NTNU
Norwegian University of
Science and Technology

Growing a Forest:

Genetic Decision tree Induction

Anders Nikolai Fuglseth

Master of Science in Computer Science
Submission date: June 2016
Supervisor: Agnar Aamodt, IDI
Co-supervisor: Ketil Bø, Trollhetta AS

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

In decision tree learning, the traditional top-down divide and conquer approach searches a limited part of the hypothesis space, often leading to sub-optimal solutions. By doing decision tree induction with the use of an evolutionary algorithm the hypothesis space can be searched globally, leading to stronger solutions, while maintaining the inherent comprehensibility that decision trees offers. We have developed EMTI, the Evolutionary Multi-class Tree Inductor, a genetic programming method for inducing parallel axis, poly-ary decision trees for multiclass classification problems. It focuses on creating accurate decision trees with a high degree of human readability. EMTI uses a genetic programming encoding-scheme representing individuals directly as decision trees, and implements tree-specific crossover and mutation operators. Initial population is generated in the form of minimal, one decision node trees, which grow rapidly in size as the evolution cycle count increases. The multi-objective fitness function rewards classification accuracy while favoring smaller trees over larger ones. Traditional decision tree pruning methods and early stopping methods are shown to be viable ways of avoiding overfitting in the algorithm. EMTI scores favorably in terms of classification accuracy compared to C4.5 and shows a strong ability to ignore data noise and irrelevant attributes.

Sammendrag

I beslutningstrelæring søker de tradisjonelle topp-til-bunn-, splitt-og-hersk-tilnærmingene et begrenset del av hypoteserommet, noe som ofte fører til sub-optimale løsninger. Ved bruk av beslutningstre-induksjon styrt av en evolusjonær algoritme kan man gjøre globalt søk av hypoteserommet, noe som leder til bedre løsninger, samtidig som man beholder den iboende forståeligheten som beslutningstrær tilbyr. Vi har utviklet EMTI, the Evolutionary Multi-class Tree Inductor: Evolusjonær Multiklasse Tre-induktør, en metode som bruker genetisk programmering for å indusere polyære, parallellaksede beslutningstrær for mange-klassede klassifiseringsproblem. Den fokuserer på å skape nøyaktige beslutnings-trær med en høy grad av menneskelig forståelighet. EMTI bruker genetisk programmering for å representere individer ved å framstille dem direkte som beslutningstrær, og implementerer tre-spesifikke kryssing- og mutasjonsoperatorer. Den initielle populasjonen blir generert som minimale beslutningstrær som vokser hurtig i størrelse etterhvert som evolusjonsprosessen utvikler seg. Den todelte evalueringsfunksjonen som avgjør kvaliteten på trærne i populasjonen gir belønning til nøyaktige trær mens den foretrekker små trær over store trær. Det blir demonstrert at tradisjonelle beslutningstrebeskjæringsmetoder og tidlig stopp-metoder er hensiktsmessige måter å unngå overtilpassing (overfitting) i algoritmen. EMTI skårer gunstig i forhold til klassifiseringsnøyaktighet i sammenligning med C4.5 og viser en sterk evne til å ignorere data-støy og irrelevante attributter.

Preface

This thesis is submitted to the Norwegian University of Science and Technology in partial fulfillment of the requirements for a Master of Science degree in Computer Science.

The work was conducted in 2016 at the Department of Computer and Information Science NTNU and at Trollhetta AS with supervisors Agnar Aamodt (NTNU) and Kjetil Bø (Trollhetta).

I would like to thank my supervisors Agnar Aamodt and Kjetil Bø for their support and guidance.

Table of Contents

List of Figures

List of Tables

List of abbreviations

AI	Artificial intelligence
AS	Aksjeselskap
AQ	Algorithm Quasi-optimal
dll	dynamic linked library
EA	Evolutionary algorithm
EMTI	Evolutionary Multi-class Tree Inductor
GA	Genetic algorithm
GP	Genetic programming
GUI	Graphical user interface
ID3	Iterative Dichotomiser 3
kNN	k-nearest neighbor
MLP	Multilayer Perceptron
NTNU	The Norwegian University of Technology and Science
NP	Nondeterministic polynomial time
OS	Operative system
RQ	Research question
SD	Standard deviation
UCI	University of California
UML	Unified modeling language
U.S.	United States (of America)

CHAPTER 1

Introduction

This master thesis project started with the task of constructing and implementing an AI method based on evolutionary algorithms. The groundwork and conception of the project was done in a pre-master specialization project in 2015 (Fuglseth [10]). It is done in collaboration with Trollhetta AS [34], and the implementation is intended to be added to Trollbrain, a module based AI method ensemble currently under development, featuring among others case-based reasoning and artificial neural network problem solving. An algorithm capable of decision tree learning was seen as an attractive addition to the ensemble.

Compared to the greedy local search algorithms traditionally used by decision tree learners, evolutionary algorithms searches the solution space globally. By broadening the search space, the hope is to be able find better, more accurate solutions.

Many learning methods such as ensemble methods and artificial neural networks provides result of high accuracy, but offers very little in terms of explaining how the methods arrive at their decisions.

EMTI: The Evolutionary Multi-class Tree Inductor was proposed, a method using evolutionary algorithms to generate decision trees, focusing on providing results with a high degree of interpretability and comprehensibility by creating single, poly-ary, parallel-axis decision trees. In analogy to growing a forest in the real world, EMTI starts out by planting a series of saplings across a field. As they grow up, the trees' quality is measured using a certain criterion (For example, if the field was growing christmas trees, they could be measured by symmetry and color.) and low quality trees are cut away. New trees are planted in their stead, made by cross-breeding the higher quality trees with each other. In such a way, as time passes, the field will gradually contain higher and higher quality trees. At the end of the process, the very best tree is chosen as the solution.

1.1 Goal and research questions

The goal of this master thesis project is to study for, and develop a working classification method capable of inducing decision tree by the use of evolutionary algorithms. The performance of the method should be analyzed, both the performance of its whole and the performance of its parts, and key weaknesses should be identified and tried to be improved upon.
Goal: Develop a robust decision tree learning algorithm using evolutionary algorithms, that can be integrated into Trollbrain.

To go along with this goal, 5 research questions will be tried answered:

The proposed version of EMTI uses a range of experimental methods to make the evolution process possible, the most important being the 4 types of mutation operators, the crossover operator and the majority vote end node setting method. All though the choice of adding each of the methods is well motivated, the individual contribution of each method remains unmeasured.
RQ1: What method components are essential for the proposed version of EMTI?

Overfitting is a prevalent problem among many classification methods. It occurs when a classifier starts modeling the random error or noise of a given training set instead of the target function it wants to describe. This leads to unnecessary complex models and degrades the classification accuracy. All though EMTI is showing some resilience to overfitting by the use of a multi-objective fitness function that tries to maximize classification accuracy and minimize tree size, overfitting still occurs on certain data sets, as shown in chapter 3.6. Implementing methods for avoiding overfitting should improve the general performance of EMTI.
RQ2: In which way can methods for avoiding overfitting be added to EMTI to improve its performance?

Overfitting in a decision tree is generally linked to an increase in the size and complexity of the tree. EMTI has the ability to limit the sizes of the trees that it creates at any given point of its duration.
RQ3: Can EMTI's ability of setting the maximum tree size be used in methods for avoiding overfitting?

Any learning algorithm will have strengths and weaknesses depending on the domain's data landscape, scope and size of data. For a user to choose the correct algorithm for the domain, it is important that these strengths and weaknesses are identified.
RQ4: What are the characteristics of the domains that EMTI functions well on?

To measure its performance, EMTI could be compared against other well established classifier algorithms, tree learning algorithms such as C4.5, ensemble methods such as Random Forest and other algorithms for example artificial neural networks or naïve bayes methods.
RQ5: How does ETMI perform compared to other well know classifier methods?

1.2 Research Method

Typical for research done in machine learning, this project is well suited for a research method based on quantitative experiments. The experiments are done by testing the learning algorithms on several different domains, represented by structured datasets. In these tests, several performance measures are observed; classification accuracies, training accuracies, run times and tree sizes, which allows comparisons between different algorithms or different sub-methods to be made. The experiments make the use of k-fold cross-validation were applicable and are repeated a number of times to minimize variance in the results.

1.3 Thesis structure

The thesis is structured in 5 chapters:

Chapter 1: Introduction introduces the project, explains the purpose and scope of the project and defines the research questions.

Chapter 2: Background theory and Related work provides the relevant background theory of the project and presents a literature review covering the research field of the project, explaining the motivations behind it.

Chapter 3: Model explains the main EMTI algorithm as it is implemented at the time of writing, describes some specific implementation details and explains a few auxiliary capabilities and functions. Some of the main weaknesses of EMTI are identified and ways to improve them are suggested, including specific implementations directed at answering the research questions.

Chapter 4: Results describes the execution and presents the results of 3 experiments, **E1**, **E2** and **E3**, set out in order to answer the research questions **Q1**, **Q2**, **Q3**, **Q4** and **Q5**.

Chapter 5: Evaluation and Conclusion evaluates the results from the experiments in chapter 4 and these evaluations are put in relation and used for discussing and answering the research questions that were posed in chapter 1. Finally, the project's potential and possibilities for future work is discussed.

CHAPTER 2

Background theory and Related work

This chapter provides the relevant background theory of the project and presents a literature review covering the research field of the project, explaining the motivations behind it.

2.1 Background theory

This section covers relevant topics within artificial intelligence and machine learning which are necessary to understand the function and purpose of the project, the main topics being decision trees, which are EMTI's main objective to create, and Evolutionary Algorithms, the tool that EMTI uses to create them.

2.1.1 Learning and Decision Trees

Russel and Norvig [29] defines learning as an agent's ability to improve its performance on future tasks after making observation about the world. Learning is useful because it allows the agent to adjust its behavior when met with changing or unforeseen situations. This also makes the job easier for the designer of the agent, as he does not need to anticipate all possible situations and changes within the agent's domain. Machine learning means making machines, typically computers, be able to learn. The ability to learn is a hallmark of intelligent behavior and machine learning is a central research subject within artificial intelligence (Quinlan [26]). This project deals with supervised inductive learning, where the agent has to model a general function or rule, based on a set of example data (input-output pairs), where there is explicit feedback given for each instance about its output function value. Below we can see an example of how this data could look, dealing with observations on whether a person decided to wake up or not:

ID	Slept for	Employed	Weekend	Time	Feel like it	Woke up
1	5 hours	*yes*	*no*	7:30	*no*	*yes*
2	10 hours	*no*	*no*	3:30	*yes*	*yes*
3	9 hours	*yes*	*no*	5:30	*no*	*no*
4	8 hours	*yes*	*yes*	9:30	*no*	*no*
5	13 hours	*no*	*yes*	13:30	*no*	*yes*
6	8 hours	*yes*	*no*	7:15	*no*	*yes*
7	3 hours	*no*	*yes*	14:50	*no*	*no*

Classification is a learning problem where the output of the function is one of a finite set of values. When there are only two possible values (e.g. *yes* or *no*) it is called a binary classification, while when there are multiple (such as *sunny*, *cloudy* or *rainy*) it is called a multiclass classification. When the output of the function is continuous it is called regression. Many tasks can be recast as classification problems, even activities like robot planning [26].

A decision tree is a function in the form of an acyclic graph that can be used to model classification problems. An example decision tree is shown in figure 2.1, based on the data above, the object is to classify whether it is worth to wake up or not, using the 5 input attributes. In this tree each attribute forms a decision node that contains a question with two or more possible answers. Starting from the top node, "Employed?", you make your way down the tree to new nodes depending on what you answer until you reach an end node. Each terminal node contains an answer for what you should do, your decision. One of the main advantages of decision trees is their comprehensibility, "decision trees are a natural representation for humans, indeed, many "How To" manuals (e.g. for car repair) are written entirely as a single decision tree stretching over hundred pages" [29].

An important part of the machine learning process that will not be focused on in this project is the data preprocessing stage. Data samples from the real world is often not usable for machine learning because it is in an unusable format, contains excessive noise or simply contains too much data than the machine can handle. Data preprocessing concerns the formatting, cleaning and sampling of such data. It also includes a number of other potentially useful things to do with a dataset before turning it over to the classification algorithms, such as data scaling, feature selection, feature decomposition and feature aggregation, which can considerably improve the final classification results.

2.1.2 Decision tree induction

Inducing a decision tree is a way to solve classification problems where an agent generates a decision tree structure based on previous observations of labeled examples (the *training set*). The optimal decision tree for a given training set is a tree that correctly classifies all the examples using the minimum possible number

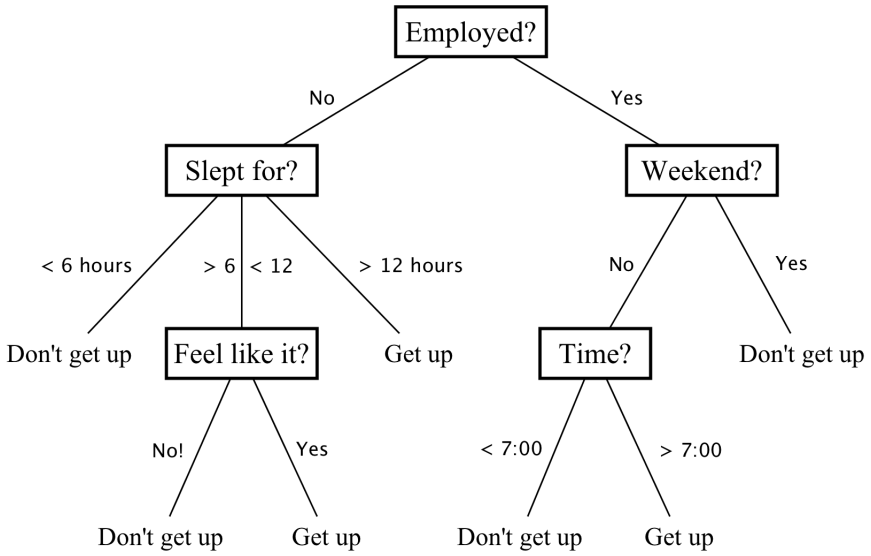


Figure 2.1: A decision tree for deciding whether a person should wake up or not.

of nodes. Finding a optimal tree is shown to be a NP-complete problem (Hyafil and Rivest [14]), so this option is only realistic for very simple problems (e.g. few attributes and labels). The traditional and most common approach for generating decision trees is using a greedy top-down recursive partitioning strategy. You start by making the root of the tree and make your way down without looking at the tree globally. Choosing which attributes to put first in a tree has a big impact on the tree's complexity and accuracy, greedy approaches typically use an heuristic to find the most "pure" attributes; the attributes that can split the training set in the most absolute way.

2.1.3 Specific learning algorithms

This section contains information about some other well-known algorithms, the k-Nearest Neighbor algorithm, that is used internally in the EMTI algorithm and ID3, C4.5, Random Forest and Multilayer Perceptron, that were used for making comparisons with EMTI.

k-Nearest Neighbor algorithm

k-Nearest Neighbor (kNN) is a classification method based on comparing a given instance to classify with the most similar instances in the training set, the nearest neighbors. It does this by iterating over the entire training data set, finding the instances that have the most similar attributes, the similarity score. How many neighbors to find depends on the k parameter, $k = 1$ for 1 neighbor, $k = 2$ for

2 neighbors, etc. When dealing with continuous values, it can also look at the distance between numbers as a similarity measure; k-Nearest Neighbor is also well suited for regression problems. Though it is a simple, well performing algorithm, it has some major drawbacks. The entire algorithm has to be re-run for each query, and the run times can increase significantly depending on the problem size. The basic implementation does also not take in consideration the weight that different attributes can carry. Finally, another problem can be that finding which k-value to set can be hard before running the algorithm.

