve a decision tree. The advantage of GTO is its use of a well-established numerical method for optimization. Its disadvantages are its inability to modify the initial structure and its inability to exploit additional resources (beyond those needed for convergence).

Freund and Mason (1999) described a new type of classification mechanism, the alternating decision tree (ADTree). ADTree provides a method for visualizing decision stumps in an ordered and logical way to demonstrate correlations. Freund and Mason presented an iterative algorithm for learning ADTrees, based on boosting. While it has not been studied as an anytime algorithm, ADTrees can be viewed as such. The learner starts with a constant prediction and adds one decision stump at a time. If stopped, it returns the current tree. The anytime behavior of ADTree, however, is problematic. Additional time resources can only be used to add more rules and therefore might result in large and over complicated trees. Moreover, ADtree is not designed to tackle the problem of attribute interdependencies because it evaluates each split independently.

Murthy, Kasif, and Salzberg (1994) introduced OC1 (Oblique Classifier 1), a new algorithm for induction of oblique decision trees. Oblique decision trees use multivariate tests that are not necessarily parallel to an axis. OC1 builds decision trees that contain linear combinations of one or more attributes at each internal node; these trees then partition the space of examples with both oblique and axis-parallel hyperplanes. The problem of searching for the best oblique split is much more difficult than that of searching for the best axis-parallel split because the number of candidate oblique splits is exponential. Therefore, OC1 takes a greedy approach and attempts to find locally good splits rather optimal ones. Our LSID3 algorithm can be generalized to support the induction of oblique decision trees by using OC1 as the sampler, and by considering oblique splits. Furthermore, OC1 can be converted into an anytime algorithm by considering more splits at each
node if resources allow. These would make challenging and interesting directions for future research.

Nijssen and Fromont (2007) presented DL8, an exact algorithm for finding a decision tree that optimizes a ranking function under size, depth, accuracy and leaf constraints. The key idea behind DL8 is that constraints on decision trees are simulated as constraints on itemsets. They show that optimal decision trees can be extracted from lattices of itemsets in linear time. The applicability of DL8, however, is limited by two factors: the number of itemsets that need to be stored, and the time that it takes to compute these itemsets. In some cases, the number of frequent itemsets is so large that it is impossible to compute or store them within a reasonable amount of time or space.

Several researchers investigated the induction of shallow decision trees. Holte (1993) reported an empirical study of the accuracy of rules that classify examples on the basis of a single test (1R). Holte concluded that, on most real world datasets, multilevel decision trees do not perform significantly better than one-level classification rules. Elomaa (1994), however, questioned the validity of these conclusions. Elomaa argued that the small difference in accuracy between 1R and C4.5 is still significant, and that the conclusions may have been the results of the use of unrepresentative databases.

Auer, Holte, and Maass (1995) presented a novel algorithm, called T2, for agnostic PAC-learning with decision trees of at most 2-levels. The computation time of T2 is almost linear in the size of the training set. When tested empirically on several datasets, T2 was shown to produce substantially simpler decision trees with little or no loss in predictive power. Since one can prove that T2 is an agnostic PAC-learning algorithm, it is guaranteed to produce close to optimal 2-level decision trees given sufficiently large training data. A generalization to depth \( d \), however, would require computation time that is exponential in \( d \). Dobkin, Gunopulos, and Kasif (1996) described several algorithms and theoretical results for learning optimal consistent decision trees of bounded depth. These methods, nevertheless, are practical only when the tree depth is tiny.

Much research effort has also been invested in the theoretical aspects of decision tree learning. A standard theoretical approach is to prove a bound on generalization error as a function of the training error and the concept size (McAllester, 1998). Then, a concept optimizing the tradeoff between training error and concept size, as expressed in the bound, is selected. These bounds depend on the size of the training sample, but not on the sample itself. To improve them, Mansour and McAllester (2000) constructed bounds that depend both on the structure of the model and on the actual examples that form the sample.

Kushilevitz and Mansour (1993) presented a polynomial time algorithm for learning decision trees with respect to the uniform distribution, using membership queries. The considered decision tree model is an extension of the traditional
Boolean decision tree model that allows linear operations in each node. Kearns and Mansour (1996) analyze the performance of greedy top-down tree learners that use impurity-based split criteria, and demonstrate that such algorithms implicitly perform boosting. Mansour, Kearns and Mansour (1997, 1998) suggest modifying C4.5’s (pessimistic) error based pruning to incorporate theoretically motivated local decision criteria, and provide risk bounds relating the pruned tree’s performance to the performance of the best possible pruned tree.

6.2 Anytime Induction of Other Cost-insensitive Classifiers

Ensemble-based methods, such as bagging and boosting (Schapire, 1999), can also be viewed as anytime algorithms. However, the classifiers constructed by the bagging and boosting algorithms consist of a committee of decision trees rather than a single tree. Therefore, the problem they face is very different from the problem we face in this work—that of learning a single tree. A major problem with ensemble-based methods is that the induced ensemble is often large, complex, and difficult to interpret (Freund & Mason, 1999). Therefore, these methods cannot be used when comprehensible models are required. Another problem is that greedy trees are unable to discover any knowledge about hard-to-learn target concepts. Therefore, combining them cannot improve performance. In our experiments, reported in Section 3.7.6, we provide empirical support for this claim by comparing our proposed anytime algorithms to bagging.

Dietterich (2000) presented the randomized-C4.5 algorithm, where a randomized version of C4.5 that chooses the split attribute at random from among the 20 best candidates is repeatedly invoked to produce an ensemble of decision trees. The experimental results indicate that ensembles of randomized-C4.5 were competitive with bagging but not as accurate as boosting. As in the case of the traditional bagging and boosting methods, our framework differs from randomized-C4.5 in that the latter produces an ensemble of trees that is obviously not as comprehensible as a single decision tree is.

Kononenko and Kovacic (1992) applied stochastic search methods, such as stochastic hill climbing and simulated annealing, for learning decision rules from examples: rules are randomly generated and iteratively improved until a local maximum is reached.

1In the contract and interruptible classification setups, our framework builds several trees. However, because only one tree participates in classification, the comprehensibility is not adversely affected.
The SVM algorithm usually depends on several parameters – kernel parameters, for example. Several works, such as (Chapelle, Vapnik, Bousquet, & Mukherjee, 2002), proposed iterative methods for automatic tuning of SVM parameters. These iterative methods can exploit additional time resources for better tuning.

A well-studied alternative to inductive learning is the theory refinement paradigm. In a theory refinement system, we first acquire a domain theory, for instance by querying experts, and then revise the obtained set of rules in an attempt to make it consistent with the training data. Opitz (1995) introduced an anytime approach for theory refinement. This approach starts by generating a neural network from a set of rules that describe what is currently known about the domain. The network then uses the training data and the additional time resources to try to improve the resulting hypothesis.

### 6.3 Cost-sensitive Classification

Cost-sensitive trees have been the subject of many research efforts. Several works proposed learning algorithms that consider different misclassification costs (Breiman et al., 1984; Pazzani, Merz, Murphy, Ali, Hume, & Brunk, 1994; Provost & Buchanan, 1995; Bradford, Kunz, Kohavi, Brunk, & Brodley, 1998; Domingos, 1999b; Drummond & Holte, 2000; Elkan, 2001; Zadrozny, Langford, & Abe, 2003; Lachiche & Flach, 2003; Abe, Zadrozny, & Langford, 2004; Vadera, 2005; Margineantu, 2005; Zhu, Wu, Khoshgoftaar, & Yong, 2007; Sheng & Ling, 2007b). These methods, however, do not consider test costs and hence are appropriate mainly for domains where test costs are not a constraint. Other authors designed tree learners that take into account test costs, such as IDX (Norton, 1989), CSID3 (Tan & Schlimmer, 1989), and EG2 (Nunez, 1991). These methods, however, do not consider misclassification costs.

Decision Trees with Minimal Cost (DTMC), a greedy method that attempts to minimize both types of costs simultaneously, has been recently introduced (Ling et al., 2004; Sheng et al., 2006). A tree is built top-down, and a greedy split criterion that takes into account both testing and misclassification costs is used. The basic idea is to estimate the immediate reduction in total cost after each split, and to prefer the split with the maximal reduction. If no split reduces the cost on the training data, the induction process is stopped.

Although efficient, the DTMC approach can be trapped into a local minimum and produce trees that are not globally optimal. For example, consider the concept and costs described in Figure 6.2 (left). There are 10 attributes, of which only \( a_9 \) and \( a_{10} \) are relevant. The cost of \( a_9 \) and \( a_{10} \), however, is significantly higher than the others. Such high costs may hide the usefulness of \( a_9 \) and \( a_{10} \), and mislead the learner into repeatedly splitting on \( a_{1-8} \), which would result in
a large, expensive tree. The problem would be intensified if \( a_9 \) and \( a_{10} \) were interdependent, with a low immediate information gain (e.g., \( a_9 \oplus a_{10} \)). In that case, even if the costs were uniform, a local measure might fail to recognize the relevance of \( a_9 \) and \( a_{10} \).

