PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Evolutionary Model Tree Induction

Rodrigo Coelho Barros

Dissertação de Mestrado apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Duncan Dubugras Alcoba Ruiz

Porto Alegre
2009

Dados Internacionais de Catalogação na Publicação (CIP)

B277e Barros, Rodrigo Coelho
Evolutionary model tree induction / Rodrigo Coelho Barros. – Porto Alegre, 2009.
83 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS
Orientador: Duncan Dubugras Alcoba Ruiz

1. Informática. 2. Mineração de Dados (Informática). 3. Algoritmos. I. Ruiz, Duncan Dubugras Alcoba. II. Título.

CDD 005.72

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada *"Evolutionary Model Tree Induction"*, apresentada por Rodrigo Coelho Barros, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas de Informação, aprovada em 10/12/09 pela Comissão Examinadora:

Prof. Dr. Duncan Dubugras Alcoba Ruiz – Orientador — PPGCC/PUCRS

Profa. Dra. Vera Lúcia Strube de Lima – PPGCC/PUCRS

Prof. Dr. Luis Otavio Campos Alvares – UFRGS

Homologada em 02/03/2010, conforme Ata No. 03/10 pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

“À minha avó,
Lygia Yara Vernetti.”

Acknowledgements

Aos meus pais, Marta e Luis Fernando, responsáveis diretos pela minha formação pessoal e profissional.

À minha avó, Lygia, pessoa muito especial na minha vida. Ela é a principal responsável por eu poder redigir este texto em inglês.

À minha irmã, Roberta e ao meu cunhado, Denis.

À minha namorada linda, Alessandra.

Aos meus colegas de apartamento e bons amigos, Mesquita e Braga, por proporcionarem dois anos de convivência excepcional e de muitas risadas.

Ao meu orientador, Duncan, pelos ensinamentos técnico-científicos e por acreditar em mim e no trabalho que propusemos. Agradeço também pela luta constante por auxílios financeiros, tornando possível minha dedicação exclusiva ao mestrado.

Ao colega e bom amigo, Márcio Basgalupp, que me apresentou ao paradigma de algoritmos evolutivos e tornou possível essa nossa caminhada, até então bem sucedida, de trabalhos em parceria. Agradeço também aos professores André Carvalho e Alex Freitas pelas ótimas contribuições no decorrer dessa parceria.

Aos colegas de GPIN - Ana, Christian, Luciano, Luzia, Nelson, Patrícia e Peterson - pelos ótimos momentos que passamos juntos e pelos auxílios técnicos indispensáveis.

Aos colegas de HP - Igor, Rita e Samuel - pelas boas risadas nas "horas improdutivas".

Aos colegas de Petrobrás/Paleoprospec, que são muitos para serem listados, mas igualmente importantes no decorrer do meu mestrado.

Àqueles que não mencionei pelo simples fato da minha memória estar se extinguindo, minhas desculpas e eterna gratidão.

Indução Evolutiva de Árvores-Modelo

Árvores-modelo são um caso particular de árvores de decisão aplicadas na solução de problemas de regressão, onde a variável a ser predita é contínua. Possuem a vantagem de apresentar uma saída interpretável, auxiliando o usuário do sistema a ter mais confiança na predição e proporcionando a base para o usuário ter novos *insights* sobre os dados, confirmando ou rejeitando hipóteses previamente formadas. Além disso, árvores-modelo apresentam um nível aceitável de desempenho preditivo quando comparadas à maioria das técnicas utilizadas na solução de problemas de regressão. Uma vez que gerar a árvore-modelo ótima é um problema NP-Completo, algoritmos tradicionais de indução de árvores-modelo fazem uso da estratégia gulosa, *top-down* e de divisão e conquista, que pode não convergir à solução ótima-global. Neste trabalho é proposta a utilização do paradigma de algoritmos evolutivos como uma heurística alternativa para geração de árvores-modelo. Esta nova abordagem é testada por meio de bases de dados de regressão públicas da UCI, e os resultados são comparados àqueles gerados por algoritmos gulosos tradicionais de indução de árvores-modelo. Os resultados mostram que esta nova abordagem apresenta uma boa relação custo-benefício entre desempenho preditivo e geração de modelos de fácil interpretação, proporcionando um diferencial muitas vezes crucial em diversas aplicações de mineração de dados.

Palavras-chave: Árvores-modelo; Mineração de Dados; Algoritmos Evolutivos.

Evolutionary Model Tree Induction

Model trees are a particular case of decision trees employed to solve regression problems, where the variable to be predicted is continuous. They have the advantage of presenting an interpretable output, helping the end-user to get more confidence in the prediction and providing the basis for the end-user to have new insight about the data, confirming or rejecting hypotheses previously formed. Moreover, model trees present an acceptable level of predictive performance in comparison to most techniques used for solving regression problems. Since generating the optimal model tree is a NP-Complete problem, traditional model tree induction algorithms make use of a greedy top-down divide-and-conquer strategy, which may not converge to the global optimal solution. In this work, we propose the use of the evolutionary algorithms paradigm as an alternate heuristic to generate model trees in order to improve the convergence to global optimal solutions. We test the predictive performance of this new approach using public UCI data sets, and we compare the results with traditional greedy regression/model trees induction algorithms. Results show that our approach presents a good trade-off between predictive performance and model comprehensibility, which may be crucial in many data mining applications.

Keywords: Model Trees; Data Mining; Evolutionary Algorithmms.

List of Figures

Figure 2.1 Knowledge discovery process. Source: [HK06] 30
Figure 2.2 Example of decision tree. Source: [TSK06] 31
Figure 2.3 Example of regression tree. Source: [WF05] 32
Figure 2.4 Example of model tree. Source: [WF05]. 32

Figure 3.1 Typical evolutionary algorithm. 38
Figure 3.2 Five scenarios compared on the basis of widget cost (x) and accidents (y). Source: [Gol89]. 42

Figure 5.1 E-Motion individual. 52
Figure 5.2 4 categorical basic trees created from a 5-category attribute. 53
Figure 5.3 Merging basic trees A, B and C into an individual. 55
Figure 5.4 Tournament selection. 59
Figure 5.5 Crossover between two individuals and the resulting offspring. 60
Figure 5.6 Two distinct kinds of mutation. 61
Figure 5.7 Filtering process. 62
Figure 5.8 Smoothing process. 63

List of Tables

Table 4.1 Related work comparison. 49

Table 5.1 Algorithms comparison. 63

Table 6.1 E-Motion default parameters. 66
Table 6.2 Data sets used in experimentation. 68
Table 6.3 Four different configurations of E-Motion. 70
Table 6.4 Error measures for E-Motion, M5P and REPTree. 70
Table 6.5 Tree Size for E-Motion, M5P and REPTree. 71
Table 6.6 E-Motion significantly better (a) or worse (b) than M5P and REP according to the corrected paired t-test. 71
Table 6.7 Results for 4 different E-Motion configurations, M5P and REPTree in two data sets. 72
Table 6.8 E-Motion's configurations E01 and E11 significantly better (a) or worse (b) than M5P and REP according to the corrected paired t-test. 73

List of Algorithms

3.1 Pseudo-code for GAs and GP. Source: [Fre08]. 39
5.1 Pseudo-code for random numeric basic trees generation. 54
5.2 Pseudo-code for clustered numeric basic trees generation. 54
.1 Pseudo-code of basic K-means. 81
.2 Pseudo-code of the lexicographic fitness analysis. 83

List of Abbreviations

KDD	Knowledge Discovery in Databases	29
SDR	Standard Deviation Reduction	33
EAs	Evolutionary Algorithms	37
GAs	Genetic Algorithms	38
GP	Genetic Programming	38
TARGET	Tree Analysis with Randomly Generated and Evolved Trees	45
RSS	Residual Sum of Squares	47
GPMCC	Genetic Programming approach for Mining Continuous-valued Classes	47
E-Motion	Evolutionary Model Tree Induction	51
RMSE	Root Mean Squared Error	55
MAE	Mean Absolute Error	55
UCI	University of California at Irvine	67

Table of Contents

1 Introduction **25**

2 Data Mining **29**

2.1 Introduction 29

2.2 Data Mining Trees 30

2.2.1 Decision Trees 30

2.2.2 Regression Trees 31

2.2.3 Model Trees 32

2.2.4 Ensemble Methods 34

2.3 Chapter Remarks 35

3 Evolutionary Algorithms **37**

3.1 Concepts 37

3.2 Genetic Algorithms and Genetic Programming 38

3.3 Multi-objective optimization 39

3.3.1 Weighted-formula approach 40

3.3.2 Pareto-dominance approach 41

3.3.3 Lexicographic approach 43

3.4 Chapter Remarks 43

4 Related Work **45**

4.1 TARGET 45

4.1.1 Initial forest 45

4.1.2 Fitness Function 45

4.1.3 Genetic Evolution 46

4.1.4 Experimentation 47

4.2 GPMCC 47

4.2.1 Initial Forest 47

4.2.2 Fitness Function 47

4.2.3 Genetic Evolution 48
4.2.4 Experimentation 48
4.3 Chapter Remarks 49

5 Evolutionary Model Tree Induction (E-Motion) 51
5.1 Solution Representation 51
5.2 Initial Forest 51
5.3 Fitness evaluation 55
5.3.1 Weighted-Formula 56
5.3.2 Lexicographic Analysis 57
5.4 Selection 58
5.5 Crossover 58
5.6 Mutation 59
5.7 Consistency check 60
5.8 Stopping criteria 61
5.9 Prediction Smoothing 61
5.10 Chapter Remarks 62

6 Evaluating E-Motion 65
6.1 Default Parameters 65
6.2 Public data sets 67
6.3 Experimental Methodology 67
6.3.1 First Experiment Protocol 68
6.3.2 Second Experiment Protocol 69
6.4 First Experiment Results 70
6.5 Second Experiment Results 72
6.6 Chapter Remarks 73

7 Final Remarks and Future Work 75

References 77

Appendix A K-Means 81

Appendix B Lexicographic analysis 83

1

Introduction

Within the data mining regression task, model trees are a popular alternative to classical regression methods, presenting good predictive performance and an intuitive interpretable output. Similarly to decision/regression trees, they are structured trees that represent graphically if-then-else rules, which seek to extract implicit knowledge from data sets. While decision trees are used to solve classification problems[1] (i.e., the output is a nominal value), both model and regression trees are used to solve regression problems (i.e., the output is a continuous value). The main difference between these approaches is that while regression trees have a single value as the output in their leaves (corresponding to the average of values that reach the leaf), model trees hold equations used for calculating the final output.

A model tree is composed by non-terminal nodes, each one representing a test over a data set attribute, and linking edges that partition the data according to the test result. In the bottom of the tree, the terminal nodes hold linear regression models built according to the data that reached each given node. Thus, for predicting the target-attribute value for a given data set instance, we follow down the tree from the root node to the bottom, until a terminal node is reached, and then we apply the corresponding linear model.

Model trees result in a clear knowledge representation, providing the user information on how the output was reached (i.e., the if-then-else rule that is provided by the tree once we follow the path until a terminal node). In contrast, approaches such as neural networks and support vector machines, while more efficient than model trees in terms of predictive performance in many problems, lack on transparency, because they do not provide the user information on how outputs are produced [RM05].

Model trees are traditionally induced by divide-and-conquer greedy algorithms which are sequential in nature and locally optimal at each node split [FG05]. Since inducing the best tree is a NP-Complete problem [TSK06], a greedy heuristic may not derive the best tree overall. In addition, recursive partitioning iteratively degrades the quality of the data set for the purpose of statistical inference, because the larger the number of times the data is partitioned, the smaller becomes the data

[1] It is not consensual in the data mining literature if decision trees are used only for classification or if regression/model trees are a particular case of decision trees. In this dissertation we adopt the first choice, i.e., data mining trees are divided in decision trees (classification task), regression trees (regression task) and model trees (regression task).

sample that fits the specific split, leading to results without statistical significance and creating a model that overfits the training data [BBC+09a].

In order to avoid the drawbacks of the greedy tree-induction algorithms, recent works have focused on powerful ensemble methods (e.g., bagging [Bre96], boosting [FS97], random forests [Bre01], etc.), which attempt to take advantage of the unstable induction of models by growing a forest of trees from the data and later averaging their predictions. While presenting very good predictive performance, ensemble methods fail to produce a single-tree solution, operating also in a black-box fashion.

We highlight the importance of validation and interpretation of discovered knowledge in many data mining applications, because comprehensible models can lead to new insights and hypotheses upon the data [FWA08]. Hence, we believe there should be a trade-off between predictive performance and model comprehensibility, so a predictive system can be useful and helpful in real-world applications. For instance, Barros et al. [BBTR08] make use of a model tree induction algorithm in a study case conducted within a large software operation with the goal of providing better effort estimates. An interpretable output was desired so the project managers could analyze the results presented by the algorithm and make decisions accordingly.

Evolutionary algorithms are a solid heuristic able to deal with a variety of optimization problems. An evolutionary approach for producing trees could enhance the chances of converging to global optima, avoiding solutions that get trapped in local-optima and that are too sensitive to small changes in the data. Evolutionary induction of decision trees is well-explored in the research community. Basgalupp et al. [BBC+09b] proposed an evolutionary algorithm for the induction of decision trees named LEGAL-Tree, which looks for a good trade-off between accuracy and model comprehensibility. Very few works however propose evolving regression/model trees as an alternative to greedy approaches.

Hence, we propose a new algorithm based on the evolutionary algorithms paradigm and also on the core idea presented in LEGAL-Tree [BBC+09b] to deal with any kind of regression problems. By evolving model trees with an evolutionary algorithm, we seek to avoid local-optima convergence by performing a robust global search in the space of candidate solutions. Moreover, our approach has to deal with multi-objective optimization, since our intention is to produce model trees that have both high predictive performance and comprehensibility[2].

We test the predictive performance and comprehensibility of this new algorithm using UCI regression data sets [AN07], and we compare the results to those produced by well-known algorithms, such as M5 [Qui92] and REPTree [WF05].

The remaining of this work is organized as follows. We briefly review the literature in data mining and evolutionary algorithms in Chapters 2 and 3, presenting general concepts and ideas that support the decisions made during the design of the proposed algorithm. Chapter 4 presents works that relate evolutionary algorithms and trees used in regression problems. Chapter 5 describes our new approach

[2]Comprehensibility, in this dissertation, is measured by counting the number of nodes in each tree. We assume that smaller trees are more legible, easier to interpret than those with a large number of nodes.

for inducing model trees, whereas the tests and comparisons to other algorithms are executed in Chapter 6. We conclude this work with some remarks and future work on Chapter 7.

2

Data Mining

This chapter presents general concepts on data mining and focuses on a specific technique that is broadly used for different mining tasks: data mining trees. It describes briefly the main algorithms for each kind of tree and it does some considerations regarding the strategy for building these trees.