ID3 algorithm

A well-known algorithm using the greedy strategy mentioned above is the ID3 algorithm by Ross Quinlan [26], which has been implemented in the project to use for making comparisons to EMTI. It is a recursive algorithm that uses the concepts of *entropy* and *information gain* to find the most appropriate attribute to split the data set on. The entropy function measure the amount of uncertainty in a data set and is given as

$$H(S) = -\sum_{x \in X} p(x)\ log_2\, p(x) \tag{2.1}$$

Where H is the entropy, S is the data set, x is a class, X is all possible classes and $p(x)$ is how many instances in the set is labeled with x, divided by the data set size. Information gain is a measure on the change in entropy after a data set is split on an attribute, given as

$$IG(A, S) = H(S) - \sum_{t \in T} p(t)\ H(t) \tag{2.2}$$

Where IG is the information gain, A is the attribute to split on, S is the data set, $H(S)$ is the entropy of the data set, T is the subsets created by splitting S on each possible value of A, $p(t)$ is how many instances in the set is labeled with the class associated with t, and $H(t)$ is the entropy of subset t.

These metrics are used for choosing which attribute to split on, the one who gives the best information gain. This is then repeated recursively, moving down the tree by splitting it on the best attributes, until all the data set has been split into pure class subsets.

C4.5 algorithm

C4.5 [27] is an extensively improved version of ID3 by Quinlan. It is able to handle continuous values, attributes with differing costs, and training data with missing values. It uses postpruning to simplify the tree, removing unnecessary and statistical insignificant nodes.

Random Forests

Random Forests [4] is an ensemble learning method by Leo Breiman and Adele Cutler. It works by creating multiple decision trees based on distinct random samples of the training data, and classifies instances based on the majority vote of the different trees. By doing this, Random Forests have high resistance to overfitting.

Multilayer Perceptron

A multilayer perceptron (MLP) is a feedforward artificial neural network model, using the backpropagation learning technique.

2.1.4 Evolutionary Algorithms

Evolutionary algorithms (EA) are optimization problem solving tools that imitate the processes of real world evolution. It does this by establishing a population of candidate solutions which are incrementally improved upon by combining solutions (crossover) and randomly changing the solutions (mutation). Each individual is evaluated according to how well they solve the problem - their fitness score. This score determines their chance of being selected as parents in the crossover operations and their chance of surviving into a new generation. A well functioning EA must maintain a balance between the pace of the solution improvement and the diversity of solutions within the population, as low diversity will increase the chance of getting stuck in a local optimum. Figure 2.2 shows the flow diagram of an evolutionary algorithm, which is explained in detail below:

Make initial child population: At the start of the algorithm, a population is established, which are typically randomly generated. This population is put into the child population pool (the new generation), while the adult population pool (the old generation) remains empty.

Determine fitness of children: At the beginning of each evolution cycle, a fitness function is applied to each individual to give them their fitness score.

Perform generation shift: The new children typically have to compete with the old pre-existing generation for survival, where the fitness score determines the chance of surviving. The surviving children are then put in the adult pool together with the surviving adults, emptying the children pool. Sometimes it can be better (in order to increase diversity) to have no competition and simply replace all the individuals in the old generation with the new one. In the first cycle of the algorithm, the adult pool is empty, so all children are transferred directly.

Select parents: Parents are individuals who's genes will be used for creating children in the next step, the crossover operations. There are many methods

for selecting parents, but the main idea is to give individuals with a high fitness score a bigger chance, while still leaving some chances for low fitness score individuals to be selected.

Perform crossover to create children: The selected parent solutions are merged together (typically two and two) creating a child solutions that take a mix of characteristics from each of its parent solutions.

Perform mutations on children: Each child will has a given chance of having a mutation, randomly changing a part of the solution. The algorithm then returns to the *Determine fitness of children* step, starting a new evolution cycle.

The algorithm will keep running until an end criterion is reached, which is typically reaching a certain fitness score for the best individual, or performing a fixed amount of cycles.

Within EA we separate between two main methods when dealing with the problem of how to encode individuals, Genetic Algorithms (GA) and Genetic Programming (GP) (Streichert [31]). GA's are perhaps the most similar to how real genes are encoded, giving an individual a genotype, a binary string of fixed length, which is then mapped unto a phenotype, the actual solution that the individual represents. All genetic operators (mutation and crossover) are performed on the genotype and are problem independent, while the fitness score is calculated on the phenotype. GP is used when the solutions are in the form of programs or functions that are not well suited for being mapped unto a binary string. Here the genetic operators are performed directly on the solution encoding, which requires problem dependent genetic operators.

2.2 Literature Review Protocol

A formal literature review was done at the start of the project in order to place it within the framework of current knowledge and research. The literature search was done on the internet using Google Scholar [11], a free web search engine for scholarly literature. Searches were done for three main subjects, evolutionary decision tree induction, early stopping and decision tree pruning. The search for evolutionary decision tree induction was by far the most extensive, attempting to cover the majority of the research conducted. Due to the wast amount of research available, a set quality and inclusion criteria where used to filter the search, described in table 2.1. The search included the following key word combinations:

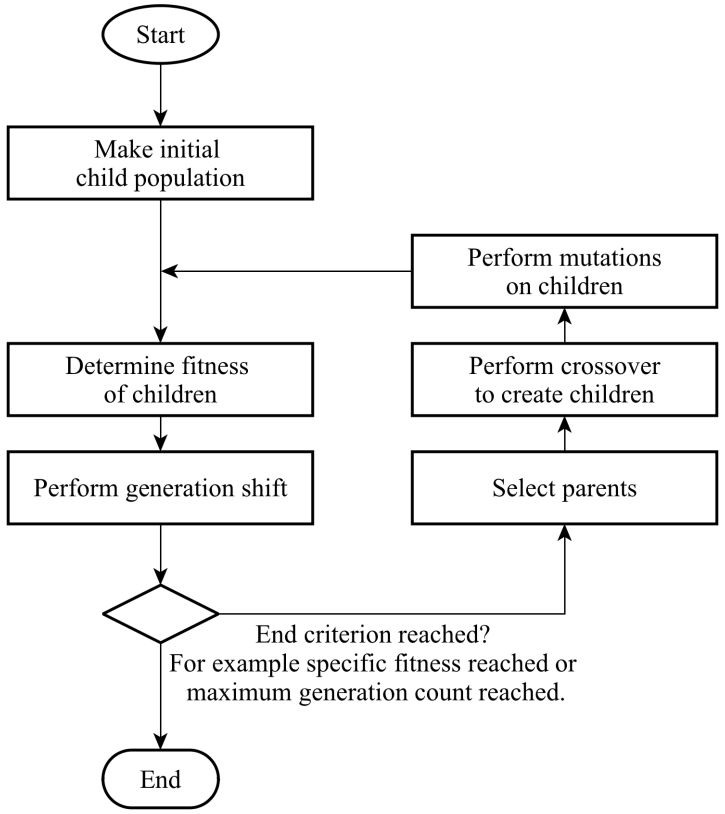


Figure 2.2: Flow diagram of an evolutionary algorithm.

Table 2.1: Quality and Inclusion criteria

#	Type	Criteria
Q1	Quality	The article should have a clear and concise abstract explaining what it is about.
I1	Inclusion	The article should contain details and explanations of an evolutionary algorithm for tree induction.
I2	Inclusion	The article should not be be about constructing an algorithm towards specific domains.
I3	Inclusion	The article should not be about using existing algorithms for specific tasks or specific domains.

1. “genetic” & “decision tree”
2. “evolutionary” & “decision tree”
3. “genetic programming” & “decision tree”

For early stopping and decision tree pruning, the goal was not to cover the topic extensively, but finding good examples of methods that EMTI could use or be inspired by The following search words were used for the early stopping search:

1. “early stopping”
2. “early stopping” & “evolutionary algorithm”
3. “early stopping” & “genetic programming”
4. “stopping criteria” & “evolutionary algorithm”
5. “stopping criteria” & “genetic programming”

For decision tree pruning, the following key words were used:

1. “decision tree pruning”
2. “decision tree pruning” & “evolutionary”
3. “decision tree pruning” & “genetic”

For both early stopping and tree pruning, combining the search with evolutionary or genetic provided very few directly relevant searches.

2.3 Related work

The greedy, top-down recursive algorithms traditionally used in decision tree induction, have the disadvantages that they search locally which often leads to suboptimal solutions and that they split the deepest part of the tree based on very small datasets, often leading to overfitting. Fayyad [7] discusses how the impurity measures (information gain, gain ratio, gini-index etc.) commonly used in these methods have several deficiencies, in that they are insensitive to inter-class separation and intra-class fragmentation, as well as insensitive to permutations of the class probability distribution (Papagelis and Kalles [24]). Ensemble tree induction have been a popular way to combat many of the weaknesses of recursive tree induction algorithms. However, ensemble algorithms have the disadvantage in that they produce a multitude of trees, lowering the comprehensibility of the solutions.

Evolutionary algorithms searches the solution space globally and can therefore find solutions that recursive tree induction algorithms are not able to find. EA’s tend to cope better with attribute interactions than greedy methods (Freitas [8]). Compared to ensemble methods, evolutionary tree induction has the advantage of being able to produce a single, comprehensive decision tree.

2.3.1 Choices in evolutionary decision tree induction

Creating decision tree classifiers using genetic algorithms and genetic programming has been tried various times. "A Survey of Evolutionary Algorithms for Decision Tree Induction" (Barros et al. 2012 [3]), provides a taxonomy and definitions of concepts based on the previous work done in the field, dividing it into two main categories: evolutionary induction of decision trees and evolutionary design of decision tree components. In the first category each individual in the population is a decision tree, while in the latter category individuals are components of decision tree classifiers. They further divide evolutionary induction of decision trees into axis-parallel trees, when there is a single attribute that splits the data set per node, and oblique, where nodes can be combinations of attributes. Using oblique trees have the potential of producing smaller and more accurate decision trees, while axis-parallel have the advantage of being generally much easier to interpret. This project uses evolutionary induction of axis-parallel decision trees because of its higher comprehensibility, which has been the most common strategy [3].

An important choice to make is whether to encode individuals using the classical GA genotype-phenotype scheme with trees represented as fixed-length binary or integer strings or the GP approach of using tree-based encoding. The GA approach seems to have been popular in many of the earlier attempts of evolutionary induction of decision trees, as seen in Kennedy et al. [16] and Bandar and McLean [2], but also in more recent papers (Smith [30]). This approach allows the use of the standard methods of mutations of crossover operations using flipping digits and combining strings. It is possible that this was done because this would being easier to implement when using less powerful computers and less expressive languages. The main drawback is that when applying mutation and crossover you get a high amount of corrupt genes (a string that doesn't map correctly to a tree), which have to be either avoided or discarded using some extra methods. The GP, tree-based approach, as seen in Aitkenhead [1], Fu et al. [9], Kretowski and Grzes [17], Papagelis and Kalles [24] and Zhao and Ciesielski [37] dissolves the genotype-phenotype scheme into a single data structure, the decision tree itself. Here it is necessary to develop new tree-specific operators for mutation and crossover, but all problems regarding corrupt genes can be avoided.

When using a tree-based encoding of individuals the standard crossover and mutation operators in the genetic algorithm needs to be rewritten. However, crossover is virtually the same operation, you select two parents, choose a random subtree from each parent and swap these subtrees, creating two distinct children. The vast majority of the evolutionary decision tree inductors uses this method [3]. For mutation it can be useful to mutate both the value of the nodes (and the test values associated the node for binary trees) ([1, 24]) and the tree structure itself ([9, 37]) (typically replacing a subtree with a random subtree).

I've found that most papers restrict their attention to binary trees (some papers are unclear which types of trees they are using), including [1, 24, 9], which when dealing with multiclass classification problems requires the same attribute to ap-

pear in many levels of the tree, and will generate incomplete trees unless special care is taken. As shown in Fu [9], binary trees can be transformed into poly-ary trees in linear time and will contain maximum twice as many nodes as a poly-ary tree. Binary trees makes it easier to split numerical and real values into separate categories (Loveard and Ciesielski [18]).

Another important decision is how one should generate the initial population. One strategy is to use an existing algorithm such as C4.5 on random subsets of the training set to generate a set of trees for the initial population ([9]). In another common strategy, called the full method, initialization is randomly choosing attributes and split values from a predefined list and halting the decision tree growth when the tree reaches a depth that is randomly selected within a pre-set interval ([37]) [3]. A more basic strategy is simply generating minimal trees that are split on a single attribute, i.e. "saplings", often referred to as *caltrops* ([1, 16, 24]).

The genetic algorithm's parent selection types and parameter settings are also of importance. For parent selection Barros' survey shows that tournament selection or roulette wheel selection is most commonly used, while rank-based selection being used on a few occasions. The parameter values such as populations size, number of generations and mutation rate are of critical value for making a genetic algorithm move towards a good solution. Barros writes that "most authors prefer to present a set of default parameter values followed by a sentence like parameter values were empirically defined." There has however been made considerable effort into developing good heuristic for finding good parameters programmatically (Michalewicz and Schmidt [19]). In Papagelis [24] they added a second layer evolutionary algorithm for finding good parameter values.

The heart of an evolutionary algorithm's effectiveness lies in its fitness function. The obvious and most common measure for the fitness function for decision tree induction is the classification accuracy (or the complement, the classification error) on the training set. Another important measure is the size of the tree, where smaller, more general trees are preferred. Some papers have focused on the single classification accuracy objective (or various similar versions of it) [1, 9], using pruning techniques to keep the trees small in size. Others use a multi-objective weighted fitness measure, taking in account both the accuracy and the size of the trees [24, 17]. Zhao [37] implements a Pareto dominance approach that, instead of providing a single optimal solution based on the weighted combination of objectives, provides an optimal set of non-dominated solutions [3].

2.3.2 The problem of overfitting

Overfitting occurs when a classifier starts modeling the random error or noise of a given training set instead of the target function it wants to describe causing the training error to become higher than the test error. This leads to unnecessary complex models and degrades the classification accuracy. A common way to avoid overfitting in evolutionary tree induction has been by using a multi-objective fitness function, that tries to maximize classification accuracy and minimize tree size using

the tree size weight parameter as balance point [3]. However making this work often requires a fine tuned tree size weight parameter, which is hard to do without resolving to a trial and error approach.

There are several other methods for combating overfitting that could also be applied for EMTI. Early stopping (pre-pruning) is an often used technique in machine learning, especially within the field of evolutionary algorithms and the field of artificial neural network, but does not seem to have been tried for evolutionary tree induction. Two main methods have most often been used in top-down recursive algorithms, ensemble methods such as Random Forests (Breiman [4]) and decision tree pruning (post-pruning) methods. Using ensemble methods in evolutionary tree decision will require many different evolution runs, making the time cost of such a method prohibitively expensive. Tree pruning is a much more applicable approach, that only needs to be done on the best tree at the end of the evolution cycle.

CHAPTER 3

Model

This chapter explains the main EMTI algorithm as it is implemented at the time of writing, describes some specific implementation details and explains a few auxiliary capabilities and functions. Some of the main weaknesses of EMTI are identified and ways to improve them are suggested, including specific implementations directed at answering the research questions.

3.1 Evolutionary Multi-Class Tree Inductor

EMTI is an evolutionary algorithm for inducing parallel axis, poly-ary decision trees for multiclass classification problems. It's focus and motivation for many of its implementation choices is to create accurate decision trees without losing comprehensibility. Parallel axis, poly-ary decision trees are in general easier to comprehend than e.g. oblique, binary trees when dealing with multiclass domains. The fact that EMTI creates poly-ary trees is perhaps the biggest factor of what sets it apart from the related work done in the field. In general EMTI does not handle continuous values, but they can be discretized during preprocessing. The use of evolutionary algorithms makes the method also applicable for reinforcement learning (See section 3.5).

3.1.1 Representation and Operators

The algorithm falls under genetic programming, an approach used by many previous methods for evolutionary decision tree induction [1, 9, 17, 24, 37]; each individual is encoded directly as a decision tree. Because of the irregular structures of the trees generated, using a GA approach would create a high number of corrupt tree solutions, the GP approach avoids this problem. This requires developing special genetic operators to be applied to the tree data structure. Figure 3.1 illustrates the different operators used in EMTI, showing only decision nodes - end nodes are

never changed by genetic operators. Crossover is done intuitively for the GP approach in the way described in section 2.3.1, replacing a random subtree from one parent with a random subtree from the other parent. The crossover operator allows for combining solutions into one, with the hope and the statistical inevitability of combining the best parts of two different solutions. To promote tree size growth in early generations, the root nodes can not be replaced during crossover. Mutations are in the form of small local changes to the tree structure and tree values and are added to the algorithm in order to stimulate the genetic diversity of the population. We have chosen to combine both mutation types that mutate the value of the nodes ([1, 24]) and the tree structure itself ([9, 37]) in order to ensure genetic diversity and to make it easier for trees to increase and decrease in size. EMTI uses 4 different types of mutation operators:

1. *Cut*: While the crossover operation promotes growth of tree sizes, this operator promotes decreasing tree sizes by cutting away a randomly chosen subtree.