DTMC is appealing when learning resources are very limited. However, it requires a fixed runtime and cannot exploit additional resources to escape local minima. Inexpensive Classification with Expensive Tests (ICET) was a pioneer in non-greedy search for a tree that minimizes test and misclassification costs (Turney, 1995). ICET uses genetic search to produce a new set of costs that reflects both the original costs and the contribution of each attribute in reducing misclassification costs. Then it builds a tree using the EG2 algorithm but with the evolved costs instead of the original ones. EG2 is a greedy cost-sensitive algorithm that builds a tree top-down and evaluates candidate splits by considering both the information gain they yield and their measurement costs. It does not, however, take into account the misclassification cost of the problem.

ICET was shown to significantly outperform greedy tree learners, producing trees of lower total cost. ICET can use additional time resources to produce more generations and hence widen its search in the space of costs. Because the genetic operations are randomized, ICET is more likely to escape local minima – into which EG2 with the original costs might be trapped. Nevertheless, two shortcomings limit ICET’s ability to benefit from extra time. First, after the search phase, it uses the greedy EG2 algorithm to build the final tree. But because EG2 prefers attributes with high information gain (and low test cost), the usefulness of highly relevant attributes might be underestimated by the greedy measure in the case of hard-to-learn concepts where attribute interdependency is hidden. This will result in more expensive trees. Second, even if ICET overcomes the above problem by randomly reweighting the attributes, it searches the space of parameters globally, regardless of the context in the tree. This imposes a problem if an attribute is important in one subtree but useless in another. To

**Figure 6.2:** Left: an example of a difficulty greedy learners might face. Right: an example of the importance of context-based feature evaluation.
better understand these shortcomings, consider the concept described by the tree in Figure 6.2 (right). There are 10 attributes with similar costs. The value of \( a_1 \) determines whether the target concept is \( a_7 \oplus a_9 \) or \( a_4 \oplus a_6 \). The interdependencies result in a low gain for all attributes. Because ICET assigns costs globally, the attributes will have similar costs as well. Therefore, ICET will not be able to recognize which one is relevant in which context. If the irrelevant attributes are cheaper, the problem is intensified and the model might end up relying on irrelevant attributes.

Davis, Ha, Rossbach, Ramadan, and Witchel (2006) presented a greedy cost-sensitive decision tree algorithm for forensic classification: the problem of classifying irreproducible events. In this setup, they assume that all tests that might be used for testing must be acquired and hence charged for before classification.

Another way to exploit additional time when searching for a less costly tree is to widen the search space. Bayer-Zubek and Dietterich (2005) formulated the cost-sensitive learning problem as a Markov decision process (MDP), and used a systematic search algorithm based on the AO* heuristic search procedure to solve the MDP. To make AO* efficient, the algorithm uses a two-step lookahead based heuristic. Such limited lookahead is more informed than immediate heuristics but still insufficient for complex domains and might cause the search to go astray (Esmeir & Markovitch, 2007b). The algorithm was shown to output better diagnostic policies than several greedy methods using reasonable resources. An optimal solution, however, could not always be found due to time and memory limits. A nice property of the algorithm is that it can serve as an anytime algorithm by computing the best complete policy found so far. Its anytime behavior, nevertheless, is problematic because policies that are optimal with respect to the training data tend to overfit. As a result, the performance will eventually start to degrade.

Greiner et al. (2002) were pioneers in studying classifiers that actively decide what tests to administer. They defined an active classifier as a classifier that given a partially specified instance, returns either a class label or a strategy that specifies which test should be performed next. Note that every decision tree that stores default class labels in its internal nodes is an active classifier. The major challenge we faced in Chapter 5 of this work is to design an anytime algorithm for learning active classifiers that can utilize any classification budget, preallocated or not.

Greiner et al. also analyzed the theoretical aspects of learning optimal active classifiers using a variant of the probably-approximately-correct (PAC) model. They showed that the task of learning optimal cost-sensitive active classifiers is often intractable. However, this task is shown to be achievable when the active classifier is allowed to perform only (at most) a constant number of tests, where
the limit is provided before learning. For this setup they proposed taking a dynamic programming approach to build trees of at most depth $d$. Their proposed method requires $O(n^d)$ time to find the optimal $d$-depth tree over $n$ attributes. While this complexity is polynomial, it is not effective, in practice, except for small $n$ and tiny $d$ values. (Farhangfar, Greiner, & Zinkevich, 2008) presented a fast way to produce near-optimal depth-bounded trees under the Naive Bayes assumption. The proposed approach resulted in a speedup of $O(n/\log n)$ and, despite the unrealistic assumption, it yielded relatively high classification accuracy.

Fan, Lee, Stolfo, and Miller (2000) studied the problem of cost-sensitive intrusion detection systems (IDS). The goal is to maximize security while minimizing costs. Each prediction (action) has a cost. Features are categorized into three cost levels according to amount of information needed to compute their values. To reduce the cost of an IDS, high cost rules are considered only when the predictions of low cost rules are not sufficiently accurate.

Arnt and Zilberstein (2005) tackled the problem of time and cost sensitive classification (TCSC). In TCSC, the utility of labeling an instance depends not only on the correctness of the labeling, but also the amount of time it takes. Therefore the total cost function has an additional component, which reflects the time needed to measure an attribute. Typically, it has a super-linear form: the cost of a quick result is small and fairly constant, but as the waiting time increases, the time cost grows at an increasing rate. The problem is further complicated when a sequence of time-sensitive classification instances is considered, where time spent administering tests for one case can adversely affect the costs of future instances. Arnt and Zilberstein suggest solving these problems by extending the decision theoretic approach introduced by Bayer-Zubek and Dietterich (2005). In our work, we assume that the time it takes to administer a test is incorporated into its cost. In the future, we intend to extend our framework to support time-sensitive classification, both for individual cases and for sequences.

Our setup assumed that we are charged for acquiring each of the feature values of the test cases. The term test strategy (Sheng, Ling, & Yang, 2005) describes the process of feature value acquisition: which values to query for and in what order. Several test strategies have been studied, including sequential, single batch, and multiple batch (Sheng et al., 2006), each of which corresponds to a different diagnosis policy. These strategies are orthogonal to our work because they assume a given decision tree.

Bilgic and Getoor (2007) tackled the problem of feature subset selection when costs are involved. The objective is to minimize the sum of the information acquisition cost and the misclassification cost. Unlike greedy approaches that compute the value of features one at a time, they used a novel data structure called the value of information lattice (VOILA), which exploits dependencies
between missing features and makes it possible to share information value computations between different feature subsets. VIOLA was shown empirically to achieve dramatic cost improvements without the prohibitive computational costs of comprehensive search.

Yang, Webb, Korb, and Ting (2007) introduced Anytime Averaged Probabilistic Estimators (AAPE) for utilizing additional computational resources during classification. At classification time, AAPE computes Naive Bayes and then exploits extra time to refine the probability estimate. Ueno, Xi, Keogh, and Lee (2006) showed how to convert nearest neighbor classifiers into anytime classifiers. The proposed algorithm, called Anytime Nearest Neighbor (ANN), utilizes additional classification time by considering more and more examples out of which the nearest neighbor is picked. Unlike TATA, both ANN and AAPE are fixed-time learners and cannot benefit from additional resources during learning. Moreover, ANN and AAPE do not deal with attribute costs and assume that the major constraint is the computation time required by the probabilistic models.

Kaplan, Kushilevitz, and Mansour (2005) studied an extension of the classical PAC model where test costs are involved. In the first setup, called offline, $f$, the target function, and $D$, the distribution of the test outcomes, are known. The goal is to minimize the expected cost of classifying a case using $f$. In the second setup, denoted distributional online, either $f$ or $D$ are unknown. When $f$ is unknown, the goal is to minimize the predictive errors while maintaining low testing costs. When $D$ is unknown, the goal is to minimize the expected cost of the tests, while collecting information regarding the distribution. In the third setup, denoted adversarial online, $f$ is given but the inputs are selected adversarially. The goal is to bound the average online cost compared to a fixed evaluation order. Each of these scenarios is analyzed and theoretical bounds are derived.

### 6.4 Other Learning Components

In addition to the induction procedure, the learning process involves other components, where additional resources can be invested. The problem of active learning is to select an example for tagging out of a pool of untagged examples. Lindenbaum et al. (2004) presented an active learning algorithm that uses lookahead to evaluate the untagged examples and selects the best example for tagging. This can be viewed as an anytime algorithm parameterized by the depth of lookahead.