2.1 Introduction

Data mining is the process of extraction of useful information from large amounts of data [HK06]. It is an integral part of knowledge discovery in databases (KDD), process in which raw data is converted into useful information (Figure 2.1). In general terms, all steps that precede the data mining process are within the *data preprocessing* stage, where raw data are properly transformed to suit different mining algorithms. Similarly, the steps that proceed the mining process are said to be part of the *data postprocessing* stage, in which data patterns are processed for visualization and interpretation purposes [TSK06].

Data Mining tasks can be divided in two major categories [TSK06] :

1. Descriptive tasks: are used to derive patterns that describe relationships in data (e.g., correlations, trends, clusters, anomalies, etc.). These tasks generally require postprocessing techniques to validate and explain the results that were achieved.

2. Predictive tasks: are used to predict the value of a particular attribute by taking into account the values of other attributes. The attribute to be predicted is called *target* or *dependent variable,* while the attributes used for making the predictions are known as *explanatory* or *independent variables*.

In this research we focus on predictive data mining tasks. Classification and regression are probably the most common predictive tasks in data mining, and they can be used to extract models that describe important data classes or that predict future data trends, respectively. We focus specifically on the regression task.

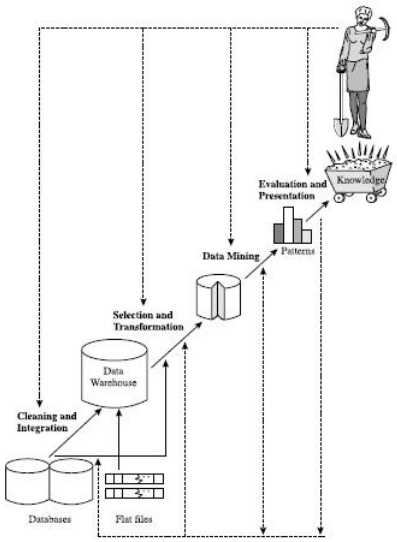

Figure 2.1: Knowledge discovery process. Source: [HK06]

Han and Kamber [HK06] differentiate classification and regression through the following sentence:

> Whereas classification predicts categorical (discrete, unordered) labels, regression models continuous-valued functions. For example, we can build a classification model to categorize bank loan applications as either safe or risky, or a regression model to predict the expenditures in dollars of potential customers on computer equipment given their income and occupation. (p. 285)

A popular strategy to deal with both classification and regression is through hierarchical models in the form of trees. Next section covers the main types of trees for classification and regression.

2.2 Data Mining Trees

Data mining trees are an efficient nonparametric method which can be used both for classification and regression. They are hierarchical data structures that usually implement the divide-and-conquer strategy to split the input space into local regions and predict the dependent variable accordingly [Alp04].

2.2.1 Decision Trees

Decision trees are hierarchical graphical representations of knowledge extracted from data sets, which are used for classifying objects. They are widely-used mostly because of the following reasons [TSK06]:

(i) ease of understanding, because they can be easily converted in a set of rules written in a natural language;

(ii) robustness to the presence of noise;

(iii) availability of computationally inexpensive induction algorithms, even for very large data sets; and

(iv) good handling of irrelevant attributes.

Figure 2.2 presents a decision tree for classifying animals as “mammals” or “non-mammals”, a typical binary-class problem.

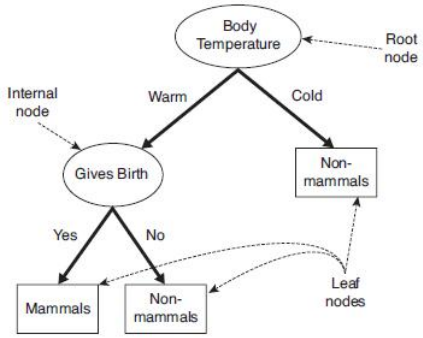

Figure 2.2: Example of decision tree. Source: [TSK06]

First, an attribute is chosen to be the root node and then it is recursively split in internal nodes until certain criterion (criteria) is (are) reached and we stop splitting the tree. The nodes in the bottom of the tree (leaf nodes) are labeled according to the most frequent class of the instances that reach that given node. The criteria to split the nodes, the stopping conditions and tree-pruning decisions vary according to the algorithm used.

Decision trees cannot deal with continuous classes problems. One possible solution would be discretize the continuous attribute in intervals spaced according to a certain heuristic. This approach often fails, partly because decision trees induction algorithms cannot make use of the implicit ordering of the discretized classes [Qui92]. Several more effective learning methods for predicting real values are available, noticeably regression and model trees, which are presented below.

2.2.2 Regression Trees

Regression trees follow the same basic principles of decision trees. The difference is the result hold by the leaf nodes, which is no longer a class label, but a real value (Figure 2.3). Typically, these real values are the average of the dependent variable values of the instances that reach each leaf node. For splitting the tree, most algorithms (e.g., CART [BFOS84]) choose a test to give the greatest expected reduction in either variance or absolute deviation.

Regression tree’s accuracy is competitive with linear regression, with the potential of being much more accurate on non-linear problems. These trees offer an interesting alternative for looking at regression problems, since they can sometimes give clues to data structure that is not apparent from a linear regression analysis perspective [BFOS84].

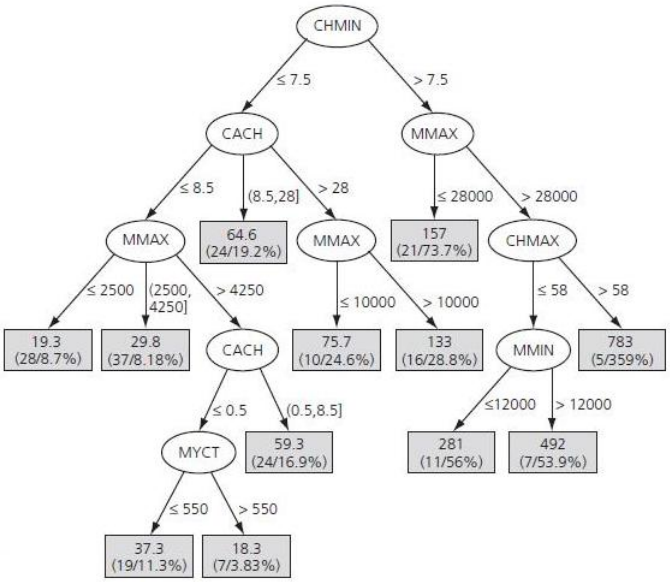

Figure 2.3: Example of regression tree. Source: [WF05]

2.2.3 Model Trees

Model trees, similarly to regression trees, are hierarchical structures for predicting continuous dependent variables. At each leaf node they hold a linear regression model that predicts the class value of instances that reach the leaf. The only difference between regression tree and model tree induction is that, for the latter, each node is replaced by a regression plane instead of a constant value (Figure 2.4).

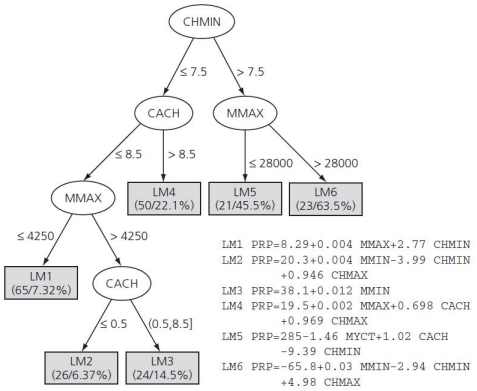

Figure 2.4: Example of model tree. Source: [WF05].

Suppose we have d predictor variables, $X_1, X_2, , ..., X_d$ and a response continuous variable Y , which is the target of the prediction model. The training set has n records, (x_i, y_i), where x_i is the vector of predictor variable values for the i^{th} training record, *i.e.* $x_i = (x_{i1}, x_{i2}, ..., x_{id})$ and y_i is the target attribute value for this same i^{th} record. A model tree holds K terminal nodes that will partition the predictor variable space into R_k regions, where $k = 1, 2, .., K$. Each region R_k will predict the

value of Y through a multivariate linear regression function, created through a least-square method [Bjö96]. A terminal node will have α predictor variables (a number $\leq d$) and β records (a number $\leq n$), the linear regression demands the creation of a $\beta \times \alpha$ **design matrix**, $M_1, M_2, ..., M_t$ such as shown in (2.1).

$$M_i = \begin{pmatrix} 1 & x_{11} & x_{12} & ... & x_{1\alpha} \\ 1 & x_{21} & x_{22} & ... & x_{2\alpha} \\ ... & ... & ... & ... & ... \\ 1 & x_{\beta 1} & x_{\beta 2} & ... & x_{\beta\alpha} \end{pmatrix} \tag{2.1}$$

Considering $\Omega = (\omega_0, \omega_1, ..., \omega_{\alpha-1})^{\mathrm{T}}$ the α-dimensional vector that represents the regression coefficients that minimize the sum of the squared error for the response variable Y, and $y = (y_1, y_2, ..., y_\beta)^{\mathrm{T}}$ the β-dimensional vector that represents the β values of the response variable Y, we can define the coefficients by solving the matrix equation given in (2.2).

$$\Omega = (M^{\mathrm{T}}M)^{-1}M^{\mathrm{T}}y \tag{2.2}$$

Once the coefficients are known, each terminal node will hold a regression model such as the one in (2.3).

$$y = \varpi_0 + \sum_{i=1}^{\alpha} w_\alpha x_\alpha \tag{2.3}$$

Hence we can notice that model trees are more sophisticated than either regression trees or linear regression. Since each terminal node will hold a regression model based on the instances that reach that node, model trees can also approximate non-linear problems, which is the case of a wide range of real-world mining applications.

For growing a model tree, most algorithms rely on a greedy top-down strategy for splitting recursively the nodes, which is also the case of both decision and regression trees. These algorithms seek to minimize some error measure that results from testing each attribute for splitting the node. M5 [Qui92] uses the standard deviation as the error measure for choosing the best split at each node. The goal is to maximize the standard deviation reduction (SDR) by testing the possible splits over the training data that reaches a particular node, as shown in (2.4).

$$SDR = sd(D) - \sum_i \frac{|D_i|}{|D|} \times sd(D_i) \tag{2.4}$$

where $sd(.)$ is the standard deviation calculation, D is the training set portion that reaches the node that is being tested and D_i the training set portion that results from splitting the node according to a given attribute and split value.

The tree induction of model trees through the greedy algorithms is sequential in nature and locally

optimal at each node split, which means that convergence for a global optimal solution is hardly feasible. In addition, minor modifications in the training set often lead to large changes in the final model due to the intrinsic instability of these algorithms [FG05]. Ensemble methods were proposed to take advantage of these unstable algorithms by growing a forest of trees from the data and averaging their predictions. They are covered in the next section.

2.2.4 Ensemble Methods

Techniques that aggregate the prediction of multiple models in order to improve predictive performance are known as ensemble methods. They tend to perform better than any single model alone when in conformity with two necessary conditions: (i) the base models should be independent of each other, and (ii) the base models should do better than a model that performs random guessing [TSK06]. Well-known examples of ensembles are:

1. Bagging [Bre96] - the training set is sampled (with replacement) according to a uniform probability distribution, and for each sample a base model is trained. For classification problems, data are classified by taking a majority vote among the predictions made by each base model. For regression, data are predicted by taking the average of the predictions made by each base model. Bagging's effectiveness depends on the (in)stability of the base models. If a base model is unstable, bagging helps to reduce the errors associated with random fluctuations in the training data (reduction of variance). Inversely, if a base model is stable, it means that the error of the ensemble is caused by bias in the base model, which cannot be solved by bagging.

2. Boosting [FS97] - the distribution of the training set is changed iteratively so that the base models will focus on examples that are hard to predict. Unlike bagging, boosting assigns a weight to each training instance and dynamically changes the weights at the end of each boosting iteration. Generally, the weights assigned to the training instances are used as a sampling distribution to draw a set of samples from the original data (meaning that instances with higher weights have higher probability of being drawn from the original data to train the base models. Data is predicted through a weighted voting scheme, where models with a poor predictive performance are penalized (e.g., those generated at the earlier boosting iterations).

3. Random forests [Bre01]: the training set is divided into a set of random vectors, generated from a fixed probability distribution. A random vector can be used to build base trees in many ways. One possible approach is to randomly select F input features (data set attributes) to split at each node of the tree. The disadvantage of this approach is the difficulty of choosing an independent set of random features when the number of original features is too small. To overcome this problem, the feature space can be increased by generating new features that are linear combinations of randomly selected features. A third approach would be to randomly

select one of the F best splits at each node of the tree, although it may potentially generate correlated trees, which is not desirable in the context of ensembles.

2.3 Chapter Remarks

Traditional decision/regression/model tree induction algorithms rely on a greedy top-down divide-and-conquer partitioning strategy. Such an heuristic is interesting because it produces trees efficiently and inexpensively. Nonetheless, there are at least two major problems related to this strategy [BBC+09a]: (i) the greedy heuristic usually produces locally (rather than globally) optimal solutions, (ii) recursive partitioning iteratively degrades the quality of the data set for the purpose of statistical inference, because the larger the number of times the data is partitioned, the smaller the data sample that fits the current split becomes, making such results statistically insignificant and contributing to a model that overfits the data.

The research community focused on the ensemble methods to alleviate the problems previously mentioned. All these ensemble methods have a major problem, which is the scheme for combining the generated trees. It is well-known that, in general, an ensemble of predictive models improves predictive performance by comparison with the use of a single model. On the other hand, the use of ensembles also tend to reduce the comprehensibility of the predictive model. A single comprehensible predictive model can be interpreted by an expert, but it is not practical to ask an expert to interpret an ensemble consisting of a large number of comprehensible models. In addiction to the obvious problem that such an interpretation would be time consuming and tedious to the expert, there is also a more fundamental conceptual problem. This is the fact that the models being combined in an ensemble are often to some extent inconsistent with each other – this inconsistency is necessary to achieve diversity in the ensemble, which in turn is necessary to increase the predictive accuracy of the ensemble. Considering that each model can be regarded as a hypothesis to explain predictive patterns hidden in the data, this means that an ensemble does not represent a single coherent hypothesis about the data, but rather a large set of mutually inconsistent hypothesis, which in general would be too confusing for an expert [BBC+09b].

In order to find a good trade-off between predictive performance and model comprehensibility, this work presents a novel algorithm based on the evolutionary algorithms paradigm [Gol89]. Instead of local search, evolutionary algorithms perform a robust global search in the space of candidate solutions [Fre08]. More on evolutionary algorithms is presented in the next chapter.

3

Evolutionary Algorithms

This chapter presents general concepts on evolutionary computation and focuses on a specific subset that is broadly used for optimization tasks: evolutionary algorithms (EAs). It describes briefly the two most common techniques used: genetic algorithms and genetic programming. It also presents some considerations regarding multi-objective optimization approaches.

3.1 Concepts

According to Olariu and Zomaya [OZ06], EAs are a collection of optimization techniques whose design is based on metaphors of biological processes. Freitas [Fre08] defines EAs as "stochastic search algorithms inspired by the process of neo-Darwinian evolution", and Weise [Wei09] states that "EAs are population-based metaheuristic optimization algorithms that use biology-inspired mechanisms (...) in order to refine a set of solution candidates iteratively".