2. *Implode*: A decision node is removed and replaced by a randomly chosen decision node child. The idea behind this operator is to promote the removal of redundant nodes deep in a tree that don't affect the classification accuracy negatively. These nodes have a small impact on the fitness of the tree, so they are less likely to disappear.

3. *Deplode*: If *implode* is called on a node that has no decision node children, *deplode* is called in stead. This operation replaces an end node child of the node with a randomly generated minimal decision node sub tree.

4. *Flip*: Randomly flip which attribute a decision node is tested on to another, legal attribute, similar to the traditional way of doing mutation in EA's. If the new attribute has less categories than the previous one, we remove children from a random location until the node has the same amount of children as the attribute has categories. If it has more categories, we add end node children to the node in the same fashion.

Though the motivation for each mutation operator is clear, it is not clear if they function the way they should and contribute positively to the algorithm. This is addressed in section 4.3, were an experiment is done to measure the individual contribution of each part.

The chance of a mutation occurring is governed by the mutation rate parameter. A mutation rate of 1.0 means that 1 mutation will occur on average on each individual in the population - for a tree with 10 decision nodes, the chance of $\frac{1}{10}$ for a mutation to occur is rolled on each node. When a mutation occurs, the mutation type is randomly chosen. The mutation rate strikes the balance between local and random searching and needs to be set quite low, not more than 1.0 has shown to be beneficial because as the mutation rate increases the algorithm's search increases in randomness, making it more and more inefficient. If the mutation rate is set too low, the search risks becoming too narrow and getting stuck in a local optimum.

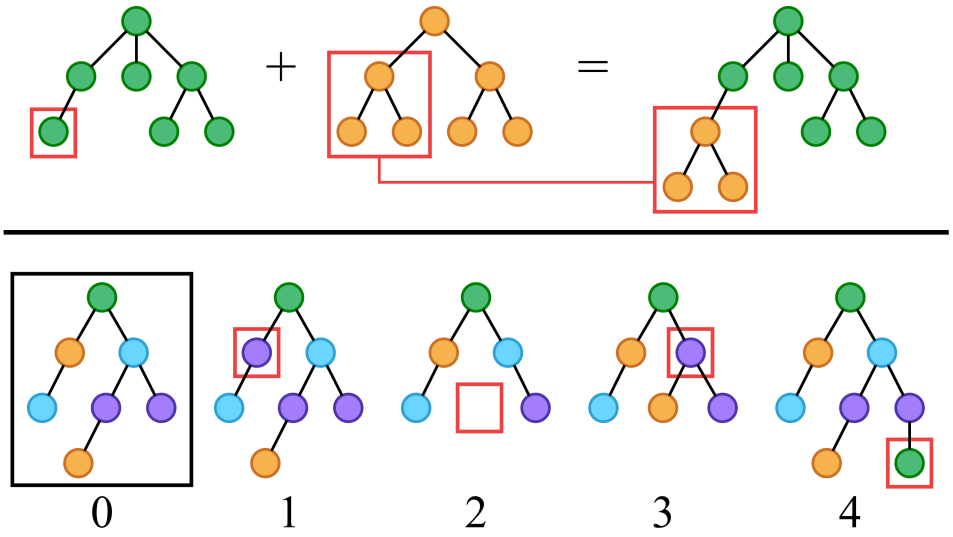


Figure 3.1: Genetic operators: crossover (top) and 4 different types of mutations: *Flip*, *Cut*, *Implode* and *Deplode* (Bottom).

3.1.2 Initial population

Inspired by the work of Papagelis and Kalles [24], the initial population consist of a series of "saplings", randomly generated minimal trees consisting of a single decision node and children end nodes for each of the corresponding attribute's possible categories. The first frame of figure 3.3 shows how the first population typically looks like for the diagonal problem.

3.1.3 Fitness function

After the initial population has been generated, it is time to evaluate the individuals using the fitness function. EMTI uses a multi-objective fitness function ([24, 17], with the objectives being maximizing classification accuracy and minimizing tree size:

$$F = \frac{1}{|X|} \left(\sum_{x \in X} C(x) - \frac{s}{w} \right) \tag{3.1}$$

Where F is the fitness score, s is the size of the tree, w is a weight parameter that controls how much impact the tree size makes on the overall fitness score, x is an instance of the training set X and $C(x)$ is 1 if an instance was classified correctly by the tree and 0 if it was classified incorrectly. To normalize the fitness between 0 and 1, the fitness is divided by the size of the training set $|X|$. Potentially, a tree with zero accuracy will have a negative fitness score, if so, it gets adjusted to zero.

The main objective should be classification accuracy, which is achieved by setting w higher than 1. But we also want the algorithm to produce as small trees as possible, not just to avoid overfitting the training data, but also to encourage the removal of redundant nodes in the tree and to increase the tree's comprehensibility. Currently EMTI uses a static weight parameter that can be set by the user. Which weight to set depends on the characteristics of the classification problem, and does not necessarily correlate with the amount of attributes and categories a problem has, but rather the size of the optimal minimal decision tree of the problem. A correctly set weight parameter can strike a perfect balance between underfitting and overfitting, but this is difficult to do without resolving to trial and error approaches. Because of this, other ways of avoiding overfitting are explored in section 3.6.

3.1.4 Adult and Parent selection

Continuing with the algorithm according to the flow diagram shown in figure 2.2, it is now time to perform the generation shift, turning children into adults. For the primordial generation of children there are no adults to compete with, so they are all immediately transferred to the adult population. But later child generations could have to compete with the existing adult pool for survival. EMTI offers tree different types of generations shifts that are typically used in EA's, that the user can chose:

1. *Full generational replacement*: All adults die and are replaced by their children.
2. *Generational mixing*: The strongest half the children and the strongest half of the adults survive.
3. *Overproduction*: Full generation replacement where we produce the double amount of children which have to compete against each other for a place in the adult pool.

In addition there are two more parameters dealing with the generation shift, reject worse generation, which gives the possibility of totally rejecting a new child generation if the best fitness is lower than in the old generation, and spice population, which gives the possibility of making a percentage of each new generation be randomly generated minimal trees as in the initial population.

Another three different methods often used in EA's are available for the parent selection:

1. *Tournament selection*: n individuals are randomly chosen from the population and the one with highest fitness is selected as a parent. There is also the added possibility of setting the parameter ϵ between 0 and 1 which gives the chance of the selection being completely random.
2. *Fitness proportionate*: Also called roulette wheel selection, each individual is given a chance of being selected as their own fitness divided by the sum of the fitness of all the individuals.

3. *Sigma scaling*: Like fitness proportionate selection, but uses the standard deviation of the population fitness score to regulate the selection pressure. When the standard deviation is low, small differences in fitness score have a bigger (positive) impact on the chance of being selected.

3.1.5 Diagonal problem

To use for testing during the implementation and to explain the methodology used in EMTI, a simple artificial classification task was created called the Diagonal problem. As show in figure 3.2, the Diagonal problem consists of a 4 square grid where each square has three possible colors: red, green or blue. Instances are classified by how many undisturbed same color diagonals they have, the possible values being 0, 1 or 2 diagonals. Using an artificial problem allows for adjusting the size and the characteristics of the data set, adding noise and irrelevant attributes, useful when testing the algorithm, in particular testing its resilience to overfitting.

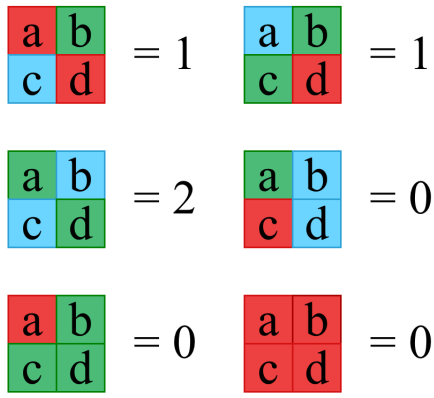


Figure 3.2: Some instances of the Diagonal problem and their class.

3.1.6 Algorithm properties

As an illustrative example, figure 3.3 shows 10 evolution cycles when running EMTI on the diagonal problem, with a population size of 10. The selection types used here are full generational replacement and fitness proportionate selection, with a mutation rate set to 2.0. These settings causes very high diversity in the solutions, which are good for illustration purposes, but slows the improvement pace significantly down and is not necessary for achieving good results. The typical parameters we have found to work well are much more greedy (less diversity but improves faster) and are to use sigma scaling and generational mixing as selection types, and a mutation rate set to 1.0. Figure 3.4 shows a plotting of the training set accuracy, tree sizes and the total fitness score of 100 generation cycles using

these settings. Notice the typically occurring saw tooth pattern in figure 3.4b of the best individual's tree size. Each sharp increase in tree size is caused by a break through in tree accuracy, normally caused by a particularly successful crossover operation. This sometimes leads to a "hypothesis explosion," with a series of accuracy improvements in a short period of time, which soon reaches a limit and then is slowly improved upon by finding smaller trees with the same accuracies. For this example, the tree weight, w, is set to its typically used value of 1000, which makes the positive reward of a smaller tree very small, resulting in the fitness curve and the accuracy curve appearing identical.

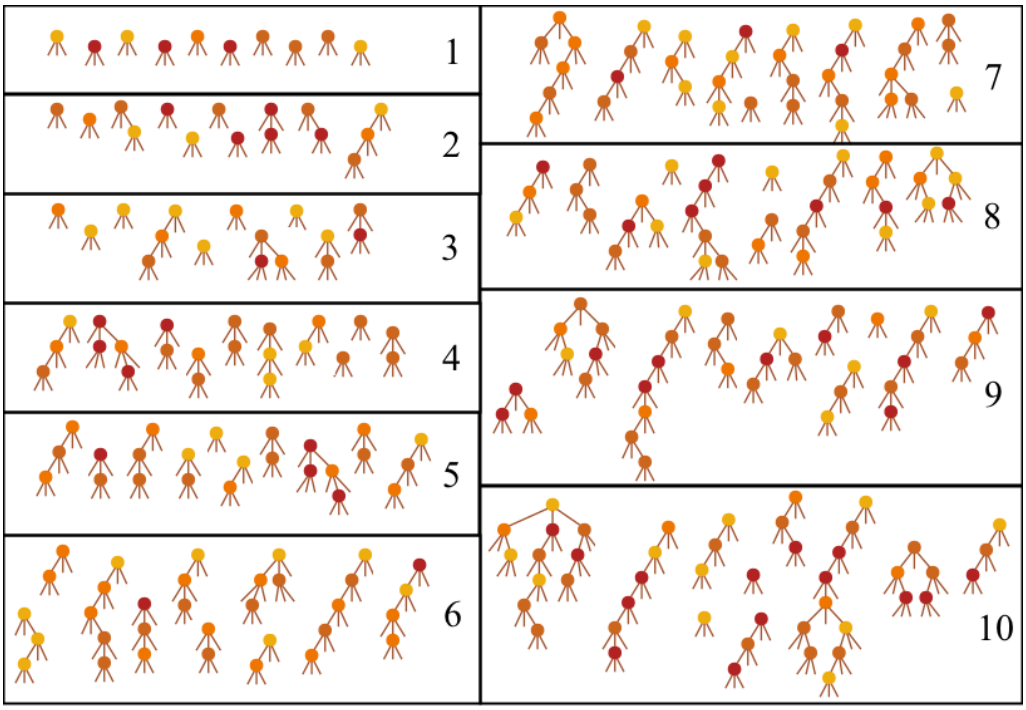
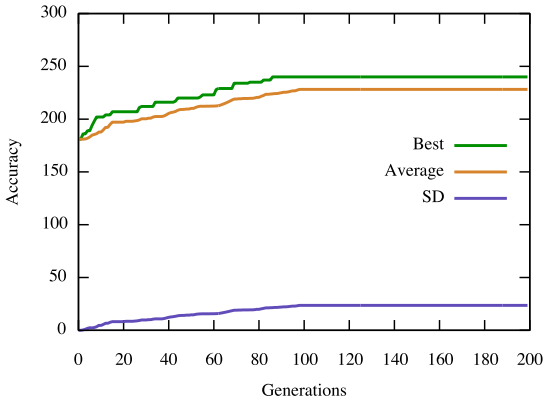


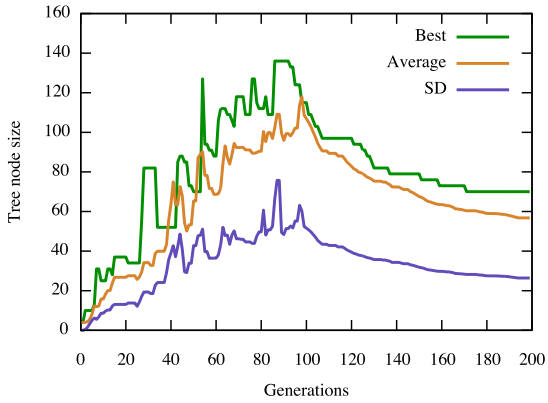
Figure 3.3: A real example of 10 EMTI evolution cycles, where each tree corresponds to an individual in a total population of 10.

3.1.7 Majority vote end node setting

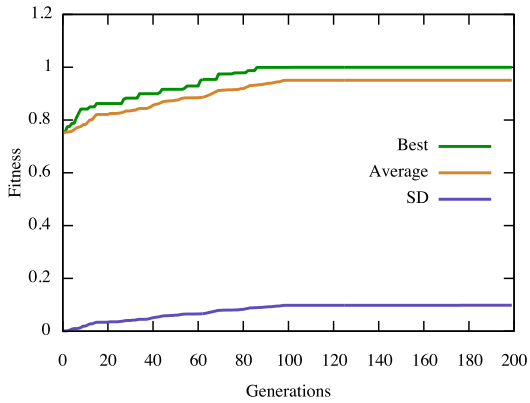
Initially during EMTI development, the end nodes values were set randomly and were a part of the evolution in the same way as the decision nodes. It was soon discovered that this was unnecessary, what we are really searching for is three structure, not which values to put in the tree. end nodes values (and decision node values, though doing this quickly becomes computationally expensive, the workload grows exponentially the further up you go in the tree) can be set when testing the tree on the training data by majority vote. This can be done in the



(a) Training set accuracy



(b) Tree size growth



(c) Total fitness

Figure 3.4: Plot of 200 generations of EMTI on the Diagonal problem.

fitness function, which already tests the tree on the training data. The way EMTI is implemented ensures that each data instance in the training set will arrive in an end node. This makes it possible to register the "traffic" of data instances that each end nodes receives, storing the instance's class label. When we finish to run through the training set, we look at each end nodes traffic information, and set the end node value as the category label that the majority of the instances ending in the node had (see figure 3.5). But what about end nodes that receive no traffic from the training set? The solution found was to keep these random and reset them after the evolution is over using k-Nearest Neighbor. But it is unclear if this is really necessary, as after many evolution cycles these end node values are not really random: they are set, through the various genetic operators, from the values of end nodes that did experience traffic. The effectiveness of the unvisited

end node scheme of using kNN on them is measured in experiment E2, section 4.4. Implementing majority vote end node has shown to dramatically the search space and run times of EMTI, this was confirmed in experiment E1 4.3.

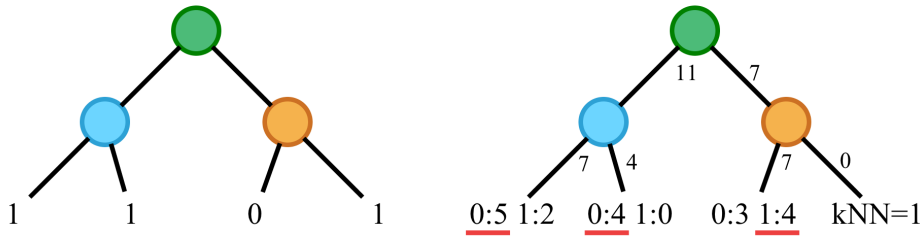


Figure 3.5: Random end node setting versus majority vote end node setting, using kNN for unvisited nodes.

