Finding Cost-Efficient Decision Trees

by

David Dufour

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2014

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Decision trees have been a popular machine learning technique for some time. Labelled data, examples each with a vector of values in a feature space, are used to create a structure that can assign a class to unseen examples with their own vector of values. Decision trees are simple to construct, easy to understand on viewing, and have many desirable properties such as resistance to errors and noise in real world data. Decision trees can be extended to include costs associated with each test, allowing a preference over the feature space. The problem of minimizing the expected-cost of a decision tree is known to be NP-complete. As a result, most approaches to decision tree induction rely on a heuristic. This thesis extends the methods used in past research to look for decision trees with a smaller expected-cost than those found using a simple heuristic. In contrast to the past research which found smaller decision trees using exact approaches, I find that exact approaches in general do not find lower expected-cost decision trees than heuristic approaches. It is the work of this thesis to show that the success of past research on the simpler problem of minimizing decision tree size is partially dependent on the conversion of the data to binary form. This conversion uses the values of the attributes as binary tests instead of the attributes themselves when constructing the decision tree. The effect of converting data to binary form is examined in detail and across multiple measures of data to show the extent of this effect and to reiterate the effect is mostly on the number of leaves in the decision tree.

Acknowledgements

I would like to thank my supervisor, Peter van Beek and my readers Robin Cohen and Forbes Burkowski for taking the time to make this thesis better than I could have on my own.

The School of Computer Science at the University of Waterloo and all those involved in making it as successful as it is.

Christian Bessiere, Emmanuel Hebrard, and Barry O'Sullivan for their work on "Minimising Decision Tree Size as Combinatorial Optimisation" which has inspired this thesis. Especially to Emmanuel Hebrard for providing direction.

The University of California Irvine Machine Learning Repository[3] and those who contribute to it. Special thanks to Moshe Lichman for the quick response to my request for data.

Most of all I would like to thank my friends and fellow students who were there with me through the duration of this work.

Dedication

I dedicate this thesis to my family, the people who have always been there for me and will continue to be there. Without them I would not have had the time nor inspiration to get to where I am today.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Decision trees are a simple yet powerful tool for predicting the class label of an example. There are many areas which can benefit from the use of decision trees, including image processing, medicine, and finance (these examples and more can be found in [28, p.26]). Trees are constructed from a set of training examples that share attributes and a selection of labels. These examples are recursively split into groups based on the value of the attribute being tested at a given node and sent down a branch that is associated with that value. At the end of these paths of branches are leaf nodes, or a collection of examples that all agree on what class they belong to. So when a new example is sent down the root of the tree, it will be examined by each attribute along the way, moving down the appropriate branches. At the end of a path of branches in the decision tree are similar examples in a leaf node, all sharing a classification that will be used to label the new example.

Hyafil and Rivest[18] show that the problem of finding an optimal decision tree for most measures of quality is NP-complete. Consequentially, a top down, single pass, heuristic-driven induction method is used in the construction of decision trees. Although, in practice on real world data, it might be possible to find decision trees that are smaller, or cost less than the trees found using the heuristic by searching the space of possible decision trees in a clever way. This chapter introduces the problem this thesis hopes to solve, outlines the motivation for this direction of work, states certain objectives, and summarizes the contributions of this thesis.

Generally, smaller trees are sought after when constructing decision trees. The depth of a tree can be kept small by considering the information content of the examples given a split using different attributes. The popular heuristic information gain is one way to accomplish the growth of decision trees with shorter path lengths to leaf nodes. A generalization of

the problem of finding decision trees with short paths from the root to its leaves is to add weights to the attributes at each split in the tree. Costs, risks, delays and other measures of preference over attributes can be included in the heuristic in a way that reduces the expected path length while preferring attributes at each point in the path with low costs. This creates a balancing act between the cost and accuracy of the resulting decision tree and optimizing for one doesn't necessarily lead to the optimization of both. The solution proposed in this thesis to the problem of minimizing the expected-cost of decision tree is to iterate over a subset of viable decision trees with a bias for trees that maximize the 'information gain over cost' heuristic.

1.1 Motivation

In past research, C. Bessiere, E. Hebrard, and B. O'Sullivan[5] made an attempt to use constraint programming and other techniques to find the smallest decision tree consistent with some training data (classifying all training examples correctly). They reasoned that smaller decision trees were preferable for their simplicity, having relatively fewer attributes and perhaps because they would have greater generalization accuracy. Bessiere et al. found that no matter which technique they used (whether it be a SAT-solver, constraint programming, or linear programming), modern optimization methods could not find the smallest decision tree on anything but the smallest of datasets.

Part of the original motivation for Bessiere et al.'s work was to minimize the number of tests conducted on a patient during a diagnosis. Where they minimized the total number of internal nodes and leaves of decision trees, I look to minimize only the number of internal nodes representing tests in the tree. Of course not all tests are the same when it comes to their monetary cost, the time they take, the risk involved, etc. Imagine in the worst case, the need for an exploratory surgery on a patient. Although a decision tree that classifies all examples could be built with just one testing node (results of an exploratory surgery), the cost of that test would be very high for the patient. In between this and conducting an infinite number of zero costs tests that provide no information, we find limited resources of hospitals, wait times for biopsy and blood test results, financial burdens of different tests and many other reasons why some tests may be preferred over others. So the work of this thesis hopes to extend Bessiere et al.'s model to include a cost for each attribute used in the construction of decision trees. Although the motivation and experiments in this thesis are focused on medical examples, the methods introduced work to generalize past results on minimizing decision tree size by adding weights to each attribute and in turn minimizing the expected cost of the decision tree.

1.2 Objectives

The objective of this thesis is to extend and simplify Bessiere et al.'s model to allow for attributes and classes with more than two values and to associate costs with attributes used to split the examples of the decision tree. The goal was to find decision trees with multi-valued attributes substantially smaller than those found by a heuristic and to find decision trees with expected-cost lower than those found by a cost heuristic.

This thesis introduces and explains a new top-down breadth-first decision tree induction algorithm with backtracking and any-time behaviour (having the best solution so far ready at any-time in the algorithms runtime). The goal is to extend past optimization attempts to allow for a more general approach to minimizing different decision tree measures.

It is also the goal of this thesis to take an in-depth look at data used for constructing decision trees. More specifically, I analyse the effects of converting data to have binary valued attributes and a binary classification.

1.3 Contributions

Bessiere et al.[5] reported finding decision trees significantly smaller than those found using Ross Quinlan's C4.5 algorithm[36] with pruning turned off. Although the D4 algorithm can find smaller decision trees, they are not nearly as small as those generated by Bessiere et al. due to them converting their data to binary. Also, in most cases the expected-cost of decision trees cannot be reduced past the expected-cost of decision trees found using the standard single pass heuristic methods. I also show that converting the attributes in data to many attributes with two values, results in smaller decision trees than the unpruned trees generated by the C4.5 algorithm. This goes against expectations when considering the straight conversion of a decision tree with multi-value attributes to a decision tree with binary value attributes, but accounts for some of the results seen in Bessiere et al.'s work.

In this thesis I introduce a new algorithm that I call D4. It is a descendant of the Ross Quinlan's ID3 algorithm[35] with some extensions that allow constraints and searching of the solution space that a simple heuristic cannot provide. It runs on categorical data with any number of attributes, values per attribute, and examples. A top-down breadth-first approach is taken when generating decision trees, with the addition of backtracking when certain bounds are passed. The D4 algorithm is highly customizable in the way it performs a search. It can generate a single decision tree equivalent to a decision tree induction algorithm that simply maximizes a heuristic or in contrast can do a complete search of

all decision trees that meet a certain criteria. Of course as the number of attributes and examples get large in a dataset, the less likely it is for a complete search to be accomplished. The D4 algorithm allows for searching a subset of the solution space as well as randomly generating decision trees with bias from a heuristic. The algorithm is an any-time algorithm and will return the best decision tree found so far when stopped. Although it cannot always find decision trees that perform better under a certain measure, the benefits from the extensibility and customizability of the algorithm make D4 a worthwhile addition to the ID3 family.

1.4 Outline of the Thesis

The remainder of this thesis will be structured as follows.

Chapter 2 starts off by explaining the history of decision tree induction. Decision trees are explained in detail and measures of both size and cost of attributes are examined. Ockham's razor as it applies to decision trees is defined and debunked as being more than a guideline for constructing accurate decision trees. Chapter 2 also discusses different heuristics used in decision tree construction, missing values in training and test data, and the pruning of decision trees.

Chapter 3 is an examination of several decision tree induction algorithms. The famous depth-first ID3 and C4.5 algorithms are explained and a best-first decision tree induction algorithm is examined to contrast D4's breadth-first approach to decision tree induction. ID3+ introduces the idea of backtracking in decision tree induction to avoid pitfalls such as insufficient examples and features during decision tree construction, which my algorithm, D4, expands on. Bessiere et al.'s work using constraint programs to optimize decision tree size is discussed and critiqued as this work is a direct extension of it. Finally, ICET, an evolutionary approach to decision tree induction is discussed for its use of attribute costs and misclassification costs.

Chapter 4 discusses the D4 algorithm introduced in this thesis, while Chapter 5 evaluates its effectiveness. Chapter 5 contains other results pertaining to the search for smaller or lower cost decision trees using backtracking and randomization.

Chapter 6 closes off the thesis with remarks on what has been accomplished and an inclusion of future directions for the optimization of decision tree cost. Appendix A at the end of this thesis is included to describe a remodelling of the constraint program in [5] to work on Choco[19] and other constraint solvers. This model does not appear anywhere

else in the thesis as it does not achieve anything that has not already been accomplished, but relates to my work and can be helpful to those looking to explore this area further.

Chapter 2

Background

The background contains a brief overview of decision trees, their size, and how to construct them. It will begin by discussing the problem of finding the smallest/lowest-cost decision tree, where it is used, what the data decisions are built from look like, why it is so hard to find small decision trees, and even why we are motivated to find them.

The first section will briefly describe decision trees and the decision tree construction problem. Then Ockham's razor will be discussed and how it relates to constructing decision trees will be explained. The popular heuristic, information gain, its use of entropy, and why heuristics are used at all will be examined. As this thesis looks to understand minimizing the expected cost of decision trees, heuristics that take cost into account will also be surveyed. Although my proposed algorithm does not take missing values into account and does not use pruning, both are important when considering decision trees and will be briefly explained at the end of this chapter.

A basic understanding of algorithms and complexity in computer science is assumed. The goal of this chapter is to give the reader an outline of the problem of finding decision trees of low expected cost and the literature surrounding that problem. Further information about decision trees can be found in the survey by Murthy[28] and information about cost-sensitive decision tree induction algorithms can be found in the survey by Lomax and Vadera[24].

2.1 Decision Trees

Decision trees have been used in machine learning for many years and their widespread use can be attributed to their ease of construction and readability. There are many application areas that decision trees are used in: some examples taken from [28, p.26] are agriculture, astronomy, financial analysis, image processing, manufacturing and production, medicine, plant diseases, and software development.

Hyafil and Rivest[18] show that constructing optimal binary decision trees, with the smallest number of expected tests to classify an example, is NP-complete. They prove completeness by taking the known NP-complete problem $3DM$, finding a three-dimensional matching, reducing it to $EC3$, the exact cover set problem where the subsets available contain exactly three elements, and then finally reducing $EC3$ to the problem of constructing an optimal binary decision tree. Their results show that finding a decision tree that classifies a set of examples with the minimum number of tests is NP-complete and the result is generalizable to assigning a cost to each test. Thus, any algorithm that finds a decision tree with the lowest average expected cost of the paths in the decision tree is not going to scale well on large data sets. In these situations, a heuristic is used to find near optimal solutions.

More so, Moret[26, p.603] shows that different decision tree measures, such as tree size and expected cost, are pairwise incompatible and thus optimizing for one measure means not optimizing for the others. Approximation is also difficult. Laber and Nogueira[21] show that the binary decision tree problem does not admit an $o(\log n)$-approximation algorithm, Adler and Heeringa[1] show this holds true for the decision tree problem with weighted tests and Cicalese et al.[9] show it holds true for the weighted average number of tests.

Table 2.1 shows how the data for decision trees are formatted. Each row in the dataset represents an example, or specifically in the case of the infection dataset, a patient that had a series of tests conducted on them. Each column contains the results from a test and is referred to as a value of an attribute. The last column represents what class the example falls in, or again more specifically, the diagnosis of the patient. The aim of constructing a decision tree is to group patients with a similar diagnosis together by comparing the results of their tests in the hopes to later find new patients' diagnoses.

Finding similar patients is accomplished by choosing an attribute and splitting the examples into groups based on which value of the attribute or result from the test they received. Figure 2.1 shows the decision tree grown using C4.5. The 'Antibiotics' test is chosen for the root and the examples 2, 6, 8, 12, and 14 are sent down the leftmost branch because those patients did not respond at all to the antibiotics. Examples 1, 9, and 15

Table 2.1: Dataset of patients labelled with their diagnosis. Each row represents a patient while each column represents the results of a test conducted on a patient. A question mark represents missing test results for a patient.