Feature generation can also be performed by anytime algorithms. Markovitch and Rosenstein (2002) presented an algorithm that searches for good features over the (potentially infinite) space of constructed features. This algorithm can be interrupted at anytime, returning the currently best generated features.

Feature selection can also benefit from the anytime approach. Last, Kandel,
Maimon, and Eberbach (2001) introduced an interruptible anytime algorithm for feature selection. Their proposed method performs a hill-climbing forward search in the space of feature subsets where the attributes are added to the selected subset one at a time, constructed from the previously selected attributes. The algorithm can be stopped at any moment, returning the set of currently selected features.

All these methods are orthogonal to our induction methods and can complement them. An interesting research direction is to determine resource distribution between the various learning stages.

Costs are also involved in the learning phase, during example acquisition and during model learning. The problem of budgeted learning has been studied by Lizotte et al. (2003). There is a cost associated with obtaining each attribute value of a training example, and the task is to determine what attributes to test given a budget.

A related problem is active feature-value acquisition. In this setup one tries to reduce the cost of improving accuracy by identifying highly informative instances. Melville et al. (2004) introduced an approach in which instances are selected for acquisition based on the accuracy of the current model and the model’s confidence in the prediction.

An extension to the work on optimal fixed depth trees (Greiner et al., 2002) is when tests have costs, both during learning and classification. Kapoor and Greiner (2005) presented a budgeted learning framework that operates under strict budget for feature-value acquisition during learning and produces a classifier with a limited classification budget. Provost, Melville, and Saar-Tsechansky (2007) discussed the challenges electronic commerce environments bring to data acquisition during learning and classification. They discussed several settings and presented a unified framework to integrate acquisition of different types, with any cost structure and any predictive modeling objective.

(Madani, Lizotte, & Greiner, 2004) studied the setup of budgeted active model selection. In this setup the learner can use a fixed budget of model probes to identify which of a given set of possible models has the highest expected accuracy. In each probe a model is evaluated on a random indistinguishable instance. The goal is a policy that sequentially determines which model to probe next, based on the information observed so far. Madani et al. formalized the task and showed that it is NP-hard in general. They also studied several algorithms for budgeted active model selection and compared them empirically.
Chapter 7

Conclusions

The application of machine learning techniques to real world problems involves several types of costs and constraints. In this thesis we proposed a novel framework for operating under different constraints during learning and classification. Our framework allows computation speed during learning to be traded for better predictive models.

In Chapter 2 we described a variety of possible scenarios of resource allocation and consumption. In contract learning, the learning resources are preallocated and provided to the learner. In interruptible learning, the learner should utilize extra time until interrupted and queried for a solution. Therefore, an interruptible learner should be ready to return a valid classifier at any time.

Using a classifier for predicting the labels of new cases carries two costs: the cost of the tests the model requires to administer, and the cost of the predictive errors the model makes. In cost-insensitive classification, tests do not have costs and the penalty of wrong classification is uniform, no matter what the error type is. Therefore, the objective of a learning algorithm in this case is to produce comprehensible and accurate models. In cost-sensitive classification, on the other hand, the goal is to minimize the total cost, i.e., the sum of testing costs and misclassification costs. Many real-world applications, however, limit the costs of the tests a model can require. Therefore, another interesting objective we studied is to minimize misclassification costs when testing costs are bounded. The bound on testing costs may be predetermined and provided to the learner (pre-contract classification), known to the classifier but not to the learner (contract classification), or unknown; the classifier is then expected to exploit resources until interrupted (interruptible classification). In this thesis we handled all these scenarios.

In Chapter 3 we presented a general framework for contract anytime induction of decision trees. The major limitation of top-down greedy tree learners is their inability to recover from wrong split decisions that might arise when local
heuristics that judge an attribute on the basis of its immediate contribution are used. Our proposed framework for top-down induction of decision trees uses more resources to evaluate candidate splits. We suggest producing a biased sample of the space of subtrees under each possible split, and to favor the split with the best sample. The evaluation of the bias of the sample and the evaluation of the sampled-trees depend on the cost scheme. Because subtrees are judged as a whole, the framework can be applied under any cost scheme. The framework can utilize additional time resources to produce larger samples and hence gain better split-utility estimations. Once the tree is built, a pruning phase is applied. The exact pruning method also depends on the objectives of the learner.

The first instantiation of the framework, called LSID3, is introduced in Chapter 3. LSID3 aims to produce comprehensible and accurate trees. The hypothesis space of trees LSID3 can produce is huge and a major question is what preference bias should be used to direct the search. Motivated by Occam’s razor, a widely adopted approach is to prefer smaller trees. The utility of this principle to machine learning algorithms, however, has been the subject of a heated debate. To resolve the debate, we conducted extensive experiments and found that the inverse correlation between the size of a tree and its accuracy is statistically significant across most domains (Appendix B). Therefore, in LSID3 we bias the sample towards small consistent trees, and prefer the split whose sample contains the smallest tree. To avoid over-fitting, we post-prune the tree by applying error-based pruning. Motivated by the need for interruptible learners, we introduced IIDT, a general method for converting contract learners that use our sampling approach into interruptible algorithms. IIDT starts with a greedy tree and continuously improves subtrees by additional sampling.

When costs are involved, we can no longer use the tree size as a heuristic. The ACT algorithm, formulated in Chapter 4, instantiates our sampling approach for the induction of cost-efficient trees that minimize the expected total cost of classification. ACT biases the sample towards low-cost trees. The testing costs of the sampled trees are estimated using the cost on the training examples. The misclassification costs are estimated using the expected error of the trees. The split whose sample contains a tree with the lowest total cost tree is chosen. Once the tree is grown, we apply a novel pruning procedure, which we have designed to reduce the expected total cost of using a tree.

For the problems that constrain classification resources, we have developed TATA (Chapter 5). TATA can be configured to: (1) get the classification budget before learning, after which it produces a classifier that exploits this budget, (2) produce a classifier that gets the budget as a parameter and operates accordingly, or (3) build a classifier that utilizes resources until interrupted, without the need to provide it with the budget in advance. For the first setup, TATA builds a tree using a modified version of the general top-down induction scheme we call
TDIDT§. TDIDT§ filters out unaffordable tests and stops growing the tree when a classification path exceeds the cost-quota. To choose a split TATA, samples the space of TDIDT§ trees under each split and favors the split whose sample contains a tree with minimal misclassification costs. For the second and third setups, TATA produces a repertoire of trees, each of which is built with a different budget. If the resource allocation is known before classification, the tree that best fits the budget is picked. Otherwise, the trees are traversed, starting from the least expensive one, until the classification resources are exhausted. The final prediction is given by the last completely traversed tree.

Each of the methods we developed has been tested extensively, on a variety of datasets and under different sets of cost. LSDI3 produced significantly smaller and more accurate trees than several greedy and non-greedy algorithms for learning decision trees. ACT induced the most cost efficient trees for the majority of domains, compared to the state-of-the-art cost-sensitive tree learners. TATA learned trees that best exploited the available classification budgets and made better predictions under the given constraints. Our proposed algorithms were also shown to exhibit good anytime behavior, with diminishing returns.

Many real-life applications that involve offline learning can benefit from our methodology (Esmeir & Markovitch, 2006). When the user is willing to allocate more resources for the learning process, our framework can produce better classifiers, and hence improve accuracy and reduce classification costs. The need for such a learner arises in many real-life artificial intelligence applications. For example:

- **Medical tasks:** Assume that a medical center has decided to acquire a classifier for diagnosis of a particular disease (e.g., (Kononenko, 2001)). The center applies C4.5 on thousands of records from previous patients, and after a few seconds receives a decision tree. During the coming months, or even years, the same induced decision tree will be used. Obviously, the medical center is willing to wait much longer to obtain a better tree.

- **Planning:** Consider an intelligent planning agent that has to take action rapidly (e.g., (Konur, Ferrein, & Lakemeyer, 2004)). This requires better long-term planning. In such cases the agent is ready to invest more resources for the learning phase to improve its real-time performance.

- **Text categorization:** Consider a machine-learning based spam filter (e.g., (Cohen, 1996)). As users, we expect the learning component to exploit the time between the arrival of new messages.

- **Computational finance:** Consider an artificial investor that uses machine learning techniques for stock portfolio selection (e.g., Borodin, El-Yaniv, &
Gogan, 2004). If we chose to update the portfolio at specific time points, we would like the learner to exploit the time between these updates.

Furthermore, researchers in the field can benefit from the automatic method for cost assignments we have developed. Only a few UCI datasets have assigned costs. In this work we designed a semi-randomized method for assigning costs to existing datasets. We applied this method on 25 datasets and established a repository, available at: http://www.cs.technion.ac.il/~esaher/cost.

This research can be extended in several directions. We intend to apply monitoring techniques for optimal scheduling of the anytime learners. We also plan to use different measures for tree quality and compare their utility. While the tree size and expected error were generally successful, our sampling approach did not, in few cases, yield significant improvement. Using other measures may improve the performance in these cases.