3.2 Implementation details

EMTI was written in $C++11$ with no dependencies other than the $C++$ standard library. Figure 3.6 shows a UML class diagram of EMTI. The front end is responsible of supplying a URL of a properly formatted dataset to the *Data* class. While algorithms like C4.5 requires a separate *.names* file containing information about the attributes, the *Data* class used in EMTI parses the data into a vector of instances and incrementally stores information about the possible attributes it finds, though it will still need information about the attribute type names when this is required. EMTI data files should be in the format of one line for each instance with space as delimiter between each attribute and the class attribute in the end of each line. *Data* also provides several options for processing the data into this format, handling different delimiters, ignoring attributes and changing the location of the class attribute.

To initiate the EA, the front end then needs to initiate a *Population* instance with the vector, attribute information and the general EA parameters such as mutation rate, parent selection type etc. as parameters. This is where the *Population* class generates the initial population. *Population* provides a method called doEvolutionCycle() to the front end, which performs one evolution cycle. The front end can then do what it wants with this method, e.g. putting it in a loop with the desired end criteria.

By default, the Population class generates information about fitness, accuracy and tree sizes for each generation, storing it in a data file. This can later be used for analysis and plotting, as seen in figure 3.4, where we have used Gnuplot [12] for drawing the graphs. After evolution is done, there is the possibility of exporting the best tree to a .dot file format, which can be visualized using Graphviz [13], as seen in figure 3.8 and 3.9.

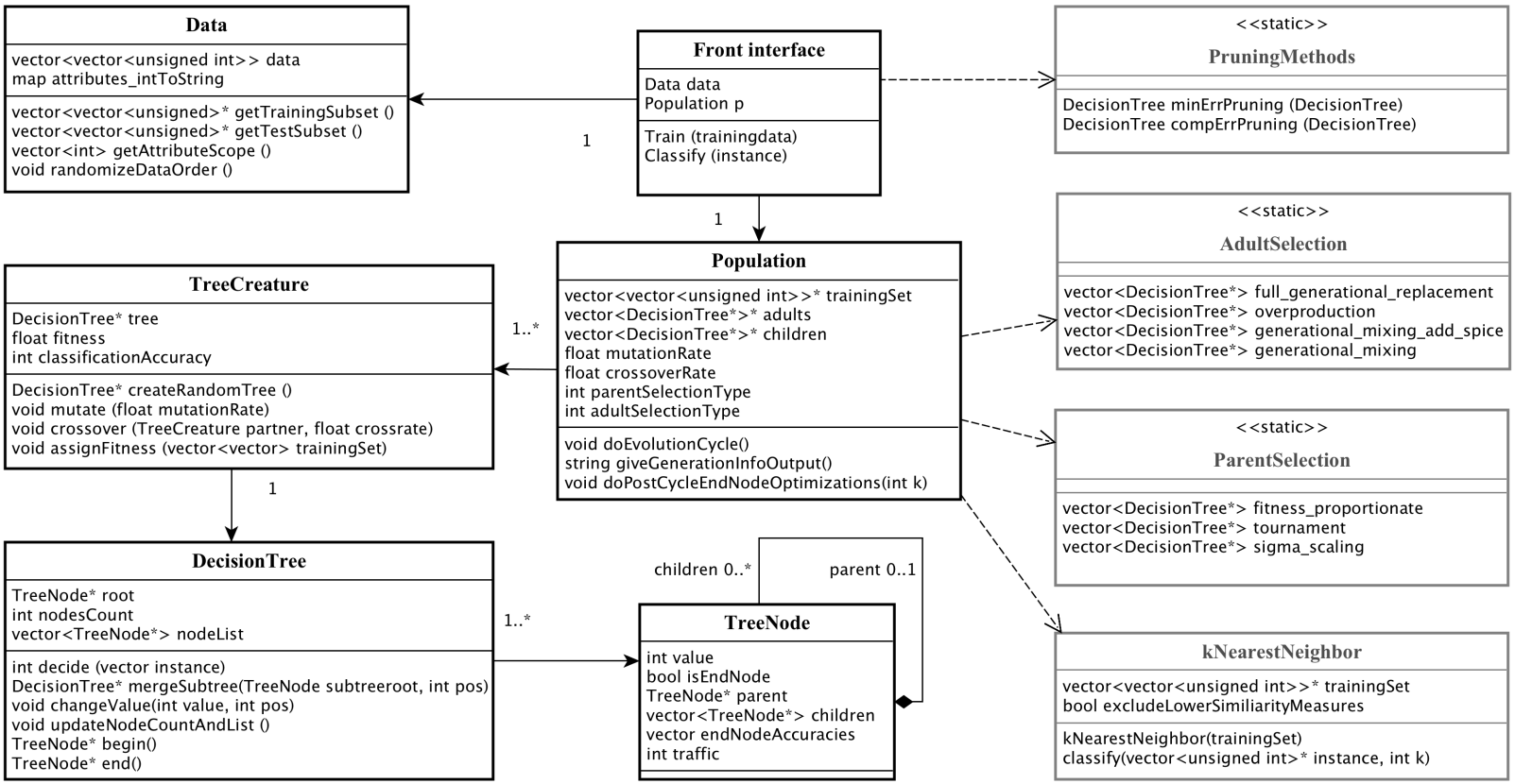


Figure 3.6: UML class diagram for the EMTI implementation.

Figure 3.7 shows the GUI for the EMTI experiments and testing grounds. It allows for real time adjustment of parameters, which data sets to train on and graph visualizations of the development of the algorithm run. In this screenshot, the top frame shows the accuracy of the current best individual on the training set (green) and the accuracy on the validation set (red) while the bottom frame shows the current best individual size (green) vs the average population size (orange). This is implemented in mac OS X using the Swift programming language, linking the C++ code via Objective-C.

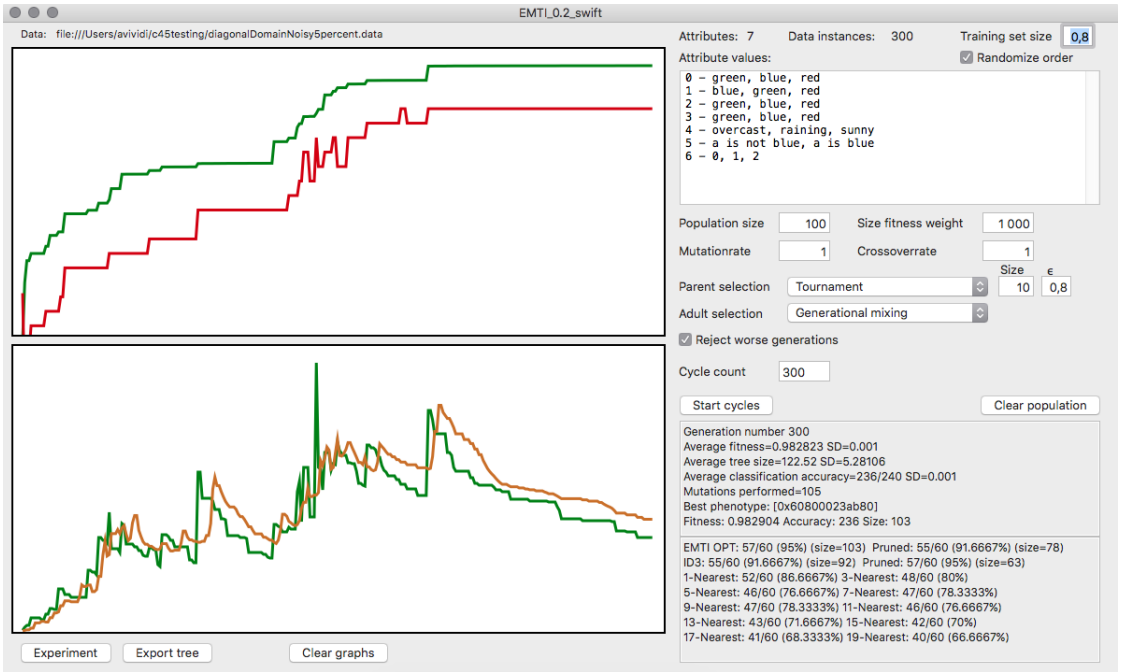


Figure 3.7: Mac OS X GUI front end screenshot

3.2.1 TrollBrain integration

A basic functioning version of EMTI was integrated with the TrollBrain interface. The main challenge was to reformat the data from TrollBrain that uses string, integer or real value types of its data into the data format supported by EMTI, were all attribute and class values are abstracted into integer values. A number of minor syntax issues also had to be resolved when the source code was migrated to the Microsoft Visual Studio $C++$ compiler. TrollBrain's AI-method component are compiled for Windows as dynamic linked libraries (dll's) that can be dynamically added to the TrollBrain interface.

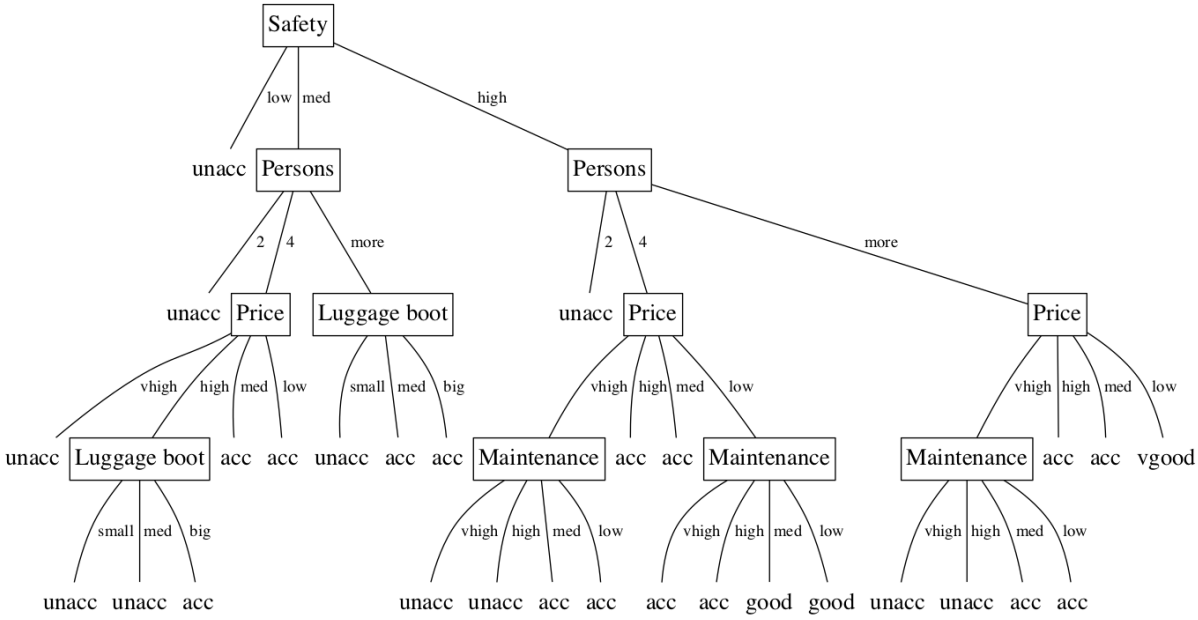
3.3 Explanation tool

Another focus, that comes naturally as it is one of the big advantages of decision trees, has been for EMTI to produce results that are easily understood by humans. By producing single parallel axis trees using no ensemble techniques or attribute mixing, the trees generated are in an easily interpretable form. An example of a tree generated by EMTI from a car evaluation domain can be seen in figure 3.8a. Here, the ability to set a maximum size of the trees that are generated is utilized to create a more comprehensive tree model. The tree in figure 3.8a has a maximum node count of 40 and achieves a 83.8% accuracy on a test set comprised of 30% of the car evaluation dataset (the other 70% was used for training.). In comparison, when running for a long time without a maximum tree size, EMTI generates a $\approx$ 200 node tree, reaching an accuracy of 93.5% (see table 4.14 and 4.15.). We can see that the decisions made by the tree (unacc = unacceptable, acc = acceptable, good = good, vgood = very good) seems in large logical and understandable. The trees shown in figure 3.8b and 3.8c shows the traffic and the accuracy when running the tree on the training set and the test set, respectively. Red nodes score less than 50% accuracy, with deeper red being worse, blue nodes score more than 50% accuracy, with deep blue being better.

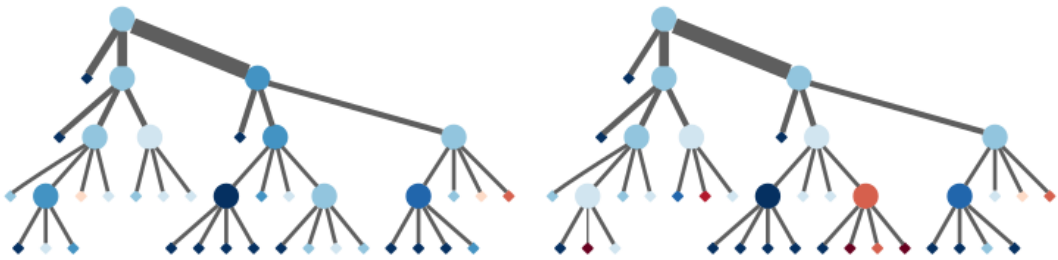
Another example is shown in figure 3.9. This tree shows what we believed to be the minimal possible decision tree for the Diagonal problem, as we have not been able to find a smaller combination of attributes that satisfies the problem. It has a total count of 67 nodes, which EMTI is normally able to find, depending on the size of the data set. Predictive algorithms like ID3 and C4.5 can usually, depending on the training set data combinations, only find larger sub-optimal solutions of this, even with all permutations of the problem available.

3.4 Generating initial population using ID3

With the objective of speeding up the algorithm, we tried another way to make the initial population, generating the initial trees by using the ID3 algorithm on random subsets of the training data, similar to the method used by Fu [9] where C4.5 was used. This makes the algorithm start up with much larger trees that already have high accuracy, jump-starting the evolutionary process. But the speed gain observed using this method were not very significant, because the normal method already has a rapid growth of tree size and accuracies in the initial cycles, quickly reaching the same size levels of the ID3 method. This method could perhaps be helpful when dealing with very large datasets, however. The main concern of using this method is that it potentially drives the evolutionary process towards a local minimum, restricting the search space to a size close to the local search space of ID3.



(a) Human readable decision tree



(b) Training data traffic tree

(c) Test data traffic tree

Figure 3.8: Decision tree for the car evaluation training set with maximum 40 nodes.

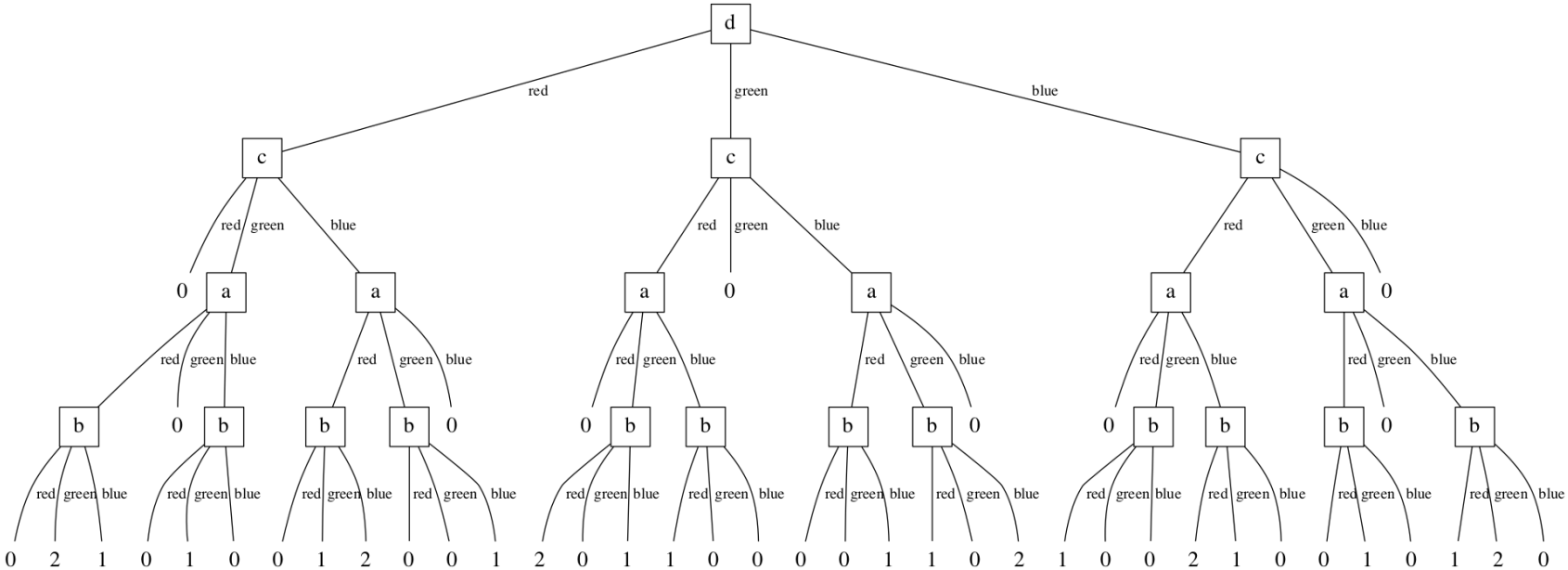


Figure 3.9: Minimal decision tree for the Diagonal problem.