Patient	Stool	Culture	Temperature	Blood Work	Antibiotics	Infection
1	?	none	normal	?	full	bacterial
2	normal	none	?	?	none	none
3	abnormal	?	high	viral	unclear	viral
4	normal	?	normal	unclear	unclear	none
5	normal	none	normal	bacterial	unclear	bacterial
6	abnormal	viral	high	?	none	viral
7	?	?	?	viral	unclear	viral
8	normal	none	high	unclear	none	none
9	normal	none	high	unclear	full	bacterial
10	normal	?	normal	viral	unclear	viral
11	?	none	high	bacterial	unclear	bacterial
12	abnormal	none	normal	viral	none	none
13	abnormal	?	high	unclear	unclear	none
14	normal	viral	normal	unclear	none	viral
15	?	none	high	?	full	bacterial

went down the rightmost branch because the antibiotics cleared the infection and the rest of the examples were unclear and went down the middle branch. Once the data have been split, it can be seen that the rightmost group of patients have all been diagnosed with a bacterial infection (or are all classified the same). These patients represent a leaf node in the tree that is labelled 'Bacterial'. Any future patients' infection cleared by antibiotics will allow for a diagnosis of a bacterial infection (or any example with the value 'full' for the attribute/test 'Antibiotic' will be classified as 'Bacterial'). The complete tree is built by repeating the splitting process on the remaining groups of examples until all of the subgroups' classifications are the same.

The details behind building decision trees have been left vague up until this point and the following sections will go on to explain the effects of decision tree size on accuracy, how to choose an attribute using a heuristic function, incorporating attribute costs into the model, what to do when values are missing and how to deal with noisy, missing, and erroneous data.

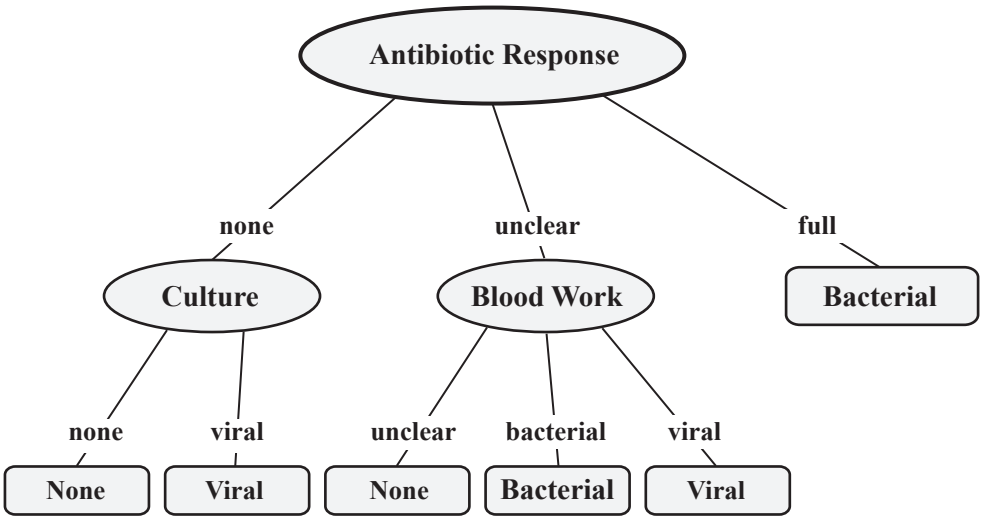


Figure 2.1: Decision tree built from Table 2.1 using the C4.5 algorithm. Patients are first tested on their response to antibiotics, with a full response immediately diagnosing patients with a bacterial infection.

2.2 Data Representation: Effect on Decision Tree Size

When working with decision trees, having attributes and a classification with only two possible values makes induction easier and has some advantages that will be explained later. Real world data does not necessarily come in this form though, so a method for converting examples into binary form is required. I will discuss the two most prominent ways of converting data into binary in this section, both a direct conversion and a method that preserves the cardinality of the attribute set. Consider the following example,

```
@attribute stool {normal,abnormal}
@attribute culture {none,viral}
@attribute temperature {normal,high}
@attribute blood_work {unclear,bacterial,viral}
@attribute antibiotic_response {none,unclear,full}
@attribute class {none,bacterial,viral}

@data
normal, none, high, bacterial, full, bacterial
```

To directly convert the example into binary form using the method used by Bessiere et al.[5], a new attribute has to be created for each value belonging to non-binary attributes, meaning nine attributes in place of five. For example, the attribute *blood_work* will become *blood_work_unclear*, *blood_work_bacterial*, and *blood_work_viral*. So since the example has the value *bacterial* for the original attribute, *blood_work_bacterial* will be set to true while the other two are set to false. Since the class has more than two values, it will be split into two subsets. The values *viral* and *none* will be grouped together as both require no treatment. The example now has nine binary attributes and a binary class (0,0,1,0,1,0,0,0,1,1) and translates to having normal stool, no culture, high temperature, clear blood work, bacterial results from blood-work, non-viral results blood-work, an antibiotic response, a clear antibiotic response, a full antibiotic response, and is classified as a bacterial infection.

Another method for converting features into binary is proposed by Friedman et al[14]. It involves splitting the values of the attribute into subsets and finding the combination of subsets that maximizes information gain. Splitting the values of the attribute into two groups allows the size of the problem to stay the same as no new attributes are created. For the above example, there would be two attributes again that need to be made into binary attributes. The attribute *antibiotic_response* will become the attributes *antibiotic_response_unclear* and *antibiotic_response_full*, which group together no antibiotic response and an unclear response into the same attribute. The class would be grouped in a similar way as before.

Concentrating on binary classes while ignoring attributes, Polat et al.[32] propose a way to turn multi-class classification problems into binary classification problems using a one-against-all approach. The method uses C4.5 to create a decision tree for each class, labelling examples positive if they are in that class and negative otherwise. Using these decision trees together greatly improves classification accuracy over using a single decision tree that has more than two classes.

2.3 Ockham's Razor

The most popular version of Ockham's razor, "entities must not be multiplied beyond necessity" is a common axiom stated without reference to Ockham by John Punch[34]. The better of two equally accurate hypotheses will be simpler and make less assumptions to avoid the possibility of errors. Historically, Ockham's razor has been used in machine learning and specifically when constructing decision trees, to justify smaller hypotheses.

This section will outline why it is important not to rely too heavily on the razor when constructing decision trees.

Ockham's razor, named so by Sir William Hamilton in 1852 in honour of William of Ockham's effective use of similar principles, has many interpretations. In this work, it will be taken to mean that a decision tree with fewer nodes is a simpler tree. There are clear benefits to a smaller tree as it is both easier to understand and contains fewer (possibly costly and time consuming) tests. A false interpretation of the razor would be to attribute higher classification accuracy to these "simpler" decision trees. It should be remembered that Ockham's razor is a heuristic, not a guarantee.

The Ockham Razor bound, derived from [6], is a theoretical bound relating to Ockham's razor. Given two hypotheses with the same error on a training set S of size m, $err_S(h)$, the upper bound on the generalization error on a testing set generated by a probability distribution D with probability greater than $1 - \delta$, $err_D(h)$ will be higher for hypotheses, h, with a longer description length, $|h|$,

$$\forall h \in \mathcal{H}, err_D(h) \leq err_S(h) + \sqrt{\frac{|h| + ln(2/\delta)}{2m}}. \tag{2.1}$$

The above formula may lead us to believe that decision trees with a smaller description length, or size, will always have better classification accuracy on unseen data. In general though, this is not the case and is merely a guarantee on the upper bound of the error.

Domingos[10] proposed separating the razor into two separate razors to consider when discussing knowledge discovery. The first is that simplicity should be preferred in itself and the second and false one can be stated as:

> Given two models with the same training-set error, the simpler one should be preferred because it is likely to have lower generalization error[10, p.2].

Domingos[10] goes on to debunk some of the theoretical arguments for this version of the razor and to discuss the "no free lunch" theorem. Wolpert and Macready[45] state that "any two algorithms are equivalent when their performance is averaged across all possible problems." Another way to look at it is: for every domain where a smaller decision tree has better accuracy on unseen data, there is another domain where the opposite is true. Thus, it is important to know your domain and know what you are optimizing for.

The reader may be familiar with the pruning of decision trees to avoid over-fitting and thus assume that the general goal is creating smaller trees to improve the accuracy on

unseen data. Over-fitting is when the error on training data is low, while the generalization error (error on testing data) is higher. The pruning that happens is to make sure that all the branches in the tree have statistical significance and have enough examples to justify a split or classification. Given sufficient data, there is no need to reduce the size of the tree.

Although Ockham's razor provides no theoretical guarantees that smaller decision trees always have higher accuracy on unseen data, does experimentation on real world data show that smaller trees do provide better classification accuracy on unseen data? Domingos[10] provides two sets of evidence against the idea that smaller decision trees provide higher classification accuracy. There have been experiments that contradict the relationship between simplicity and accuracy, as well as the success of more complex models increasing accuracy in practice.

Murphy and Pazzani[27] investigate the relationship between the size of decision trees and their accuracy on test data. They consider decision trees with perfect accuracy on training data. Their findings show, on average, the smallest consistent decision trees are less accurate than slightly larger decision trees. However, with no prior knowledge, simpler hypotheses should be preferred as they improves accuracy when learning simple concepts and the opposite bias provides no benefit for more complex concepts.

Webb[44] examines Ockham's razor and specifically addresses what he calls Ockham thesis, "Given a choice between two plausible classifiers that perform identically on the training set, the simpler classifier is expected to classify correctly more objects outside the training set." This version of the razor is more general than Murphy and Pazzani's version as it includes decision trees that are not consistent with the training data. By adding complexity to both pruned and unpruned decision trees generated by C4.5, Webb was able to increase predictive accuracy. The increased accuracy provides experimental evidence against Ockham thesis.

Needham and Dowe[29] look to support the validity of Ockham's Razor by considering message length of a hypothesis (the log probability of a hypothesis given the data) as opposed to node cardinality (or the size of the decision tree). Although the new measure for simplicity outperforms node cardinality on certain tasks, the results "did not provide undisputed evidence for Ockham's Razor principle." In fact, they found the shortest message lengths preformed worse than those slightly larger, despite a trend of shorter message lengths having lower prediction error across all the trees considered.

2.4 Heuristics

Heuristics are a way to use available data in a problem to provide guidance in finding a solution when an exhaustive search is not practical. Bessiere et al.[5] show this by finding three exact methods for finding optimal decision trees do not scale well to anything but the smallest data sets. The use of a heuristic allows an algorithm to estimate a solution or part of a solution to aid in the search for an optimal or good solution. In regard to decision trees, heuristics are used to choose which attributes should be used to structure the data. For example, the information gain heuristic uses the number of examples with certain properties to choose an attribute that reduces entropy and consequently the number of splits required to reach a leaf node[35].

Murthy[28, p.10], in work on evaluating and comparing attribute evaluation criteria, comes to the conclusion that no attribute selection rule is superior to any of the others, but the comparison of the heuristics can outline which heuristics one would use in different settings. Although the choice of heuristic can be unclear, choosing the right stopping criterion and pruning methods to avoid over-fitting is important. I found these conclusions to be true when comparing and combining the heuristics used in [2], although the similarity of the heuristics played a significant role.

The following two subsections will describe entropy and information gain, the widely popular splitting criteria that is also used in this thesis.

2.4.1 Entropy

The now famous Shannon's entropy was popularized by Quinlan in ID3 and C4.5's information gain heuristic. Entropy is the measure of unpredictability and the information gain heuristic seeks to reduce the entropy from a set of examples by grouping them so their classifications are more in agreement.

The entropy, H, of a discrete random variable, X, with possible values $x_1, ..., x_n$ and probability $P(x_i)$ is defined as,

$$H(X) = -\sum_i P(x_i) log_2 P(x_i). \tag{2.2}$$

It is assumed $P(x_i) log_2 P(x_i) = 0$ when the probability of a value is 0 or 1.

In reducing entropy, the predictability of a classification becomes higher and with no entropy it can be predicted that the group of examples represents their common classification.

2.4.2 Information Gain

To continue with the process of creating a decision tree, I have included a worked out example of calculating the information gain of an attribute. Information gain is the change in entropy from the current node to the weighted sum of the entropies of its children given the selection of a splitting attribute.

The entropy function for the infection database example is the sum of the probabilities of a certain classification/diagnosis (bacterial, viral, none) multiplied by the information content of that classification,

$$\begin{aligned} entropy(\frac{b}{b+v+n}, \frac{v}{b+v+n}, \frac{n}{b+v+n}) = &- \left(\frac{b}{b+v+n}\right) \log_2 \left(\frac{b}{b+v+n}\right) \\ &- \left(\frac{v}{b+v+n}\right) \log_2 \left(\frac{v}{b+v+n}\right) \\ &- \left(\frac{n}{b+v+n}\right) \log_2 \left(\frac{n}{b+v+n}\right). \end{aligned}$$

The above equation uses b to represent the number of examples classified as bacterial infections, v as the number of examples classified as viral infections and n as examples classified as no infection.

Information gain calculates the entropy of the examples after the split on some attribute and subtracts it from the entropy of the examples at the node you are splitting. In this example, $|E_i^b|$ is the number of examples in node i classified as bacterial and $|E_i|$ is the number of all of the examples associated with node i. The set V_k contains the values for attribute k. The set $E_j \subset E_i$ is all of the examples in E_i that have value $v_j \in V_k$ when tested on attribute k. The information gain is then given by,

$$IG(node_i, attribute_k) = entropy(\frac{|E_i^b|}{|E_i|}, \frac{|E_i^v|}{|E_i|}, \frac{|E_i^n|}{|E_i|}) - \sum_{v_j \in V_k} \frac{|E_j|}{|E_i|} entropy(\frac{|E_j^b|}{|E_j|}, \frac{|E_j^v|}{|E_j|}, \frac{|E_j^n|}{|E_j|}). \tag{2.3}$$

Starting at the top of the decision tree for the infection dataset, there are 15 examples with 5 of each of the classifications. The attribute chosen for this example is the one with the highest information gain, antibiotic response. There are 5 examples that have no response, 3 examples with a full response and 7 examples with an unclear result. For each of the three values of antibiotic response, the examples are again split up into their classification to calculate entropy for the examples with each value,

$$\begin{aligned} IG(root, Antibiotics) = entropy(\frac{5}{15}, \frac{5}{15}, \frac{5}{15}) &- \frac{5}{15} entropy(\frac{0}{5}, \frac{2}{5}, \frac{3}{5}) \\ &- \frac{3}{15} entropy(\frac{3}{3}, \frac{0}{3}, \frac{0}{3}) \\ &- \frac{7}{15} entropy(\frac{2}{7}, \frac{3}{7}, \frac{2}{7}) \\ = 1.5 - 0.333 &\times (0 + 0.529 + 0.442) \\ &- 0.200 \times (0 + 0 + 0) \\ &- 0.467 \times (0.516 + 0.524 + 0.516) \\ = 0.535. & \end{aligned}$$

The calculation of each attribute's information gain is done every time an attribute needs to be selected for splitting the examples up and the one with the highest information gain is chosen. An attribute unused by an ancestor node will have to be selected each time the data is recursively split until the examples at each leaf node can be used to confidently label them with a classification.