The idea surrounding EAs is the following. There is a population of individuals, each one a possible solution to a given problem. This population evolves towards better and better solutions through stochastic operators. After the evolution is completed, the fittest individual represents a "near-optimal" solution for the problem at hand.

For evolving individuals, an EA evaluates each individual through a fitness function that measures the quality of the solutions that are being evolved. After all individuals from an initial population have been evaluated, the iterative process of the algorithm starts. At each iteration (hereby called *generation*), the best individuals have a higher probability of being selected for reproduction to increase the chances of producing good solutions. The selected individuals undergo stochastic genetic operators such as crossover (swapping of genetic material from two individuals) and mutation (modification of the genetic material of an individual), producing new offspring. These new individuals will replace the current population of individuals and the evolutionary process is repeated until a stopping criterion is satisfied (e.g., a fixed number of generations or until a satisfactory solution has been found). Figure 3.1 presents a typical EA flowchart.

Among the advantages of EAs is that their "black box" character makes only few assumptions

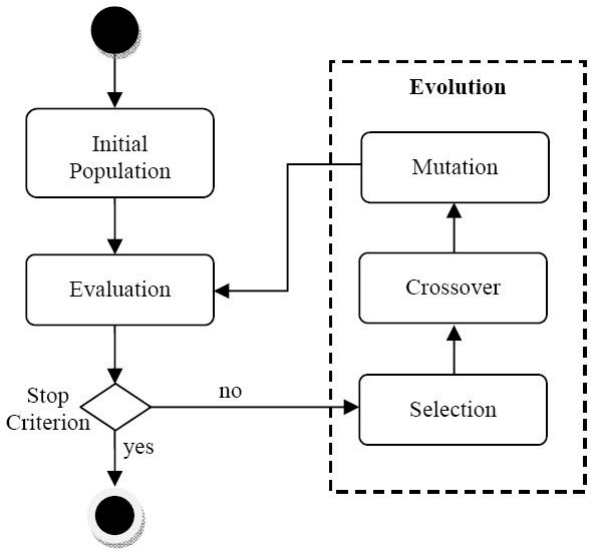

Figure 3.1: Typical evolutionary algorithm.

about the underlying fitness functions. Furthermore, the definition of fitness functions usually requires lesser insight to the structure of the problem space than the manual construction of an admissible heuristic, which enables EAs to perform consistently well in many different problem categories [Wei09].

There are several kinds of EAs, such as genetic algorithms, genetic programming, classifier systems, evolution strategies, evolutionary programming, estimation of distribution algorithms, etc. This chapter will focus on genetic algorithms (GAs) and genetic programming (GP), the two kinds of EAs most commonly used for data mining, according to Freitas, in [Fre08].

3.2 Genetic Algorithms and Genetic Programming

At a high level of abstraction, both GAs and GP can be described by the pseudocode in Algorithm 3.1. Even though GAs and GP share the structure depicted in this pseudocode, there are important differences between them.

GAs, which were initially presented by Holland in his pioneering monograph [Hol75], is defined as:

> (...) search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among *string structures*[our italics] with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search. (p. 1)

Representation is a key issue in GAs, and while they are capable of solving a great many problems, the use of fixed-length character strings may not work on a variety of cases. John Koza, the researcher responsible for spread the GP concepts to the researching community, argues in his text book on GP [Koz92] that the initial selection of string length limits the number of internal states of the system and

also limits what the system can learn. Moreover, he states that representation schemes based on fixed-length character strings do not provide a convenient way of representing computational procedures or of incorporating iteration or recursion when these capabilities are desirable or necessary to solve a given problem. Hence, he defines GP as:

> (...) a paradigm that deals with the problem of representation in genetic algorithms by increasing the complexity of the structures undergoing adaptation. In particular, the structures undergoing adaptation in genetic programming are general, hierarchical computer programs of dynamically varying size and shape. (p. 73)

Algorithm 3.1 Pseudo-code for GAs and GP. Source: [Fre08].

```
Create initial population
Calculate fitness of each individual
repeat
    Select individuals based on fitness
    Apply genetic operators to selected individuals, creating new individuals
    Compute fitness of each new individual
    Update the current population
until (stopping criteria)
```

Freitas [Fre08] affirms that the use of GP in the context of data mining is quite interesting once the dynamic representation of individuals provides GP with a potentially robust strategy for knowledge discovery purposes, which is not so naturally done when using GAs.

Although the definition of GP claims each individual to be a computer program or algorithm, in practice (in the context of data mining) most GP algorithms evolve a solution (for instance, a decision tree) specific for a single data set, rather than a generic program that can be used for deriving solutions (e.g., a decision tree induction algorithm such as C4.5). An exception is the work of Pappa and Freitas [PF09], where each GP individual is a full rule induction algorithm, that evolves in a grammar-based approach. This approach has the advantage of not limiting the evolution of solutions to a single data set, and also of avoiding the bias inserted by human beings coding.

The evolutionary algorithm proposed in this dissertation does not evolve an algorithm or computer program but solutions (model trees) for predictive regression problems. Nonetheless, its individual representation is not based on fixed-length character strings, but in dynamic tree structures. Thus, the proposed algorithm is presented as a GP-based approach instead of a GAs-based approach.

3.3 Multi-objective optimization

A crucial issue in data mining is how to evaluate the models that are generated. In many cases, researchers focus their attention on a single measure (objective) that indicates predictive performance of the models they generate. For instance, in classification models the goal is often to maximize accuracy (rate of correctly predicted labels) while in regression models the goal is to minimize a

generic error measure (such as the mean squared error). However, there are cases where we must pay attention to several objectives and it is not possible to concentrate on one of them. When this is the case, the problem is said to be a multi-objective or multi-criteria optimization problem [Koz92].

In a multi-objective problem, seeking for the optimal solution is not obvious. Consider the following example. Two model trees are generated for predicting a numeric value of a given data set. Model tree A has an error rate of 0.7 while model tree B has one of 0.9. If the goal is simply to minimize the error rate, model tree A would be selected as the optimal solution. However, model tree A has 79 leaf nodes (and thus 79 linear models) while model tree B has 10 leaf nodes (10 linear models). Model tree B may be the preferred option, since it is much smaller and consequently easier to interpret and less prone to overfiting. But what if there is a third model tree, C, with an error rate of 0.8 and 15 leaf nodes? Which one should we choose, B or C? The answer depends on the user needs and the problem domain. For instance, if we want to predict trends of the stock market, we would choose a predictive model that has the highest accuracy overall, whereas in a cancer detection system we would prefer a model that is biased towards detecting the disease, but not necessarily the one with highest accuracy overall. In software metrics estimation, we would like to have a predictive model that is both accurate and easy to interpret, so the project managers can make decisions to improve the estimation process and the software development process as a whole.

Freitas in [Fre04] discusses three general approaches to cope with multi-objective optimization problems: (i) weighted-formula; (ii) lexicographic analysis; and (iii) Pareto dominance. Next sections detail each of the mentioned approaches.

3.3.1 Weighted-formula approach

This is by far the most common approach in data mining applications [Fre04]. It transforms a multi-objective problem into a single-objective one by assigning numeric weights to each objective (evaluation measure) and then combining the values of the weighted criteria into a single value through arithmetic operations such as addition and multiplication. Typically, the quality Q of a given predictive model is given by one of the two kinds of formula [Fre04]:

$$Q = w_1 \times c_1 + w_2 \times c_2 + ... + w_n \times c_n \tag{3.1}$$

$$Q = c_1^{w_1} \times c_2^{w_2} \times ... \times c_n^{w_n} \tag{3.2}$$

where w_i,$i = 1, ..., n$, denotes the assigned weight to the c_i corresponding criterion (objective) and n the number of objectives. The quality measure Q becomes the sole objective and each predictive model is evaluated according to its value.

The weighted-formula approach has the advantage of being conceptually easy to use and implement, and also of being computationally inexpensive for appointing the optimal solution.

One of its disadvantages is the well-known "magic number" problem, which is the fact that, in general, weights are assigned in an ad-hoc fashion, based on the intuition of the user about the relative importance of each objective. Researchers generally justify their options with sentences like "the weights were empirically determined".

Another disadvantage is the fact that the manual setting of weights may prevent the predictive system to find a model with a better trade-off among different objectives. In particular, weighted-formula that are a linear combination of different objectives are quite problematic. Freitas [Fre04] presents the following scenario to show the limitations of linear combinations. Suppose we want to select a candidate for a position as a data miner in a company and we want to evaluate them according to two criteria: c_1, which is the knowledge of the candidate in machine learning and c_2, the knowledge in statistics. Suppose we have three candidates with the following scores: candidate 1, $c_1 = 9.5$ and $c_2 = 5$; candidate 2, $c_1 = 7$ and $c_2 = 7$ and candidate 3, $c_1 = 5$ and $c_2 = 9.5$. Freitas notices that while it is trivial to think of weights for the criteria that would make candidate 1 or candidate 3 the winner, it is impossible to choose weights for c_1 and c_2 that would make candidate 2 the new company employee, assuming the weighted-formula is a linear combination like in (3.1). The second candidate might be a better choice, whatsoever, once he/she has a more balanced knowledge in both subjects.

There is also another drawback of weighted-formula approaches, which is mixing non- commensurable criteria. For instance, consider we have two objectives, "liters of water wasted" and "age of the oldest family member". Common sense tells us that it does not make sense to add 500 liters to 83 years-old. The result would be meaningless 583 of an unknown unity. It should be noticed that the problem in here is not only the fact that measures with different units are being mixed, because the values could be normalized according to some criterion. The main problem is that the quantity produced would be meaningless to the user, which goes against the trend that discovered knowledge should be understandable to the user [FPSSU96].

3.3.2 Pareto-dominance approach

To illustrate the concept of Pareto-dominance or Pareto optimality, consider the following example presented by Goldberg [Gol89]. Suppose a widget manufacturer wishes to minimize both on-the-job accidents and widget cost. Both of these criteria are important to the successful operation of the plant, and furthermore, it is not trivial to estimate the cost of an accident. Suppose also the five scenarios below (A, B, C, D and E) which result in the following widget cost and accident count values, respectively:

A = (2, 10)

B = (4, 6)

C = (8, 4)

D = (9, 5)

E = (7, 8)

These data are plotted in Figure 3.2, a graph of accident count (y) versus widget cost (x). By scanning the graph, we can notice that the best points in our multi-objective optimization task are the ones nearer the origin $(0, 0)$, since both objectives should be minimized. More specifically, scenarios A, B and C seem to be good possible choices, though none is best for the two objectives. There are trade-offs from one of these three scenarios to another - for instance, scenario A has a better widget cost than scenario B but also a higher accident count. These three points are said to be *nondominated* because there are no points better than these for all criteria. Conversely, scenarios D and E are said *dominated* by another point. Scenario $E(7, 8)$ is dominated by $B(4, 6)$ once $4 < 7$ and $6 < 8$, whereas scenario $D(9, 5)$ is dominated by $C(8, 4)$ because $8 < 9$ and $4 < 5$.

In this example, instead of having a single optimal scenario, we have the Pareto optimal set {A, B, C}. We can state that a point (vector) $\mathbf{x}$ *dominates* $\mathbf{y}$, symbolically expressed by $\mathbf{x} \prec \mathbf{y}$, when the following conditions hold:

$$(\mathbf{x} \prec \mathbf{y}) \Leftrightarrow (\forall i)(x_i \leq y_i) \wedge (\exists i)(x_i < y_i) \tag{3.3}$$

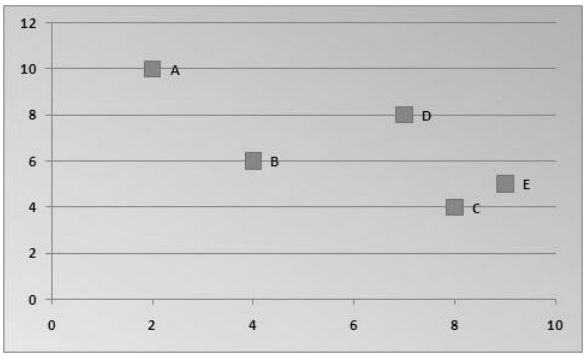

Figure 3.2: Five scenarios compared on the basis of widget cost (x) and accidents (y). Source: [Gol89].

In practice, the Pareto-dominance approach does not help the user to select a single alternative from the Pareto optimal set. The ultimate decision to which single option is best should be done by the decision maker, considering the application domain and his/her own expertise on the subject. This can be seen as an advantage, since data mining is only one step of a comprehensive and highly interactive knowledge discovery process.

Freitas [Fre04] argues that the difference between the weighted-formula approach and the Pareto approach is the timing when the user has to make a subjective decision. For instance, in the weighted-formula approach, the user has to make a choice of the weight values for each criterion, which is in practice a subjective decision made *a priori*, before the data mining algorithm is run. By contrast,

in the Pareto approach the user makes a subjective choice *a posteriori*, after he/she has seen all nondominated solutions returned by the data mining algorithm. We can notice that it is much more comfortable to analyze trade-offs associated with the different solutions produced by the algorithm and choose one based on our preference instead of making, in general, a very uninformed decision by choosing weights before the algorithm is run.

The robustness of the Pareto approach, however, comes with a price. When there is a high number of objectives to be optimized, the manual exploration of nondominated solutions may become tedious and even impracticable, due to the combinatorial explosion of trade-offs among objectives.

3.3.3 Lexicographic approach

The lexicographic analysis intends to determine priorities among the objectives, and the best solution is the one that is significantly better according to a higher-priority objective. If there is no decision whether a solution is better than the other in a given objective, the next objective is chosen in order of priority. To better exemplify this approach, consider the following example. Let x and y be two model trees and a and b two evaluation measures. Besides, consider that a has the highest priority between the measures and that t_a and t_b are tolerance thresholds associated with a and b respectively. The lexicographic approach works according to the following analysis: if $|ax - ay| > t_a$ then it is possible to establish which model tree is "better" considering objective a alone. Otherwise, the lower-priority measure b must be evaluated. In this case if $|bx - by| > t_b$ then the fittest tree between x and y can be decided by considering measure b alone. If it is still the case the difference between values falls within the assigned threshold t_b, the best value of the highest-priority measure a is used to determine the fittest tree.

The lexicographic approach might be an interesting choice considering that it recognizes the non-commensurability of the different criteria, and it allows the user to determine which criteria are more important without the need of identifying the correct weight of each measure, while preserving the simplicity of the weighted-formula approach and returning a single solution as the best one. Its disadvantage is the threshold values that must be defined *a priori*. Although defining thresholds can be critically appointed as a "magic number" problem, there is a commonplace approach to deal with this situation: statistics-oriented procedures [Fre04]. For instance, standard-deviation based thresholds that allow us to reject a null hypothesis of insignificant difference between two criterion values with a certain degree of confidence. This is a statistically sound solution for this approach, even though the degree of confidence is a parameter that has to be chosen.