3.5 Reinforcement learning

EMTI is made for use in supervised learning where the agent models its function based on labeled example data. But by modifying the way fitness values are assigned to the decision trees of the population, EMTI could also be used for reinforcement learning, where the agent models its function based on the positive or negative consequences of its actions. This was tried for a simple "robot" world, visualized in figure 3.10, consisting of a 10×10 grid of squares, that are either empty, contain food or contain poison. The robot has to navigate this world trying to eat food (positive reward) and avoiding eating poison (negative reward). Three sensory inputs are provided telling the robot what the squares to the left, front and right contain and three actions are possible, to go left, front or right. The objective is to make the robot learn the correct actions for the different possible inputs.

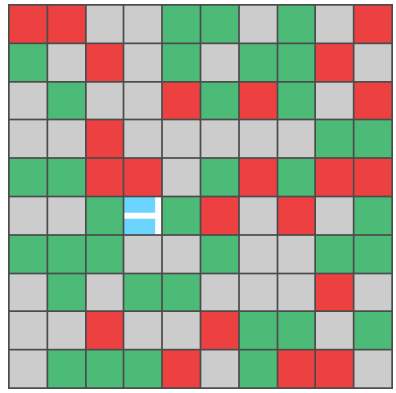


Figure 3.10: Visualization of the robot world, the blue and white square indicates the position and direction of the robot, grey indicates empty squares, green indicates squares containing food and red indicates squares containing poison.

This is possible to do in EMTI by making a decision tree serve as the brain of the robot. The initial population is set up normally, with trees containing one random decision node of one of the inputs and three leaf nodes, one for each possible action. The fitness score of a tree is then calculated by making the robot use the tree to decide what to do, doing 50 actions in the world. This is done for each tree in the population resetting the world every time between each tree. Then the normal evolutionary operations, mutation, crossover, adult selection and parent selection are performed, constituting a full evolution cycle. In this way evolutionary search is made possible, resulting in stronger and stronger scoring decision trees as evolution progresses. Because of the lack of training data, majority vote end node setting is impossible, making the evolutionary progress significantly slower. Figure 3.11 shows a decision tree learned by EMTI using this method. The decision tree lacks some expressive powers; if it finds poison to the front and to the right it will always go left, regardless of what the square left of the robot contains, if all squares around

the robot are empty, it will always go straight. An improvement could be to let a fourth action be allowed, making the robot go in a random direction, which it would learn to do when all options are equal.

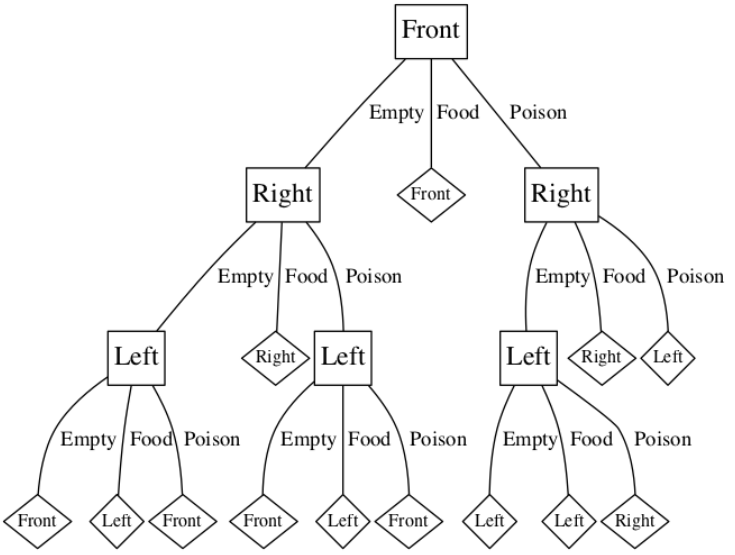


Figure 3.11: Decision tree for a robot brain. Square boxes are the inputs with branches for each different types of input, while diamond boxes are the resulting actions.

3.6 Avoiding overfitting and other improvements

This section identifies three main weaknesses of EMTI, and proposes ways to improve upon them. Overfitting is seen as the most important weakness, and five different methods to avoid overfitting is proposed which are later tested in chapter 4. Three involves early stopping techniques; the single look back method and the size control method and a method combining the two, while the two other are tree pruning methods; Error-complexity pruning and Minimum-error pruning.

3.6.1 EMTI weaknesses

At the start of the project, the potential flaws and weaknesses of the algorithm were tried identified, mainly in comparison to other conventional decision tree learners. Three main weaknesses were found:

1. *Long run times*: Compared to typical greedy approach of decision tree learners, EMTI requires much more calculation and hence is slower.

2. *Many parameters*: In its current form, EMTI has many parameters (mutation rate, population size, cycle count etc.) that the user must set and learn to use.

3. *Overfitting*: In its current form, EMTI tries to model the training data perfectly, unless the tree size weight parameter is set very high. This will cause overfitting on many data sets.

It is recognized that evolving decision trees will be fundamentally slower than greedy approaches, as an EA's computational complexity is somewhere between the complexity of a greedy search and a brute force search. As a dataset grows in size in terms of both instances and attributes, the speed difference grows larger. EMTI is therefore not well suited for very large data sets. The biggest potential for decreasing EMTI run times would be to allow parallelization of the many independent tasks that make up the algorithm, such as the crossover operator, the mutation operator and the fitness setting. There is also potential for optimizing the existing code, especially looking into ways to optimize how the tree structures are handled.

Typically for EA's, EMTI has a wide range of parameters that can be set: population size, cycle count, mutation rate, crossover rate, parent selection type, adult selection type, reject worse generation and the tree size weight parameter. This could potentially be a daunting task for an eventual user of the system. The issue of controlling values of various parameters of an evolutionary algorithm is one of the most important and promising areas of research in evolutionary computation (Eiben [6]). However, in qualitative testing of EMTI has shown very little need for setting different parameters for different domain problems and shows to work well setting the parameters more towards a greedy search mode, with low mutation rate (≈ 1.0) and low population sizes (≈ 100), retaining and improving the already established solutions by using generational mixing adult selection. The chance of falling into a local optima are small and happens close to the optimal solution. Just as greedy searching has been shown to work well in traditional tree learners, it works well in evolutionary tree learning. Therefore EMTI can work well with a standard set of parameters as proposed in table 3.1, where only the amount of evolution cycles will differ depending the data set.

Using this preset parameters, a method called *autoEMTI* was created, that takes a training set as input, runs the evolutionary algorithm and returns the final decision tree, without any parameter needed to be set by the user. In stead of using a preset cycle count value, the method stops when the best fitness of the population has been the same $n = 100$ times in a row, indicating that the evolution has converged to the best solution it can find. Qualitative test has shown that it achieves the same results as when externally setting the parameters, though it often uses unnecessary many cycles.

By using a multi-objective fitness function, EMTI show resilience to overfitting when the tree size weight is correctly adjusted. In table 3.1, the tree size weight parameter is set high, such that the algorithm will prefer an improvement in clas-

Table 3.1: EMTI standard parameters

Population	Mutation rate	Crossover rate	Parent selection	Adult selection	Reject worse generation	Tree size weight
100	1.0	1.0	Sigma scaling	Gen. mixing	Yes	1000

sification accuracy, even at the cost of a much larger decision tree. This parameter could be tuned and used as an instrument for avoiding overfitting, but this is very hard to do without resolving to a trial and error approach, as setting it too low will cause significant underfitting, i.e. very simple solutions that don't describe the domain correctly. Figure 3.12 shows how overfitting will occur on certain data sets. The graphs on the figure shows EMTI being run on the Wisconsin Breast Cancer Data set, using 70% of the instances as training set and the 30% remaining as a test set. The top frame shows training set accuracy of the best individual of the population (red) relative to the test set accuracy (blue) with the y axis covering roughly between 90% and 100% classification accuracy. The bottom frame shows the relation between the tree size of the best individual (red) and the average tree size of the population (green).

As the evolutionary algorithm progresses, we see a steady improvement of the training set accuracy, together with a steady increase of the tree sizes (and therefore complexity of the solution) of the population, until the training set reaches a close-to maximum accuracy. At this point the only way for the algorithm to increase fitness values is by finding smaller, equivalent accuracy decision trees, as shown in the decreasing tree sizes at the end of the bottom frame. But looking at the classification accuracy scores for the test set, we see a very different development. It increases quickly in correlation with the training set, but soon diverges and starts decreasing as the training set accuracy increases. A clear sign of overfitting: the algorithm introduces rules into the decision trees based on noisy and extraordinary instances of the data set, the generalization error increases. The tendency to overfit is a critical weakness in the algorithm, and was therefore chosen as the most important weakness to improve upon. We look into two commonly used methods for avoiding overfitting, the more general approach of early stopping, that can be used by a variety of learning algorithms, and decision tree pruning, commonly used by tree learning algorithms.

3.6.2 Early stopping

Looking at figure 3.12, we can see that if the algorithm could be stopped at (or brought back to) the point when the test set accuracy was the highest, it should yield a better general solution to the problem. This is what is typically done in an already well established technique within machine learning called *Early stopping*, also often referred to as pre-pruning.

The procedure is relatively simple, making a portion (typically 30%) of the train-

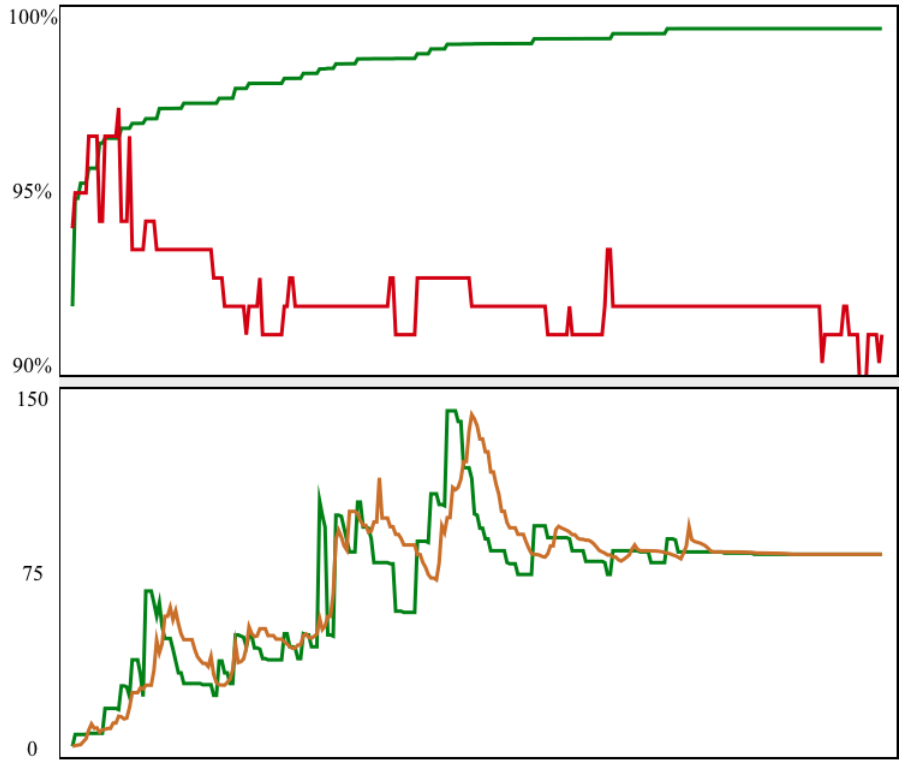


Figure 3.12: Example of overfitting when running on the Wisconsin Breast Cancer data set. The top frame shows training set accuracy of the best individual in the population (green), relative to the validation set accuracy (red). The bottom frame shows the relation between the tree size of the best individual (green) and the average tree size of the population (orange).

ing dataset into a validation set and stopping at a point when the generalization error increases, which happens when the classification accuracy of the validation set starts worsening. The main challenge is knowing when to stop, the stopping criterion. Prechelt [25] shows how learning rate curves in general are not smooth and have high variation, so generalization error can still go down after it increased. Prechelt proposes a function that chooses a stopping criteria with a trade-off between training time and generalization. For simplicity, minimizing training time was seen as a low priority for this project, focusing on maximizing validation set accuracy. Two candidate methods were implemented for testing:

Single look-back method

This simple method does in reality not doing any early stopping at all, but the principle behind it remains the same. Using a validation set during training, the

best tree of generation with the so far lowest generalization error is retained until the end of the evolution (typically when a certain amount of cycles has been reached). This tree is then chosen as the final tree.

Size control method

A decision tree retrieved from the middle of an evolution is likely to have room for optimizations and can contain duplicate attributes in the same branches of the tree. This method seeks to continue evolution after such a tree is retrieved, without increasing the generalization error. It does this by setting a limit at how big the tree can grow. When overfitting happens in decision tree learners, it is in relation to the tree learner making overly large and complex trees. Utilizing EMTI's ability of setting a maximum size of the trees it generates, a stopping criterion is set, saying that when the generalization error has grown for n cycles, instead of stopping the algorithm, the maximum size of the tree is set to the size of the best tree of the generation that had the lowest generalization error, continuing the algorithm until completion, possibly setting lower limits to the tree size, but never higher.

It is important to be careful when using the validation set multiple times during training, as you risk that it becomes biased in the same way as the training set. For example, a multiple look back method was proposed, where after n cycles of higher generalization errors, the evolution was set back to the lowest point, and the evolution would continue from there. This method would slowly create trees that fit the training set and the validation set well, but the actual generalization error could still be high. But simply limiting the tree size as in the size control method will not create any bias, because it does not directly influence the classification accuracy. Finally, a method combining single look-back and the size control method will also be tested.

3.6.3 Decision tree pruning

Decision tree pruning is a way to simplify decision trees in order to achieve two main objectives: making the tree easier to understand and improving the accuracy of overfitted trees. The process involves finding unnecessary parts of the tree and removing them, using a certain criterion for deciding what makes the parts unnecessary. A lot of research has been done in decision tree pruning starting from the 1980's, often employing more and more complex statistical methods for deciding which nodes to prune. As the main interest is finding out whether pruning is applicable for EMTI, some of the earlier, simpler methods have been implemented. In "An Empirical Comparison of Pruning Methods for Decision Tree Induction"[21] by John Mingers, 1986, a comparison of five pruning methods is made, Error-complexity pruning, (Breiman [5]), Critical value pruning (Mingers [22]), Minimum-error pruning (Niblett and Bratko [23], Reduced error pruning and Pessimistic error pruning (both by Quinlan [28]). Two of these methods were chosen for testing with EMTI, Error-complexity Pruning because of its strong results

in Minger's paper and Minimum-error pruning because of the different approach it uses and that it is not reliant on a validation set.

Error-complexity pruning (Err-comp)

A two stage method that starts by generating a series of trees, each more and more pruned until we are left with the root node of the tree. It does this by pruning one subtree at at time choosing a node that has the subtree with the smallest error-complexity measure α, given by the equation

$$\alpha = \frac{R(t) - R(T_t}{N_T - 1} \tag{3.2}$$

$R(t)$ is the error cost of the node, given as

$$R(t) = r(t)\, p(t) \tag{3.3}$$

where $r(t)$ is the error rate of the node given by the training data and $p(t)$ is the proportion of training data going through compared to the total training data. $R(T_t)$ is the error rate of the subtree associated with the node and is the sum of the error rate of all the leaves of the node. N_T is how many leaves the node has.