2.4.3 Cost Heuristics

The original goal of this thesis was to extend the constraint programming method to minimize the size of decision trees proposed by Bessiere et al.[5] to include a cost for each attribute. The reasoning behind extending their method of searching for decision trees using a tree size heuristic is that not all tests are created equal. If the data for an example to be classified need to be collected in some way, finding the value of an attribute could take time, cost resources, or have a significant risk associated with it. The task of minimizing the size of a decision tree without taking into consideration the cost of the tree is incomplete in domains such as health care. Although this thesis only concentrates on the cost of a test, there are many other ways to include costs when building decision trees[43].

There has been some work done on adding costs to attributes in the past, through the inclusion of costs into the greedy search function in some way[30][31][41]. There has also been work done to minimize the cost incurred from misclassification and some work that does both in a way similar to my approach[42]. These are discussed further in Chapter 3.

The cost heuristics considered when designing my algorithm are adopted from Turney's[42] cost-sensitive classification of decision trees. This section looks at the heuristics used in cost-sensitive decision tree construction algorithms IDX, CS-ID3, EG2, and ICET. All of the heuristics share the ratio of information gain over cost and thus share similar performance when minimizing cost measures of decision trees.

The cost heuristic used in Norton's algorithm IDX[30] which was inspired by GOTA[16] considers both change in information gain (ΔI_i) and the cost (C_i) of adding the next branch level i,

$$\frac{\Delta I_i}{C_i}. \tag{2.4}$$

No justification for this ratio is given beyond the past successes of information gain in reducing the height of a decision (the length of the path to a leaf node) combined with the fact that we wish to reduce the expected cost.

Cost-Sensitive ID3[41] (CSID3) modifies Nez's heuristic used in EG2[31] without any justification and ends up with the heuristic,

$$\frac{{\Delta I_i}^2}{C_i}. \tag{2.5}$$

I chose to use the heuristic used in CSID3 in my experiments as none of the heuristics performed better than any other[42] and it contained no parameter to adjust.

The information cost function (ICF) considers a cost/benefit approach to designing the heuristic[31]. Nez also includes the parameter $0 \leq \omega \leq 1$ to allow the user to calibrate the factor of economy, or how much cost is taken into consideration when choosing an attribute,

$$ICF_i = \frac{2^{\Delta I_i} - 1}{(C_i + 1)^{\omega}}. \tag{2.6}$$

There is no information on how to set the parameter ω. Turney[42] uses the heuristic from EG2 in ICET (with the ω parameter set to 1) to allow for greater control over the cost bias, although he claims that there is no reason to choose one heuristic over another.

2.5 Missing Values

A lot of real world data is not complete or error free. There is a strong possibility of encountering missing values for the attributes of training examples and any future examples we hope to classify. There are many approaches to dealing with missing values and some of those will be discussed below. Information about dealing with missing values in data that have attribute costs and misclassification can be found in [23], [22], and [47]. Although I am discussing missing values in the background, my work does not handle or deal with missing values in any way.

When building a classifier, the easiest route would be to completely ignore any examples that have missing values in them. However, if the number of training examples is limited and a high percentage of the data is missing, doing so would be wasting information. Another approach is to only ignore examples with a specific attribute missing in the calculations for the heuristic of that same attribute. This still wastes information and an alternate approach can be found in Quinlan's C4.5[36]. Quinlan handles missing values for attributes by sending each example down every branch with a weight attached to it. If the value is known, then that weight is 1 for its corresponding branch, while the weight is 0 for all other branches. Otherwise, the weight of an example for a given value's branch is the frequency of that value in the examples at the current node. When encountering a missing value for an attribute in an example to be classified, the example is similarly split and weighted according the the probabilities of each possible value.

It is possible that the reason for missing data is the same across all of the examples collected for training. In this situation, missing data is actually information and can be taken advantage of when building decision trees. One way to do this is to add an additional branch to associate with examples that have missing values for an attribute. That way, when future examples with a missing value for an attribute need to be classified and are tested on that attribute, they can be sent down the branch associated with missing values. A similar approach could be to hold examples with a missing value for an attribute at the node where the attribute is tested. This node would be treated as a leaf node when an example to be classified is missing a value for that attribute.

A classification centric design for dealing with missing values builds a decision tree for each unclassified example. If the example has a missing value for an attribute, that attribute is not considered in the construction of the decision tree. This problem relates to the problem of selecting subsets of attributes to learn on and would be used in conjunction with other approaches for building a decision tree when values are missing in the training data.

Lastly, it is also possible to use imputation to replace missing values in the data with a value chosen using statistics about the data. This could be as simple as choosing the average value among the examples or the average value among the examples with the same classification. It can also be much more complicated and use a statistical method or machine learning itself to impute the missing values. For example, setting the attribute with missing values as the class and using the examples with known values, will allow C4.5 to classify the missing values in the data.

2.6 Pruning

A common way to improve the generalization accuracy of decision trees is to prune off sub-trees and replace them with leaves. To avoid over-fitting (the problem of matching the training data while losing accuracy on test data) it is important to have enough data to support a classification. Pruning sacrifices accuracy on training data by combining examples in multiples leaves into one leaf in the hopes of improving accuracy on unseen data. There are two main types of pruning, pre-pruning and post-pruning. Pre-pruning stops the growth of trees when splitting the data is no longer beneficial and results in decision trees that are not consistent with the training data. Post-pruning is done after a consistent decision tree is grown and uses a subset of the training data that was set aside to determine where to prune in the decision tree.

A difficulty with real world data is that it can contain noise that our learners would use in the construction of the model. The idea of pruning was proposed by Breiman[8] as an alternative to stopping the growth of a decision tree. Pruning is the act of revisiting a decision tree after it is constructed and removing sections of the tree that are not supported by the data or contributing to the classification accuracy of the tree. Otherwise trees can become too large and over-fit the data. This means that errors and noise have been used in the construction of the decision tree and the accuracy of the decision tree on unseen data will suffer. The reader is directed towards [36],[28],[24],[25],[8],[7], and [12] for more information about specific pruning methods [1].

2.7 Summary

In this chapter I covered the background for this thesis, namely decision trees and research related to constructing them. I discussed what decision trees are, how they are created

[1] No decision trees were pruned in the making of this thesis.

and the hard problem of finding an optimal decision tree. I also looked at Ockham's razor and how it relates to the decision tree problem. Heuristics, including cost heuristics, were discussed as they are a necessary component when building small, low-cost decision trees. I also briefly looked at missing values in data and what pruning is and how it can improve decision trees.

In the next chapter I will discuss research related to my thesis, including the C4.5 algorithm which I compare my approach against as well as a constraint programming approach that I re-evaluate in Chapter 5.

Chapter 3

Related Work

This chapter covers a few algorithms to give a historical sense of the D4 algorithm introduced in this thesis, it covers different decision tree construction strategies related to depth, breadth, and best first growth of decision trees and describes the algorithms extended by D4 and that I compared D4 against to determine its performance.

The D4 algorithm is a successor to the ID3 algorithm as it does not include the extensions made in the C4.5 algorithm, although extensions of my own are made. In Chapter 5, I evaluate D4 on data with attributes that have costs and use C4.5 to create a simple decision tree using the same cost heuristic used in D4. The ID3+ algorithm, another extension to ID3, introduces the idea of backtracking to resolve issues seen in the construction of decision trees, such as running out of attributes or examples as the tree is grown. I found this idea useful and extended the use of backtracking to include upper-bounds on decision tree size and cost. I also discuss a constraint programming approach that works on binary data, as the results of the approach will be re-evaluated later in this thesis.

3.1 ID3

In 1986, Ross Quinlan introduced ID3, a way to inductively build decision trees[35]. ID3 belongs to a family of learning systems called 'Top-Down Induction of Decision Trees', where a tree is built recursively by splitting up data and sending subsets of the data down branches until some stopping condition is met. Quinlan recognized that the generation of all decision trees for a data set would only be possible for small induction tasks and designed ID3 to build a single, although not optimal, tree from a subset of the available

data. If the decision tree does not classify all of the training data correctly, a subset of the incorrectly classified data is added to the original subset and ID3 is run again. Algorithm 1 presented below uses information gain to choose which attribute to use when splitting the data.

Algorithm 1 Quinlan's ID3

```
Let E be all training examples, A be the attribute
while Decision tree made with E' ⊂ E does not classify E do
    Randomly select subset of examples, E'' where E' ⊂ E''
    E' ← E''
    ID3(E', A)
end while

function ID3(E', A)
    if e ∈ E' belong to the same class then
        return a leaf node labelled with this class
    end if
    if A = ∅ then
        return a leaf node with the most common class as label
    end if
    a ← a' ∈ A that maximizes gain(a')
    for v_i ∈ a do
        Let E'_i ⊂ E' with value v_i ∈ a
        if E'_i = ∅ then
            create a leaf node with the most common class as label
        else
            ID3(E'_i, A − {a})
        end if
    end for
end function
```

The following equation for entropy is the same as Shannon's entropy (equation 2.2) seen in Chapter 2. The number of positively classified examples, p, and the number of negatively classified examples, n, are used to find the probabilities used in the entropy calculation. It is used in ID3's information gain calculation as an impurity measure,

$$I(p,n) = -\frac{p}{p+n}log_2\frac{p}{p+n} - \frac{n}{p+n}log_2\frac{n}{p+n}. \tag{3.1}$$

The equation for E(A) is a weighted sum of the entropies of the possible values, value 1 to value v, an attribute can have,

$$E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I(p_i, n_i). \tag{3.2}$$

Below is the classic information gain heuristic seen in ID3, C4.5 and many other decision tree induction algorithms,

$$gain(A) = I(p, n) - E(A). \tag{3.3}$$

It is noted in the original paper that $I(p, n)$ is redundant as it stays constant for all attributes. The same results can be obtained by minimizing $E(A)$.

A problem with the gain criterion is that it favours attributes with more values. Imagine an attribute called "label" that had a unique value for each example in the data set. This attribute would be chosen at the root using the gain criterion because the information gain would be maximized as all of the branches would contain exactly one example (one classification). This is obviously not helpful as the labelling of the data was arbitrary and provides no information. A solution to the problem proposed in Quinlan's paper is to include the intrinsic value(IV) of an attribute, A, in the selection criteria,

$$IV(A) = -\sum_{i=1}^{v} \frac{p_i + n_i}{p + n} log_2 \frac{p_i + n_i}{p + n}. \tag{3.4}$$

This allows the number of values that an attribute has to be used in the new 'gain ratio criterion'. This ratio takes the attributes with above average gain and finds the attribute that maximizes the following,

$$gainratio(A) = gain(A)/IV(A). \tag{3.5}$$

Although ID3 is simple, the spirit of the algorithm lives on in most decision tree induction algorithms to this date. The D4 algorithm introduced in this thesis is a direct descendant of ID3 with the inclusion of backtracking and costs for attributes.

3.2 C4.5

C4.5 is Quinlan's extension of his original ID3 algorithm[36]. It has the same basic structure as ID3:

1. Run through examples and check for stopping criteria.
2. For each attribute available, calculate information gain/gain ratio.
3. Chose the attribute with highest gain and split examples across groups based on their values. Each value will correspond to a branch that leads to another node with either an attribute to split the new subset of examples or a leaf node with a classification of the examples.
4. Recursively build the sub-trees from the child nodes' examples.

Although the structure is the same, C4.5 is able to handle missing values and continuous attributes, and has pruning and stopping criteria built in.

Missing values in an example are handled by ignoring those examples in gain calculations for the attribute whose value is missing. For the classification of an example with missing values, the example is split at any nodes with attributes for which the example is missing values, into probabilities based on the fraction of examples int the node. For example, if there is no value for culture on an unseen example, the example will be classified as 0.25 viral, 0.75 none as there is one example in the viral leaf and three examples in the none leaf.

Continuous or numeric attributes are handled by creating an attribute associated with a binary split of the values for each attribute in the data set. This groups all of the examples into two parts for the gain calculation, all examples with values less than a particular example are considered to have the same value in the new binary attribute and all examples with values greater than or equal to that example are the other value. The split in the examples with the highest gain is chosen and the examples are passed down two branches.

C4.5 allows the user to specify how many examples are needed to create a leaf node. This prevents classification of unseen data without any data to support that classification. C4.5 also uses all of the training examples to create confidence intervals for post pruning where depending on these error estimates, sub-trees are replaced by leaves or their own sub-trees. This is known as sub-tree replacement and sub-tree raising.

In the results section of this thesis, C4.5 is extended to use the heuristic from Cost-Sensitive ID3 seen in Chapter 2 in place of the information gain heuristic and to acts as a baseline of performance when finding decision trees of low cost.