3.4 Chapter Remarks

Evolutionary algorithms are a solid search-algorithms class for solving optimization problems. Its biological inspiration allows diversity of solutions, generally avoiding convergence to local optima.

In data mining, the two most commonly used approaches of EAs are GAs and GP. While sharing their basic structure, GAs differ from GP in the way they represent the candidate solutions: fixed-length character strings. In GP, the structures undergoing adaptation are computer programs of dynamically varying size and shape, which allows the development of potentially robust strategies for coping with knowledge discovery problems.

EAs evolve candidate solutions based on a defined fitness function that ranks the individuals according to some criteria. In real data mining applications, a good solution is often selected based on multiple objectives to be optimized, meaning that an ideal solution must excel in all criteria used to its evaluation, or the one that presents the better trade-off among criteria. Different approaches for multi-objective optimization were proposed in the literature, all presenting advantages and disadvantages. Among those, we highlight (i) the weighted-formula approach, which reduces the multi-objective problem into a single-objective one by assigning weights to the objectives and combining them into a single equation; (ii) the Pareto-dominance approach, which separates all solutions that are better than others in all objectives, in a Pareto optimal set; and (iii) the lexicographic analysis, which assigns priorities to the objectives and evaluates the solutions in a "cascade" fashion.

Next chapter details related works, which consist of evolutionary algorithms that evolve tree-structures for the data mining regression task.

4

Related Work

To the best of our knowledge, there are two works that relate evolutionary algorithms and trees used for mining continuous-valued classes: TARGET [FG05, GF08] and GPMCC [PE08], both presented in the next sections. We analyze them according to the following criteria: (i) initial forest initialization; (ii) fitness function; (iii) genetic evolution; and (iv) experimental results.

4.1 TARGET

TARGET [FG05, GF08] (Tree Analysis with Randomly Generated and Evolved Trees) is an EA proposed to evolve regression trees. Each candidate solution is a regression tree of variable size and shape and random initialization.

4.1.1 Initial forest

The initial population consists of 25 randomly created trees. The authors claim to have experimented with forest sizes between 10 and 100 trees, but with no apparent impact on the outcome. The random generation of trees starts with a single root node, with probability p_{split} that dictates whether the node is split and two child nodes are created or the node becomes a terminal node. If the node is split, then an attribute from the data set and a split value are randomly chosen from a pool of candidate attributes and split values. The default value of p_{split} is 0.5.

After the tree is randomly created in TARGET, the training data are applied to the tree from the root down to the leaves. Since the split rules are randomly chosen, some nodes might be empty or have a value of observations below the minimum required. These nodes are pruned from the tree before the evaluation step is performed.

4.1.2 Fitness Function

For evaluating the trees, TARGET makes use of the Bayesian information criterion (BIC) as a measure of tree fitness. BIC, which was proposed by Schwarz [Sch78], is a statistical model selec-

tion criterion based on likelihood. In other terms, it is a weighted-formula that penalizes for model complexity (size of the tree). It is expressed as

$$BIC = max(ln\, L) - \frac{1}{2} p\, ln\, n = -\frac{n}{2} ln\, 2\pi - \frac{n}{2} ln \frac{SSE}{n} - \frac{n}{2} - \frac{1}{2} p\, log\, n \tag{4.1}$$

where p is the effective number of parameters in the tree model, L is the likelihood function and the maximum is taken over the possible values of the terminal node means and the constant variance. The last term in (4.1) is the model complexity penalty, which is a function of both the effective number of parameters p and the training sample size n. SSE is the residual sum of squares, expressed as

$$SSE = \sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2 \tag{4.2}$$

where y_i is the actual value of the i^{th} instance and $\hat{f}(x_i)$ the predicted tree value for the same instance.

One critical problem of using the BIC criterion in TARGET is determining the value of p, essential part of the model complexity penalty term. Increasing p by too much leads to smaller trees with less predictive performance. The authors state that further research is required to determine the appropriate adjustment of the model complexity penalty term.

4.1.3 Genetic Evolution

TARGET implements the elitism technique, meaning that the best trees in the current generation are kept to the next generation. The default elitism value is 3. It also implements two different crossover types: (i) subtree swap crossover, where entire subtrees are swapped between two selected trees; and (ii) node swap crossover, where only split rules (attribute and split value) are actually swapped. TARGET generates two offspring, and the best of the four (parents and offspring) is kept to the next generation.

TARGET implements four distinct types of mutation, with probability proportional to fitness, and the user specifies the number of mutations that are to occur at each generation (default value is 2). The possible types of mutation are: (i) split set mutation: randomly change the split value of a randomly selected node; (ii) split rule mutation: randomly change both attribute and split value of a randomly selected node; (iii) node swap mutation: swap both attribute and split value of two randomly selected nodes within the same tree; (iv) subtree swap mutation: randomly swap subtrees of two randomly selected nodes within the same tree.

Finally, TARGET implements the "transplant" operation, which consists of adding new randomly generated trees in each new generation. Default value of transplant is 3. The new generation of trees after each iteration is composed of 25 trees resulting from elitism (3), crossover (19) and transplant (3). The stopping criterion of the algorithm is the lack of improvement in the best trees of the generations, though the authors do not make clear which is the default number of generations without improvement.

4.1.4 Experimentation

TARGET is compared to the traditional greedy regression tree induction algorithm CART, to Bayesian CART [CGM97, DMS98] (Markov chain Monte Carlo technique based algorithm) and also to the ensemble methods Random Forests and Bagging.

The first comparison is with Bayesian CART. It uses the published examples and results in [CGM97] and [DMS98]. In the data set used by Chipman et al. [CGM97], TARGET manages to identify the correct tree model (R^2 of .6457 with 5 terminal nodes) in 6 out of 10 runs. Bayesian CART identified the correct tree model in only 2 out of 10 runs. In the data set proposed by Denison et al. [DMS98], TARGET presented smaller trees than Bayesian CART, though with similar residual sum of squares (RSS) values.

The comparison with CART, Random Forests and Bagging was made through two public regression data sets from UCI [AN07]: Boston housing and Servo. TARGET outperformed CART in terms of mean squared error, but it is outperformed by both ensemble methods. The authors claim that TARGET presents a good trade-off in terms of accuracy (error measures) and model interpretability (final result is a single tree), whereas CART presents only good interpretability and the ensemble methods only a good accuracy.

4.2 GPMCC

GPMCC [PE08] (Genetic Programming approach for Mining Continuous-valued Classes) is a framework proposed to evolve model trees. Its structure is divided in three different parts: (1) GASOPE [PE07], which is a genetic algorithm to evolve polynomial expressions; (2) K-Means [Llo82], which is a traditional clustering algorithm and is used in this framework to sample the training set; and (3) a GP to evolve the structure of model trees.

4.2.1 Initial Forest

Trees are generated by randomly expanding a node and randomly selecting attributes and split values. The growth of the trees is dictated by a parameter that indicates the maximum tree depth. Default value of population size is 100.

4.2.2 Fitness Function

The fitness function used by GPMCC is an extended form of the *adjusted coefficient of determination* (4.3)

$$R_a^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y}_i)^2} \times \frac{n-1}{n-d} \tag{4.3}$$

where n is the size of the sample set, y_i is the actual output of the i^{th} instance, $\hat{y}_i$ is the predicted output for the same instance, $\overline{y}_i$ is the average of all instances and d is a complexity factor that penalizes the size of an individual (in number of nodes) and the complexity of each model of the terminal nodes (number of terms and their corresponding order). The higher the value of complexity factor d, the lower the value of R_a^2. The best individuals are those with the highest values of R_a^2.

4.2.3 Genetic Evolution

GPMCC implements the crossover operator as follows. Two individuals A and B are chosen by tournament selection. A non-terminal node N_A is randomly selected from A and a node N_B from B. A new individual C is created by installing N_B as a child of N_A. It is not clear though which child of N_A is replaced by N_B.

GPMCC makes use of 8 different mutation strategies: (i) expand-worst-terminal-node; (ii) expand-any-terminal-node; (iii) shrink; (iv) perturb-worst-non-terminal-node; (v) perturb-any-non-terminal-node; (vi) perturb-worst-terminal-node; (vii) perturb-any-terminal-node; and (viii) reinitialize.

The expand operators (i) and (ii) partition a terminal node (the one with highest mean squared error or any node) in an attempt to increase the fitness of an individual. The shrink operator (iii) replaces a non-terminal node of an individual with one of its children. The perturb operators (iv), (v), (vi) and (vi) seek to modify the split values of non-terminal nodes or to replace the non-regression expressions of the terminal nodes with expressions of the fragment pool. Finally, the reinitialize operator (viii) is a reinvocation of the procedure that expands a node into a tree according to an informed maximum depth (this procedure is used for building the initial forest of individuals).

Fragment Pool

GPMCC uses GASOPE [PE07], a genetic algorithm that evolves polynomial expressions to be used in the terminal nodes of the model trees. K-Means is used to cluster the training data, and GASOPE produces a non-linear regression model from each cluster. The number of clusters is iteratively reduced in order to produce non-linear regression expressions that vary from specific to more general approximations (the higher the number of clusters, the more specific the expressions, and vice-versa). The expressions generated in this fragment pool receive a lifetime mapping so those that are used the most in the model trees have their lifetime index reset (i.e., their lifetime increases) while non-used expressions are taken out of the pool. The lifetime index ensures the usefulness of the created expressions.

4.2.4 Experimentation

GPMCC is compared to a neural network approach called NeuroLinear [SLZ02] and to a commercial version of Quinlan's M5 [Qui92], in various data sets from UCI [AN07]. GPMCC outperformed

these approaches only by the number of rules generated (paths of the tree), and was consistently outperformed in terms of accuracy (mean absolute error).

One important remark is that GPMCC has a total of 42 configurable initialization parameters that control several steps of the algorithm. The authors claim that the influence of GPMCC parameters on performance was empirically investigated and that it was shown that performance was insensitive to different parameter values. Since the authors make such claim, one has to ask the real utility of presenting so many configurable options that in practice will not alter the algorithm's results. As a second point of criticism, GPMCC seems to be overly-complex, presenting results that do not justify all the choices made during the algorithm's development.

4.3 Chapter Remarks

We present our approach for evolving model trees (E-Motion) in the next chapter. It differs from TARGET since we are evolving model trees and not regression trees. It is important to notice that model trees are a more robust and effective approach when compared to regression trees, often presenting more accurate results [WF05]. Our strategy, while having the same goal as the GPMCC framework, makes use of a single evolutionary algorithm to evolve model trees, which makes the induction of model trees faster and simpler. Our goal throughout this work was to achieve at least the same level of accuracy than GPMCC and traditional algorithms such as M5, but with more legible trees (trees with fewer nodes).

Table 4.1 summarizes the typical steps of an evolutionary algorithm adopted in TARGET and GPMCC. This table is further filled in after we present E-Motion, in the next chapter.

Table 4.1: Related work comparison.

Algorithm	Initial Forest	Fitness	Selection	Crossover	Consistency Check	Stopping Criteria
TARGET	random controlled through p_{split}	W.F	roulette	2 children, 1 kept for next gen	node pruning	n^o gen. without improvement
GPMCC	random controlled through $maxDepth$	W.F	tournament	1 child, 1 kept for next gen.	no	max number of gen.

5

Evolutionary Model Tree Induction (E-Motion)

E-Motion is a novel multi-objective genetic programming algorithm for model trees induction. Each step of E-Motion is presented in this chapter following the flow previously depicted in Figure 3.1. In addition, we describe two specific steps that are not presented in Figure 3.1: (i) consistency check; and (ii) prediction smoothing.

5.1 Solution Representation

We adopt the tree representation to represent the candidate solutions of E-Motion, because it seems logical that if each individual is a model tree, the solution is best represented as a tree. Thus, each individual is a set of nodes, with each node being either a non-terminal or a terminal node.

Each non-terminal node contains an attribute, and each leaf node contains a linear regression model that is calculated after each genetic iteration (generation). A set of edges linking each non-terminal node with its respective children is also a part of the tree representation. In E-Motion, every node-split is binary.

There are two distinct possible cases of node relations: (i) the relationship between a categorical node and its children: if a node x represents a categorical attribute, there will be 2 edges representing different categories aggregations, which will be further explained in Section 5.2; and (ii) the relationship between a numeric continuous node and its children: if a node x represents a numeric continuous attribute, there will be a binary split according to a given threshold chosen by the algorithm, also further clarified in Section 5.2. Figure 5.1 presents an example of an E-Motion individual.

5.2 Initial Forest

The initial forest generation implemented by E-Motion is twofold. First, the algorithm produces *basic trees*, based on each data set attribute. Basic trees are trees with a root node and two children that are terminal nodes.

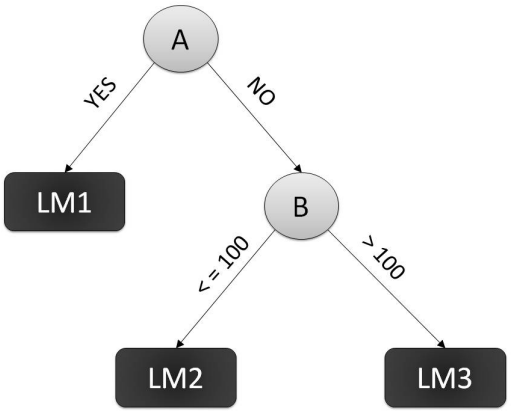

Figure 5.1: E-Motion individual.

For creating basic trees where the root node is a categorical attribute, the average target-attribute value corresponding to each possible category in the enumeration is calculated from the training instances, and the values in the enumeration are sorted according to these averages. Then, if the categorical attribute has k categories, it is replaced by $k-1$ *synthetic binary attributes*, the i^{th} being 0 if the value is one of the first i in the ordering and 1 otherwise. This strategy is also implemented in M5 [Qui92].

Witten and Frank [WF05] state that it is possible to prove analytically that the best split at a node for a categorical attribute with k values is one of the $k-1$ positions obtained by executing the process aforementioned. Even though this sorting process should be executed each time a node is included in a model tree, there is no significant loss by performing such ordering just once, before starting to build the tree [WF05]. Hence, for each categorical attribute, E-Motion produces $k-1$ different basic trees, where k is the number of categories the attribute owns.

To exemplify the categorical basic trees generation, consider the following scenario. The categorical attribute $Algorithm$ has 5 categories: CART, M5P, REPTree, Neural Networks and Random Forests. The target attribute is $ErrorValue$. E-Motion calculates the average value of $ErrorValue$ grouping the data set instances by the attribute $Algorithm$ categories. The result is the following: CART = 0.75, M5P = 0.64, REPTree = 0.83, Neural Networks = 0.59 and Random Forests = 0.58. These values are sorted resulting in the following order: Random Forests, Neural Networks, M5P, CART and REPTree. Since we have 5 categories, we will have 4 different binary splits, by aggregating the categories in: (i) {Random Forests} X {Neural Networks, M5P, CART, REPTree}; (ii) {Random Forests, Neural Networks} X {M5P, CART, REPTree}; (iii) {Random Forests, Neural Networks, M5P} X {CART, REPTree}; and (iv) {Random Forests, Neural Networks, M5P, CART} X {REPTree}. These 4 possible binary splits will generate 4 categorical basic trees, as presented in Figure 5.2.