After the series of trees is generated, the tree with the smallest misclassification rate using an independent validation set is chosen as the final pruned tree.

Minimum-error pruning (Min-err)

This method uses Laplace probability estimates to find the theoretical minimum expected error rate when classifying independent data, with the assumption that that all classes are equally likely. The expected error rate E_k of tree node is given by the equation

$$E_k = \frac{n - n_c + k - 1}{n + k} \tag{3.4}$$

Using a data set with k classes, were n instances have been observed in the nodes of which n_c where in class c. At each decision node in the tree, the expected error rate is calculated both for if its subtree is pruned or if it is not pruned, using the error rates of each of its branches, combined by weighting according to the proportion of instances along each branch. If pruning the node leads to a smaller expected error rate, the node is pruned.

CHAPTER 4

Results

This chapter describes the execution, including the conditions they were performed in and the parameter settings that were used, and presents the results of 3 experiments, **E1**, **E2** and **E3**, set out in order to answer the research question stated in section 1.1.

4.1 Experimental plan

Three different main experiments were done, labeled **E1**, **E2** and **E3**.

E1 attempted to measure the performance of the inner components of the EMTI algorithm, in relation to **RQ1**.

E2 measured the performance of different methods of avoiding overfitting in EMTI, mainly in relation to **RQ2** and **RQ3** and to some extent **RQ1**.

E3 measured the performance of EMTI against four other known classification algorithms on a variety of datasets, in relation to **RQ4** and **RQ5**.

Experiments were conducted by running different algorithms on various datasets, listed in table 4.1, measuring statistics such as training and test classification accuracies, tree sizes and run times. An evolutionary algorithm is non-deterministic and results can have high variance between each run, therefore the algorithms are run many times on the same dataset to increase the accuracy of the results. Because the data amount for the datasets are limited, using the same random training and test set every time can cause particularly skewed results for certain combinations of the data. Except for on the datasets that have explicit, designed test sets (the monk, corral and mofn 3-7-10 datasets), k-fold cross-validation is used to combat this problem. This is done by splitting the datasets into k random parts (folds) were one part is chosen as the test set and the rest are chosen to make up the training set. This is repeated so that each part serves as test set one time. For all experiments where cross-validation was used in this project, 10-fold cross-validation

was used. This means that the evolutionary algorithm has to run 10 times for each datasets, making run times considerably (10 times) longer. For each run, the random seed is generated based on the current time of the experiment. Setting a static seed based on a constant was considered, as this should allow the experiments to be perfectly recreated, but was discarded since there was uncertainty whether or not this could put any bias into the results.

4.2 Description of the datasets used

Table 4.1 show the characteristics of the datasets used in the three experiments. They were all retrieved from the UCI machine learning repository [35], a collection of benchmark datasets for classification and regression, with the exception of the Corral and the Mofn 3-7-10 datasets by Ronny Kohavi, found at sgi.com [33].

Table 4.1: Dataset characteristics.

Dataset	Type	Attributes	Categories	Instances	Artificial	Year
Diagonal problem	Categorical	4	3	300	yes	2015
Car evaluation	Categorical	6	4	1728	yes	1997
W. Breast cancer	Integer	10	2	699	no	1992
Zoo	Mixed	17	7	101	yes	1990
Balance scale	Categorical	4	3	625	yes	1994
House votes '84	Categorical	16	2	435	no	1987
German Credit Data	Categorical	20	2	1000	no	1994
Lymphography	Categorical	18	4	148	no	1988
Monk's problems	Categorical	6	2	432	yes	1991
Corral	Categorical	6	2	128	yes	1994
Mofn 3-7-10	Categorical	10	2	1024	yes	1994

Diagonal domain

Artificial domain created for the testing of EMTI, see section 3.1.5.

Car evaluation dataset

The Car evaluation dataset evaluates whether a car is a good buy based on buying price, maintenance price, number of doors, number of people it can carry, size of luggage boot, and safety, giving the cars rating as either unacceptable, acceptable, good or very good.

Wisconsin Breast cancer dataset

The Wisconsin Breast cancer dataset uses a number of numerical attributes to determine whether a patient has benign or malignant cancer. Though it is numerical, each attribute value falls between 0 and 10, making it possible to use with categorical-only classifiers. The dataset include 16 instances with missing values in this set.

Zoo dataset

A simple dataset for classifying animals containing boolean values for properties such as *feathers*, *predator*, *fins* etc. and the numerical value *legs* (*0*, *2*, *4*, *6*, *8*). Based on this information they are classified into one of 7 numerical types. The meaning of the different types are unknown.

Balance scale dataset

This dataset was apparently generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The attributes are the left weight, the left distance, the right weight, and the right distance.

1984 United States Congressional Voting Records (House votes '84)

Is the congressman a democrat or a republican? This dataset includes *Yes* or *No* votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the Congressional Quarterly Almanac, including questions like *immigration*, *crime*, and *duty-free-exports* etc.

German Credit Data

In the German Credit dataset the objective is to decide if a person represents a *good* or *bad* credit risk based on a number of numerical and categorical attributes such as credit history, checkin account balance, employment status, age etc. The classification should be considered a cost matrix, where it more costly to declare a person good when they are bad than to declare a person bad when they are good, but this was ignored in the experiments for the sake of simplicity. Since EMTI normally does not handle numerical and real values, the dataset was discretized using Weka's [36] discretizing tool.

Lymphography

Dataset within a medical domain obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia.

Monk's problems

The Monk's problems [32] (Thun et al.) datasets are a set of 3 artificial classification problems generated for the specific use of comparing different learning algorithms. A robot is described using six attributes with a total of 432 possible permutations:

x_1 :	*head_shape*	$\in$	round, square, octagon
x_2 :	*body_shape*	$\in$	round, square, octagon
x_3 :	*is_smiling*	$\in$	yes, no
x_4 :	*holding*	$\in$	sword, ballon, flag
x_5 :	*jacket_color*	$\in$	red, yellow, green, blue
x_6 :	*has_tie*	$\in$	yes, no

While the class is binary and is described by a logical sentence. The first problem uses a training set of 124 instances randomly chosen from the possible permutations and is expressed as: **(head_shape = body_shape) or (jacket_color = red)**. It is in standard conjunctive form, which should be easily solved and expressed by decision tree learning algorithms. The second problem has a training set of 169 randomly chosen instances and is expressed as: **exactly two of the six attributes have their *first* value**. This problem is complicated to express in standard conjunctive form and requires very large decision trees to express perfectly. The third problem has a training set of 122 randomly chosen instances and is expressed as: **(jacket_color = green and holding = sword) or (not jacket_color = blue and not body_shape = octagon)**. This is again in standard conjunctive form, but this time 5% misclassification, i.e. noise was added to the training set.

Corral dataset

The correlated attribute (Corral) dataset is an artificial dataset by Ronny Kohavi, first used in "Irrelevant features and the subset selection problem", John, Kohavi and Pfleger [15]. It has 6 binary attributes and a binary class. The first four attributes determine the class by the logical sentence $(A0 \wedge A1) \vee (B0 \wedge B1)$, while the 5th attribute is irrelevant and the 6th attribute is correlated with the class value, but with 25% noise. The dataset comes with a distinct training and test set.

Mofn 3-7-10 dataset

This is another artificial dataset by Ronny Kohavi with 10 attributes 3 of which are irrelevant. The dataset comes with a distinct training and test set.

4.3 E1 Method components effectiveness study

The objective of this experiment was to measure the effectiveness of the different method components that drives the evolutionary search forwards in EMTI. This was done by individually deactivating the components and measure the performance without them against the performance of EMTI with all components activated. The three major contributors are the crossover operator, mutation operator and the majority vote end node setting.

4.3.1 E1.1 Majority end node setting effectiveness

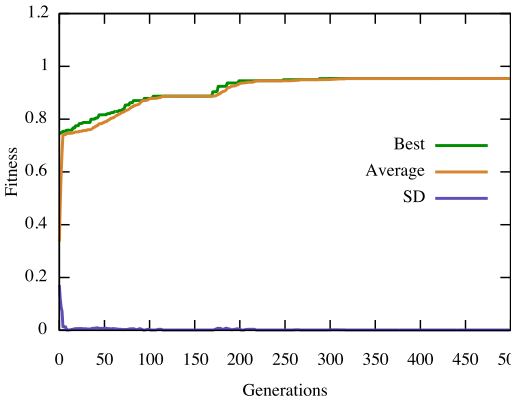
Changing the algorithm from random end node setting to majority vote end node setting had already been observed to greatly increase the search effectiveness but had not been quantitatively measured. Two algorithm runs were made on the diagonal domain dataset using the settings described in table 4.2, one using the normal version and one where the algorithm had been reverted to using random end node setting. Figure 4.1 shows fitness (learning curves) and tree size development over time for the two runs.

Table 4.2: E1.1 settings

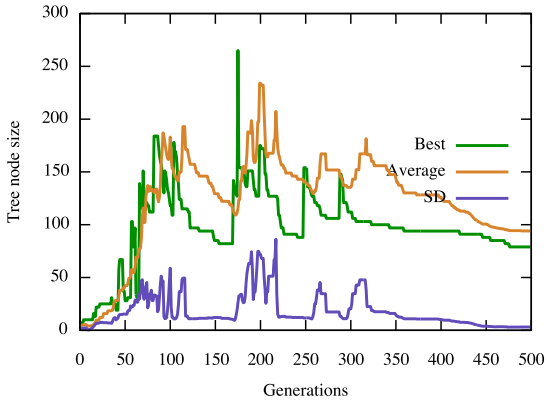
Dataset	Cycles	Population	Mutation rate	Crossover rate	Parent selection	Adult selection
Diagonal	500	500	2.0	1.0	Sigma scaling	Gen. mixing

4.3.2 E1.2 Crossover and mutation components effectiveness

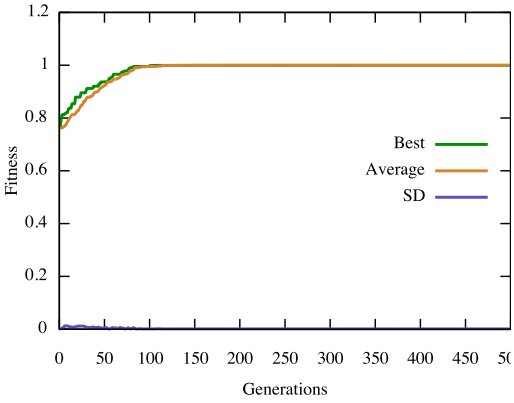
Similarly to experiment 1.1, the crossover operator and the 4 different types of mutation operators (cut, implode, deplode and flip) were individually deactivated and the performance compared was compared with EMTI with all components activated. In an effort to measure search effectiveness, the experiment measured how many generation cycles it took to reach a certain training accuracy for two datasets, the diagonal domain and the Wisconsin breast cancer dataset. For the diagonal domain, the target accuracy was *training set size*-2, for the cancer set, the target accuracy was *training set size*-9. To ensure accurate results, the experiment



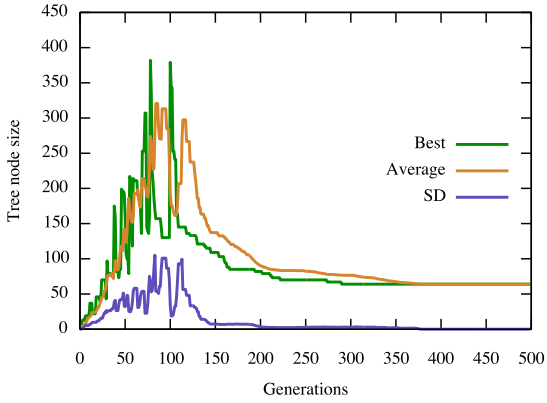
(a) Training set fitness, random end node.



(b) Tree size growth, random end node.



(c) Training set, fitness majority vote.



(d) Tree size growth, majority vote.

Figure 4.1: E1.1: Random end node setting vs. majority vote end node setting.

was repeated 50 times for each disabled method for each dataset. The results can be seen in table 4.3. Disabling certain methods causes EMTI to strongly slow down or not reach the desired accuracy at all. For this reason a cut-off point of 1000 cycles was set, this is reflected in the values for no crossover and no deplode in the table, which would be higher without this cut-off point.

Table 4.3: E1.2: EMTI crossover and mutation components effectiveness

	Diagonal Domain		Breast cancer	
Change to EMTI	Mean cycle count	SD	Mean cycle count	SD
Normal	118.05	35.59	106.69	25.59
No crossover	1000.00	0.00	1000.00	0.00
No branch cut	120.07	40.23	104.67	24.19
No implode	120.77	41.68	112.46	24.27
No deplode	796.17	373.38	108.62	23.75
No flipping	110.44	26.73	117.87	27.72

4.4 E2 Early stopping and tree pruning methods comparisons

This experiment measures the differences between three different forms of early stopping (Single look-back, Size control and Size control with single look-back) and pruning (Error complexity pruning and Minimum error pruning) for EMTI as described in section 3.6. For comparison, the normal version of EMTI is included and a normal EMTI version using the same reduced size training set as in the early stopping versions and the complexity pruning version, to measure the potential drawback of using parts of the training set as a validation set. Finally, also added is a measure of EMTI's majority end node optimization method for unvisited nodes (see section 3.1.7) by showing classification accuracies before and after optimizations. The methods are labeled in the tables accordingly:

Basic pre opt: The normal EMTI algorithm without any unvisited end node optimization.

Basic: The normal EMTI algorithm.

Min-err pruning: The normal EMTI algorithm using minimum error pruning on the best tree after evolution.

Basic small set: The normal EMTI algorithm, using the same size training set as the methods below that require a validation set.

Err-comp pruning: The normal EMTI algorithm using complexity-error pruning on the best tree after evolution.

Single look-back: EMTI using the single look-back method.

Size control: EMTI using the size control method.

Size control look-back: EMTI using the size control method combined with the single look back method.

Three evolutions were done for each experiment, as combining several of the methods in the same evolution run is possible, one combining Basic pre opt, Basic and Min-err pruning, one combining Basic small set, Err-comp pruning and Single-look back and one combining Size control and Size control look-back. The experiments were conducted on 6 different datasets with settings as described in table 4.4. For all the experiments 10-fold cross-validation was used, with mutation rate and crossover rate always set to 1.0 and using sigma scaling for parent selection and generational mixing for adult selection. To ensure accurate results, the cross-validation was repeated 10 times for the four first smaller Datasets, and 5 times for the last two bigger datasets. For all experiments, the validation sets used were 25% percent of the training set chosen randomly, and the majority end node optimization used $k = 3$.

Table 4.4: Overview of E2 settings.

#	Folds	Times	dataset	Pop. size	Generations
2.1	10	10	Diagonal domain 0% noise	100	300
2.2	10	10	Diagonal domain 20% noise	100	300
2.3	10	10	W. B. Cancer	100	300
2.4	10	10	House Votes '84	100	300
2.5	10	5	German Credit Data	50	1000
2.6	10	5	Car Evaluation	50	1000

4.4.1 E2.1 Diagonal domain 0% noise

With no noise in the dataset and a clear underlying structure, neither of the methods for avoiding overfitting should have a positive effect on the classification accuracy. The results are shown in table 4.5.

Table 4.5: E2.1 Diagonal domain 0% noise results. Classification accuracies, standard deviations and training accuracy shown as percentage, run time shown as seconds.

EMTI type	Mean accuracy	SD	Mean size	Run time	Training accuracy
Basic pre opt.	99.23	1.88	67.66	392.00	100.00
Basic	99.53	1.63	"	"	"
Min-err pruning	96.36	4.39	58.30	"	98.42
Basic small set	97.60	3.68	66.58	438.64	99.99
Err-comp pruning	96.33	5.26	64.60	"	99.56
Single look-back	97.76	3.97	67.21	"	99.99
Size control	97.63	3.41	64.99	438.44	99.87
Size control look-back	97.83	3.38	66.07	"	99.87

4.4.2 E2.2 Diagonal domain 20% noise

Adding significant noise in the dataset, by adding one irrelevant and one co-related attribute and 20% misclassifications (appearing both in the training sets and test sets), it was expected that methods for avoiding overfitting could improve upon the classification accuracy. The results are shown in table 4.6.