3.3 ID3+

Xu, Wang, and Chen's[46], Improved Decision Tree Algorithm: ID3$^+$ (see algorithm 2), hopes to improve on the ID3 algorithm by including backtracking. They claim that ID3's information gain heuristic favours decision trees with short path lengths from root to leaf as opposed to correct decision trees that have at least one supporting example in each leaf. Instead of stopping the growth of the tree when the branch runs out of available examples or features, ID3$^+$ will backtrack up the tree and choose the next best feature given some heuristic. This allows their algorithm to search the solution space for a tree that classifies the data perfectly and with enough data in each leaf to support the classification.

Through their experimental results, Xu et al. show that they can find decision trees that more accurately fit the training data on two of the four data sets that were not classified with 100% accuracy by ID3. Although the idea of backtracking is useful, their algorithm can result in no tree being found if there isn't a decision tree consistent with the data.

3.4 Best-first Decision Tree

Friedman et al.[14] introduce the idea of a best-first expansion of decision tree nodes in their work on boosting as seen in algorithm 3. The strategy of stopping growth early to save computation works best when you expand the nodes that have the best information gain first, leaving nodes with less potential to reduce entropy unexpanded. Growth can be stopped after a certain size, amount of time, or entropy has been reached.

Shi[40] proposed a best-first expansion of decision tree nodes, which has certain advantages despite generating the same tree that a depth-first expansion will generate. By expanding nodes in best-first order, pruning can be approached differently. In contrast, depth-first expansion is the simplest expansion with the least overhead and breadth-first expansion is useful for backtracking and making decisions at each layer of the decision tree.

As with most decision tree induction on real world data, after a certain number of expansions and as the tree grows, it will become less accurate. To deal with over-fitting, Shi uses both pre-pruning and post-pruning to contain the size of the decision tree.

Algorithm 2 ID3$^+$

Let E contain the examples and A contain the attributes
function ID3$^+$(E, A)
    if all $e \in E$ are classified the same **then**
        return a leaf node with this classification
    end if
    if A is empty or E is empty **then**
        return NIL
    end if
    while $A \neq \emptyset$ **do**
        $a \longleftarrow$ next best attribute from A
        for all value v_i in a **do**
            Let E_i be the subset of examples with v_i
            if ID3$^+$(E_i, $A \setminus \{a\}$) $= NIL$ **then**
                destroy sub-tree
                goto while
            end if
        end for
        return an internal node with children from ID3$^+$
    end while
    return NIL
end function

Algorithm 3 Best-First Decision Tree

Let A be a set of attributes,
Let E be the training instances,
Let N be the number of expansions,
Let M be the minimal number of instances at a terminal node
function BFTree(A, E, N, M)
  if $E = \emptyset$ **then**
    return failure
  end if
  Calculate the reduction of impurity for each attribute in A on E at the root node RN
  Find the best attribute $A_b \in A$
  Initialise an empty list NL to store nodes
  Add RN (with E and A_b) into NL
  expandTree(NL, N, M)
  return a tree with the root RN;
end function
function expandTree(NL, N, M)
  if $NL = \emptyset$ **then**
    return ;
  end if
  Get the first node FN from NL;
  Retrieve training instances E and the best splitting attribute Ab of FN;
  if $E = \emptyset$ **then**
    return failure
  end if
  if the reduction of impurity of $FN = 0$ or N is reached **then**
    Make all nodes in NL into terminal nodes
    return
  end if
  if the split of FN on A_b would result in a successor node with less than M instances **then**
    Make FN into the terminal node
    Remove FN from NL
    expandTree(NL, N, M)
  end if
  Let SN_1 and SN_2 be the successor nodes generated by splitting FN on A_b on E
  Increment the number of expansions by one;
  Let E_1 and E_2 be the subsets of instances corresponding to SN_1 and SN_2;
  Find the corresponding best attributes A_{b1} for SN_1;
  Find the corresponding best attributes A_{b2} for SN_2;
  Put SN_1 (with E_1 and A_{b1}) and SN_2 (with E_2 and A_{b2}) into NL according to the reduction of impurity;
  Remove FN from NL;
  expandTree(NL, N, M)
end function

Best-first pre-pruning stops the expansion of the trees when the error estimates increase with further splitting. The size of these trees is used as a stopping criteria for the final decision tree. Like all pre-pruning, Shi's method has the potential to stop too soon. Best-first post-pruning solves this problem at the cost of additional computation time.

3.5 Constraint Programming Approach

Constraint programming[38] is a programming paradigm used to solve combinatorial problems where a user declares a problem and specifies constraints for a constraint solver. These constraints limit the values a variable can take on within its domain based on relationships between the variables. A constraint solver assigns values to the variables as it searches for a solution, shrinking the available values for a variable by constraint propagation and backtracking to variable assignments that agree with the constraints.

Bessiere, Hebrard, and O'Sullivan[5] use constraint programming to attempt to solve the smallest decision tree problem. Most approaches to decision tree learning use greedy algorithms and heuristics to grow a single tree from the data, while Bessiere et al. take the principle of Ockham's razor to the extreme and look to systematically search the space of consistent decision trees for the smallest. An example presented for the motivation of finding the smallest decision tree is to reduce the number of tests in a diagnosis.

They formulate the problem of finding the smallest decision tree that is consistent with a set of training examples as follows in Table 3.1.

A decision tree classifies a set of examples $\mathcal{E}$ if and only if $\forall(e_i \in \mathcal{E}^+, e_j \in \mathcal{E}^-)$, $l(e_i) \neq l(e_j)$ (no two examples in a leaf have different classifications). So, given a set of examples $\mathcal{E}$, the goal is to find a decision tree that classifies $\mathcal{E}$ with a minimum number of nodes.

To solve the smallest decision tree problem, Bessiere et al. propose using a constraint programming model. They set an upper bound (initially found using the information gain heuristic) on the size of the decision tree and search for a binary tree with nodes less than or equal to the bound. Most of the variables in the constraint program pertain to the relationship between the nodes and defines the structure of the tree. This is unnecessary overhead as the structure of the decision tree is implied in it's top-down induction and can be reduced to just the variables concerning the feature and examples associated with a node. As well, the majority of the constraints are associated with the structure of the tree. This is unnecessary as they find that this model does not scale well enough to explore any significant portion of the search space. To solve this problem, they subvert any gains their approach may have by employing the information gain heuristic seen in

Table 3.1: Formulation of the problem of finding the smallest decision tree. The left column contains variables related the structure of the decision tree and the data it is built from. The right column contains more information about the variables and how they are related to the decision tree.

Variable	Description
$\mathcal{E} = e_1, ..., e_m$ be a set of examples and $\mathcal{E}^+, \mathcal{E}^-$ be a partition of $\mathcal{E}$	$\mathcal{E}^+$ contains positively classified and $\mathcal{E}^-$ contains negatively classified examples of $\mathcal{E}$
$\mathcal{F} = f_1, ..., f_k$ be a set of features	$e[f]$ is the binary valuation of feature $f \in \mathcal{F}$ in example $e \in \mathcal{E}$
$T = (X, U, r)$ be a binary tree rooted by $r \in X$	$L \subseteq X$ is the set of leaves of T and a decision tree based on T where internal node $x \in X \setminus L$ is labelled by $f(x) \in \mathcal{F}$
$(x, y) \in U$ is an edge labelled with boolean $g(x, y)$	$g(x, y) = 0$ if y is the left child of x and $g(x, y) = 1$ if y is the right child of x
$p(l)$ be a path in T	for $l \in L$, denotes the path in T from the root r to leaf l
$\forall e \in \mathcal{E}$, we can associate the unique leaf $l(e) \in L$	every edge (x, y) in $p(l(e))$ has $e[f(x)] = g(x, y)$

C4.5 to construct the tree top down. Essentially running the C4.5 algorithm multiple times through backtracking, the authors found that this itself is even too slow and must constrain the available features to just the features with top three information gain values at each node. It should be noted that limiting the available features to the top three features when constructing the decision tree is NP-complete.

> DECISION-TREE. Given a set S of n points in R^3, divided into two concept classes 'red' and 'blue', is there a decision tree T with at most k nodes that separates the red points from the blue points?
>
> Theorem: DECISION-TREE is NP-complete[15]

This search method, coupled with a restart strategy, makes it appear as though significantly smaller decision trees are found. In Chapter 5, this thesis claims that the size of the decision tree approaches that of the decision tree found using C4.5 and only after that are some gains made. Their experimental results show that the first decision trees they find are significantly smaller than unpruned decision trees found by C4.5. How can this

be when their algorithm does not look to maximize the same heuristic employed by C4.5? This can be explained by the conversion of the data to have binary attributes to allow it to be used in their model. They do not convert the data when passing it to C4.5 and thus it generates larger decision trees containing empty leaves. It then comes as no surprise that the pruned decision trees created by C4.5 are almost always smaller than the best decision trees found by the constraint program. There are no empty leaves and C4.5 always chooses the feature that maximizes information gain, instead of mucking around with lesser feature choices. Chapter 5, containing the results of this thesis, goes into greater detail on the effect that converting the data to contain binary attributes has on the problem, as well as showing some of the results of Bessiere et al. can be obtained merely by passing the binary data to C4.5.

3.6 ICET

Turney [42] looks at dealing with two types of costs when building decision trees, both costs associated with attributes and the cost of classification errors. The task of minimizing the cost of the decision tree is accomplished by employing genetic programming, an algorithm that evolves a population of biases for the decision trees generated by a modified C4.5. Turney is motivated in the same way as Bessiere et al., in that medical diagnosis requires tests, although Turney also deals with the fact that these tests have varying costs and costs associated with classification errors.

Turney's algorithm for cost-sensitive classification, called ICET (Inexpensive Classification with Expensive Tests) uses a modified version of C4.5 which includes costs. It evaluates the fitness of a bias by the classification cost of the decision tree it grows combined with the cost of classification errors. The biases undergoing evolution contain the costs used in the generation of the decision tree, a parameter for controlling the sensitivity to cost, and a parameter to control the level of pruning used in C4.5.

Turney's experiments show that ICET outperforms basic heuristic single growth algorithms when minimizing cost. It is unclear though if their results are due to a superior algorithm or if the difference in decision tree cost between C4.5 using EG2 and ICET (which uses EG2 as its heuristic) is caused by C4.5 with EG2 not optimizing at all for misclassification cost and thus missing out on half of the available information. Unfortunately, they do not test the performance of ICET when only considering attribute costs and thus it is not known whether or not they can reduce the average cost of a decision tree where D4 cannot. It would be interesting to see if an algorithm that took both kinds of costs

into consideration as well as performing a more straight-forward search of the decision tree space would be competitive with ICET in both speed and performance.

3.7 Summary

In this section I discussed several methods for constructing decision trees: traditional heuristic based methods such as ID3 and C4.5 as well as more modern approaches that use backtracking and different forms of search to optimize a decision tree in size or cost. Overall I looked at different methods of decision tree induction to give a sense of where the D4 algorithm I introduce fits in.

In the next chapter, I will introduce and explain my proposed method for solving the problem of minimizing the expected cost of decision trees.

Chapter 4

Proposed Method

In this chapter I introduce the D4 algorithm, a supervised machine learning algorithm. The input it takes is a set of labelled examples, such as the data in Table 2.1. Following the running example, the input to the algorithm would be a set of patients who have already been diagnosed as having a viral, bacterial or no infection based on some tests that were run on the patients. What D4 outputs is a decision tree, a grouping of patients with the same diagnosis and similar test results. The decision tree is used to diagnose future patients, using their tests results to find similar patients and assign a classification. Figure 4.1 gives a pictorial of the process.

The decision tree is computed layer by layer until the cost of the decision tree becomes too high and new combinations are tried at previous layers. The process begins at a root node containing all of the examples and the attributes available with which to split the examples with. Similar examples are sent down each branch to repeat the process until all of the examples have the same classification. If at any point the size, cost or other measure of the decision tree surpasses the running upper-bound(starting off with the decision tree found using a heuristic), the examples are regrouped up to a point and subsequently split down new branches using different attributes. This can be done randomly or exhaustively until all decision trees are considered. After the search is complete, a decision tree optimized according to some criteria will be returned.

4.1 D4 Algorithm

The D4 algorithm, in its simplest form, is a breadth-first decision tree induction treatment of the ID3 algorithm with added backtracking. The decision tree is grown layer by layer,

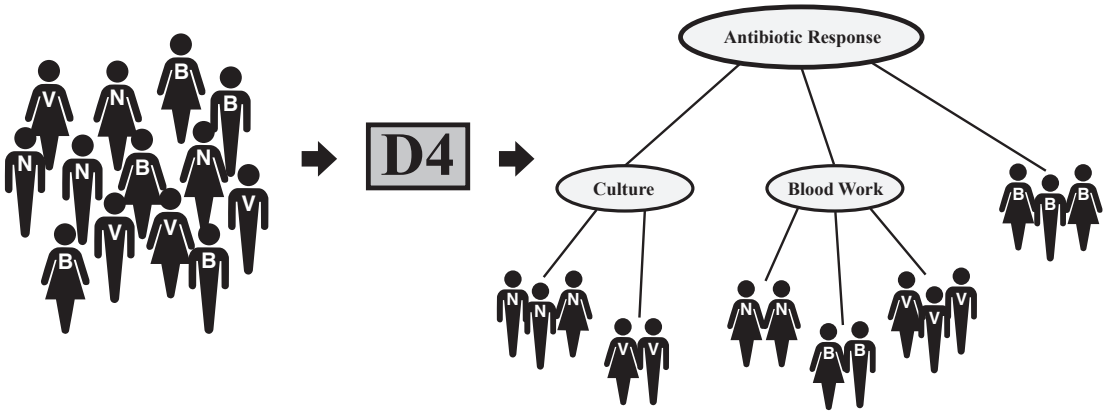


Figure 4.1: Pictorial of the D4 algorithm. The left-most group of people represent patients or examples used as input into the algorithm. After being passed through the algorithm the patients are grouped by their diagnosis and organized in the decision tree by the values of their tests.

choosing a combination of attributes to split the nodes at that layer and trying a new combination when some upper-bound is passed. The upper-bound can be the size, depth, expected cost, or any other measure of the decision tree. After the upper-bound is hit a given number of times, D4 backtracks to the parent layer of the tree where it tries a new combination. Choosing the combination of attributes can either be random or methodical. For the random attribute selection method, a attribute is chosen from the list of available attributes for each node in the layer given some probability distribution. For example, the three attributes with highest information gain can have an equally likely chance of being chosen while the remaining attributes are never chosen. Another way of choosing the attributes, introduced by Esmeir and Markovitch[11], is assigning probabilities to the attributes based on the information gain of that attribute, giving preference to attributes with higher information gain. The other approach to choosing the combination of attributes to use at a given layer is to step through the combinations of the top n attributes based on information gain. This allows an upper-bound to be set that maximizes the heuristic by choosing the single combination from the best attribute in every node. After that, the top $n+1$ attributes as determined by the heuristic can be iterated through, although this does not scale well and even the top two attributes cannot be completely iterated through. There are too many possible decision trees to search through given the combination of attributes. Recall, Goodrich[15] showed that constructing a decision tree with only two attributes to choose from per node is NP-complete.