For numeric attributes, E-Motion implements two different strategies that seek to incorporate task-specific knowledge in order to derive reasonable threshold values. Both consist of dividing the

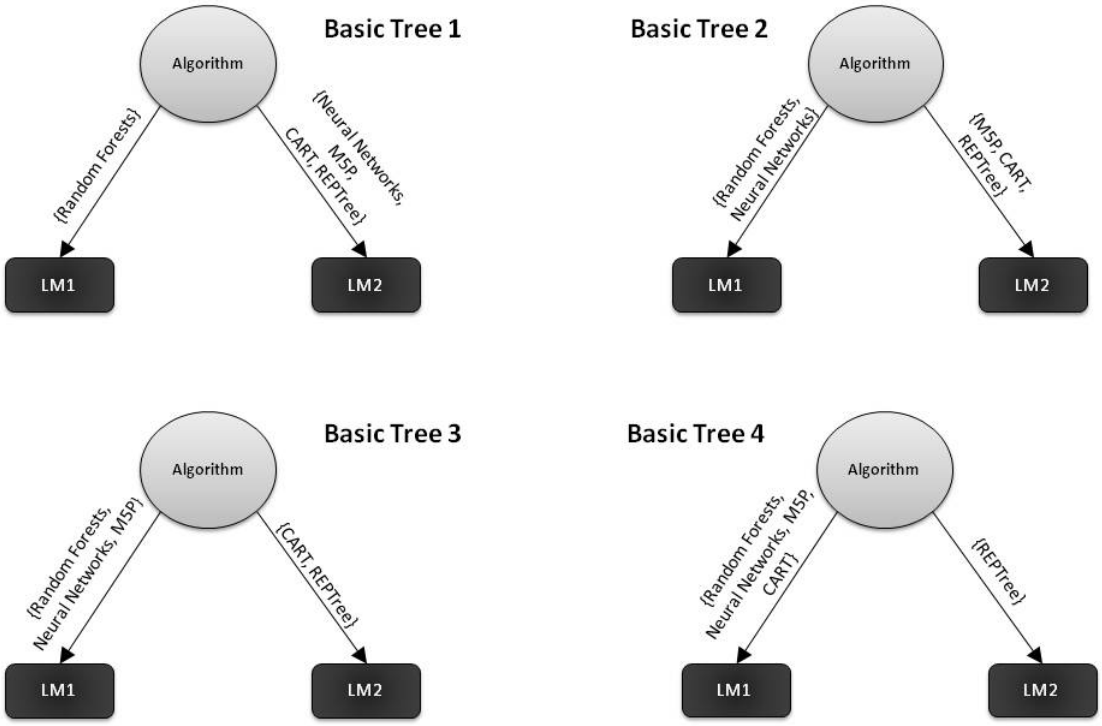

Figure 5.2: 4 categorical basic trees created from a 5-category attribute.

training set in different pieces and calculating the SDR of each piece, generating five different basic trees with possibly different threshold values. The difference between them is in the way the training set is divided.

In the first approach, E-Motion calculates the SDR of the entire training set and then divides it in four random equally-sized pieces, calculating the SDR of each piece. This strategy is depicted in Algorithm 5.1.

In the second strategy, the training set is clustered in 5 clusters with K-Means[1] [Llo82], and the SDR of each cluster is calculated, generating 5 numeric basic trees with possibly different threshold values. The distance function used is the Euclidean Distance. This strategy is presented in Algorithm 5.2.

These approaches have two main advantages: (a) the thresholds are defined in a data-driven manner (i.e., by using the standard deviation reduction for selecting interesting threshold values), instead of selecting random values, which is the case of most evolutionary approaches; and b) a certain degree of heterogeneity is achieved by partitioning the training set into different pieces (random pieces or clustered pieces), increasing the chances of selecting a good threshold value. Both advantages are a result of incorporating general knowledge about the regression task being solved into the GP

[1] Appendix 7 presents brief information on K-Means.

Algorithm 5.1 Pseudo-code for random numeric basic trees generation.

```
Let x be the training data set
Let n be the number of data set attributes
Let a[i] be the i^th attribute of x
for i = 1 to n do
   root = a[i]
   threshold = SDR(a[i], x)
   new basicTree(root,threshold)
   Divide x in 4 different random pieces
   Let y[k] be the k^th piece of x
   for k = 1 to 4 do
      threshold = SDR(a[i], y[k])
      new basicTree(root,threshold)
   end for
end for
```

algorithm, which tends to increase its effectiveness [BBC+09b, BBR+10].

Algorithm 5.2 Pseudo-code for clustered numeric basic trees generation.

```
Let x be the training data set
Let n be the number of data set attributes
Let a[i] be the i^th attribute of x
Cluster x in 5 clusters
Let y[k] be the k^th cluster of x
for i = 1 to n do
   root = a[i]
   k := 1
   repeat
      threshold = SDR(a[i], y[k])
      new basicTree(root,threshold)
      k := k + 1
   until k = 6
end for
```

The choice of which approach to use for generating numeric basic trees is a user-defined parameter. After creating the basic trees, E-Motion aggregates them to build the initial forest. The user sets the maximum depth value for the trees that will be part of the initial forest, and E-Motion randomly combines different basic trees so as to create a tree with depth that can range from 1 to the maximum depth value informed. Figure 5.3 shows this rationale, where basic tree A was chosen as the root node and its leaves were replaced by basic trees B and C.

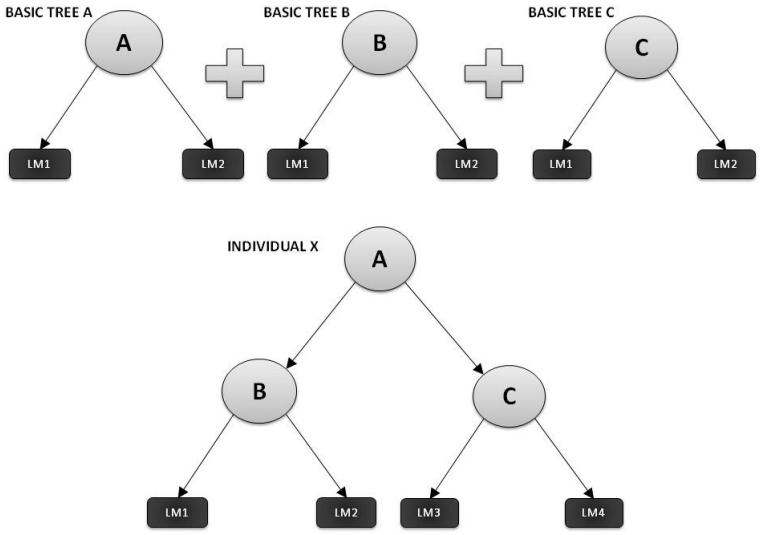

Figure 5.3: Merging basic trees A, B and C into an individual.

5.3 Fitness evaluation

After generating an initial forest, E-Motion evaluates each single tree in terms of its predictive performance. It makes use of two error measures to evaluate the fitness of each model tree: root mean squared error (RMSE) and mean absolute error (MAE). The mean squared error is the most common measure to evaluate numeric predictions. We take the square root to give it the same dimensions as the predicted value itself. The RMSE is given by

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}} \tag{5.1}$$

where y_i is the actual value of the target attribute and $\hat{y}_i$ is the estimated value of the target attribute. MAE, which is another common choice for evaluating regression problems, is given by

$$MAE = \frac{\sum_{i=1}^{n}|\hat{y}_i - y_i|}{n} \tag{5.2}$$

It is known that any given evaluation measure is biased to some extent. RMSE for instance is quite sensitive to outliers, while MAE treats all size of errors evenly according to their magnitude. It is clear that each evaluation measure can serve a different purpose and its use depends on a large extent upon the objectives of the prediction system user.

We have chosen these two error measures because they are the most common for regression problems, and even though they present significant differences when dealing with particular cases, it turns out that in most practical situations the best numeric prediction method is still the best for whatever

error measure used.

Although predictive performance is clearly of great importance and has been the primary focus of researchers when developing prediction models, we believe it is not sufficient to indicate how robust a predictive approach is. Comprehensibility or the capability of explaining to the end-user how a prediction was obtained is crucial in several applications. Understanding the predictions made by a model helps the end-user to get more confidence in the prediction, and more importantly, can provide the basis for the end-user to have new insight about the data, confirming or rejecting hypotheses previously formed. A comprehensible model can even allow the end-user to detect errors in the model or in the data [FWA08].

Burgess and Lefley [BL01] suggest that an evolutionary algorithm that optimizes a single evaluation measure is faded to degrade the other measures, and that a fitness function that is not tied to one particular measure may present more acceptable overall results. Based on such assumption, we have decided to use three different measures to evaluate how fit an individual is: RMSE and MAE as error measures and tree size (number of nodes) as a measure of model comprehensibility, assuming that smaller trees are more comprehensible than larger ones.

Since E-Motion has to deal with three different measures to evaluate each tree, it was necessary to define which strategy would be used to cope with the multi-objective optimization. In Section 3.3 we have presented three well-known strategies for handling multi-objective optimization. E-Motion implements two of them: weighted-formula and lexicographic analysis. It does not implement the Pareto-dominance approach due to time restrictions but we propose to extend E-Motion by implementing the Pareto approach in a near future and test it properly.

5.3.1 Weighted-Formula

The weighted-formula approach was implemented through the equation defined in (5.3).

$$\begin{aligned} Fitness(I_x) = 0.4 \times (\frac{RMSE(I_x) - argmin(RMSE)}{argmax(RMSE) - argmin(RMSE)}) + \\ 0.4 \times (\frac{MAE(I_x) - argmin(MAE)}{argmax(MAE) - argmin(MAE)}) + \\ 0.2 \times (\frac{Size(I_x) - argmin(Size)}{argmax(Size) - argmin(Size)}) \end{aligned} \tag{5.3}$$

where I_x is a given individual of the current population P. The highest value of a given measure is given by $argmax(measure)$ (5.4) and the lowest value by $argmin(measure)$ (5.5).

$$argmax(measure) = \{measure(I_n) \mid \forall y \in P, measure(I_n) \geq measure(I_y)\} \tag{5.4}$$

$$argmin(measure) = \{measure(I_n) \mid \forall y \in P, measure(I_n) \leq measure(I_y)\} \quad (5.5)$$

The idea behind this weighted-formula is to give the same priority to the error measures ($RMSE$ and MAE) and a lower priority to tree size. The choice of weights (40% for the error measures and 20% for tree size) was empirically defined, based on common-sense and non-exhaustive experimentation (as discussed before, this approach suffers from the "magic number" problem, i.e., how to select the "correct" weights?).

5.3.2 Lexicographic Analysis

E-Motion also implements the lexicographic analysis of the error measures and tree size. The priority set was: (i) $RMSE$; (ii) MAE; and (iii) tree size, respectively.

Differently from the weighted-formula approach, now one of the error measures ($RMSE$) has priority over the other (MAE). We have chosen $RMSE$ as the highest-priority measure because it is the principal and most commonly used measure [WF05]. MAE comes second because the goal is to seek for high predictive performance. And finally tree size is used to decide which model is best among those with similar error values.

The lexicographic analysis works as described in Algorithm .2 - Appendix 7. Basically, given two individuals I_A and I_B, it evaluates which is best by analyzing, in order of priorities, each measure. For instance, it checks if the difference of $RMSE$ between the two individuals falls within a given t_{RMSE} threshold. If the difference is higher than the threshold, the individual with the lower value of $RMSE$ is the fittest one. Otherwise, it checks if the difference of MAE between the two individuals falls within a given t_{MAE} threshold. The same analysis is made, and tree size is the last measure to be checked, in case there is no clear winner regarding MAE values.

In case all differences fall within the thresholds assigned to each measure, the lowest absolute value of $RMSE$ indicates the winner. If it is the case both individuals have the exact same value or $RMSE$, the lowest absolute value of MAE is evaluated, and so on. If both individuals have the exact same values of $RMSE$, MAE and tree size, any of them can be considered the winner of the analysis.

Even though the lexicographic analysis is made 2 individuals per time, all individuals are ranked according to the results in a list, from the fittest to the worst individual. It should be noticed that the error measures calculations are made considering the validation set. This set is a 30%[2] sample from the data set and it is used exclusively for fitness calculation with the purpose of avoiding data overfiting.

The threshold values assigned to each measure were defined as follows. For the error measures, a tolerance of 5% of the average value of each measure in a population is used to indicate which solution

[2]We divide the data set in 60% for training, 30% for validation and 10% for testing, as recommended in [WF05].

is best. For tree size, a tolerance of 20% the average tree size of individuals within the population was chosen. To illustrate the rationale behind the choice of these threshold values, suppose we have two individuals I_A and I_B with the following measure values:

I_A	I_B	Pop. Average
$RMSE = 0.7355$	$RMSE = 0.7588$	$RMSE = 0.7740$
$MAE = 0.8640$	$MAE = 0.9020$	$MAE = 0.9544$
Size $= 9$	Size $= 7$	Size $= 12$

The tolerance thresholds regarding the population average are: $RMSE = 0.0387$, $MAE = 0.0477$ and Size $= 2.4$. The difference in $RMSE$ between I_A and I_B (0.0233) falls within the tolerance threshold, so the next measure is analyzed. The difference in MAE (0.038) also falls within the tolerance threshold, so tree size is now analyzed. Since the difference in tree size (2 nodes) also falls within the tolerance threshold, the best individual is I_A for having the lowest absolute value of $RMSE$.

Several tolerance threshold values were tested non-exhaustively, and the tolerance thresholds of 5% and 20% seem to provide a good selection of the best trees. As future work, a statistics-oriented procedure based on standard deviation will be implemented to automatically select interesting threshold values.

5.4 Selection

E-Motion uses tournament selection, a popular and effective selection method. t individuals from the current population are selected randomly (default value is $t = 3$), and they "battle" against each other, i.e., the fittest individual is chosen to undergo crossover or mutation. This process is repeated until the reproduction pool is full. E-Motion also implements the elitism technique, which means it preserves a percentage x of the fittest individuals of the current population for the next one (default value is 5%). Figure 5.4 shows the rationale of the tournament selection.