We also intend to test the performance of our framework on other cost schemes that involve other types of cost. We believe that the generality of our framework will allow excellent results to be obtained under other setups as well. To reduce the runtime of our anytime algorithms, we plan to cache some of the lookahead trees and use them, rather than resampling at each node. If a split is chosen, the sample of already available subtrees can be used to evaluate its descendants as well. Finally, an important advantage of our method is that it can be easily parallelized. Assume, for example, that we decided on samples of size $r$. Then, $r$ different machines can independently form the sample and speed up the induction process by a factor of $r$. We intend to consider this direction in the future.
Appendix A

Datasets

Below we give a more detailed description of the datasets used in our experiments:

1. **Automobile**: This problem, taken from the UCI Repository, consists of three types of entities: (1) the specification of an automobile, (2) its assigned insurance risk rating, and (3) its normalized losses in use as compared to other cars. Several of the attributes in the database could be used as class attributes: we chose to use both the make (model) and the symbolling (risk degree).

2. **Balance Scale**: This dataset is taken from the UCI Repository and was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The attributes are the left weight, the left distance, the right weight and the right distance.

3. **Breast Cancer**: This problem is taken from the UCI Repository. Each instance represents the characteristics of a patient and the class is whether or not there are recurrence events.

4. **Bupa**: The BUPA Liver Disorders dataset is taken from the UCI Repository. The tests are blood tests that are thought to be sensitive to liver disorders that might arise from excessive alcohol consumption. The target concept is defined as $drinks < 3$ or $drink \geq 3$.

5. **Car**: This dataset, taken from the UCI Repository, was derived from a simple hierarchical decision model that evaluates cars according to basic characteristics such as price, safety, and comfort.

6. **Connect-4**: This dataset, taken from the UCI Repository, contains all legal 8-ply positions in the connect-4 game in which neither player has won yet and the next move is not forced.
7. **Corral:** An artificial dataset first used by John, Kohavi, and Pfleger (1994).

8. **Glass:** In this domain, taken from the UCI Repository, the goal is to determine the type of glass from its characteristics.

9. **Heart:** In this domain, taken from the UCI Repository, the goal field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

10. **Hepatitis:** In this problem, taken from the UCI Repository, the diagnosis is known, and the problem is to determine the likely outcome of the disease.

11. **Iris:** This dataset is taken from the UCI Repository and contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

12. **Monks problems:** This set, taken from the UCI Repository, contains three problems. Each example is represented by 5 nominal attributes in the range 1, 2, 3, 4. The problems are:
   - **Monks-1:** \((a_1 = a_2)\text{ or } (a_5 = 1)\).
   - **Monks-2:** exactly two of \((a_1 = 1, a_2 = 1, a_3 = 1, a_4 = 1, a_5 = 1)\).
   - **Monks-3:** \(((a_5 = 3)\text{ and } (a_4 = 1))\text{ or } ((a_5 \neq 4)\text{ and } (a_2 \neq 3))\), with an added 5% class noise.

The original datasets are already partitioned into training and testing sets.

13. **Mushroom:** This dataset, taken from the UCI Repository, includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota family. Each species is identified as edible or poisonous.

14. **Nursery:** This database, taken from the UCI Repository, was derived from a hierarchical decision model originally developed to rank applications for nursery schools.

15. **Pima:** The Pima Indians Diabetes dataset, taken from the UCI Repository, includes several medical tests and the possible classes are either the patient is healthy or the patient has diabetes.

16. **Solar Flare:** This problem is taken from the UCI Repository. Each instance represents captured features for one active region on the sun. Among the three possible classes, we considered the C-class flare where the instances are more distributed.
17. **TAE**: The Teaching Assistant Evaluation dataset, taken from the UCI Repository, consists of evaluations of teaching performance; scores are "low", "medium", or "high".

18. **Tic-Tac-Toe**: The problem, taken from the UCI Repository, deals with the classification of legal tic-tac-toe end games, as wins or non-wins for the x player. Each example is represented by 9 nominal attributes, which represent the slot values.

19. **Thyroid**: In this dataset, taken from the UCI repository, the class is diagnostic (normal or hyperthyroid). The features represent medical information and basic characteristics of the patient.

20. **Voting**: This dataset, taken from the UCI repository, includes votes for each member of the U.S. House of Representatives on the 16 key votes identified by the CQA. The class of each record is Democrat or Republican.

21. **Wine**: This problem, taken from the UCI Repository, deals with the classification of wines into 3 class types. Each example is represented by 13 continuous attributes, which represent measures of chemical elements in the wine.

22. **Zoo**: In this domain, taken from the UCI Repository, the goal is to determine the type of the animal from several attributes.

23. **Multiplexer**: The multiplexer task was used by several researchers for evaluating classifiers, e.g., Quinlan (1993). An instance is a series of bits of length \( a + 2^a \), where \( a \) is a positive integer. The first \( a \) bits represent an index into the remaining bits and the label of the instance is the value of the indexed bit. In our experiments we considered the 20-Multiplexer \((a = 4)\). The dataset contains 500 randomly drawn instances.

24. **Boolean XOR**: Parity-like functions are known to be problematic for many learning algorithms. However, they naturally arise in real-world data, such as the Drosophila survival concept (Page & Ray, 2003). We considered XOR of five and ten variables with additional irrelevant attributes.

25. **Numeric XOR**: A XOR based numeric dataset that has been used to evaluate learning algorithms, e.g., Baram, El-Yaniv, and Luz (2003). Each example consists of values for \( x \) and \( y \) coordinates. The example is labeled 1 if the product of \( x \) and \( y \) is positive, and \(-1\) otherwise. We generalized this domain for three and four dimensions and added irrelevant variables to make the concept harder.

141
26. **Multi-XOR / Multi-AND-OR:** These concepts are defined over 11 binary attributes. In both cases the target concept is composed of several subconcepts, where the first two attributes determine which of them is considered. The other 10 attributes are used to form the subconcepts. In the Multi-XOR dataset, each subconcept is an XOR, and in the Multi-AND-OR dataset, each subconcept is either AND or OR.
Occam’s razor, attributed to the 14th-century English logician William of Ockham, is the principle that, given two hypotheses consistent with the observed data, the simpler one should be preferred. This principle has become the basis for many induction algorithms that search for a small hypothesis within the version space (Mitchell, 1982). Several studies attempted to justify Occam’s razor with theoretical and empirical arguments (Blumer et al., 1987; Quinlan & Rivest, 1989; Fayyad & Irani, 1990). But a number of recent works have questioned the utility of Occam’s razor, and provided theoretical and experimental evidence against it.

Schaffer (1994) proved that no learning bias can outperform another bias over the space of all possible learning tasks. This looks like theoretical evidence against Occam’s razor. Rao et al. (1995), however, argued against the applicability of this result to real-world problems by questioning the validity of its basic assumption about the uniform distribution of possible learning tasks.

Domingos (1999a) argued that the disagreement about the utility of Occam’s razor stems from the two different interpretations given to it: the first is that simplicity is a goal in and of itself, and the second is that simplicity leads to better accuracy. While accepting the first interpretation, Domingos questioned the second one.

Webb (1996) presented C4.5X, an extension to C4.5 that uses similarity considerations to further specialize consistent leaves. Webb reported an empirical evaluation which shows that C4.5X has a slight advantage in a few domains and argued that these results discredit Occam’s thesis.

Murphy and Pazzani (1994) reported a set of experiments in which all possible consistent trees were produced and their accuracy was tested. Their findings were inconclusive. They found cases where larger trees had, on average, better accuracy. Still, they recommend using Occam’s principle when no additional information about the concept is available. The major limitation of their work
is the exhaustive enumeration of the version space. Such an approach is only applicable to domains with very few features.

In this work we present an alternative approach that performs statistical testing of Occam’s thesis on a sample of the version space. This approach allows us to use high-dimensional domains and complex concepts. One problem with random sampling of the version space is the rarity of small trees in the sample. We therefore use, in addition to random sampling, biased sampling methods based on modern anytime induction algorithms (Esmeir & Markovitch, 2004). These methods produce samples with much higher concentrations of small trees.

The major contribution of this work is to provide convincing empirical evidence for Occam’s razor in the context of classification trees. Furthermore, the various sampling techniques we applied help to better understand the space of consistent decision trees and how top-down induction methods explore it. Note that this empirical demonstration of the utility of Occam’s principle does not pretend to provide a philosophical proof for Occam’s thesis.