Table 4.6: E2.2 Diagonal domain 20% noise results. Classification accuracies, standard deviations and training accuracy shown as percentage, run time shown as seconds.

EMTI type	Mean accuracy	SD	Mean size	Run time	Training accuracy
Basic pre opt.	75.70	7.06	142.48	742.02	90.54
Basic	76.20	7.27	”	”	”
Min-err pruning	76.60	7.70	86.81	”	86.45
Basic small set	74.53	7.75	128.78	645.72	91.86
Err-comp pruning	71.70	8.82	25.16	”	75.56
Single look-back	73.63	8.33	120.04	”	86.47
Size control	72.67	7.79	87.40	453.52	88.99
Size control look-back	72.17	7.74	95.61	”	85.22

4.4.3 E2.3 Wisconsin Breast Cancer

This dataset taken from real world medical data has previously shown to cause some overfitting in EMTI, so it was expected that the methods for avoiding overfitting would improve the classification accuracy. The results are shown in table 4.7.

Table 4.7: E2.3 Wisconsin Breast Cancer results. Classification accuracies, standard deviations and training accuracy shown as percentage, run time shown as seconds.

EMTI type	Mean accuracy	SD	Mean size	Run time	Training accuracy
Basic pre opt.	92.59	2.92	179.23	777.05	99.58
Basic	93.51	2.72	”	”	”
Min-err pruning	94.38	2.67	96.35	”	98.04
Basic small set	93.33	2.78	140.85	624.76	99.77
Err-comp pruning	93.16	3.27	32.16	”	95.42
Single look-back	93.79	2.65	90.91	”	97.78
Size control	93.52	2.87	95.16	530.09	98.68
Size control look-back	93.63	3.29	85.59	”	97.87

4.4.4 E2.4 House Votes '84

Another real world dataset were EMTI has shown signs of overfitting, it was expected that methods for avoiding overfitting would improve results. The results are shown in table 4.8.

Table 4.8: E2.4 House Votes '84 results. Classification accuracies, standard deviations and training accuracy shown as percentage, run time shown as seconds.

EMTI type	Mean accuracy	SD	Mean size	Run time	Training accuracy
Basic pre opt.	94.99	3.54	40.45	431.84	98.95
Basic	95.03	3.63	”	”	”
Min-err pruning	95.33	3.60	28.78	”	98.40
Basic small set	94.74	3.12	35.11	348.61	98.96
Err-comp pruning	95.66	2.89	7.27	”	95.62
Single look-back	95.38	2.91	28.60	”	97.87
Size control	94.90	3.46	26.86	270.26	98.61
Size control look-back	95.61	3.29	24.52	”	97.94

4.4.5 E2.5 German Credit Data

This is a dataset with many attributes and complex relations (i.e. hard for decision trees to express) that EMTI has shown to perform poorly on. It was unclear if methods for avoiding overfitting would improve results. The results are shown in table 4.9.

Table 4.9: E2.5 German Credit Data results. Classification accuracies, standard deviations and training accuracy shown as percentage, run time shown as seconds.

EMTI type	Mean accuracy	SD	Mean size	Run time	Training accuracy
Basic pre opt.	67.86	4.75	878.50	2290.08	93.14
Basic	68.22	4.65	”	”	”
Min-err pruning	69.76	4.77	611.40	”	90.62
Basic small set	67.66	4.59	680.82	1940.07	95.26
Err-comp pruning	69.64	4.12	29.60	”	72.73
Single look-back	69.94	4.14	207.26	”	80.26
Size control	70.20	4.49	99.04	626.84	82.06
Size control look-back	70.60	4.50	89.60	”	78.12

4.4.6 E2.6 Car Evaluation

An artificial dataset with a complex relations between the attributes, but a clear underlying structure and no noise. It was not expected that methods for avoiding overfitting would improve classification accuracies. The results are shown in table 4.10.

Table 4.10: E2.6 Car Evaluation results. Classification accuracies, standard deviations and training accuracy shown as percentage, run time shown as seconds.

EMTI type	Mean accuracy	SD	Mean size	Run time	Training accuracy
Basic pre opt.	91.47	2.25	545.50	2583.71	99.15
Basic	92.05	1.89	”	”	”
Min-err pruning	90.20	2.50	230.68	”	93.93
Basic small set	91.82	2.26	424.14	2144.33	99.49
Err-comp pruning	90.54	2.86	225.40	”	96.46
Single look-back	91.55	2.49	446.68	”	98.48
Size control	90.82	2.38	209.26	1338.30	96.58
Size control look-back	91.04	2.49	225.92	”	96.28

4.4.7 E2 Summary

Table 4.11 and 4.12 shows a summary of the results of the different methods compared to basic EMTI in terms of tree sizes and classification accuracies respectively.

Table 4.11: E2 size summary in terms of node count.

Dataset	Basic	Min-err pruning	Err-comp pruning	Single look-back	Size control	Size control look-back
Diagonal 0	67.66	58.30	64.60	67.21	64.99	66.07
Diagonal 20	142.48	86.81	25.16	120.04	87.40	95.61
WB Cancer	179.23	96.35	32.16	90.91	95.16	85.59
Votes	40.45	28.78	7.27	28.60	26.86	24.52
Credit	878.50	611.40	29.60	207.26	99.05	89.60
Car eval.	545.50	230.68	225.40	446.68	209.26	225.92
Mean	308.97	185.39	64.03	160.12	97.12	97.89

Table 4.12: E2 accuracy summary (%).

Dataset	Basic	Min-err pruning	Err-comp pruning	Single look-back	Size control	Size control look-back
Diagonal 0	99.53	96.36	96.33	97.76	97.63	97.83
Diagonal 20	76.20	76.60	71.70	73.63	72.67	72.17
WB Cancer	93.51	94.38	93.16	93.79	93.52	93.63
Votes	95.03	95.33	95.66	95.38	94.90	95.61
Credit	68.22	69.76	69.64	69.94	70.20	70.60
Car eval.	92.05	90.20	90.54	91.55	90.82	91.04
Mean	87.42	87.10	86.17	87.01	86.62	86.81

4.5 E3 Comparisons with other learning methods

In this experiment Basic EMTI and EMTI with minimum-error pruning was tested on a variety of datasets and compared with three well known tree decision learning algorithms, ID3, C4.5 and Random Forest and one neural network classifier, the Multilayer Perceptron (MLP). A brief explanation of these algorithms can be found in chaper 2.1.3. The external algorithm testing was done using the Weka [36] machine learning software while EMTI was tested in the EMTI OSX environment using the exact same datasets. To ensure there were no significant differences between the software, for example differences created by the way data is read or in the way cross-validation is handled, a local implementation of ID3 in the EMTI OSX environment was also added. The Min-err method for decision tree pruning was added to the experiment on the basis of its results in experiment E2, as discussed in section 5.1.2.

Two statistics are measured in the experiment, tree sizes (only applicable for EMTI, ID3 and C4.5), shown in table 4.14, and classification accuracies, shown in table 4.14. The Weka ID3 algorithm does not give a size measure, so it could not be included in the tree size comparison. The different algorithms are labeled accordingly in the tables:

ID3 local: The ID3 algorithm used in the EMTI OSX environmnent.

ID3 weka: The ID3 algorithm used in Weka.

C4.5: Weka's J48 algorithm, the java equivalent to C4.5, with pruning.

C4.5 no pruning: Weka's J48 algorithm without pruning.

Random Forest: Weka's version of the Random Forest algorithm.

MLP: Weka's version of the Multilayer Perceptron algorithm.

EMTI: the basic EMTI algorithm.

Min-Err: EMTI using minimum-error pruning.

10-fold cross-validation was done 10 times for all the results, except for the monk, corral and mofn datasets, that supply their own test set. While the other algorithm's results are deterministic when using the same test set, EMTI results may still vary. For this reason EMTI was repeated 50 times and the results were averaged for each run on each monk dataset, the corral and mofn dataset. As in E2, mutation rate and crossover rate was set to 1.0 and sigma scaling for parent selection and generational mixing for adult selection was used. The rest of the experiment settings for EMTI are listed in table 4.13.

For the Weka algorithms the default parameter settings were used:

C4.5 used the following parameters: binarySplits = false, confidencefactor = 0.25, minNumObj = 2, numFolds = 3, reducedErrorPruning = false, subtreeRaising = true, unpruned = false (true for C4.5 no pruning), useLaplace = false.

Random Forest used the following parameters: maxDepth = 0 (unlimited), numFeatures = 0, numTrees = 100, seed = 1.

MLP used the following parameters: autobuild = false, decay = false, hiddenLayers = a, learningRate = 0.3, momentum = 0.2, nominalToBinaryFilter = true, normalizeAttributes = true, normalizeNumericClass = true, reset = true, seed = 0, trainingtime = 500, validationSetSize = 0, validationThreshold = 20.

There are no values for Weka's ID3 algorithm for the house votes and W.B. cancer datasets in table 4.15 because they contain missing values, which ID3 normally does not handle. The ID3 for the EMTI OS X environment uses the same data reading as EMTI, which reads and registers all attributes found in the data file without the use of .names file, making it possible for it to handle missing values.

Table 4.13: Overview of E3 settings.

#	Dataset	Pop. size	Generations	k
1	Diagonal 0	100	500	5
2	Diagonal 10	100	500	5
3	Diagonal 15	100	500	5
4	W. B. cancer	100	1000	5
5	Lymphography	100	1000	5
6	House votes	100	500	5
7	Zoo	100	500	5
8	Car eval.	100	1500	5
9	Bal. Scales	100	1500	3
11	Monk 1	100	1000	7
12	Monk 2	100	1000	7
13	Monk 3	100	1000	7
14	Corral	100	500	5
15	Mofn 3-7-10	100	1000	5

Table 4.14: E3 tree sizes in terms of node count.

Dataset	ID3 local	C4.5 no pruning	C4.5	EMTI	Min-err
Diagonal 0	69.02	66.49	58.30	67.06	58.84
Diagonal 5	127.13	78.30	46.70	77.41	62.46
Diagonal 15	225.11	90.47	3.85	84.96	54.71
W. B. Cancer	208.14	133.40	30.90	195.7	100.14
Lymphography	106.52	46.22	26.23	79.04	27.14
House votes	64.85	37.93	15.49	44.26	28.09
Zoo	22.92	18.46	18.30	20.22	18.84
Car eval.	207.29	186.37	169.88	197.17	100.23
Bal. Scales	486.10	111.60	42.60	498.90	120.20
Monk 1	90.00	43.00	18.00	39.96	38.34
Monk 2	171.00	73.00	31.00	164.00	132.30
Monk 3	41.88	25.00	12.00	41.88	22.62
Corral	23.00	11.00	11.00	13.00	13.00
Mofn 3-7-10	113.00	63.00	27.00	104.72	77.16
Mean	139.71	70.30	36.51	116.31	61.01

Table 4.15: E3 classification accuracies with standard deviations (%).

Dataset	ID3 local	ID3 weka	C4.5	Random Forest	MLP	EMTI	Min-err
Diag. 0	97.97 (2.90)	97.83 (2.86)	93.23 (4.82)	99.20(1.96)	93.37(5.03)	99.87 (0.81)	97.33 (4.03)
Diag. 5	88.86 (6.70)	89.19 (2.02)	87.70 (10.40)	91.67 (4.53)	90.80 (5.15)	95.50 (4.01)	93.77 (5.09)
Diag. 15	68.86 (8.09)	70.03 (7.38)	73.77 (1.99)	80.90 (5.95)	80.53 (6.66)	86.20 (6.48)	84.13 (7.60)
Cancer	89.80 (3.29)	-	94.57 (2.92)	96.35 (2.32)	95.55 (2.18)	93.52 (2.94)	94.28 (2.76)
Lympho	70.00 (11.31)	72.90 (11.06)	77.84 (9.33)	82.79 (8.19)	80.81 (9.51)	74.32 (11.42)	77.29 (9.70)
Votes	92.97 (3.56)	-	96.57 (2.56)	96.50 (2.52)	94.85 (3.19)	95.08 (3.27)	95.44 (2.94)
Zoo	95.05 (6.59)	97.12 (4.96)	92.61 (7.33)	97.02 (5.20)	95.25 (5.69)	94.95 (7.26)	92.77 (8.31)
Car eval.	90.16 (3.53)	89.19 (2.02)	92.22 (2.01)	94.63 (1.56)	99.41 (0.74)	93.58 (2.97)	94.22 (2.64)
Scales	38.69 (5.67)	37.74 (4.92)	64.14 (4.16)	79.55 (3.79)	98.23 (2.25)	71.36 (5.18)	68.10 (4.58)
Monk 1	77.08	78.47	75.69	92.36	100.00	100.00 (0.00)	98.50 (1.38)
Monk 2	66.67	64.58	65.05	69.90	100.00	69.79 (1.14)	69.00 (1.04)
Monk 3	93.98	94.44	97.22	96.06	93.52	94.25 (1.72)	95.38 (1.12)
Corral	81.25	87.50	81.25	96.88	100.00	98.88 (1.50)	98.88 (1.50)
Mofn	89.06	91.02	85.55	93.65	100.00	94.38 (2.93)	92.60 (2.47)
Mean	81.46	80.83	84.10	90.53	94.45	90.12	89.41

CHAPTER 5

Evaluation and Conclusion

In this chapter the results from the experiments in chapter 4 are evaluated and these evaluations are put in relation and used for discussing and answering the research questions that were posed in chapter 1. Finally, the project's potential and possibilities for future work is discussed.

5.1 Evaluation

In this section the results of the experiments presented in chapter 4 are analyzed and evaluated.

5.1.1 E1 evaluation

The results of experiment **E1** are shown in figure 4.1c and table 4.3.

E1.1 clearly shows how majority vote end node setting drastically improves EMTI search speed. Looking at figure 4.1c, using majority vote node setting the fitness quickly reaches closes the maximum, while random end node setting progresses a lot slower and never reaches the maximum fitness as seen in figure 4.1a.

Looking at table 4.3, E1.2 presents more unclear results, apart for the fact that the crossover operator is shown to be a vital part of the algorithm. For the diagonal domain dataset, no branch cut and no implode slightly weakens the search effectiveness while no deplode nearly halts the search completely (most of the time exceeding the 1000 cycle threshold). Curiously, as it is corresponds to the traditional form of mutation in evolutionary algorithms, no flipping improves the search effectiveness significantly. But when looking at the results for the Wisconsin breast cancer dataset, we see that using no flipping provides significantly worse results, while no deplode has just a small negative impact. This suggest that different

mutation types have different effectiveness depending on the dataset and its qualities (the diagonal dataset has a very clear underlying structure, while the breast cancer dataset is a real life dataset with significant noise.) This is also reflected in how no deplode has just a small negative effect for the Wisconsin breast cancer dataset.

5.1.2 E2 evaluation

The results of experiment **E2** are shown in detail from table 4.5 to 4.10 and summarized in table 4.11 and 4.12.

At a glance, we can see that the avoiding overfitting methods generally performed better on the real life datasets (Cancer, Votes and Credit), while basic generally performed better on the artificial ones (Diagonal 0, Diagonal 20 and Car). For all the experiments, basic scores higher accuracies using unvisited end-node optimizations. Looking at the results for basic small set, we can observe that reducing the size of the training set, which is required for many of the methods, already significantly worsens the classification accuracy for the small size datasets used in the experiments. This worsening is expected to be much less when training on larger datasets.

Starting at E2.1, the diagonal dataset with 0% noise, basic scores best with an average accuracy of 99.5%, the early stopping methods scores between 97.6-97.8%, slightly improving upon the score of basic small set. The two pruning methods scores worst with 96.3%. In E2.2, the diagonal dataset with 20% noise, the avoiding overfitting methods were expected to improve upon basic EMTI. But all methods score significantly worse and also worse than basic small set, except for min-err pruning which delivers slightly improved results[1], scoring 76.6% compared to the score of 76.2% for basic. Min-err pruning scores consistently well for E2.3 cancer, E2.4 votes and E2.5 credit, but performs worst in E2.6 car, though not significantly. Compared to min-err pruning, comp-err pruning prunes more radically, creating often much smaller tree sizes, but classification accuracy tends to be lower, except for the votes dataset, were it had the best accuracy of all methods, scoring 95.7%. Comp-err pruning overall created the smallest trees in the test, followed by Size control look-back. Size control look-back scores best of the early stopping methods, scoring especially high in E2.4 votes (95.6%) and E2.5 credit (70.6%), perhaps scoring only badly in E2.2 diagonal 20 (72.2%).