5.5 Crossover

E-Motion implements the crossover operation as follows. First, two individuals randomly chosen among the selected ones (selection operation) will exchange sub-trees. According to a randomly selected number which varies from 1 (root node) to n (total number of tree nodes), E-Motion performs an adapted pre-order tree search method, visiting recursively the root node and then its children from left to right. The search method is equivalent to the traditional binary pre-order search. After identifying the nodes according to the randomly selected number in both parents, E-Motion will exchange the whole sub-trees which are represented by the selected nodes, generating two new individuals (offspring). Figure 5.5 illustrates the crossover operation, where the offspring is created by keeping the

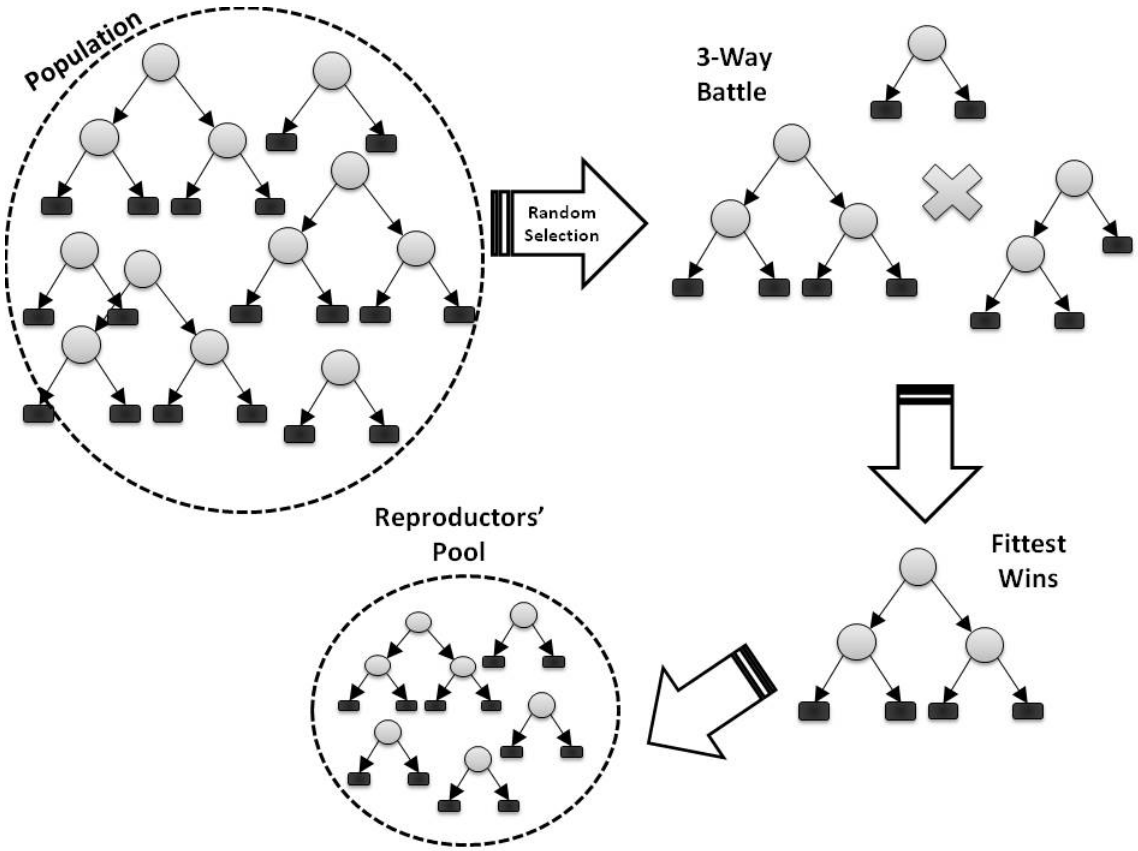

Figure 5.4: Tournament selection.

structure of the parents but with the selected nodes being replaced. Linear models are updated after the genetic operators take place. In the figure, we suppose the children's linear models have already been updated (that is the reason for the apparent change in the numbering of each LM).

By exchanging the whole sub-trees from the selected nodes and not only specific nodes, we avoid domain irregularities, because each edge refers to attribute characteristics that are represented by a node. It does not prevent, however, redundant rules and inconsistencies. See Section 5.7 for details on how E-Motion addresses these issues.

5.6 Mutation

E-Motion implements two different strategies for mutation of individuals. The first one considers the exchanging of a whole sub-tree, selected randomly from an individual, by a leaf node, acting like a pruning procedure. The second strategy replaces a randomly selected leaf node in an individual by a basic tree generated during the initial forest creation. Figure 5.6 presents both strategies.

These strategies aim at increasing or diminishing the individual's size, improving the heterogeneity of the population and avoiding convergence to local optima.

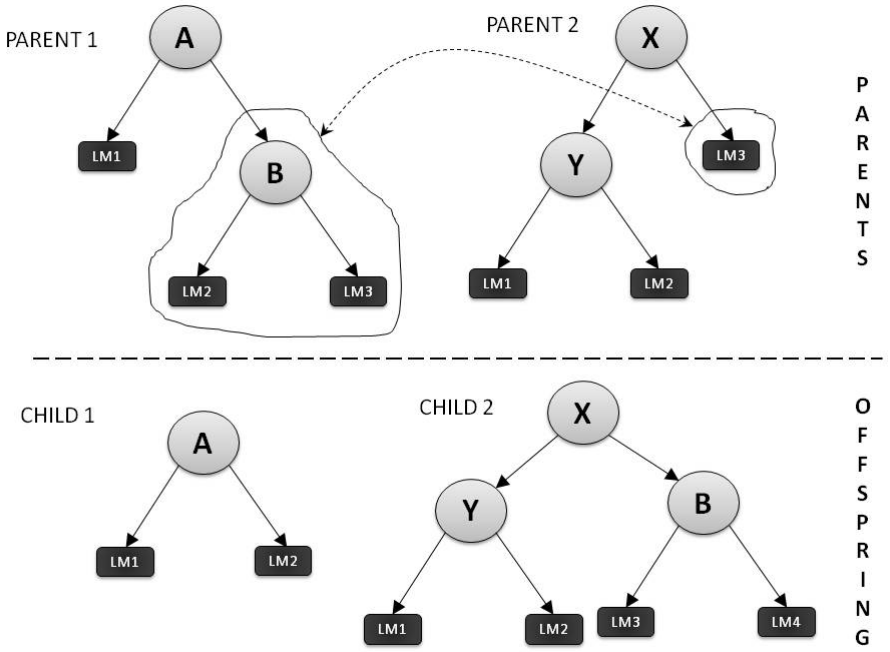

Figure 5.5: Crossover between two individuals and the resulting offspring.

5.7 Consistency check

The stochastic operators of the evolutionary algorithm, due to their own nature, may create incoherent scenarios regarding the logical structure of model trees. E-Motion implements a filtering process to deal with these validity issues.

After each evolutionary iteration and before each individual is evaluated, the training data set is distributed along each individual so the linear models are calculated according to the instances that reach the leaf nodes. The following inconsistent scenarios may occur:

- A categorical attribute appears more than once in a sub-tree. It is well-known that the same categorical attribute test should not be done more than once in a same sub-tree, thus such a situation should be prevented. E-Motion deals with this case by pruning the sub-tree in which the categorical attribute appears more than once.

- Incoherent thresholds for repeated numeric nodes. If a numeric attribute appears in a same sub-tree more than once, a special caution should be taken regarding the threshold values. For instance, if a given numeric attribute x is tested according to a threshold 10, the sub-tree that results from the test $x < 10$ may possibly contain another test over the attribute x, but no threshold larger than 10 would result in a node that holds instances. Incoherent thresholds also should be prevented from happening. E-Motion recalculates the SDR and, thus, the thresholds for these cases, preventing the existence of incoherent thresholds.

This filtering process generally provides smaller trees that are consistent with the training data set.

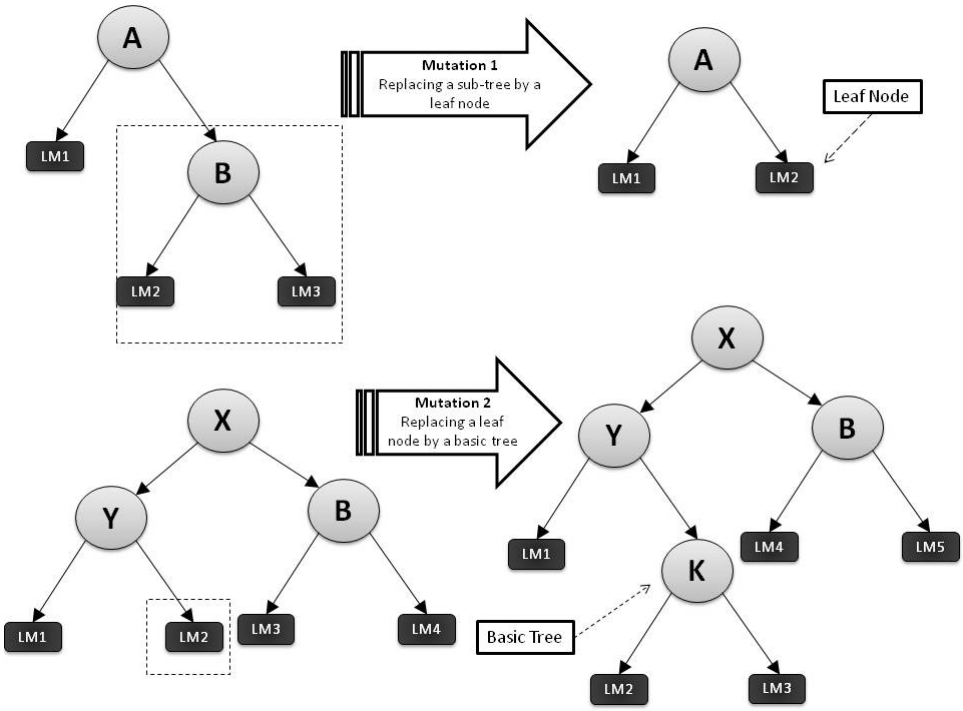

Figure 5.6: Two distinct kinds of mutation.

Figure 5.7 presents both cases, with the original and filtered trees.

5.8 Stopping criteria

E-Motion implements two distinct stopping criteria. The first one is the maximum number of generations (iterations) in which the solutions will evolve. Such a number is a user-defined parameter (E-Motion's default is 200 generations). The second criterion is the maximum number of generations without improvement in the best solution, which is a percentage of the maximum number of generations. It is also a user-defined parameter (E-Motion's default is 10% the max number of generations).

5.9 Prediction Smoothing

In a model tree, for predicting the outcome of a test instance, we have to walk along the tree until reaching a leaf node, using the instance's attribute values to make routing decisions at each node. The leaf contains a linear model that was built according to training instances, and it is used for the test instance to yield a predicted value.

Quinlan [Qui92] states that the prediction accuracy of tree-based models can be improved by a smoothing process. This process is described as follows: (i) generate a linear model for each node of the tree (instead of generating linear models for the leaves only); and (ii) once the leaf model has been used to obtain the predicted value for a test instance, filter this value along the path back to the

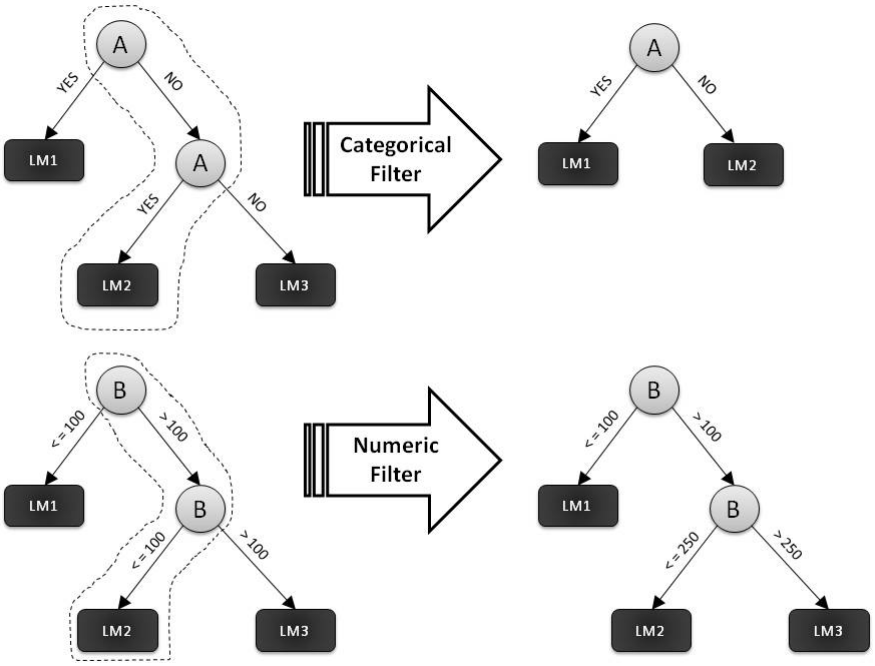

Figure 5.7: Filtering process.

root, smoothing it at each node by combining it with the value predicted by the linear model for that node. Figure 5.8 presents a draft of this process.

The combination of values in the smoothing process is given by (5.6):

$$p' = \frac{(n \times p) + (k \times q)}{n + k} \tag{5.6}$$

where p' is the prediction that will be passed up to the next higher node, p is the prediction passed to this node from below, q is the value predicted by the linear model at this node, n is the number of training instances that reach the node below and k is a smoothing constant. The default value for the smoothing constant is 15, as suggested by Quinlan [Qui92].

Smoothing achieves good results specially on cases when models along the path predict very different values and when models are constructed from few training cases [Qui92]. The smoothing process also compensates for the sharp discontinuities that inevitably occur between adjacent linear models at the leaves of pruned trees [WF05].

5.10 Chapter Remarks

E-Motion is a multi-objective genetic programming algorithm that seeks a trade-off between predictive performance and model comprehensibility. It implements two different strategies for coping

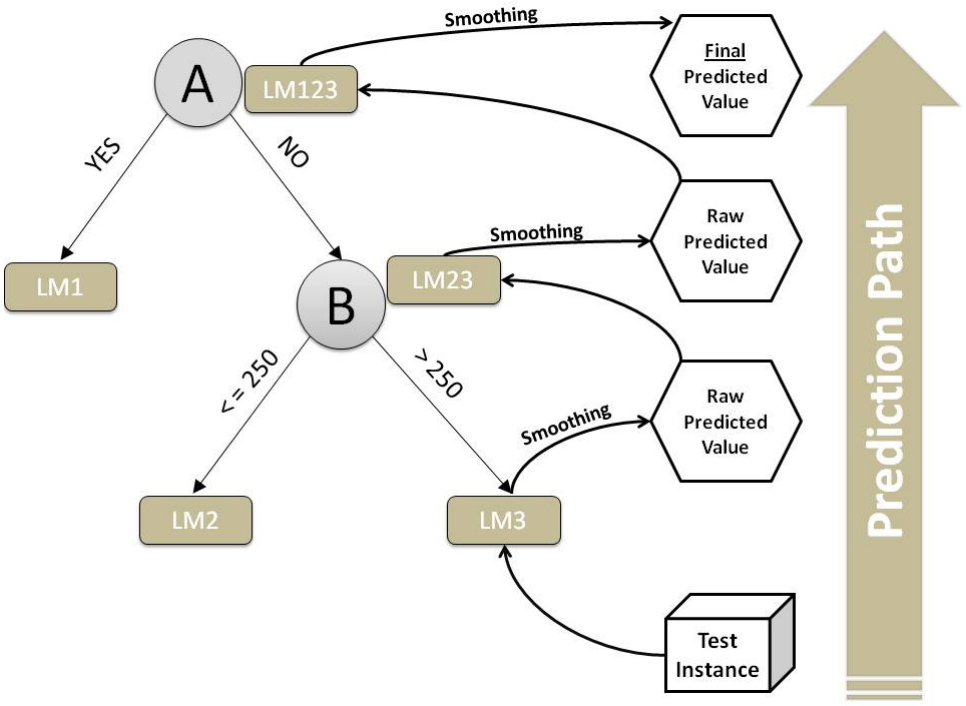

Figure 5.8: Smoothing process.