B.1 Occam’s Empirical Principle

In the context of machine learning, the widely accepted interpretation of Occam’s razor is that given two consistent hypotheses, the simpler one is likely to have a lower error rate. Fayyad and Irani (1990) have formally defined this notion: for two decision trees $T_1$ and $T_2$ and a fixed $\epsilon$, $0 < \epsilon < 1$, $T_1$ is likely to have a lower error rate than $T_2$ if $\Pr \{ P(T_1, \epsilon) < P(T_2, \epsilon) \} > 0.5$, where $P(T, \epsilon)$ is the probability that $T$ has an error rate greater than $\epsilon$.

Fayyad and Irani (1990) also provided theoretical support for favoring smaller trees. They showed that under a set of assumptions, given two trees $T_1$ and $T_2$ consistent with the observed data, $T_1$ is likely to have a lower error rate than $T_2$ if $T_1$ has fewer leaves. Berkman and Sandholm (1995), however, have questioned this set of assumptions and argued that the opposite conclusion can be drawn from it.

The main challenge we face in this work is to empirically test the validity of Occam’s razor in decision tree induction. We therefore define the Occam’s empirical principle:

**Definition 1** Let $E_{\text{train}}$ and $E_{\text{test}}$ be a training and a testing set respectively. Let $H$ be a set of hypotheses consistent with $E_{\text{train}}$. We say that $H$ satisfies Occam’s empirical principle with respect to $E_{\text{train}}$ and $E_{\text{test}}$ if, for any $h_1, h_2$ drawn from $H \times H$,

$$\Pr \left( \text{Acc} (h_1, E_{\text{test}}) \geq \text{Acc} (h_2, E_{\text{test}}) \mid |h_1| \leq |h_2| \right) \geq 0.5,$$

where $|h|$ is the size of hypothesis $h$ and $0 \leq \text{Acc}(h, E) \leq 1$ is the accuracy of $h$ on a test set $E$. 

144
Procedure TDIDT\((E, A)\)

If \( E = \emptyset \)
    Return Leaf\((\text{nil})\)

If \( \exists c \) such that \( \forall e \in E \, \text{Class}(e) = c \)
    Return Leaf\((c)\)

\( a \leftarrow \text{Choose-Attribute}(A, E) \)
\( V \leftarrow \text{domain}(a) \)

\text{Foreach} \( v_i \in V^* \)

\( E_i \leftarrow \{ e \in E \mid a(e) = v_i \} \)
\( S_i \leftarrow \text{TDIDT}(E_i, A - \{a\}) \)

Return Node\((a, \{ \langle v_i, S_i \rangle \mid i = 1 \ldots |V| \})\)

* When \( a \) is numeric, a cutting point is chosen and \( a \) is not filtered out when calling TDIDT recursively.

**Figure B.1:** Top-down induction of decision trees. \( E \) stands for the training set and \( A \) stands for the set of attributes.

### B.2 Sampling the Version Space

Given a learning problem, we would like to produce all the possible trees consistent with the observations and test whether their size and accuracy are correlated. Such an exhaustive enumeration, however, is not practical for most real-world domains. Therefore, in what follows we propose sampling as an alternative. First we define the population, i.e., the space of decision trees we sample from, and then we describe 3 different sampling techniques, each of which focuses on a different part of the sampled space.

#### B.2.1 Defining the Version Space

Given a set of attributes \( A \), the hypothesis class we deal with is the set of decision trees over \( A \), denoted by \( DT_A \). Let \( E \) be a set of examples. Because Occam’s razor is applicable only to hypotheses that can explain the observations, we limit our discussion to the **version space**—the set of all trees consistent with \( E \), denoted by \( DT_A(E) \). Furthermore, since most decision tree learners build a tree top-down, we focus on a subset of the consistent trees—the trees obtainable by top-down induction, denoted by \( TDIDT_A(E) \). Under the TDIDT scheme, the set of examples is partitioned into subsets by testing the value of an attribute and then each subset is used to recursively build a subtree. The recursion stops when all the examples have the same class label. Figure B.1 formalizes the basic procedure...
for top-down induction.

While $TDIDT_A(E)$ is a strict subset of $DT_A(E)$, we claim that the trees in $DT_A(E)$ that are not in $TDIDT_A(E)$ are not interesting for the purpose of model learning. There are two types of trees in $DT_A(E) - TDIDT_A(E)$:

1. A tree containing a subtree with all leaves marked with the same class. Obviously, such a subtree could have been replaced by a single node marked with the class.

2. A tree with an internal node that has no associated examples from $E$. The subtree rooted at this node is not supported by training examples and is therefore not interesting for induction.

Note that including the above trees could unjustly distort the results. In the first case, larger trees that are logically equivalent will be included, arbitrarily weakening the negative correlation. In the second case, the extra branches are not supported by any training example. Thus, their leaves must be labeled randomly, lowering the accuracy and hence arbitrarily strengthening the negative correlation.

While we restrict the definition of Occam’s empirical principle to consistent hypotheses, in our experiments we also examine its applicability to pruned $TDIDT_A(E)$ trees. This allows us to draw conclusions for noisy datasets as well.

B.2.2 Sampling Techniques

Our goal is to sample the $TDIDT_A(E)$ space in order to test Occam’s empirical principle. Our first proposed sampling technique uses TDIDT with a random selection of the splitting attribute (and cutting point, where the attribute is numeric). We refer to this method as the Random Tree Generator ($RTG$). Observe that although it has no bias with respect to generalization quality, $RTG$ does not uniformly sample $TDIDT_A(E)$. For example, if the concept was $\pi_1$ and the attributes were $\{a_1, a_2, a_3\}$, the probability of constructing the smallest tree (with a single split) is much higher than that of constructing a specific large tree. We will later show that the non-uniform sampling should not affect the validity of our conclusions.

One problem with random sampling is the rarity of small trees. Many induction methods, however, are likely to concentrate on small trees. Theoretically, the correlation could have been statistically significant when sampling the $TDIDT$ space but not when sampling a subspace consisting of small trees. To test this hypothesis we need a sample of small trees. Such a sample could be obtained by repeatedly invoking $RTG$ and keeping the smaller trees. Nevertheless, the number of $RTG$ invocations needed to obtain a reasonable number of small trees...
Procedure SID3-CHOOSE-ATTRIBUTE($E, A$)

Foreach $a \in A$

\[ p(a) \leftarrow \text{gain-1} (E, a) \]

If $\exists a$ such that $\text{entropy-1}(E, a) = 0$

\[ a^* \leftarrow \text{Choose attribute at random from} \quad \{a \in A \mid \text{entropy-1}(E, a) = 0\} \]

Else

\[ a^* \leftarrow \text{Choose attribute at random from } A; \quad \text{for each attribute } a, \text{ the probability of selecting it is proportional to } p(a) \]

Return $a^*$

Figure B.2: Attribute selection in SID3

is prohibitively high. Another alternative is to use ID3. Repeated invocations of ID3, however, result in similar trees that can vary only due to different tie-breaking decisions. Esmeir and Markovitch (2004) introduced SID3, a stochastic version of ID3 that is designed to sample the version space semi-randomly, with a bias to smaller trees. In SID3, instead of choosing an attribute that maximizes the information gain, we choose the splitting attribute semi-randomly. The likelihood that an attribute will be chosen is proportional to its information gain.\(^1\) However, if there are attributes that decrease the entropy to zero, then one of them is picked randomly. The attribute selection procedure of SID3 is listed in Figure B.2.

For many hard learning tasks such as parity concepts, ID3’s greedy heuristic fails to correctly estimate the usefulness of the attributes and can mislead the learner to produce relatively large trees. In such cases, SID3 can, in theory, produce significantly smaller trees. The probability for this, however, is low and decreases as the number of attributes increases. To overcome this problem, we use a third sampling technique that is based on the recently introduced LSID3 algorithm for anytime induction of decision trees (Esmeir & Markovitch, 2004). LSID3 adopts the general TDIDT scheme, and invests more time resources for making better split decisions. For every candidate split, LSID3 attempts to estimate the size of the resulting subtree were the split to take place, and favors the one with the smallest expected size. The estimation is based on a biased sample of the space of trees rooted at the evaluated attribute. The sample is obtained using SID3. LSID3 is parameterized by $r$, the sample size. When $r$ is greater, the sample is larger and the resulting estimate is expected to be more

\(^1\)SID3 ensures that attributes with gain of zero will have a positive probability to be selected.
### Procedure LSID3-Choose-Attribute($E, A, r$)

If $r = 0$

Return ID3-Choose-Attribute($E, A$)

Foreach $a \in A$

Foreach $v_i \in \text{domain}(a)$

$E_i \leftarrow \{ e \in E \mid a(e) = v_i \}$

$min_i \leftarrow \infty$

Repeat $r$ times

$T \leftarrow \text{SID3}(E_i, A - \{a\})$

$min_i \leftarrow \min (min_i, |T|)$

$\text{total}_a \leftarrow \sum_{i=1}^{[\text{domain}(a)]} min_i$

Return $a$ for which $\text{total}_a$ is minimal

---

**Figure B.3:** Attribute selection in LSID3

accurate. Therefore, LSID3 is expected to improve with the increase in $r$. Figure B.3 lists the procedure for attribute selection as applied by LSID3. Because our goal is to sample small trees and not to always obtain the smallest tree, we use $LSID3(r = 1)$ as a sampler. Observe that $LSID3$ is stochastic by nature, and therefore we do not need to randomize its decisions.