Looking at run times, we can see that basic and min-err, which use all available data for training, naturally uses longer time than the methods using parts of the data as validation set, because they have more data to evaluate the population on for each tree in each cycle. The size control methods speed up their run times further, up to 4 times faster than basic, by keeping their tree sizes lower, which makes traversing them faster.

[1] It should be noted that, as seen in E3, min-err pruning scores worse than basic for the diagonal domain with 15% noise, and also for any lower noise levels.

Because of the high accuracy score, the solid theoretical support of the method and the tendency to prune quite conservatively, which should give smaller variance in accuracies, min-err pruning was chosen as the pruning method to use for testing against other algorithms in E3.

5.1.3 E3 evaluation

The results of experiment **E3** are shown in table 4.14 and 4.15.

Looking at table 4.14, basic EMTI produces smaller trees than ID3 for all datasets except the Balance scales dataset. EMTI with Min-Err naturally produces even smaller trees. A surprising result is how the unpruned version of C4.5 generally produces smaller trees than EMTI, as it was expected that C4.5 without pruning would behave similarly to ID3. Normal C4.5 seems to prune much more aggressively than EMTI with Min-err, often creating much smaller tree sizes, for example for the Diagonal 15 dataset, 3.85 to 54.71, Balance scales, 42.60 to 120.20 and the Monk 2 dataset 31.00 to 132.30.

Table 4.15 shows MLP performing far better on average in terms of accuracy, providing top results for many datasets and only scoring significantly lower for the Diagonal domain datasets. Random forest scores the second best results, but only slightly better on average than EMTI. All though C4.5 often scores on par or better than EMTI, it has a few result dragging its average down, scoring particularly bad on the Monk 1 and Corral dataset. EMTI with min-err scores slightly worse on average than basic EMTI, but scores higher for the W.B. Cancer, Lymphography, House Votes, Car evaluation and Monk 3 datasets. Surprisingly, min-err scored worse than basic ETMI for the car evaluation dataset in experiment 2, but better here. This could be caused by the bigger population and higher generation cycle count used in experiment 3 for this dataset. EMTI scores the strongest of all the methods on some of the artificial dataset; the diagonal, monk 1, corral and mofn datasets.

5.2 Discussion

This section discusses and answers the research questions defined in section 1.1, using the results and evaluation of the experiments presented in chapter 4.

5.2.1 RQ1 discussion

Which parts of EMTI are vital for its performance and which parts are not contributing?

Experiment E1 showed the importance of the majority vote end node setting method and the crossover operator for algorithm's search effectiveness. The differ-

ent mutation operators (cut, implode, deplode and flip) showed to have differing effectiveness impacts depending on the domain the algorithm was run on, with all methods being able to increase the effectiveness in certain domains. Experiment E2 showed that using kNN for unvisited end nodes improves the EMTI's ability to classify accurately.

5.2.2 RQ2 discussion

In which way can methods for avoiding overfitting be added to EMTI to improve its performance?

Three early stopping methods and two decision tree pruning methods were tested against each other and against the basic EMTI implementation in experiment E2. It was observed and it stands for reason that for domains with no potential for overfitting to occur, these methods, aimed at simplifying the modeled functions, can never achieve higher classification accuracies, only the same or worse than the basic EMTI implementation. However, machine learning is ultimately intended to be used in interactions with the real world, and domains in the real world are more likely to contain noise and have potential for overfitting than artificial, abstract domains. The experiments showed that the methods for avoiding overfitting increased the classification accuracies for a number of real world datasets, which should be given higher importance than the fact that they often degraded the accuracies for artificial domains.

The pruning methods tried in E2 were relative simple and many, more sophisticated decision tree pruning algorithms exist. Seeing how the C4.5 algorithm aggressively prunes its decision tree and still scores higher than EMTI on real world datasets such as the Wisconsin breast cancer, lymphography and house votes datasets suggests that EMTI could with success utilize even more aggressive and sophisticated pruning methods. But C4.5 also overly simplifies many of the problems, particularly noticeable in the diagonal 15, balance scales and monk 2 datasets, causing lower classification accuracies. If the choice was to either employ no method for avoiding overfitting or always employing it, the latter is recommended. Min-err has shown to be a good algorithm for such a scenario, by not pruning too aggressively, it does not degrade EMTI significantly on the domains that have no need for pruning and is still able to improve the accuracies in domains were pruning is useful.

5.2.3 RQ3 discussion

Can EMTI's ability of setting the maximum tree size be used in methods for avoiding overfitting?

The experimental size control with look-back method achieved the third highest mean accuracy score out of the five methods for avoiding overfitting and scored higher accuracies than the basic EMTI implementation for the german credit,

house votes and Wisconsin breast cancer datasets, greatly reducing the tree sizes found.

5.2.4 RQ4 discussion

What are the characteristics of the problems that EMTI functions well on?

In experiment E3, EMTI scored particularly low on the monk 2, balance scales and lymphography datasets. The monk 2 dataset is a relative simple problem with two different categories. Still EMTI classifies just 69.8% of the test data correctly, which is just 19.8% more than the expected result for someone guessing the category randomly. The reason for this is that the monk 2 problem, stated as "exactly two of the six attributes have their first value", is designed to be difficult to express using decision trees by not being in standard conjunctive form. To express this problem with decision trees you would have to test for almost all the attributes [32] which requires a very large tree. Here it should be noted how methods for avoiding overfitting will directly work against the tree learning algorithm's ability to model problems like this, because these problem require large and complex solutions which methods for avoiding overfitting will undermine, causing underfitting. The balance scale domain touches on the same problem, though it is an artificial data set with no noise, all the tree learning algorithms struggle to express the relationship between the attributes. The MLP algorithm is able to handle this higher order logic and scores nearly perfect for both the monk 2 and balance scales datasets. The real world lymphography dataset seems to be a high noise dataset, with a high potential for overfitting. EMTI with min-err pruning scoring worse than C4.5 and random forest for this dataset showing that EMTI could benefit from a more aggressive pruning method.

EMTI scores particularly well on the diagonal, corral and mofn datasets, which are datasets with a clear underlying structure and in standard conjunctive form, but with added noise and/or irrelevant and correlated attributes with a relative small amount of training instances compared to the amount of possible permutations of the problem. The risk for the greedy top-down algorithms when running on such datasets is that it will do the first split of the dataset on the wrong attribute. We can conclude that EMTI consistently avoids this and shows a strong ability to ignore noise and to find the underlying structure of such problems, even achieving higher accuracies than MLP. EMTI also scores perfectly on the monk 1 dataset, where it seems that the other tree learning algorithms struggle with the same problem of splitting the dataset on the wrong attributes. Further, EMTI shows a strong ability to model a logical function based on a small set of example instances, such as in the diagonal domain with 0% noise, were it scores the highest accuracy (99.87%), scoring significantly better than MLP (93.37%).

5.2.5 RQ5 discussion

How does ETMI's performance compare to other well know classifier methods?

On average, EMTI performs in terms of classification accuracies worse than MLP and slightly worse than Random Forests. The main advantage EMTI has over these algorithms is its ability to clearly explain its decision making process. EMTI performs better on average than ID3 and C4.5 in terms of classification accuracies, though C4.5 shows somewhat better results for certain real world datasets that are structured in such away that aggressive pruning is beneficial. EMTI is shown to be a robust algorithm, providing consistent results with few negative outliers. The limitations in the expressiveness of decision trees means that EMTI cannot compete in terms of accuracy with learning methods such as artificial neural networks on certain problem types.

The main motivation and hypothesis of the project was that by widening the solution search space of traditional greedy top-down tree learning algorithms such as ID3 and C4.5 by the use of EA search, from local to global search, better solutions can be found. The results show how this approach is able to avoid the problem of splitting on the wrong attribute early in the tree, by testing many different attribute splits in the genetic population, making it able to often find better solutions than the traditional learning algorithms. Compared to ID3, the unpruned version of EMTI consistently finds different, smaller solutions, showing how ID3, whose basic methodology forms the basis of many top-down tree learners, consistently finds solutions far from the optimal one.

5.3 Conclusion and Future Work

EMTI has been shown to be a robust learning algorithm that is able to create accurate decision trees while maintaining their inherent comprehensibility. It can make use of post decision tree pruning to avoid overfitting and is able to handle strong data noise and to disregard irrelevant attributes, making it perform well against other learning algorithms.

This master thesis project deals with the function modeling stage of a supervised learning algorithm and explores some post modeling options like decision tree pruning. There is potential for improving the performance of the algorithm both in the data-preprocessing and the post modeling stages of the learning process. These stages are modular; different methods can be used interchangeably without changing the EMTI algorithm. For the preprocessing stage there is a need for a data discretization method, but also other methods could be useful, such as data scaling and feature selection methods. For the post modeling stage, other tree pruning methods using more sophisticated statistical pruning criteria is believed to be able to improve the performance of EMTI in domains susceptible to overfitting.

One of EMTI's expected abilities is to be able to adjust to new data in the domain

without resetting the solutions; it should be able to dynamically adjust the existing trees in the population. This would need to be verified through testing.

The possibly novel concept of majority vote end node setting (see section 3.1.7) could be extended upwards in the tree to decision nodes, though with an exponential complexity cost for each extra tree layer looked at. If this is done all the way up to the root node, the algorithm is essentially searching only for the structure of the tree, no longer caring for which values it contains. This is believed to often be achievable as it has been observed that for most domains the trees generated extend in depth only in a small proportion to the attribute count of the domain. Another less complex possibility is to test for different values just for the end nodes and the root node of the tree, as the root nodes is the single most decisive value of the decision tree. The main challenge will be to adjust the tree in domains were different attributes have different amounts of categories.

In terms of improving run times, the potential for parallelization shows the most promise, as EMTI is made up by many independent operations, such as the fitness setting, mutation operator and the crossover operator, which together form the largest part of the algorithm's time complexity. There has been no particular focus on the time performance of the implementation, so there is likely to be many ways of improving run times by restructuring and optimizing the code, especially in the way the tree structures are handled.

Decision trees are limited in their expressiveness, resulting in universal poor performance on certain problems such as boolean satisfiability problems. It could be interesting to research the possibility of applying evolutionary algorithms to other, more expressive logical languages such as the propositional decision rule learning used in the AQ-algorithm (Michalski [20]).

References

[1] Aitkenhead, "A co-evolving decision tree classification method", Expert Syst. Appl., vol. 34, no. 1, pp. 18-25, 2008

[2] Bandar and McLean "Genetic algorithm based multiple decision tree induction", 6th International Conference on Neural Information Processing, pp. 429-434, 1999

[3] Barros, Basgalupp, Carvalho and Freitas, "A Survey of Evolutionary Algorithms for Decision Tree Induction", IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, vol. 42, issue 3, 2012

[4] Breiman, "Random Forests", Machine learning 2001

[5] Breiman, Freidman, Olshen and Stone, "Classification and regression trees", California: Wadsworth International, 1984

[6] Eiben, Hinterding and Michalewicz, "Parameter control in evolutionary algorithms" in IEEE Transactions on Evolutionary Computation, vol. 3, no. 2, pp. 124-141, Jul 1999

[7] Fayyad, "On the Induction of Decision Trees for Multiple Concept Learning", Doctoral dissertation, Department of Electrical, Engineering and Computer Science, University of Michigan, 1991

[8] Freitas, "Data Mining and Knowledge Discovery with Evolutionary Algorithms" Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002

[9] Fu, Golden, Lele, Raghavan and Wasil, "A Genetic Algorithm-Based Approach for Building Accurate Decision Tree", INFORMS Journal on Computing/Vol. 15, No. 1, Winter 2003

[10] Fuglseth "Evolutionary induction of decision trees", (Project report, TDT4501 Computer science, Specialization project, 2015)

[11] Google Scholar, web search engine of scholarly literature `https://scholar.google.com`, last accessed June 2016

[12] Gnuplot, portable command-line driven graphing utility `http://www.gnuplot.info`, last accessed June 2016

[13] Graphviz, open source graph visualization software `http://www.graphviz.org`, last accessed June 2016

[14] Hyafil and Rivest, "Constructing Optimal Binary Decision Trees is NP-complete". Information Processing Letters 5 (1): 15-17, 1976

[15] John, Kohavi and Pfleger, "Irrelevant features and the subset selection problem", Machine Learning: Proceedings of the Eleventh International Conference, 1994

[16] Kennedy, Chinniah, Bradbeer and Morss, "The construction and evaluation of decision trees: A comparison of evolutionary and concept learning methods", Evolutionary Computing, ser. Lecture Notes in Computer Science, D. Corne and J. Shapiro, vol. 1305, pp. 147-161, 1997

[17] Kretowski and Grzes, "Mixed decision trees: An evolutionary approach", International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2006) pp. 260-269, 2006

[18] Loveard and Ciesielski, "Representing classification problems in genetic programming", Proc. Congr. Evolutionary Computation, May 27-30, pp. 1070-1077, 2001

[19] Michalewicz and Schmidt, "Parameter control in practice", Parameter Setting in Evolutionary Algorithms, ser. Studies in Computational Intelligence, vol. 54, pp. 277-294, 2007

[20] Michalski, "On the quasi minimal solution of the general covering problem" Proc. 5th International Symposium on Information Processing (FCIP 1969), Bled, Yugoslavia, vol. A3, pp. 25-128, 1969

[21] Mingers, "An Empirical Comparison of Pruning Methods for Decision Tree Induction", Machine Learning, Volume 4, Issue 2, pp 227-243, 1989

[22] Mingers, "Expert systems rule induction with statistical data", Journal of the Operational Research Society, 38, 39-47, 1987

[23] Niblett and Bratko, "Learning decision rules in noisy domains", Proceeding Expert System 86, Brighton, Cambridge: Cambridge University Press, 1986

[24] Papagelis and Kalles, "Breeding Decision Trees Using Evolutionary Techniques", ICML '01 Proceedings of the Eighteenth International Conference on Machine Learning, pages 393-400, 2001

[25] Prechelt, "Early stopping - but when?", Orr, G.B. and Müller, K.-R. (Eds.): LNCS 1524, ISBN 978-3-540-65311-0, 1998

[26] Quinlan, "Induction of Decision Trees", Machine Learning, Volume 1, Issue 1, pp 81-106, March 1986

[27] Quinlan, "C4.5: Programs for Machine Learning". Morgan Kaufmann Publishers, Inc., 1993

[28] Quinlan, "Simplifying decision trees", International Journal of Man-Machine Studies, 27, 221-234, 1987

[29] Russell and Norvig, "Artificial intelligence: a modern approach", 3rd edition, Prentice-Hall, 2009

[30] Smith, "Rna search acceleration with genetic algorithm generated decision trees", iSeventh International Conference on Machine Learning and Applications, pp. 565-570, 2008

[31] Streichert, "Introduction to Evolutionary Algorithms", University of Tuebingen

[32] Thrun, Bala, Bloedorn, Bratko, Cestnik, Cheng, Jong, Dzeroski, Hamann, Kaufman, Keller, Kononenko, Kreuziger, Michalski, Mitchell, Pachowicz, Roger, Vafaie, Van de Velde, Wenzel, Wnek and Zhang, "The MONK's Problems: A Performance Comparison of Different Learning Algorithms", CMU-CS-91-197, Carnegie Mellon University, Computer Science Department, 1991

[33] The Corral and Mofn 3-7-10 dataset were found here: `https://www.sgi.com/tech/mlc/db/`, last accessed 2016

[34] Trollhetta AS, `http://www.trollhetta.com`, last accessed June 2016

[35] UC Irvine Machine Learning Repository, `http://archive.ics.uci.edu/ml/`, last accessed 2016

[36] Weka 3: Data Mining Software in Java, `http://www.cs.waikato.ac.nz/ml/weka/`, last accessed June 2016

[37] Zhao and Ciesielski, "Multi-objective genetic programming approach to developing pareto optimal decision trees", Decis. Support Syst., vol. 43, no. 3, pp. 809-826, 2007