The inspiration for this algorithm comes from Bessiere et al's[5] work optimizing the size of a decision tree using constraint programming. As their solution did not scale well on larger data sets, they had to use a search heuristic that turned their approach into a top down induction of decision trees with backtracking. The D4 algorithm improves upon their model by eliminating the overhead of the constraint library by inferring the structure of the decision tree while keeping the power of backtracking and by generalizing to include data that is not in binary form and weights associated with attributes. Thus, D4 can handle any number of values for each attribute and any number of classes for each example.

4.2 Input/Output

The D4 program currently takes in training data in ARFF (Attribute-Relation File Format) files. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the WEKA[17]. Although this file format allows data to have missing values and attributes with continuous values, the D4 program does not currently handle them and assumes all data sets are complete and contain only categorical attributes. The format of the file is as follows: It begins with the name of the relation at the top, followed by a line for each attribute and a line for the class. The attribute name is followed by the values it can take on. After the attributes and below the data tag is a list of examples with values separated by commas and ending with a classification. The costs associated with each attribute are stored in a separate file in the same order that the attributes appear in the ARFF.

```
@relation infection

@attribute stool {normal,abnormal}
@attribute culture {none,viral}
@attribute temperature {normal,high}
@attribute blood_work {unclear,bacterial,viral}
@attribute antibiotic_response {none,unclear,full}
@attribute class {none,bacterial,viral}

@data
normal,none,normal,unclear,unclear,none
normal,none,high,unclear,none,none
normal,none,normal,bacterial,unclear,bacterial
```

```
normal,none,normal,unclear,full,bacterial
normal,viral,normal,unclear,none,viral
abnormal,viral,high,viral,unclear,viral
...
```

Once a final decision tree is decided on, the D4 program stores that tree as a compilable C++ program. The C++ program takes in a CSV file and returns the accuracy of the decision tree on all testing examples in that file.

4.3 Algorithm

The D4 algorithm is split up into three sections (algorithm 4, algorithm 5, and algorithm 6): the main D4 algorithm that builds new trees after failed searches, the GenerateLayer algorithm that builds a single decision tree using a breadth-first backtracking search and the CostHeuristic algorithm that gives weights to each attribute.

The main body and starting point of the D4 algorithm is shown in algorithm 4. It begins with a set of training examples, a set of attributes, and a set of costs associated with each attribute. It first checks to see if all of the examples are already classified the same, in which case it returns a leaf node with that classification. If this is not the case, D4 calculates the cost heuristic value for every attribute given all of the available examples. Once those are sorted, it jumps into a loop that will continue searching until a set amount of time has passed or some other stopping criteria has been met.

The main while loops starts with a root node containing all of the examples and attributes available to D4 and places it in the open list. The open list contains all of the internal nodes at a given layer (non-leaf nodes). It then calls GenerateLayer, a recursive function for building a decision tree. If a decision tree is returned, the cost and size of the tree is saved as upper-bounds. These upper-bounds are used in successive calls to GenerateLayer to backtrack to a parent layer when the decision tree becomes to expensive or too large.

Algorithm 4 D4

```
Let E be a set of training examples
Let A be a set of attributes
Let C be a set of costs associated with each a ∈ A
Let N be the open list of internal nodes
Let UB be an upper bound on the size of the tree
Let UBC be an upper bound on the cost of the tree
function D4(E, A, C)
    if ∀e ∈ E, e is classified with Y then
        return single node tree with classification Y
    end if
    for all attribute a_i ∈ A do
        h_i ← CostHeuristic(a_i, c_i, E)
    end for
    Sort attributes in A by heuristic value h_i
    while current X < maximum Z do
        Create new node, n, containing E and A
        Add n to open list N
        if T ← GenerateLayer(T, N, UB, UBC) ≠ NIL then
            UB ← size of tree T
            UBC ← expected cost of tree T
        end if
    end while
    return T
end function
```

The GenerateLayer (algorithm 5) portion of the D4 algorithm is more complicated. It works on all of the nodes in an open list that represents the internal nodes of the decision tree at that layer. New combinations of attributes for nodes are tried the until no combinations of attributes is left or some maximum number of combinations is reached. As there are time constraints for running most algorithms, the maximum number of combinations tried in a layer should be set to allow enough time spent at a given layer while still allowing time to spent in other layers. For my experiments I started with trying one combination and then linearly increasing this every time a tree is not found. The algorithm can be stopped at any time and the current best solution will be returned.

Inside the while loop, a attribute is assigned to each node in the open list (in this case, randomly based on the heuristic). Once a attribute is chosen, the node is expanded and

a leaf or internal node is created for each branch. The internal nodes are fitted with the examples that have the corresponding values for their parent nodes' attribute, the inherited list of attributes (not including its parent's attribute) and the heuristic values for all of those attributes. These internal nodes are added to a new open list that will be the basis for the next layer.

After the next layer is constructed, the decision tree may surpass the cost or size upper-bound. If so, the new layer is scrapped and a new combination is tried at the current layer. Otherwise, if there are still internal nodes in the open list, GenerateLayer is called again until all of the examples are grouped together with the same classification. This section of the algorithm is stopped when the predetermined amount of time has passed or enough combinations have been tried. If the upper-bound is surpassed too many times at a given layer, it will give up and return to its parent layer where a new combination will be tried there. It is possible that the root node will run out of attributes to try and the D4 algorithm will give up at that point.

Algorithm 5 GenerateLayer

```
Let N be the open list of internal nodes
Let UB be an upper bound on the size of the tree
Let UBC be an upper bound on the cost of the tree
function GenerateLayer(T, N, UB, UBC)
    while current X' < maximum Z' do
        for all n ∈ N do
            Assign attribute a ∈ A in n randomly
            for all v_i ∈ V_f do
                if E_{v_i} = ∅ or A_{v_i} = ∅ or all e ∈ E_{v_i} all have the same classification then
                    Add leaf node to N' and set as child of n in T
                else
                    A' ← A \ a
                    for all attribute a_i in A' do
                        h_i ← CostHeuristic(a_i, c_i, E_{v_i})
                    end for
                    Sort a_i by heuristic value h_i
                    Create new node, n, containing E_{v_i} and A'
                    Add n to open list N' and set as child of n in T
                end if
            end for
        end for
        if Cost of T > UBC or Cost of T = UBC and size of T > UB then
            Update X', X
        else if N' = ∅ then
            return T
        else if T ← GenerateLayer(T, N, UB, UBC) ≠ NIL then
            return T
        end if
    end while
    return NIL
end function
```

Although any heuristic can be used with D4, the heuristic chosen (algorithm 6) is the heuristic used in Cost-Sensitive ID3[41] seen in equation 2.5. It finds the information gain for each attribute on a subset of examples, squares it and then divides it by the cost of the attribute. Although the costs are unweighted in the selection of the attributes, the

expected cost is used when calculating the cost of the tree and subsequent upper-bounds.

Algorithm 6 CostHeuristic

Let E be a subset of training examples
Let A be a attribute
Let c be a cost associated with $a \in A$
function CostHeuristic(a, c, E)
    $I \leftarrow$ InformationGain(E, a)
    return $\left(\frac{I^2}{c}\right)$
end function

4.4 Summary

In this chapter I introduced the D4 algorithm. On the surface it is a straightforward extension of the ID3 algorithm to include backtracking. It allows the use of any heuristic, attribute selection method, stopping criteria, and upper-bounds for backtracking. It is useful in finding decision trees beyond those created using simple heuristics and allows for attributes to have costs associated with them.

In the next chapter I evaluate effectiveness of the D4 algorithm and I explain the effect of transforming data into binary form.

Chapter 5

Experimental Evaluation

This chapter looks to examine the effectiveness of the D4 algorithm. Before going into the experiments, I address the effect that converting attributes into binary form has on decision tree size. It appears that smaller decision trees can be generated solely by converting the training data into binary form, reducing the number of leaves in the tree. Binary decision trees can contain no empty leaves and thus have an advantage. All of the data used in the experiments are either randomly generated or from the UCI Machine Learning Database[1].

The first experiment compares the decision trees generated by the D4 algorithm and a modified version of C4.5. It is shown that decision trees of lower expected cost generally cannot be found by backtracking in the decision tree with the D4 algorithm. The experimental results show that the heuristic used in both C4.5 and D4 is powerful and the number of decision trees in the solution space is intractably large.

The second experiment revisits the results of Bessiere et al.[5]. They used constraint programming to search through decision trees constructed from the top three attributes maximizing information gain to find decision trees of smaller size. They found that by doing this search, they could find substantially smaller decision trees. This thesis attributes some of those results to the conversion of non-binary attributes to many binary attributes.

The last experiment looks at trends across data of varying size measures. Increasing the number of examples, attributes and values per attribute changes how large the effect the conversion of attributes to binary has on the size of the decision tree generated.

[1] UCI Machine Learning Repository at http://archive.ics.uci.edu/ml/index.html

5.1 Data

The data used in this thesis come from the UCI Machine Learning Repository. Thirteen datasets were chosen with varying numbers of examples, attributes and classes. Table 5.1 shows the size of the dataset as well as how many features can be considered when doing a complete search of the decision tree space in less than a day.

Table 5.1: Datasets from the UCI Machine Learning Repository used to test D4 algorithm. The datasets are sorted by second column (the number of examples in the dataset). The third and fourth columns are the number of attributes in the dataset and the average number of values for those attributes. The second to last column contains the number of classes, while the last column is the number of top attributes that can be searched completely in under an hour. The 'Lenses' and 'Hayes Roth' datasets can look at every possible decision tree with all combinations of all the attributes at all depths considered.

Name	Examples	Attributes	Avg Values	Classes	Complete
Lenses	24	4	2.3	3	*4
Promoters	106	57	4.0	2	3
Hayes Roth	132	4	3.8	3	*4
Lymphography	148	18	3.3	4	3
SPECT	187	22	2.0	2	1
Balance Scale	625	4	5.0	3	1
TicTacToe	958	9	3.0	2	1
Car	1728	6	3.5	4	1
Splice	3190	60	8.0	3	1
Mushroom	8124	22	5.7	2	8
Nursery	12960	8	4.0	5	1
Chess	28056	6	7.3	18	1
Connect 4	67557	42	3.0	3	1

The healthcare data with associated costs in Table 5.2 also comes from the UCI Machine Learning Repository but is not publicly listed. There are eight data sets related to various health issues with four of the sets sharing attributes and costs as they are all involving the detection the presence of heart disease in a patient.

Table 5.2: Cost datasets from the UCI Machine Learning Repository used to test D4 algorithm. The columns in the table represent the number of examples, attributes, average number of values, and classes for each dataset.

Name	Examples	Attributes	Avg Values	Classes
Cleveland Heart	303	13	3.3	5
Diabetes	768	8	4.0	2
Hepatitis	155	19	3.2	2
Hungarian Heart	294	13	3.3	2
Liver	345	6	4.0	2
Switzerland Heart	123	12	3.3	5
Thyroid	7200	21	2.6	3
VA Heart	200	13	3.3	5

5.2 Decision Trees with Cost

This section contains the results of an experiment conducted to find whether a backtracking search of the decision tree solution space can result in decision trees of lower expected cost than those generated with the same heuristic built into C4.5. The original code for Quinlan's C4.5 was used with the minor change to the heuristic to maximize information gain squared over cost as opposed to merely maximizing information alone to separate the benefits of the D4 algorithm from the underlying heuristic employed. All pruning and requirements to continue splitting as opposed to creating a leaf have also been removed. Table 5.3 shows the average size, cost and classification accuracy over the ten trees generated using ten-fold cross-validation. The experiment compared C4.5 (with no pruning: c4.5 -m 1 -u -g -f $< filename >$) written in C, to D4 (randomized selection from the top three attributes minimizing the heuristic) written in C++ with a running time of 30 minutes. After that time, the best tree found by D4 so far is returned and used in the average.

These results, though expected, are disappointing. Only one dataset resulted in finding a tree of lower cost than C4.5 (about 75% of the cost). Trees with lower expected cost may exist, but finding a decision tree cheaper than the one found using the heuristic by searching the space of decision trees generated using the top three attributes as decided by the heuristic is both too slow and there is no guarantee of better solutions.

Table 5.3: Expected costs of decision trees grown on the datasets from the UCI Machine Learning Repository. The left three columns contain information about the decision tree generated by C4.5 and the three columns on the right generated by D4. The three columns for each algorithm contain the average expected costs, sizes and generalization accuracies of the decision trees generated. The dark gray highlighted cell represents the only dataset where the D4 algorithm was able to find a decision tree of lower expected cost than C4.5. The light gray highlighted cells represent an insignficant cost reduction by D4.