### B.3 Empirical Evaluation

We tested Occam’s empirical principle, as stated in Definition 1, on 20 datasets, 18 of which were chosen arbitrarily from the UCI repository (Asuncion & Newman, 2007), and 2 which are artificial datasets that represent hard concepts: XOR-5 with 5 additional irrelevant attributes, and 20-bit Multiplexer. Each dataset was partitioned into 10 subsets that were used to create 10 learning problems. Each problem consisted of one subset serving as a testing set and the union of the remaining 9 as a training set, as in 10-fold cross validation. We sampled the version space, $TDIDT_A(E)$, for each training set $E$ using the three methods described in Section B.2.2 and tested the correlation between the size of a tree (number of leaves) and its accuracy on the associated testing set. The size of the sample was ten thousand for $RTG$ and $SID3$, and one thousand for $LSID3$ (due to its higher costs). We first present and discuss the results for consistent trees and then we address the problem of pruned, inconsistent trees.
Table B.1: Testing Occam’s empirical principle using different sampling methods that produce consistent trees. For each method we report the accuracy, tree size, and Spearman’s correlation coefficient ($\rho$) averaged over all 10 partitions. We also report the number of times (out of 10) that a negative correlation was found to be statistically significant with $p = 0.95$.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Acc. (RTG Size)</th>
<th>Acc. (SID3 Size)</th>
<th>Acc. (LSD3 Size)</th>
<th>$\rho$</th>
<th>$\rho$</th>
<th>$\rho$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREAST-W</td>
<td>92.9±2.8, 128±14</td>
<td>93.1±2.7, 108±11</td>
<td>94.3±1.6, 77±4</td>
<td>-0.1</td>
<td>8</td>
<td>-0.1</td>
</tr>
<tr>
<td>BUPA</td>
<td>59.7±8.0, 213±10</td>
<td>63.4±7.3, 94±6</td>
<td>61.9±7.4, 69±3</td>
<td>0</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>CAR</td>
<td>72.8±4.6, 647±27</td>
<td>79.7±5.8, 520±95</td>
<td>91.9±1.1, 285±11</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>CLEVELAND</td>
<td>51.6±6.9, 188±7</td>
<td>50.2±7.2, 134±7</td>
<td>46.1±7.4, 98±5</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CORRAL</td>
<td>73.3±22.9, 15±3</td>
<td>81.6±19.6, 10±2</td>
<td>89.8±8.3, 7±1</td>
<td>0.2</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>GLASS</td>
<td>55.6±9.9, 135±8</td>
<td>62.3±9.3, 57±5</td>
<td>68.8±8.4, 39±3</td>
<td>-0.1</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>HUNGARIAN</td>
<td>72.8±7.4, 125±10</td>
<td>73.3±7.2, 65±6</td>
<td>69.4±7.5, 47±3</td>
<td>-0.1</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>IRIS</td>
<td>88.9±7.3, 39±9</td>
<td>92.8±4.5, 12±2</td>
<td>93.8±2.7, 8±9</td>
<td>-0.1</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>MONKS-1</td>
<td>91.1±4.6, 203±42</td>
<td>97.0±3.8, 113±55</td>
<td>100.0±0.0, 28±4</td>
<td>NA</td>
<td>NA</td>
<td>0.2</td>
</tr>
<tr>
<td>MONKS-2</td>
<td>77.8±4.4, 294±9</td>
<td>75.5±4.6, 289±8</td>
<td>77.7±3.1, 259±3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MONKS-3</td>
<td>88.3±5.4, 171±46</td>
<td>96.0±2.5, 77±33</td>
<td>96.7±0.4, 38±2</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MUX-20</td>
<td>55.7±6.3, 388±14</td>
<td>56.6±6.5, 249±13</td>
<td>86.1±11.8, 89±35</td>
<td>-0.9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NURSERY</td>
<td>77.2±5.9, 327±55</td>
<td>93.1±1.8, 1583±295</td>
<td>98.1±0.5, 656±54</td>
<td>-0.4</td>
<td>0.10</td>
<td>0</td>
</tr>
<tr>
<td>SCALE</td>
<td>72.2±4.2, 394±11</td>
<td>71.7±4.1, 389±11</td>
<td>70.7±3.8, 352±25</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SPLICE</td>
<td>61.1±17, 1977±101</td>
<td>60.5±4.2, 1514±112</td>
<td>89.3±1.8, 355±23</td>
<td>-0.5</td>
<td>0.10</td>
<td>0</td>
</tr>
<tr>
<td>TIC-TAC</td>
<td>72.3±4.8, 468±34</td>
<td>80.2±4.5, 311±30</td>
<td>87.7±3.0, 166±11</td>
<td>-0.1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>VOTING</td>
<td>89.2±5.8, 52±12</td>
<td>92.8±4.3, 26±5</td>
<td>94.5±3.1, 15±2</td>
<td>0.2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WINE</td>
<td>78.6±10.2, 73±12</td>
<td>90.6±6.7, 13±3</td>
<td>91.7±4.3, 7±1</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>XOR-5</td>
<td>50.7±10.6, 136±8</td>
<td>51.9±11.8, 108±11</td>
<td>96.5±7.7, 39±11</td>
<td>-0.8</td>
<td>0.10</td>
<td>0</td>
</tr>
<tr>
<td>ZOO</td>
<td>90.0±7.4, 24±5</td>
<td>91.9±6.2, 18±4</td>
<td>94.2±3.5, 11±1</td>
<td>-0.1</td>
<td>NA</td>
<td>0</td>
</tr>
</tbody>
</table>

B.3.1 Consistent Decision Trees

Figure B.4 plots size-frequency curves for the trees obtained by each sampling method for three datasets: Nursey, Glass, and Multiplexer-20 (for one fold out of the 10). Each of the three methods focuses on a different subspace of TDIDT$_A(E)$, with the biased sampling methods producing samples consisting of smaller trees. In all cases we see a bell-shaped curve, indicating that the distribution is close to normal. Recall that RTG does not uniformly sample TDIDT$_A(E)$: a specific small tree has a better chance of being built than a specific large tree. The histograms indicate, however, that the frequency of small trees in the sample is similar to that of large trees (symmetry). This can be explained by the fact that there are more large trees than small trees. To further verify this, we compared the distribution of the tree size in an RTG sample to that of all trees, as reported in (Murphy & Pazzani, 1994) (Mux-11 dataset). The size-frequency curves for the full space and the sampled space were found to be similar.

Occam’s empirical principle states that there is a negative correlation between
the size of a tree and its accuracy. To test the significance of the correlation, we used the non-parametric Spearman correlation test on each of the samples.\footnote{All statistics were computed using the The R Project package (R Development Core Team, 2005).} Spearman’s coefficient $\rho$ measures the monotonic association of two variables, without making any assumptions about their frequency distribution. For a paired sample of $X$ and $Y$, $\rho$ is defined as $1 - 6 \sum d_i^2/(n(n^2 - 1))$, where $d_i$ is the difference in the statistical rank of $x_i$ and $y_i$. There is a special correction for this formula in the presence of ties.

Table B.1 lists summarizing statistics for the RTG, SID3, and LSID3 samplers. The validity of Occam’s empirical principle, tested by Spearman’s method, is listed in the rightmost column. For each sampling method, for each dataset, we count how many times, out of the 10 folds, the null hypothesis $H_0$ (which states that the variables are not correlated) can be rejected at an $\alpha = 5\%$ significance level, against the alternative hypothesis that the correlation is negative.

The results indicate that when random sampling (RTG) is used, Occam’s empirical principle, as measured by Spearman’s test, is valid for almost all problems, except for Scale. The results for the SID3 sampling method indicate that even when focusing on smaller trees, simplicity is beneficial as a bias for accuracy. Again, except for the Scale dataset, there is a strong inverse correlation between size and accuracy. The numbers indicate that the correlation is weaker than the RTG case, yet still significant across most domains.

The LSID3 samples focus on very small trees. In several cases, LSID3 could reach the smallest tree possible. Again the negative correlation was found to be significant for most domains.\footnote{Note that in some cases, significance tests could not be computed due to the large number of ties (indicated by NA in the table).} However, the number of cases where the null hypothesis could not be rejected is higher than in the previous samplers. One possible reason for this phenomenon is that LSID3 samples trees from a much tighter size-range. Hence, there is an increased probability for finding two trees, $T_1$ and $T_2$, with the size and accuracy of $T_1$ being greater than $T_2$.