Table 5.1: Algorithms comparison.

Algorithm	Initial Forest	Fitness	Selection	Crossover	Consistency Check	Stopping Criteria
TARGET	random controlled by p_{split}	W.F	roulette	2 children, 1 kept for next gen	node pruning	n^o gen. without improvement
GPMCC	random controlled by $maxDepth$	W.F	tournament	1 child, 1 kept for next gen.	no	max number of gen.
E-Motion	data-driven controlled by $maxDepth$	W.F and Lex.	tournament	2 offspring, 2 kept for next gen.	node pruning and thresholds adjustment	max number of gen. and n^o gen. without improvement

with the multi-objective optimization problem. It also implements the traditional genetic operators such as selection, crossover and mutation. Moreover, E-Motion avoids inconsistent scenarios by filtering each individual after it undergoes the genetic operators. Table 5.1 includes E-Motion in the comparison among related work.

We consider the initial forest initialization of E-Motion an advantage over the other algorithms, once we are trying to reduce the search space through a data-driven heuristic, which may help speeding the convergence to the final solution. The possibility of choosing between two distinct types of multi-objective optimization is also a clear advantage of E-Motion. Furthermore, E-Motion's consistency check seems to implement a more robust strategy when compared to the other approaches. Finally, E-Motion implements two stopping criteria, which may also help speeding the algorithm's convergence.

6

Evaluating E-Motion

This chapter presents some experiments conducted in order to evaluate E-Motion in terms of predictive performance and comprehensibility. It is divided in two stages. The first one was published in the proceedings of the *2010 ACM Symposium of Applied Computing*, in the track of evolutionary computation [BBR+10], where 8 UCI data sets were used to compare E-Motion, M5 and REPTree. The second stage presents variations in the evolutionary parameters of E-Motion and the respective results for 2 data sets.

6.1 Default Parameters

The parameter values of an evolutionary algorithm can greatly influence whether the algorithm will find a near-optimum solution, and whether it will find such a solution efficiently. Choosing correctly the parameters, however, is a time-consuming task and considerable effort has gone into developing good heuristics for it [MS07]. For instance, Lobo et al. [LLM07] compiled a series of papers presented in the 2005 Genetic and Evolutionary Computation Conference, as well as other papers from invited authors in the theme *parameter setting in evolutionary algorithms*, presenting the state of the art in parameter tuning and control. While parameter setting is an important step of this work, a deeper discussion on different strategies for choosing the optimal set of parameters is out of the scope of this dissertation. Table 6.1 presents the default parameter values of E-Motion, and we discuss our choices below.

The *initial forest max depth* parameter controls the random merging of basic trees. It means that the initial population can have trees of one level (for instance, basic trees have only one level), two levels (like the individual in Figure 5.1) or three. The choice of this parameter affects the average size of the population, and since our target is to produce comprehensible trees, 3-levels depth was a good choice in our experiments. By setting this parameter with values higher than 3, we have ended up generating large trees overall, and initializing the forest with values lower than 3 affected the search of optimal trees in problems where the near-optimal solution is a large tree.

The *population size* parameter controls the algorithm's number of possible candidate solutions.

Table 6.1: E-Motion default parameters.

Parameter	Value
Initial forest max depth	3 levels
Population size	200 individuals
Convergence rate	10% max nº of generations
Max nº of generations	200 generations
Tournament size	3 individuals
Elitism rate	5% population size
Crossover rate	90%
Mutation rate	10%

De Jong and Spears [JS91] work is said to present the *de facto* parameters standard for most GAs. It works with populations with 50 individuals. Since GP is more complex than GAs in its solution representation, we believe that more individuals are necessary to achieve good diversification. Zhao's work [Zha07] that evolves decision trees through GP suggest 5000 individuals is the recommended value, based on their *pilot testing*. GPMCC [PE08] on the other hand suggests that 100 individuals are enough to find near-optimal solutions. TARGET [FG05] uses 25 individuals, and the authors argue that values between 10 and 100 were tested, without drastic changes in the outcome. We have opted for 200 individuals because the number of basic trees generated are based on the number of data set attributes. For instance, a data set with 10 numeric attributes and 10 categorical attributes will have 50 numeric basic trees and around 40 categorical basic trees (assuming each attribute has 5 categories). Thus, with 90 distinct basic trees, we think 100 individuals in the initial forest would not explore the several possible combinations of basic trees, so we have doubled this value and obtained good results in non-exhaustive experimentation.

The *convergence rate* parameter is one of the algorithm's stopping criteria. If there are no modifications in the best solution for a certain number of generations (E-Motion's default is 20 generations without improvement), the algorithm ends its execution. Horn [Hor93] states that in an EA with selection, crossover and low mutation, the population should converge in just a few generations (e.g., 30 to 50 generations for a population of 1000 individuals). Since E-Motion works with 200 individuals, it is a fair assumption that 20 generations without improvement are enough to stop its execution.

The *max nº of generations* parameter is the other algorithm's stopping criterion. De Jong and Spears [JS91] suggest 1000 generations is a good value, and TARGET [FG05] follows this suggestion. GPMCC [PE08], on the other hand, has a default value of only 10 generations for evolving the population of model trees. We have tested values between 100 and 500 generations in E-Motion, and there was no significant gain between 200 and 500 generations, so we set the default value to 200 generations. The main reason for not using a higher value of generations like 1000 was due to time constraints, but we will try to address this issue in future works.

The *tournament size* parameter defines the number of individuals that battle in each tournament round. E-Motion's default value is 3. Blickle and Thiele [BT96] state that typical values are 2 (binary tournament) and 3, while there is a great loss of diversity when the value is higher than 5.

The *elitism rate* parameter defines the percentage of the current population that is kept unaltered for the next generation. E-Motion's default is 10 individuals (5% the population size). By doing so, we make sure that the 5% best solutions are always kept for the next generations, preventing that crossover or mutation eventually ruin good solutions.

The *crossover rate* parameter is the probability that two selected parents from the reproduction pool generate two offspring. Crossover rates are usually high. For instance, De Jong and Spears [JS91] set a 60% crossover rate while Grefenstette's work [Gre86], which presents another well-accepted set of parameters, suggests a crossover rate of 90%. E-Motion implements the crossover rate of 90%, indicating that two selected parents will generate offspring that will replace them in 90% of the times, whereas there is a chance of 10% that the selected parents will be copied into the next generation. E-Motion can afford such a high crossover rate once it implements the elitism technique, so it seeks for high diversification with the certainty that the best trees will not be altered.

Finally, the *mutation rate* parameter indicates the probability of each individual to suffer mutation. Mutation in GAs is usually bitwise, which means each bit of the chromosome has the chance of suffering mutation, with a rate of around 1%. In GP, because the chromosomes are complex structures, the rate of 1% is not sufficient. Since each chromosome in E-Motion is a tree composed by nodes, it is a fair assumption that mutation could be tested per node. The problem here is that E-Motion's trees have dynamic sizes, so we have fixed the mutation rate in 10% per chromosome (tree). GPMCC [PE08], for example, sets a default mutation rate of 20% and Zhao's MOPG (multi-objective genetic approach) [Zha07] a default of 10%.

6.2 Public data sets

Table 6.2 depicts the 10 regression data sets tested in this dissertation. They were randomly chosen from the UCI (University of California at Irvine) Machine Learning Repository [AN07].

6.3 Experimental Methodology

We have divided our experimentation in two moments: (i) a comparison among E-Motion, M5 [Qui92] and REPTree [WF05] using 8 data sets from those in Table 6.2; and (ii) a comparison among 4 different configurations of two data sets from Table 6.2 (Abalone and Cloud). The machine used for both experiments is a dual Intel Xeon Quad-core E5310 running at 1.6 GHz each, 8 MB L2 and 8 GB RAM.

Table 6.2: Data sets used in experimentation.

Data set	#Instances	#Num. Attrib.	#Cat. Attrib.	Description
Abalone	4177	7	1	Predicting the age of abalone from physical measurements. Abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope - a boring and time-consuming task.
AutoMPG	398	4	3	Concerns city-cycle fuel consumption in miles per gallon, to be predicted in terms of 3 multivalued discrete and 5 continuous attributes.
Breast Tumor	286	1	8	This is one of three domains provided by the Oncology Institute that has repeatedly appeared in the machine learning literature.
Cloud	108	4	2	These data are those collected in a cloud-seeding experiment in Tasmania between mid-1964 and January 1971.
Fish Catch	158	5	2	159 fishes of 7 species are caught and measured. Altogether there are 8 variables. All the fishes are caught from the same lake (Laengelmavesi) near Tampere in Finland.
Machine CPU	209	6	0	The problem concerns Relative CPU Performance Data.
Quake	2178	3	0	Dataset from Smoothing Methods in Statistics, [Sim96]
Stock	950	9	0	The data provided are daily stock prices from January 1988 through October 1991, for ten aerospace companies.
Strike	625	5	1	The data consist of annual observations on the level of strike volume and their covariates in 18 OECD countries from 1951-1985. The average level and variance of strike volume varies across countries. The data distribution also features a long right tail and several large outliers.
Veteran	137	3	4	Veteran's Administration Lung Cancer Trial.

6.3.1 First Experiment Protocol

For the first experiment, we have tested E-Motion with the lexicographic approach and random division of the training set to generate the numeric basic trees. We have analyzed the $RMSE$, MAE and tree size obtained by the M5P algorithm (WEKA's implementation of the well-known M5) [Qui92] and REPTree [WF05], a regression tree algorithm also implemented in WEKA, for all

data sets listed in Table 6.2.

The M5P and REPTree parameter settings used are the default ones of each algorithm and we have used 10-fold cross-validation, a widely disseminated approach for validating prediction models. In each of the ten iterations of the cross-validation procedure, the training set is divided into sub-training and validation sets, which are used to produce the basic trees and linear models (sub-training set), filtering process (sub-training set) and fitness function (validation set). The sub-training set represents 60% of the full training set whereas the validation set represents 30%, as recommended in [WF05]. This split is intended to avoid training data overfiting.

E-Motion was executed according to the parameters listed in Table 6.1. We have made no attempt to optimize these parameter values, a topic left for future research. For the lexicographic analysis, as mentioned in Chapter 5, RMSE is the highest-priority measure, followed by MAE and tree size, respectively. Thresholds were calculated dynamically, where each error measure has a threshold of 5% of its average value within the current population, and 20% for the average tree size. These parameters were defined empirically, through previous experimentation.

Due to the fact that GP is a non-deterministic technique, we have run E-Motion 30 times (varying the random seed across the runs) for each one of the ten training/test set folds generated by the 10-fold cross-validation procedure. These folds were the same ones used by M5P and REPTree, to make a fair comparison among the algorithms. After running E-Motion on each of the ten data sets, we have calculated the average and standard deviation of the 30 executions for each fold and then the average of the ten folds. We have calculated the averages and standard deviations for the ten folds of M5P and REPTree since these are deterministic algorithms.

To assess the statistical significance of the differences observed in the experiments for each data set, we have executed the corrected paired t-test [NB03], with a significance level of 1% and 9 degrees of freedom. The measures we analyzed were the same we used in the lexicographic analysis: tree size, $RMSE$ and MAE.

6.3.2 Second Experiment Protocol

In this second experiment, we have executed four different versions of E-Motion in two data sets from Table 6.2: *Abalone* and *Cloud*. The four different versions are summarized in Table 6.3. The goal of this experiment is to check whether the different fitness and numeric basic trees implementations provide distinct results.

We have also executed 30 times the 10-fold cross-validation procedure to collect the results, and the statistical test used was once again the corrected paired t-test [NB03]. The choice of the data sets was made randomly, and we have not tested the four configurations to all data sets listed in Table 6.2 due to time constraints, but it is also in our list of future work.

Table 6.3: Four different configurations of E-Motion.

Configuration Type	Numeric Basic Trees Generation	Fitness Function
E00	Random division of training set	Lexicographic Analysis
E01	Random division of training set	Weighted-Formula
E10	Clustering of training set	Lexicographic Analysis
E11	Clustering of training set	Weighted-Formula

6.4 First Experiment Results

Table 6.4 presents the average values for the error measures for E-Motion (E), M5P and REPTree (REP). Standard deviation values are within parentheses. Table 6.5 presents the results regarding the average tree size (number of nodes) of the solutions produced by each algorithm.

Table 6.4: Error measures for E-Motion, M5P and REPTree.

	RMSE			MAE		
Dataset	E	M5P	REP	E	M5P	REP
AutoMPG	2.98 (0.65)	2.72 (0.52)	3.44 (0.59)	2.20 (0.38)	2.00 (0.33)	2.53 (0.36)
Breast Tumor	10.45 (1.24)	9.94 (1.36)	10.47 (1.02)	8.36 (1.15)	8.05 (1.08)	8.30 (0.92)
Fish Catch	62.72 (0.98)	59.92 (15.96)	139.30 (60.56)	43.41 (0.81)	40.76 (10.45)	81.72 (29.36)
Machine CPU	45.91 (21.82)	54.81 (27.38)	93.13 (58.08)	29.51 (11.13)	29.82 (10.27)	49.73 (24.28)
Quake	0.19 (0.01)	0.19 (0.01)	0.19 (0.01)	0.15 (0.01)	0.15 (0.01)	0.15 (0.01)
Stock	1.04 (0.07)	0.92 (0.20)	1.21 (0.24)	0.81 (0.05)	0.67 (0.08)	0.84 (0.12)
Strike	440.44 (282.66)	436.99 (277.65)	459.87 (274.23)	217.52 (52.36)	211.29 (48.91)	225.40 (54.68)
Veteran	133.35 (86.34)	126.85 (78.12)	140.01 (81.91)	92.68 (47.97)	91.95 (42.88)	98.98 (37.39)

For the next tables, the data sets were abbreviated as follows: AutoMpg {A}, BreastTumor {B}, FishCatch {F}, MachineCPU {M}, Quake {Q}, Stock {So}, Strike {Si} and Veteran {V}.