	C4.5			D4		
Data Set	Cost	Size	Accuracy	Cost	Size	Accuracy
Cleveland Heart	76.2	390.3	48.5%	76.2	390.3	48.5%
Diabetes	8.4	805.0	65.0%	8.4	805	65.0%
Hepatitis	4.7	65.4	74.2%	3.5	66.2	77.5%
Hungarian Heart	93.4	333.6	72.5%	93.3	344.2	71.8%
Liver	24.5	342.2	60.0%	24.4	345.0	60.6%
Switzerland Heart	157.4	252.6	25.4%	157.4	252.6	25.4%
Thyroid	31.3	3667.0	91.9%	31.2	3712.2	92.0%
VA Heart	115.7	458.2	30.5%	115.7	458.2	30.5%

5.3 The Benefit of Binary

The D4 algorithm is an extension to the constraint program by Bessiere et al.[5] that allows it to handle non-binary attributes with costs associated with them. Why is it then that they can find significantly smaller decision trees whereas most of the time D4 cannot find decision trees with lower expected cost? Even the first decision tree they find is always smaller than the decision tree found by C4.5. This is puzzling as they use the same heuristic as C4.5 and elect to choose from the top three attributes that maximize information gain instead of the single best attribute that maximizes information gain. Table 5.4 shows a comparison between WEKA's J48, the constraint program (CP), and my algorithm (D4). In this experiment, costs were not considered and the D4 algorithm was able to reduce the size of the decision tree on all datasets. Although when binary data was used with J48, the decision trees were even smaller in size (converting data to binary before passing it to J48 can be emulated by using an option that causes J48 to treat categorical data as numerical). The constraint program was able to further reduce the decision tree size, although not by half like Bessiere et al. found. When the binary decision trees created by J48 were pruned, they were often much smaller than the constraint program. Table 5.5 shows that the accuracy of the decision trees remained similar, except in the case where the accuracy of the constraint program on unseen data was significantly lower. It should

be noted that the accuracy on training data would out perform that of the pruned decision trees as it aims to be consistent with the training data.

The J48 columns all represent WEKA's implementation of C4.5 running with different settings. The normal J48 represents the algorithm with pruning turned off, no minimum number of examples per leaf, and normal splits (Options: J48 -v -O -J -U -M 0). The 'B' label represents the algorithm with pruning turned off, no minimum number examples per leaf, and binary splits (Options: J48 -v -O -J -U -M 0 -B). The 'P' label represents the algorithm with its default settings of pruning turned on, a minimum of two examples per leaf, and a pruning confidence of 0.25 (Options: J48 -v -C 0.25 -M 2). The 'BP' label represents the algorithm with its default settings of pruning turned on, a minimum of two examples per leaf, a pruning confidence of 0.25, and binary splits (Options: J48 -v -C 0.25 -M 2).

Table 5.4: Comparison between WEKA's J48, the D4 algorithm and Bessiere et al.'s constraint programming (CP) method on categorical data. Both D4 and CP were allowed to run for five minutes after finding it's last solution, while J48 had no timelimit. Each cell represents the average decision tree size from each of the 10-fold cross-validation runs.

Dataset	J48	D4	J48(B)	CP	J48(P)	J48(BP)
Lenses	14.0	12.7	12.4	8.8	6.4	6.6
Promoters	41.4	33.0	24.4	18.6	23.0	14.4
Lymphography	88.2	64.5	51.2	32.8	27.5	26.4
Splice	1144.2	1093.0	282.4	195.0	358.6	133.2
Mushroom	30.0	27.0	25.0	15.0	30.0	16.8

Table 5.5: Comparison between WEKA's J48, the D4 algorithm and Bessiere et al.'s constraint programming (CP) method on categorical data. Each cell represents the average decision tree classification accuaracy from each of the 10-fold cross-validation runs.

Dataset	J48	D4	J48(B)	CP	J48(P)	J48(BP)
Lenses	80.00	70.37	75.00	30.00	83.33	78.33
Promoters	75.55	83.64	76.36	51.82	76.55	80.09
Lymphography	75.62	78.90	80.19	67.05	77.00	79.67
Splice	91.10	90.82	92.10	70.53	93.89	93.98
Mushroom	100.00	100.00	100.00	69.89	100.00	99.98

Figure 5.1 and figure 5.2 show table 5.4 and table 5.5's data in graphs. It is much clearer to see the reduction in decision tree size by searching the decision tree space, converting the data to binary, and by pruning decision trees. The effect of each difference correlates with the number of training examples, number of attributes for each example, and number of values for each attribute, but the effect cannot be predicted as seen by the mushroom dataset having relatively similar decision tree sizes.

Figure 5.1: Bar graph showing the average size of the decision trees generated by each algorithm on each dataset. The Y-axis marks the average size of the ten decision trees generated by each algorithm, while the X-axis contains the datasets. Each dataset contains six bars representing the six algorithms. J48 with all pruning turned off generated the largest decision trees while J48 using binary data with pruning turned on generated the smallest decision trees. The D4 algorithm searches through many possible trees to find a smaller decision tree using multi-variate data, while the CP algorithm does the same thing but using binary data.

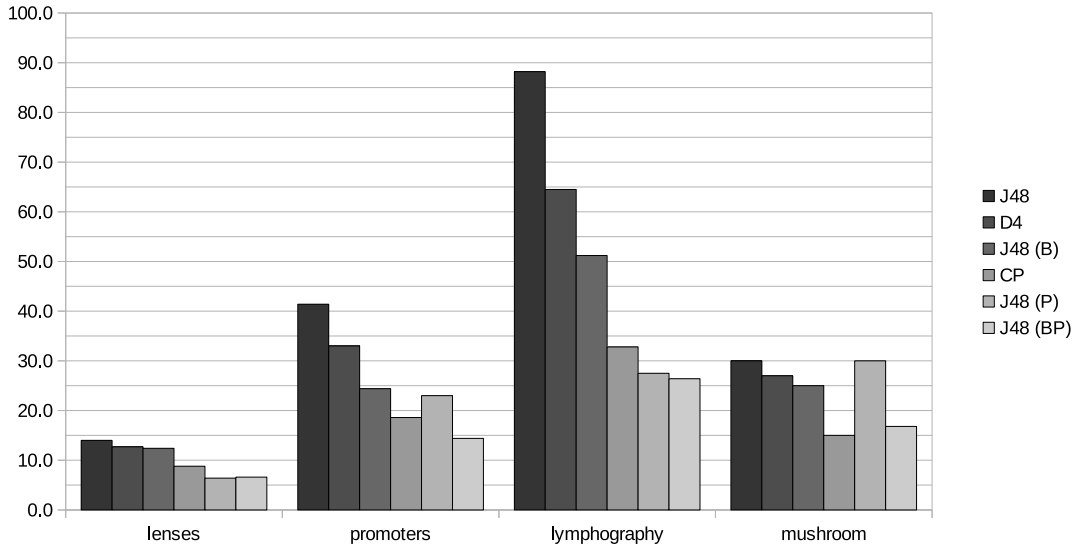
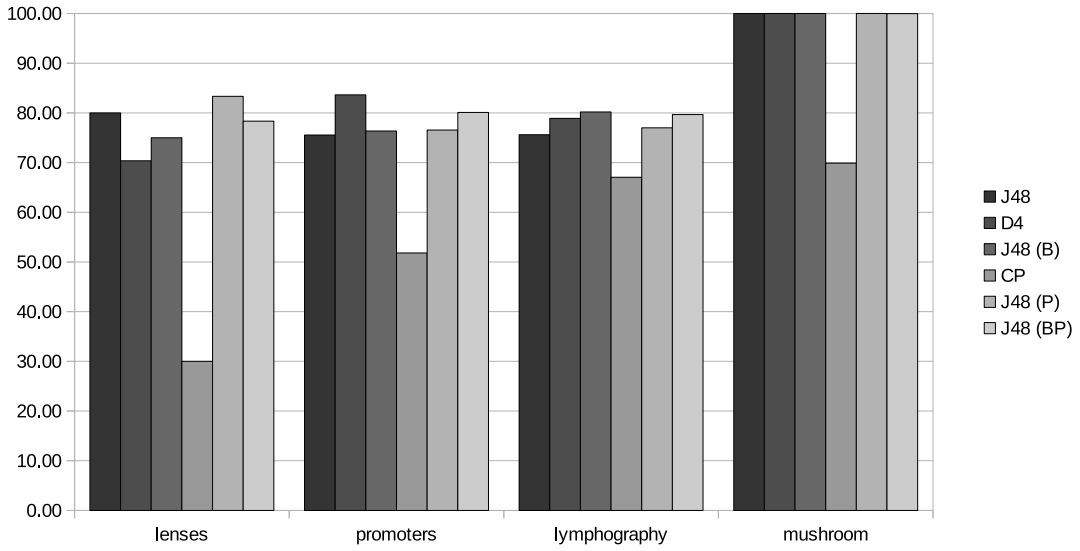


Table 5.6 shows why Bessiere et al.[5] were able to reduce the size of decision trees by half in their paper and I was not able to find such an improvement running their algorithm on different data. I chose to directly use their data as they used a modified version of WEKA's J48 and also did not specify the settings used in their constraint program. The columns 'WEKA', 'CP First', and 'CP Best' are taken directly from Bessiere et al.'s paper

Figure 5.2: Bar graph showing the average accuracy of the decision trees on test data generated by each algorithm on each dataset. The Y-axis marks the average accuracy of the ten decision trees generated by each algorithm, while the X-axis contains the datasets. Each dataset contains six bars representing the six algorithms. The CP algorithm generated decision trees with considerably lower accuracies.



and the last two columns (Non-Binary and Binary) are WEKA's J48, the open source implementation of C4.5, on both non-binary and binary data. Table 5.6 has the datasets from their paper with at least some categorical attributes.

Both of the columns 'WEKA' and 'Non-Binary' in Table 5.6 should have similar values as they are the unpruned decision trees built from the non-binary data. The first decision tree found by the constraint program should be the same size or larger than the decision tree found by WEKA as the constraint program does not optimize the information gain heuristic. The best tree found by the constraint program can be anywhere from slightly larger to slightly smaller depending on whether the heuristic was ever maximized or whether a smaller tree is found. The additional two columns 'Non-binary' and 'Binary' show that they also did not reduce the decision tree size as much as they thought as a lot of the reduction comes from the effect transforming the data to binary has.

Table 5.6: Comparison between WEKA's J48 and constraint programming (CP) method on categorical data. Average decision tree size (number of nodes) is reported for each algorithm. The '% of Data' column represents the percentage of data used in the training set. The 'WEKA' column contains the results of J48 run on non-binary data from [5]. The 'CP First' and 'CP Best' columns contain the results of the CP method from the same paper (both the first and best decision trees found). The last two columns are original and contain WEKA's J48 algorithm's result when run on both binary and non-binary data.

Dataset	% of Data	WEKA	CP First	CP Best	Non-binary	Binary
Car	5	30.1	24.8	18.5	29.3	23.0
	10	46.7	40.2	30.1	46.7	33.8
	20	71.0	59.8	47.7	72.8	55.2
	30	87.7	74.2	60.1	91.1	64.2
	50	114.5	93.3	75.6	117.0	81.5
	70	139.2	105.5	86.3	136.1	88.6
	90	161.7	115.2	92.0	156.6	92.6
Income	1	185.9	85.1	76.2	150.5	97.9
	1.5	265.2	123.6	112.9	208.8	150.3
	5	791.0	390.7	364.8	639.1	379.3
Chess	1	126.8	81.5	66.6	127.6	93.2
	1.5	172.4	119.6	98.9	176.6	139.2
	5	434.5	317.2	274.7	449.8	294.8
	10	735.3	525.5	458.8	717.7	601.8
Average		240.1	154.0	133.1	222.8	156.8

So in summary, after analysis of the experiments in [5], the general method of iterating through a subset of the decision tree solution space (or only considering the three attributes with the best information gain) is not as effective of an approach to optimizing decision trees in size as once thought. It is also strange that the D4 algorithm was able to see a reduction in decision tree size but not in expected costs of decision trees.

5.4 Effect of Binary Attributes on a Variety of Data

Although Bessiere et al.'s[5] method for minimizing decision tree size is effective, the reason it appears significantly more effective than other methods lies in the conversion of their data to include only binary attributes. The act of doing so combines nominal and numeric attributes, removes resolution from numeric attributes and greatly increases the number

of attributes to choose from. The resulting trees from the converted data end up much smaller than the trees created using the original data as seen in Figure 5.3.

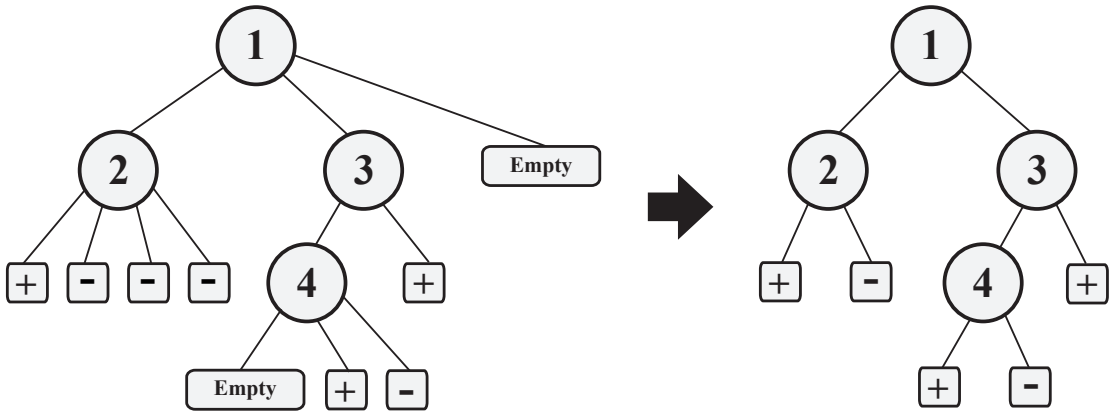


Figure 5.3: Effect of converting attributes to binary. The left decision tree has multi-valued attributes while the right decision tree has binary attributes. Notice the number of tests stays constant despite the decrease in size.