As indicated by the frequency curves, the different sampling methods cover different portions of the space, but sometimes overlap. An interesting question is whether the conclusions hold if we analyze all samples together. Note that it is not statistically valid to merge the samples of RTG, SID3, and LSID3 due to their different distributions. Nevertheless, we measured the correlation statistics for the combined samples and found that the results are very similar, with a strong negative correlation between the size of a tree and its accuracy.

To illustrate the correlation between the size of a tree and its accuracy, we grouped the trees from a single run into bins, according to their size, and calculated the average accuracy for each bin. We did this for each of the sampling
Table B.2: Testing Occam’s empirical principle when the trees are pruned. For each method we report the accuracy, tree size, percentage of pruned leaves, and Spearman’s correlation coefficient (ρ) averaged over all 10 partitions. We also report the number of times (out of 10) that a negative correlation between the size of the unpruned trees and their accuracy after pruning was found to be statistically significant with $p = 0.95$.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>RTG</th>
<th>SID3</th>
<th>LSID3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc.</td>
<td>Size</td>
<td>%P</td>
</tr>
<tr>
<td>Breast-w</td>
<td>93±3</td>
<td>128±14</td>
<td>82</td>
</tr>
<tr>
<td>Bupa</td>
<td>63±7</td>
<td>213±10</td>
<td>86</td>
</tr>
<tr>
<td>Car</td>
<td>81±3</td>
<td>647±77</td>
<td>93</td>
</tr>
<tr>
<td>Cleveland</td>
<td>55±6</td>
<td>188±7</td>
<td>73</td>
</tr>
<tr>
<td>Corral</td>
<td>75±20</td>
<td>15±3</td>
<td>67</td>
</tr>
<tr>
<td>Glass</td>
<td>59±9</td>
<td>135±8</td>
<td>73</td>
</tr>
<tr>
<td>Hungarian</td>
<td>78±5</td>
<td>125±10</td>
<td>92</td>
</tr>
<tr>
<td>Iris</td>
<td>92±7</td>
<td>39±9</td>
<td>82</td>
</tr>
<tr>
<td>Monks-1</td>
<td>91±6</td>
<td>203±42</td>
<td>89</td>
</tr>
<tr>
<td>Monks-2</td>
<td>66±2</td>
<td>294±9</td>
<td>97</td>
</tr>
<tr>
<td>Monks-3</td>
<td>94±5</td>
<td>171±46</td>
<td>92</td>
</tr>
<tr>
<td>MUX-20</td>
<td>57±6</td>
<td>388±14</td>
<td>84</td>
</tr>
<tr>
<td>Nursery</td>
<td>88±2</td>
<td>327±51</td>
<td>97</td>
</tr>
<tr>
<td>Scale</td>
<td>69±4</td>
<td>394±11</td>
<td>90</td>
</tr>
<tr>
<td>Splice</td>
<td>63±5</td>
<td>1977±101</td>
<td>91</td>
</tr>
<tr>
<td>Tic-Tac</td>
<td>78±4</td>
<td>468±34</td>
<td>88</td>
</tr>
<tr>
<td>Voting</td>
<td>94±5</td>
<td>52±12</td>
<td>92</td>
</tr>
<tr>
<td>Wine</td>
<td>82±10</td>
<td>73±12</td>
<td>84</td>
</tr>
<tr>
<td>XOR-5</td>
<td>55±12</td>
<td>136±8</td>
<td>85</td>
</tr>
<tr>
<td>Zoo</td>
<td>90±8</td>
<td>24±5</td>
<td>53</td>
</tr>
</tbody>
</table>

methods. Bins with less than 5 observations were discarded. Figure B.5 plots the results for the Nursery, Glass, and Multiplexer-20 datasets. The error bars represent confidence intervals, with $\alpha = 5\%$.

Again, the graphs show a strong correlation between size and accuracy, confirming Occam’s empirical principle. For the Nursery and Multiplexer-20 datasets the correlation is strong for all 3 samplers. For Glass, the correlation is weaker when the trees are very small. These graphs, which represent each size range in its own bin, indicate that the positive support we showed for Occam’s principle is not a result of a size bias in our sampling methods.

B.3.2 Pruned Decision Trees

Formally, Occam’s empirical principle, as defined in Section B.1, is applicable only to consistent hypotheses. Most decision tree learners, however, do not necessarily
produce consistent models because they output a pruned tree in attempt to avoid overfitting the data. This is usually done in two phases: first a tree is grown top-down and then it is pruned. In our second set of experiments, we examine whether taking simplicity as a bias in the first stage is beneficial even when the tree is later pruned. Therefore, we measure the correlation between the size of unpruned trees and their accuracy after pruning.

Table B.2 summarizes the results. The same statistics were measured with a single change: the accuracy was measured after applying error-based pruning (Quinlan, 1993). As in the case of consistent trees, examining the overall correlation between the size of a tree and its accuracy indicates that for most datasets the inverse correlation is statistically significant.

Table B.2 also gives the percentages of pruned leaves. While the trees produced by RTG were aggressively pruned, the percentage of pruned leaves in LSID3 was relatively low. This is due to the stronger support for the decisions made at the leaves of smaller consistent trees.

### B.4 Conclusions

Occam’s razor states that given two consistent hypotheses, the simpler one should be preferred. This principle has been the subject of a heated debate, with theoretical and empirical arguments both for and against it. In this work we provided convincing empirical evidence for the validity of Occam’s principle with respect to decision trees. We state Occam’s empirical principle, which is well-defined for any learning problem consisting of a training set and a testing set, and show experimentally that the principle is valid for many known learning problems. Note that our study is purely empirical and does not attempt to reach an indisputable conclusion about Occam’s razor as an epistemological concept.

Our testing methodology uses various sampling techniques to sample the version space and applies Spearman’s correlation test to measure the monotonic association between the size of a tree and its accuracy. Our experiments confirm Occam’s empirical principle and show that the negative correlation between size and accuracy is strong. Although there were several exceptions, we conclude that in general, simpler trees are likely to be more accurate. Observe that our results do not contradict those reported by Murphy and Pazzani (1994), but complement them: we do not claim that a smaller tree is always more accurate, but show that for many domains smaller trees are likely to be more accurate.

We view our results as strong empirical evidence for the utility of Occam’s razor to decision tree induction. It is important to note that the datasets used in our study do not necessarily represent all possible learning problems. However, these datasets are frequently used in machine learning research and considered
typical tasks for induction algorithms.

We also examined the applicability of Occam’s razor to pruned trees. For most domains we found the correlation between the size of a tree before pruning and its accuracy after pruning to be statistically significant. These results indicate that taking Occam’s razor as a preference bias when growing a tree is useful even if the tree is post-pruned.

An interesting research direction that we plan to pursue is to test the correlation between a tree’s accuracy and a variety of other measures, such as expected error and description length.
Figure B.4: Frequency curves for the Nursery (top), Glass (middle), and Multiplexer-20 (bottom) datasets
Figure B.5: Correlation between size and accuracy using RTG (left-most), SID3 (middle), and LSID3 (right-most). The upper graphs represent the results for the Nursery dataset, the graphs in the middle row stand for the Glass dataset and the lower graphs stand for the Multiplexer-20 dataset.
References


157


160


אלגוריתמים כל-עת לليمית מסורגי כל-עת

מאמר אסמי
אלגוריתמי כל-עת לلزمדת מסוגי כל-עת

טיפוג על מחק

לSEM מילוי חלקי של הדרישות לكلفות ההזרה

דוקטור לפילוסופיה

מאמר אסמי

וונש לטכנולוגיה — מכון טכנולוגי לישראל

אול הצעות

חיפה

ספטמבר 2008

Technion - Computer Science Department - Ph.D. Thesis PHD-2008-12 - 2008
המתקד מועצה להנדסת פרויקטים של שאלת מרחבי
הפכילות למודי המ瑚ב

אצי מודנה לסכין. מלונים (המשגחה הלשון של הנושא) קשור墨西哥 ואחר
וג congפיך על התחמך הספיםקות הדייבר ברוחדמורי

Technion - Computer Science Department - Ph.D. Thesis PHD-2008-12 - 2008
תקציר

השימוח בשיטות מתוחכם המערמריםの大まとめ לייצר מוסון עבורי יישומית מחייה
הוס-ים וMethodManagerים תורם ויתור. יישומיות של מערכות מחשבים שנירו צביר.
вед腩 הצרשות על ידי המפעלים. אף Baths התיאוריה והתוכנה של פיתוי עכל הפרובוקט והஓיב
ולא התאמנה בחרילה המ닙דית והissions הפיך לארוג כדי עקב הפרובוקט והוחדל
והפלס של יישומיות אלה. עבורה זו לא מצוינת מערכות תשיות המאשחרות לפועל
ובDebugger מראוקוויט צאל.

הנה,公告称. שכרפ יפואיה yalית הלתшение בעטניקה צדיפת הפרוגרה
הוסמיה שהופך במחמודי מכונת לאונובקאי מבית השכול בל הבודק השגות אוות ששי
הוסמק הליחות מהמד רמות. למשל, מחירים של חניכים פושטים זמנים
 haciendo זה וממק מעדכון. דובק והמק כל הש城市管理 ספירות הזמנים
בשלעית ומקוהו יותר. שדיקות מראוקוויט הוא. משבח, שלעת ינורש רכז לפפנוכיס
כרות ביס_Columns. ובלשנווד שיתוף浦ליל לשלעש מ⚘ים שזנים. הפק שבסיסונ
אפרל הליחות הבינר דודי שערטיים מה القوم מעסיקו אדם בינר מחלה. המרפ הuptools
Mspווניג במדוץ אושי יקח הוחובה את כל סוכי הזרות היה לيمة את.

cדונה נספה. הינוח שמעלב ליווור חמורה מעטני לבודית מעגש ואטרקטית
לאורך בקרת איקון והקשרו ורכיבים המיועדים. בושו הלחין ייאטו ALSO רכיב מעגלה.
מקולה צוות ניצן חלף מדיו צע וברמות תכנית של רכיבי. על סוס מسعد הזיו ל הזהלט

Technion - Computer Science Department - Ph.D. Thesis PHD-2008-12 - 2008
האמノ הריכוזתקין: עטאזד,חלק מהמודלים בכלים לחם רן עלולותlastic뼈
יתאצוי.כל,משייקוליעיל僬,הפועלוכללהרשאותבולטיםשלושה
בשתיוםוןבבלבכלרכז,מכן,עלהפוסעיםשהבלונכםלהשתחמותשם
הישובסמהכליםאינועלולהעלשליהשונף.

בשני המק_indicesuggle,מקסילבנהיהבשמועשלורןключаלהשלוט
השורוה bày.על-ך,ך เมษכוי הפרת-risk וכלпередוהייהוכמסוליחק
משיאהבריםיתויותっこלהחמידאםמודליםהמודיםישחקעלמוןבשלhisor.
אלגוריתמיםходитוליםשםشرحאתביוגנטיקוםהמודליםשמושביםמקליפלםלתו
נקראיםאלגוריתמיםכל-רד.התרומותהמודוליתשלעדנהויאשהמערכותהמידההדוע.
יתשמשוכל rampantמשאותכלמידהתמפוסכדיליופלמודיםטוביםיתוי,ךומביןית
יכלצלתוריישליוהןומבחינותעלילותשמשםבה.

בלינכתשלעומכםשםאילרוגיםיהקורסיבי𝚕למידתאםחלチンשלשווה
לעם.האלגוריתמיםמאפרשלhamsterידעאםחלチンשלשווהכדילכלחלチンשלשווה
שוביתיתויות mmcיתים:במעךشهادאותשלמעךהפוסעיםלל Usa חלק
ה睫מידבורהמודימה Cornwallتخמדופרת.ה더מהוםהמודיםשלהפוסעיםאינודרומאתמרות
הعجزיםשהמתחתכלפעל,ושופטיםאתהפוסעיםעלפיי mấyדומים
במקסימיםבטאם potràלתלותשל SqlConnection LIBRARY אזור דוחה,ויכאשתמחפיש
התעשיםאכתבית,משתמעהםלאכתםשלמהوضوعיםוכדיליעсрקר
המודיםיתוי.אלגוריתמיםלתמידתויביםשל😷וספיםלאירוםמודיםמודיםפועליםלי
יעקוריםחורשלאפסותמודיםיאחרמודיםפועליםכלשפחה,המסביכים
היותואדונאותהמידה.עלפיוותを集め,אךשרكيفהםбережשנימה
ל الاسلامהתופעה,שילאבדעתאתההפוסעיםיתוי.ברحصرשלמערכותלألمורה:אם
 заявкלהסבסאתדונאהוםלתמידתהבודעהשליםמודים,סיבורשלמוד地坪ותיתור
tוהיההלוכדתכללשהורהית.ניסוייםראששכרואאיםאימתישחקיו mới產業
ועציו-הלכלכה.אילך,ברוחמתэтомуודים lehetפסולאםומכייםאתהשרמותתי

Technion - Computer Science Department - Ph.D. Thesis PHD-2008-12 - 2008
 besar מוערים מתרשים שוניות של בדיקת הלמידה המ раствוחית איננה הבחזר לחדר

ה독ק אלא למעור את העזרת הכהן: יתיימו עיס קניוט בין מבצעים מחוז
מקים. ואעך מזריים שוניות בלתי צעויות קורים. מצט, נגלה ולא משמע ממד תמר ליאכון. במקרא דצ או مضיים לשלוט על עזרת השימוע האפשרי. עלייה וקללות של מרכזים: עזרת המibrator, אורח זא ומיצרisce על-

עלון המבנה המᴗעת על כבושת הלמידה, עלון התשווית, אורח זא ומיצרisce

בأمפוזוט מדות סטטיסטיים.

בנסס אלילגנטו על חום הממקצה הלוחמל הלמידה, מקימי רוב שעלה האמונות
שנה הבבל של המושאלים המוקצים להחליל הסיווג. כי טלעב מבואים, פיתוחו-
משמורת אלגוריתמים למשה שמשותלים לפני מתוחצנים כל-זע, הן זאויים ממסורש שופ-
קבלי התלחות חותת מבצל מבProcessEvent על שהוא הסיווג. תקרור עזרת הסיווג יולד
חלוב בצרא שולוים דהוד הלמד.adoo בנובה רוק לע-
לא הלוביצ מראשם לולות דווע קלמד.adoo בנובה רוק לע-
לא הלוביצ מראשם כלא. עזר המקרה הריאון, אנ בוגו שפ מזרחיים עלילת זאויים
שון י WIFI חנה והפ מוסיג את חחר המתחנץ המפרו. עזר המקרה בח-
התקלע אנוי דיר מראשם. אנ בוגו אוכס עייס המזורים שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שמו שمو-

ביס שום. אנ bi קצוב יוד קליפ השמחה, מהתשומת ביען הממדיא יבוד. אורח

סריס סדרתיות את השפעה מחויל ליקר עד שואליים משישונים. במקרא דצ ההחל.

בשו ממקסמלת על-סמס יז רואן שטסיק בבלן.

כאלאגוריתמים סטטיסטי, הולמד שחיב מפגיעת טפי תלחת על פולי קדוקות מועמות
מקומי. אילויג עליים הלוחמל האלגוריתמים המהמודים. את גישה של-

גא הרות נתי-מש של התלחת, המזימה מידע להפיעל משטחי הזרות: מצוות זו-
הזהב be המים של המוקצים תלחל תלחל
החלימה של בטיחות מראשים. וזרורי ברות-פיריה be קווי לוגר את האלגוריתמים הלוד
בכל מפל 들בוק והדוב. בז公園 צוד קושאיין מנסים מאמציה המהלוד התמודדות

iii
쉽משו לברית הקדומות.

cדי לתحوال את הביצועים של השיטות המערבות לאלח הקיימות. ערכו מספר

הלכה לפנים: בפרק 2 אוגנטי רקע במנחתי הלמריד -על חיות מנצlund

מושבעים, מתחמיים את הסביבתיות השונות את חיות הלמריד את מטרת הלמרידה

שאותו ממוגנים חלקית, ג'י. בנסף, מגדירים בפרק 3 מספר מתמטים בחת גאווה

משתמשים במדרש העבידה. פור 3 ובעכנכת הלאורטימים הלמדדה המחודשים מ-

אז גווע לכל-עד ואת השיא של הלמריד את התהליך הלמריד עלייה או אופני מקריסטיבי

להויכר את אימט המנהיגים הצהרה לפלטול צמחיים בו עזרתי דינמיה סטטוסטיאיט

של ממרחק הפיסים מהתכנס לכל ממצ. בנוסף, פור 4ajaran ט饨 יז מלהenheim את השיג

וזי לכל חלב פעמיים וממדידת חיות והמדידת אורח חיים כל nors.

מצא込まれ ממצודה הלמריד או הלמלת העולים שמראות את עולפו השימור של-

לז. בפרק 5 אני מתריכים לממדידת מצרכים שמסכות לעבר התהליך הלמריד את-

הלאורטימים בפרק 5-5 ממקהל עם תובנות את אפימיי הכמות הלאורטימים בין

הפטריטים שונים. בפרק 6 אני מתריכים על מצרכים של המרתה נוספות בין

הסיבות שונים. בפרק 7 אני מתריכים על מצרכים שונים בין

משכים ומחקר זה המופעים כלויים מחקר עדימי.