There are two reasons that binary trees end up smaller: (*i*) if only one leaf is classified positive while the rest are negative, a binary tree will require only one test and (*ii*) the binary trees contain no empty branches. Every split worth doing will have examples in a 'true' branch and a 'false' branch. In contrast, the original data has attributes with multiple values and a split can result in no examples of a certain value. In both of these cases though, the cost of the tests used are fully realized in both forms of decision trees; they only differ in the number of leaf nodes. Finding smaller decision trees with binary data is in agreement with past research as seen by what's stated in the Binary Tree Hypothesis:

> **Binary Tree Hypothesis**: For a top-down, non-backtracking, decision tree generation algorithm, if the algorithm applies a proper attribute selection measure, then selecting a single attribute-value pair at each node and thus constructing a binary tree, rather than selecting an attribute and branching on all its values simultaneously, is likely to lead to a decision tree with fewer leaves.[13, p.107]

The hypothesis is restricted to decision trees with fewer leaves, as the data still require a similar amount of tests to split. Another approach is combining the set of values into

two sets, as opposed to creating a new attribute, that represents an example having that value or not, for every value in the old attribute. Quinlan[35] mentions Kononenko[20] and Shepherd[39] both found that taking this approach leads to smaller decision trees with improved classification performance while reducing human comprehension of the decision tree.

It is known that converting attributes to a binary representation has an effect on the structure of decision trees. Although Bessiere et al.'s [5] work benefited from the use of binary data, they merely did the conversion with no mention of analysis of the benefits seen from using binary data. Table 5.7 and 5.8 shows the relative effect that converting attributes to binary has on decision tree size when the number of attributes, varying number of values per attribute, average attribute value, and number of examples. The experiment is run using all of the training data to create a single decision tree. The D4 algorithm was used to create the decision trees on both the binary and non-binary data with no pruning and a minimum number of leaves per node (this was done in a single pass with no backtracking). The binary columns for D4 have the same decision tree size and accuracy as requiring a minimum number of examples in a leaf makes no difference. For non-binary data, requiring at least one example to make a classification, removes the empty leaves from the decision tree. WEKA's implementation of C4.5 is used on it's default setting in the last two columns to show the effects of pruning on decision tree size and accuracy on training data.

Table 5.7: Comparison of decision tree size between binary and non-binary features on an assortment of data. The first column on the left gives a description of the data being used. The code 4f 4v 10e means the data has four features, four values per features and ten examples, while 2-5v specifies a range of values. The table is broken up into three sections: D4 run with no minimum number of examples required for a leaf and no pruning, D4 run with a minimum of one example required for a leaf and no pruning, and the default pruning settings of WEKA's J48. In each of those three sections there are two columns representing the normal data as well as the data converted to have binary attributes using the method used by Bessiere et al.[5]. The three highlighted columns are to show that binary decision trees are normally smaller than their non-binary counterparts. When there is a minimum number of examples required in a leaf, binary decision trees are no longer smaller, as there are no longer empty leaves in the non-binary tree. This removes empty leaves and shrinks the size of the decision trees on data with non-binary attributes. In this experiment only the best attribute was examined as determined by the heursitc, D4 preformed no searching or backtracking.

	D4 unpruned		D4 minimum		WEKA pruned	
	Non-binary	Binary	Non-binary	Binary	Non-binary	Binary
4f4v10	13	9	5	9	1	7
4f4v100	117	95	53	95	13	17
4f4v1000	341	457	337	457	33	73
4f4v10000	341	509	341	509	105	117
4f2-5v10	10	9	10	9	1	5
4f2-5v100	83	95	73	95	10	13
4f2-5v1000	184	221	184	221	35	59
4f2-5v10000	194	237	194	237	61	91
4f8v10	25	11	1	11	1	1
4f8v100	217	93	33	93	1	33
4f8v1000	1793	931	305	931	25	163
4f8v10000	4681	6205	2737	6205	209	1049
4f2-10v10	19	7	9	7	1	5
4f2-10v100	181	105	131	105	1	11
4f2-10v1000	681	767	629	767	65	119
4f2-10v10000	1019	1237	1019	1237	225	325

Table 5.8: Comparison of decision tree size between binary and non-binary features on an assortment of data. Continuation of Table 5.7 with ten features as opposed to four.

	D4 unpruned		D4 minimum		WEKA pruned	
	Non-binary	Binary	Non-binary	Binary	Non-binary	Binary
10f4v10	9	5	5	5	5	3
10f4v100	101	63	69	63	21	41
10f4v1000	1041	663	629	663	141	303
10f4v10000	11137	7927	6277	7927	1761	2939
10f2-6v10	9	7	9	7	1	3
10f2-6v100	94	61	88	61	23	41
10f2-6v1000	944	717	897	717	226	321
10f2-6v10000	10484	8263	9472	8263	1361	2771
10f8v10	9	5	9	5	1	5
10f8v100	153	55	49	55	1	39
10f8v1000	1433	539	505	539	113	351
10f8v10000	15713	6029	4353	6029	1265	3295
10f2-11v10	9	7	9	7	5	3
10f2-11v100	105	47	89	47	22	33
10f2-11v1000	1139	629	1038	629	153	347
10f2-11v10000	11053	6483	9652	6483	1253	3187

Table 5.9 is from the same experiment but shows the accuracies of the decision trees. The accuracies of the decision trees are tested on the original training data. Interestingly, the accuracies of the unpruned decision trees grown on binary data and non-binary date are the same. Although, the accuracy of decision trees requiring a minimum number of examples per leaf drops as leaves no longer contain examples of the same classification. This minimum does not effect decision trees grown on binary data because an empty leaf would imply all of the examples going down a single branch resulting in no information gain.

Table 5.10 summarizes some trends. The first column states which types of datasets are averaged, for example 10e represents all datasets with 10 examples. The second column is the ratio between non-binary attributes and binary attributes. The third column is the ratio between non-binary attributes with an enforced minimum of one example in each leaf and binary attributes. The advantage of binary attributes on the size of decision trees disappears when a minimum number of examples per leaf node is required. The size of the

decision tree for non-binary attributes even becomes smaller than the binary trees as the number of examples increases (this is expected as the direct conversion of a decision tree to binary increases its size when attributes have four or more values). As the number of examples increases, the ratio between unpruned non-binary trees and binary trees decreases as there are more available data to fill every leaf node. As the number of features increases, the ratio between the two decision trees increases. Increasing the values per attribute also increases the ratio as the potential for empty leaves increases. Attributes with two values would have a ratio of one to one, as the attribute is a binary attribute.

Table 5.10: Trends found when comparing decision tree sizes of datasets with similar dimensions. The first column is the group that is being averaged, whether it be datasets with a hundred examples (100e), four features (4f), or eight values per feature (8v). The second column is the ratio between the size of decision trees found using the original data and the decision trees found using the data converted to have binary attributes. The third column is the ratio between the size of decision trees found using the original data with a requirement that every leaf have at least one example and the decision trees found using the data converted to have binary attributes.

	Non-binary/Binary	Minimum/Binary
10e	1.71	1.05
100e	1.79	1.03
1000e	1.47	0.94
10000e	1.26	0.86
4f	1.32	0.72
10f	1.79	1.23
small range	1.13	1.08
4v	1.31	0.79
large range	1.65	1.31
8v	2.14	0.70

In conclusion, there is clearly an effect on the size of decision trees caused by the conversion of the data to have binary attributes. This effect is not equal across all data though and varies according to different measures of the data. The larger the data set, the size of both decisions trees on the same data becomes closer. Having more features or values per feature though, increases the size differential between decision trees created with non-binary attributes and decision trees created with binary attributes.

5.5 Summary

In this Chapter I evaluated the D4 algorithm as well as re-evaluated the results found by Bessiere et al[5]. I found that in most cases the expected cost of decision trees could not be reduced beyond the cost of decision trees found by a simple heuristic. I also found that converting attributes into binary form reduces the size of decision trees greatly, although not necessarily the number of tests. The effects of binary attributes was also evaluated across different measures of data size.

Table 5.9: Comparison of decision tree accuracy between binary and non-binary features on an assortment of data. This table contains the training accuracies of the decision trees in Tables 5.7 and 5.8.

	D4 unpruned		D4 minimum		WEKA pruned	
	Non-binary	Binary	Non-binary	Binary	Non-binary	Binary
4f4v10	100%	100%	80%	100%	60%	90%
4f4v100	89%	89%	76%	89%	71%	74%
4f4v1000	69%	69%	69%	69%	58%	64%
4f4v10000	56%	56%	56%	56%	54%	55%
4f2-5v10	100%	100%	100%	100%	60%	90%
4f2-5v100	84%	84%	83%	84%	64%	68%
4f2-5v1000	63%	63%	63%	63%	57%	60%
4f2-5v10000	54%	54%	54%	54%	54%	54%
4f8v10	100%	100%	40%	100%	60%	60%
4f8v100	100%	100%	69%	100%	56%	81%
4f8v1000	94%	93%	72%	93%	59%	73%
4f8v10000	74%	74%	68%	74%	56%	66%
4f2-10v10	100%	100%	90%	100%	60%	90%
4f2-10v100	97%	97%	92%	97%	56%	69%
4f2-10v1000	78%	78%	78%	78%	60%	67%
4f2-10v10000	60%	60%	60%	60%	57%	58%
10f4v10	100%	100%	90%	100%	90%	90%
10f4v100	100%	100%	91%	100%	75%	92%
10f4v1000	100%	100%	90%	100%	72%	87%
10f4v10000	100%	100%	88%	100%	73%	83%
10f2-6v10	100%	100%	100%	100%	70%	90%
10f2-6v100	100%	100%	98%	100%	78%	93%
10f2-6v1000	100%	100%	99%	100%	77%	86%
10f2-6v10000	99%	99%	97%	99%	70%	82%
10f8v10	100%	100%	100%	100%	60%	100%
10f8v100	100%	100%	83%	100%	58%	95%
10f8v1000	100%	100%	85%	100%	66%	92%
10f8v10000	100%	100%	82%	100%	67%	89%
10f2-11v10	100%	100%	100%	100%	90%	80%
10f2-11v100	100%	100%	95%	100%	82%	94%
10f2-11v1000	100%	100%	97%	100%	70%	88%
10f2-11v10000	100%	100%	96%	100%	67%	87%

Chapter 6

Conclusion

This chapter summarizes the thesis and provides direction for future work. The main contribution I have made is the D4 algorithm, which I have used to show that past research on optimization of decision trees is not as effective as claimed. It is a generalization of past work and opens up the door for many extensions for other research on improving decision tree induction. I will also discuss my analysis of converting training data to binary and how it effects the size of decision trees grown.

6.1 Conclusions

To summarize the work of this thesis succinctly, the number of decision trees that provide a classification of the training data is too large and the heuristic for finding cost-efficient decision trees is too efficient to improve the expected-cost of the decision tree. A randomized search of the decision tree space using the heuristic as a guide cannot always find a decision tree that outperforms the original tree found by the heuristic alone. Although I was not able to subvert the worst case scenario by using intelligent search methods on real world data, I have shown that past research in the area is not as promising as it may seem. The results of Bessiere et al.[5] in finding smaller decision trees through biased random search of the decision tree space, is skewed by the transformation of training data to binary form (i.e. changing each attribute into a set of multiple binary attributes corresponding to each value of the original attribute). Past research along with my findings agree that decision trees made using data with binary attributes are both smaller and more accurate than their the decision trees made with the same data in its original non-binary form. Although, the difference between binary and non-binary decision trees varies depending on the number

of examples, attributes and values per attribute of the original data set when making a direct comparison. Future research in decision tree construction should consider closely the effects of data representation and look closely at the details of experiments to explain results more precisely.

6.2 Future Work

The D4 algorithm only handles one aspect of costs associated with decision tree learning. Along with costs associated with using an attribute to split the data, there are costs incurred when a new example is misclassified. For example, what are the costs of misdiagnosing a bacterial infection as viral, the costs of misdiagnosing a viral infection as bacterial? These costs are difficult to assign and would require extensive research and data, but play an important role in the balance between decision tree cost and its accuracy.

The main downfall of the D4 algorithm, is the random search through the decision tree space. Currently, the top three attributes, as decided by the heuristic, are chosen between with equal weighting. Another approach is to systematically search through a majority of the attributes at the top of the tree, but after a certain depth, only consider the best attribute. This would greatly reduce the search space, while not relying too heavily on the heuristic until deeper in the decision tree, where the heuristic preforms better.

It would also be interesting to find out if searching the decision tree space through backtracking can allow for additional information that a heuristic cannot capture, to be used in the selection of attributes to split the examples with. An example of information that cannot be captured by a heuristic could the relationship between attributes that appear in the same branch. As well, once this method is extended to perform more reliably, it would be beneficial to handle missing values and include the option to prune the decision tree found.

References

[1] Micah Adler and Brent Heeringa. Approximating optimal binary decision trees. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 1–9. Springer, 2008.

[2] Abdulaziz Alkhalid, Igor Chikalov, and Mikhail Moshkov. Comparison of greedy algorithms for decision tree optimization. In *Rough Sets and Intelligent Systems—Professor Zdzisław Pawlak in Memoriam*, pages 21–40. Springer, 2013.

[3] Kevin Bache and Moshe Lichman. UCI machine learning repository, 2013.

[4] Nicolas Beldiceanu, Pierre Flener, and Xavier Lorca. The tree constraint. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 64–78. Springer, 2005.

[5] Christian Bessiere, Emmanuel Hebrard, and Barry OSullivan. Minimising decision tree size as combinatorial optimisation. In *Principles and Practice of Constraint Programming-CP 2009*, pages 173–187. Springer, 2009.

[6] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Occam's razor. *Information processing letters*, 24(6):377–380, 1987.

[7] Jeffrey P Bradford, Clayton Kunz, Ron Kohavi, Cliff Brunk, and Carla E Brodley. Pruning decision trees with misclassification costs. In *Machine Learning: ECML-98*, pages 131–136. Springer, 1998.

[8] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and Regression Trees*. CRC press, 1984.

[9] Ferdinando Cicalese, Tobias Jacobs, Eduardo Laber, and Marco Molinaro. On the complexity of searching in trees and partially ordered structures. *Theoretical Computer Science*, 412(50):6879–6896, 2011.

[10] Pedro Domingos. The role of occam's razor in knowledge discovery. *Data mining and knowledge discovery*, 3(4):409–425, 1999.

[11] Saher Esmeir and Shaul Markovitch. Lookahead-based algorithms for anytime induction of decision trees. In *Proceedings of the twenty-first international conference on Machine learning*, page 33. ACM, 2004.

[12] Floriana Esposito, Donato Malerba, Giovanni Semeraro, and J Kay. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, 1997.

[13] Usama M Fayyad and Keki B Irani. The attribute selection problem in decision tree generation. In *AAAI*, pages 104–110, 1992.

[14] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Special invited paper. additive logistic regression: A statistical view of boosting. *Annals of Statistics*, pages 337–374, 2000.

[15] Michael T Goodrich, Vincent Mirelli, Mark Orletsky, and Jeffery Salowe. Decision tree conctruction in fixed dimensions: Being global is hard but local greed is good. Technical report, Technical Report TR-95-1, Johns Hopkins Univ., Department of Computer Science, Baltimore, MD 21218, 1995.

[16] Carlos RP Hartmann, Pramod K Varshney, Kishan G Mehrotra, and C Gerberich. Application of information theory to the construction of efficient decision trees. *IEEE Transactions on Information Theory*, 28(4):565–577, 1982.

[17] Geoffrey Holmes, Andrew Donkin, and Ian H Witten. Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pages 357–361. IEEE, 1994.

[18] Laurent Hyafil and Ronald L Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[19] Narendra Jussien, G Rochart, and X Lorca. The choco constraint programming solver. In *CPAIOR08 workshop on Open-Source Software for Integer and Contraint Programming (OSSICP08)*, 2008.

[20] Igor Kononenko, Ivan Bratko, and E. Roskar. Experiments in automatic learning of medical diagnostic rules. 1984.

[21] Eduardo S Laber and Loana Tito Nogueira. On the hardness of the minimum height decision tree problem. *Discrete Applied Mathematics*, 144(1):209–212, 2004.

[22] Charles X Ling, Victor S Sheng, and Qiang Yang. Test strategies for cost-sensitive decision trees. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1055–1067, 2006.

[23] Charles X Ling, Qiang Yang, Jianning Wang, and Shichao Zhang. Decision trees with minimal costs. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 483–486. ACM, 2004.

[24] Susan Lomax and Sunil Vadera. A survey of cost-sensitive decision tree induction algorithms. *ACM Computing Surveys*, 45(2):16, 2013.

[25] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine learning*, 4(2):227–243, 1989.

[26] Bernard M. E. Moret. Decision trees and diagrams. *ACM Computing Surveys*, 14(4):593–623, December 1982.

[27] Patrick M Murphy and Michael J Pazzani. Exploring the decision forest: An empirical investigation of occam's razor in decision tree induction. *Journal of Artificial Intelligence Research*, pages 257–275, 1994.

[28] Sreerama K Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2(4):345–389, 1998.

[29] Scott L Needham and David L Dowe. Message length as an effective ockhams razor in decision tree induction. In *Proc. 8th International Workshop on Artificial Intelligence and Statistics (AI+ STATS 2001)*, pages 253–260, 2001.

[30] Steven W Norton. Generating better decision trees. In *IJCAI*, volume 89, pages 800–805, 1989.

[31] Marlon Núñez. The use of background knowledge in decision tree induction. *Machine Learning*, 6(3):231–250, 1991.

[32] Kemal Polat and Salih Güneş. A novel hybrid intelligent method based on C4.5 decision tree classifier and one-against-all approach for multi-class classification problems. *Expert Systems with Applications*, 36(2):1587–1592, 2009.

[33] Patrick Prosser and Chris Unsworth. Rooted tree and spanning tree constraints. In *17th ECAI Workshop on Modelling and Solving Problems with Constraints*, 2006.

[34] John Punch. *Johannes Ponciuss commentary on John Duns Scotus's Opus Oxoniense*, volume 15. 1639.

[35] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[36] J. Ross Quinlan. *C4.5: Programs for Machine Learning*, volume 1. Morgan Kaufmann, 1993.

[37] Guillaume Richaud, Xavier Lorca, and Narendra Jussien. A portable and efficient implementation of global constraints: The tree constraint case. In *Proceedings of CICLOPS*, pages 44–56, 2007.

[38] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.

[39] Barry Shepherd. An appraisal of a decision tree approach to image classification. In *IJCAI*, volume 83, pages 473–475, 1983.

[40] Haijian Shi. *Best-first Decision Tree Learning*. PhD thesis, The University of Waikato, 2007.

[41] Ming Tan. Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning*, 13(1):7–33, 1993.

[42] Peter Turney. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research (JAIR)*, 2:369–409, 1995.

[43] Peter Turney. Types of cost in inductive concept learning. *In Proceedings of the ICML'2000 Workshop on Cost-Sensitive Learning*, pages 15–21, 2000.

[44] Geoff Webb. Further experimental evidence against the utility of occam's razor. *Journal of Artificial Intelligence Research*, 4:397–417, 1996.

[45] David H Wolpert and William G Macready. Coevolutionary free lunches. *IEEE Transactions on Evolutionary Computation*, 9(6):721–735, 2005.

[46] Min Xu, Jian-Li Wang, and Tao Chen. Improved decision tree algorithm: Id3+. In *Intelligent Computing in Signal Processing and Pattern Recognition*, pages 141–149. Springer, 2006.

[47] Shichao Zhang, Zhenxing Qin, Charles X Ling, and Shengli Sheng. "Missing is useful": Missing values in cost-sensitive decision trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(12):1689–1693, 2005.

APPENDICES

Appendix A

Constraint Program

Described in this appendix is a reworking of the constraint model for creating decision trees by Bessiere et al[5]. Although not as efficient, it is a good starting point for creating a model for a standard constraint programming tool such as Choco. The constraints used in this model to enforce a tree structure are a brute force approach to the problem and can be greatly improved using the work in [37], [33], and [4]. This model does not consider the search aspect of the problem and is only a brute force base for a more complex model.

A.1 Variables

Let n be the number of internal nodes in the decision tree, m be the number of examples being used to build the tree, and k be the number of features available to test on. Examples are contained in the array e. The following variables are used in the constraint model:

Integer Variable $P_i \in [0, n-1]$: index of the parent of node i,

Integer Variable $L_i \in [0, n-1]$: index of the left child of node i,

Integer Variable $R_i \in [0, n-1]$: index of the right child of node i,

Integer Variable $N_i \in [0, 2]$: number of children of node i,

Integer Variable $F_i \in [0, k-1]$: feature assigned to node i,

Integer Variable $D_{ij} \in [0, 1]$: boolean representing if node i is a descendant of node j,

Set Variable $E_i = \{0, ..., m-1\}$: set of examples associated with each internal node,

Integer Variable $T8_{ijk} \in [0,1]$: for flagging if an example goes left when tested at node i,

Integer Variable $T9_{ijk} \in [0,1]$: for flagging if an example goes right when tested at node i,

Integer Variable $T10_{ijk} \in [0,1]$: for flagging if two examples go down the same branch when tested at node i.

A.2 Constraint Program

The first constraint states that the root must be its own parent and the parent of its first node:

$$P_0 = 0 \wedge P_1 = 0. \tag{A.1}$$

Unlike the root node, no other node has itself as a parent,

$$\forall i \in [1, n-1], P_i \neq i. \tag{A.2}$$

All non-root nodes have less than three children,

$$\forall i \in [1, n-1], N_i \neq 3. \tag{A.3}$$

If a node has two children then the left and right child must exist,

$$\forall i \in [1, n-1], N_i = 2 \rightarrow (L_i \neq i \wedge R_i \neq i). \tag{A.4}$$

If a node has one child then the left xor the right child must exist,

$$\forall i \in [1, n-1], N_i = 1 \rightarrow (L_i = i \oplus R_i = i). \tag{A.5}$$

If a node has zero children than the left and right child must not exist,

$$\forall i \in [1, n-1], N_i = 0 \rightarrow (L_i = i \wedge R_i = i). \tag{A.6}$$

The first of the tree constraints defines the descendant variables. If j is $i's$ parent then i is a descendant of node j. Otherwise, j and i share the same descendants,

$$\forall i \neq j \in [0, n-1], (P_i = j) \rightarrow (D_{ji} = 1). \tag{A.7}$$

$$\forall i \neq j \neq l \in [0, n-1], (P_i = j) \rightarrow (D_{li} = D_{lj}). \tag{A.8}$$

The following constraint prevents cycles in the tree,

$$\forall i \neq j \in [0, n-1], (D_{ij} = 1) \rightarrow (D_{ji} = 0); \tag{A.9}$$

The tree must be binary, meaning that if i is $j's$ parent then j is either the left or the right child of i (but not both),

$$\forall i \neq j \in [0, n-1], (P_j = i) \leftrightarrow ((L_i = j) \oplus (R_i = j)). \tag{A.10}$$

The number of children of i is equal the number of times i appears in the 'parents of' array,

$$\forall i \in [0, n-1], occurrence(N_i, P, i). \tag{A.11}$$

No two features F_i, F_j are tested twice along a branch, meaning they are not descendants of one another,

$$\forall i \neq j \in [0, n-1], (D_{ij} = 1) \rightarrow (F_i \neq F_j)). \tag{A.12}$$

Create the set of examples to be passed to the left child. For each feature l, the variable $T8_{ijl}$ holds the F_i^{th} element of the example (the value node i has for feature l),

$$\begin{aligned} \forall i \neq j \in & [0, n-1], \forall l \in [0, m-1], \\ & nth(constant(l), F_i, e, T8_{ijl}) \\ & L_i = j \rightarrow (l \in E_i \wedge T8_{ijl} = 0) \leftrightarrow l \in E_j. \end{aligned} \tag{A.13}$$

Create the set of examples to be passed to the right child. For each feature l, the variable $T9_{ijl}$ holds the F_i^{th} element of the example (the value node i has for feature l),

$$\begin{aligned} \forall i \neq j \in & [0, n-1], \forall l \in [0, m-1], \\ & nth(constant(l), F_i, e, T9_{ijl}) \\ & R_i = j \rightarrow (l \in E_i \wedge T9_{ijl} = 1) \leftrightarrow l \in E_j. \end{aligned} \tag{A.14}$$

If two examples have different classification but have the same value for the feature on node i, then node i must have children,

$$\begin{aligned} \forall i \in [0, n-1], & \forall x \in [0, m-1], \forall y \in [x+1, m-1], e[x][k] \neq e[y][k], \\ & nth(constant(x), F_i, e, T10x_{ixy}), \\ & nth(constant(y), F_i, e, T10y_{ixy}), \\ & (x \in E_i \wedge y \in E_i \wedge T10x_{ixy} = T10y_{ixy}) \rightarrow N_i > 0. \end{aligned} \tag{A.15}$$

Extension of A.15 where two examples have different classifications and both go down the left branch making it an internal node ($L_i \neq i$),

$$\begin{aligned} \forall i \in [0, n-1], & \forall x \in [0, m-1], \forall y \in [x+1, m-1], e[x][k] \neq e[y][k], \\ & nth(constant(x), F_i, e, T10x_{ixy}), \\ & nth(constant(y), F_i, e, T10y_{ixy}), \\ & (x \in E_i \wedge y \in E_i \wedge T10x_{ixy} = 0 \wedge T10y_{ixy} = 0) \rightarrow L_i \neq i). \end{aligned} \tag{A.16}$$

Extension of A.15 where two examples have different classifications and both go down the right branch making it an internal node ($R_i \neq i$),

$$\begin{aligned}\forall i \in [0, n-1], &\forall x \in [0, m-1], \forall y \in [x+1, m-1], e[x][k] \neq e[y][k], \\ &nth(constant(x), F_i, e, T10x_{ixy}), \\ &nth(constant(y), F_i, e, T10x_{ixy}), \\ &(x \in E_i \wedge y \in E_i \wedge T10x_{ixy} = 1 \wedge T10y_{ixy} = 1) \rightarrow R_i \neq i). \end{aligned} \tag{A.17}$$

A.3 Symmetry Breaking

Although these constraints are not necessary, they reduce the number of decision trees that can be modelled by the constraint by ordering the nodes from top to bottom and left to right. The following constraint is to force an ordering of the nodes from top to bottom,

$$\forall i \in [0, n-1], P_i \leq min(i, P_{i+1}). \tag{A.18}$$

The next two constraints ensure that the nodes are ordered from left to right in the decision tree,

$$\forall i \in [0, n-1], (i \leq R_i) \wedge (R_i \leq 2i + 2). \tag{A.19}$$

$$\forall i \in [0, n-1], (i \leq L_i) \wedge (L_i \leq 2i + 1) \wedge (R_i \neq i \rightarrow L_i \leq R_i). \tag{A.20}$$