Table 6.6 shows in which data sets the differences regarding the error measures and tree size were statistically significant, according to the corrected paired t-test. This table is divided into two parts. The left part indicates the data sets in which E-Motion was significantly better than M5P or REP according to each of the three criteria in column (a). Each entry in the column E-Motion contains 8 values, one for each of the data sets, and the value in question is a data set identifier if E-Motion was significantly better than the corresponding algorithm in the second column (M5P or REP) according to the corresponding measure in column (a); otherwise the value in question is "-". The second part of the table has a similar structure, but now each entry in the column E-Motion indicates for which data set E-Motion was significantly worse than the corresponding algorithm in the second column, according to the corresponding measure in column (b).

Table 6.5: Tree Size for E-Motion, M5P and REPTree.

	Tree Size		
Dataset	E	M5P	REP
AutoMPG	3.17 (0.26)	6.60 (2.63)	71.90 (14.08)
Breast Tumor	2.59 (0.80)	1.80 (1.03)	10.30 (12.60)
Fish Catch	3.34 (0.60)	7.80 (4.34)	38.60 (10.72)
Machine CPU	6.18 (0.83)	6.00 (3.16)	21.20 (14.28)
Quake	1.00 (0.00)	3.60 (2.99)	18.00 (33.88)
Stock	11.78 (0.67)	87.80 (15.70)	160.40 (15.69)
Strike	4.93 (0.72)	10.40 (7.55)	41.20 (16.10)
Veteran	3.31 (0.75)	1.80 (2.53)	9.80 (8.44)

Table 6.6: E-Motion significantly better (a) or worse (b) than M5P and REP according to the corrected paired t-test.

(a)		E-Motion	(b)		E-Motion
	M5P	---M----		M5P	--------
RMSE	REP	--F-----	RMSE	REP	--------
	M5P	---M----		M5P	-----So--
MAE	REP	--FM----	MAE	REP	--------
	M5P	--F-QSo--		M5P	---M----
Size	REP	A-FM-SoSi-	Size	REP	--------

Regarding statistical significance results, we can notice that E-Motion performs similarly to M5P and REPTree in terms of error measures. It outperforms M5P in the MachineCPU data set ($RMSE$ and MAE), and it outperforms REPTree in FishCatch ($RMSE$ and MAE) and MachineCPU (MAE). E-motion is never outperformed by REPTree considering the error measures, and it is outperformed by M5P only in Stock (MAE). Considering absolute values only, E-Motion is superior to REPTree in 7 data sets for $RMSE$ and 6 for MAE. However, it is outperformed by M5P, though by statistically insignificant margins.

Once E-Motion induces trees with predictive performance similar to M5P and REPTree, it is interesting to analyse the comprehensibility of the models generated. As we can see in Table 6.5, E-Motion always produces smaller trees when compared to REPTree. In addition, it produced trees that are smaller than M5P ones in 5 out of the 8 datasets we used. This difference is statistically significant in 5 out of 8 datasets when comparing E-Motion to REPTree, and in 3 out of 8 when comparing E-Motion to M5P. Only in the MachineCPU dataset M5P was able to generate trees that are smaller than E-Motion's trees with statistical significance.

Overall, these results suggest that E-Motion seems to produce trees which are often both accurate and significantly smaller than the other two traditional algorithms. We conclude this experiment by

arguing that the lexicographic analysis used as fitness function has achieved its goal by providing a good trade-off between predictive performance and model comprehensibility, which we consider one of the main contributions of this work.

We point out that E-Motion's execution time for this experiment is coherent to most evolutionary algorithms, which means it turns out to be around 100 times slower (per fold) than most greedy algorithms for generating model trees. Note that in predictive data mining tasks such as regression, computational time is normally considered much less important than solution-quality criteria such as prediction error and tree size.

6.5 Second Experiment Results

Table 6.7 presents the average values for the error measures and tree size for 4 different E-Motion configurations (E00, E01, E10, E11), M5P and REPTree (REP). Standard deviation values are within parentheses. Values in bold indicate the best algorithm/configuration according to absolute values.

Table 6.7: Results for 4 different E-Motion configurations, M5P and REPTree in two data sets.

Data set	Algorithm	RMSE	MAE	Size
Abalone	E00	2.19 (0.14)	1.57 (0.06)	3.80 (1.20)
	E01	**2.19 (0.12)**	**1.57 (0.06)**	**2.94 (0.08)**
	E10	2.19 (0.13)	1.57 (0.06)	3.60 (0.99)
	E11	**2.19 (0.13)**	**1.57 (0.06)**	**2.93 (0.05)**
	REP	2.35 (0.35)	1.63 (0.14)	75.90 (30.25)
	M5P	**2.14 (0.12)**	**1.54 (0.07)**	**9.60 (4.32)**
Cloud	E00	0.44 (0.17)	0.30 (0.07)	10.20 (3.10)
	E01	**0.41 (0.19)**	**0.29 (0.08)**	**3.07 (0.31)**
	E10	0.43 (0.16)	0.29 (0.07)	8.04 (2.46)
	E11	**0.41 (0.18)**	**0.28 (0.08)**	**3.10 (0.25)**
	REP	0.61 (0.30)	0.38 (0.17)	13.90 (4.77)
	M5P	**0.39 (0.17)**	**0.28 (0.08)**	**10.47 (3.30)**

By analyzing Table 6.7, we can notice that the difference in error measures for the different configurations of E-Motion is not representative, but there is variation regarding tree sizes. For instance, approaches that make use of the weighted-formula strategy seem to generate smaller trees than those that implement the lexicographic analysis. This happens in both data sets, with a greater difference in *Cloud*.

Moreover, the approaches with lower error values are also those that implement the weighted-formula. This was particularly a surprise, specially due to the fact that there was no extra effort in the definition of a complex weighted-formula. Perhaps the superiority of the weighted-formula approach in this small experiment explains why it is the most used approach in evolutionary algorithms for data

mining, even though it presents the drawbacks described in Chapter 3. Another possible explanation for the low performance of the lexicographic approach is the difficulty of establishing good threshold values for each measure, and also of choosing which error measure has the priority over the other.

E01 and E11 presented the best results among E-Motion's configurations. Since they presented statistically identical results, Table 6.8 shows in which data sets the differences between E01/E11 and M5P or E01/E11 and REPTree were statistically significant, according to the corrected paired t-test. The left part of the table indicates the data sets in which E01/E11 were significantly better than M5P or REP according to each of the three criteria in column (a). Each entry in the column E01 contains 2 values, and the value in question is a data set identifier (*A* stand for Abalone and *C* for Cloud) if E-Motion was significantly better than the corresponding algorithm in the second column (M5P or REP) according to the corresponding measure in column (a); otherwise the value in question is "-". The second part of the table indicates for which data set E01/E11 were significantly worse than the corresponding algorithm in the second column, according to the corresponding measure in column (b).

Table 6.8: E-Motion's configurations E01 and E11 significantly better (a) or worse (b) than M5P and REP according to the corrected paired t-test.

(a)		E01/E11	(b)		E01/E11
	M5P	--		M5P	--
RMSE	REP	AC	RMSE	REP	--
	M5P	--		M5P	--
MAE	REP	AC	MAE	REP	--
	M5P	AC		M5P	--
Size	REP	AC	Size	REP	--

By analyzing Table 6.8, we can notice that E01/E11 are statistically better than REPTree for both data sets, as expected. They also provide smaller trees than M5P for both data sets. Regarding error measures, there is practically no difference between E01/E11 and M5P because even though M5P provides lower absolute values, the difference is not enough to be consider statistically significant.

6.6 Chapter Remarks

In this chapter we presented two different experiments for trying to assess quantitatively E-Motion's performance. In the first experiment (documented in [BBR+10]) we have executed E-Motion with its default parameters plus the lexicographic approach and random division of the training set to generate numeric basic trees. We compared the performance of E-Motion in 8 data sets from UCI machine learning repository to well-established regression algorithms such as M5P and REPTree. The results were quite encouraging, since E-Motion was able to consistently provide smaller

trees while keeping its predictive performance very similar to M5P, the state of the art algorithm for model trees induction.

In the second experiment, the goal was to analyze whether changing the fitness evaluation and the strategy for generating numeric thresholds would increase or decrease E-Motion's performance. Results have shown that, in two data sets from UCI repository, the configurations that made use of the weighted-formula fitness evaluation approach provided better trees (smaller trees with similar/slightly lower error values). This was unexpected since our first belief was that the lexicographic approach would be a robust and efficient alternative for multi-objective optimization.

Considering the good results obtained by the weighted-formula approach, we propose as future work to repeat the first experiment with E-Motion's fitness set to perform this approach, for we believe the results can be significantly enhanced. We are aware that more tests should be done until a definitive conclusion about E-Motion's true performance can be presented. Nonetheless, the results of these two experiments seem to indicate that our goal of providing model trees with high predictive performance and also high comprehensibility was achieved.

Finally, we have considered comparing E-Motion to TARGET and GPMCC, but the authors did not reply our e-mail that asked for the source code of both. We need the source code of them otherwise the comparison will not be fair since each algorithm divides the data sets in different partitions, and this can affect severely the final results.

7

Final Remarks and Future Work

Model trees are a popular alternative to classical regression methods, mainly because the models they provide resemble the human reasoning. We emphasize that the comprehensibility of the discovered model is important in many applications where decisions will be made by human beings based on the discovered knowledge. Therefore, there is a clear motivation to provide model trees that are not only accurate but also relatively simple.

Traditional model tree induction algorithms which rely on a recursive top-down greedy strategy are relatively fast but susceptible to converging to local optima, while an ideal algorithm should choose the correct tree partitioning in order to converge to a global optimum. With this goal in mind, we have proposed a novel evolutionary algorithm for inducing model trees called E-Motion. It is based on Legal-Tree [BBC+09b], a GA for induction of decision trees.

E-Motion avoids the greedy search performed by conventional tree induction algorithms, and performs instead a global search in the space of candidate model trees. Additionally, while other approaches typically rely on a single objective evaluation, we allow the user to choose between the lexicographic approach and weighted-formula. In the lexicographic approach, multiple measures are evaluated in order of their priority. it is relatively simple to implement and control and does not suffer from the problems the weighted-formula and Pareto dominance do, as discussed earlier.

E-Motion considers the two most common error measures used for evaluating regression problems: root mean squared error and mean absolute error. Also, it considers tree size as a measure of model comprehensibility, assuming that smaller trees are easier to interpret.

In an experiment with 8 UCI data sets, E-Motion's results did not significantly differ from the popular M5 algorithm regarding the error measures, but E-Motion consistently induced smaller model trees. This means an overall improvement in simplicity was obtained without any statistically significant loss in predictive performance in most data sets. This is a clearly positive result, which is also supported by the well-known principle of Occam's razor, a principle very often used in data mining and science in general. Regarding the comparison with REPTree, E-Motion is superior both in error measures and tree size, which is partially explained by the different types of trees that are induced (model trees versus regression trees), but also by the ability of E-Motion of producing reduced and

accurate trees.

In a second experiment with 2 data sets, we have compared four different configurations of E-Motion. Surprisingly, those that implement the weighted-formula provided smaller trees and similar (if not slightly lower) error values. We did not expect that to happen, specially due to the lack of scientific evidence to support the weights chosen in the formula and its apparent simplicity. Hence, this initial evidence suggests that more experiments are necessary to check whether E-Motion's weighted-formula version can consistently provide better results in comparison to any tree-based algorithm designed for regression problems.

The greatest challenge of this work was the continuously attempt of reducing E-Motion's execution time. As most evolutionary algorithms, E-Motion is very time-consuming. Due to the fact it is a non-deterministic algorithm, it was necessary to execute E-Motion 30 times per fold, in a 10-fold cross-validation procedure, which means the algorithm was executed 300 times per data set. Considering that each execution can run up to 200 generations, each one evolving 200 individuals, it is not hard to see why E-Motion is not so efficient in terms of execution time when compared to most traditional tree-based induction algorithms. We have started an attempt to parallel the processing of some genetic operators as selection and crossover, as well as the filtering process, in order to speed-up the algorithm's execution time. This is our first goal in future works.

Other possibilities for future research are as follows. We plan to test the weighted-formula version of E-Motion in more than 30 data sets, so we can properly assess its true performance. Moreover, we intend to implement the Pareto dominance approach and test it properly. The lexicographic analysis will also be modified so we can set the threshold values through statistical techniques, as previously mentioned. We also plan to explore different basic trees generation to improve diversification and convergence to global-optima. Furthermore, the setting of input parameters for E-Motion could be done through a supportive GA, helping to achieve convergence to a global optimum and avoiding empirical-based parameters setting. Finally, we intend to extend the leaf models in order to generate polynomial expressions, so each model tree will hold non-linear models in its leaves, allowing optimal exploration of time-series data, for instance.

References

[Alp04] E. Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2004.

[AN07] A. Asuncion and D. Newman. UCI Machine Learning Repository. url: http://www.ics.uci.edu/ mlearn/. visited in june, 24th, 2009., 2007.

[BBC+09a] M. Basgalupp, R. Barros, A. Carvalho, A. Freitas, and D. Ruiz. Legal-tree: A lexicographic multi-objective genetic algorithm for decision tree induction. In *Proceedings of the 24th Annual ACM Symposium on Applied Computing*, pages 1085–1090, Hawaii, USA, 2009.

[BBC+09b] M. Basgalupp, R. Barros, A. Carvalho, A. Freitas, and D. Ruiz. Lexicographic multi-objective evolutionary induction of decision trees. *International Journal of Bio-Inspired Computation*, 1:105–117, 2009.

[BBR+10] R. Barros, M. Basgalupp, D. Ruiz, A. Carvalho, and A.. Freitas. Evolutionary model tree induction. In *Proceedings of the 25th Annual ACM Symposium on Applied Computing (to appear)*, pages 1132–1137, Sierre, Switzerland, 2010.

[BBTR08] R. Barros, M. Basgalupp, N. Tenório, and D. Ruiz. Issues on estimating software metrics in a large software operation. In *Proceedings of the 32nd Annual IEEE Software Engineering Workshop*, pages 152–160, Kassandra, Greece, 2008.

[BFOS84] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[Bjö96] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

[BL01] C. Burgess and M. Lefley. Can genetic programming improve software effort estimation? A comparative evaluation. *Information and Software Technology*, 43(14):863–873, 2001.

[Bre96] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[Bre01] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[BT96] T. Blickle and L. Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394, 1996.

[CGM97] H. Chipman, E. George, and R. Mcculloch. Bayesian cart model search. *Journal of the American Statistical Association*, 93:935–960, 1997.

[DMS98] D. Denison, B. Mallick, and A. Smith. A Bayesian CART algorithm. *Biometrika*, 85:363–377, 1998.

[FG05] G. Fan and J. Gray. Regression tree analysis using TARGET. *Journal of Computational and Graphical Statistics*, 14(1):206–218, 2005.

[FPSSU96] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in knowledge discovery and data mining*. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[Fre04] A. Freitas. A critical review of multi-objective optimization in data mining: a position paper. *SIGKDD Explor. Newsl.*, 6(2):77–86, 2004.

[Fre08] A. Freitas. *Soft Computing for Knowledge Discovery and Data Mining*, chapter A Review of Evolutionary Algorithms for Data Mining, pages 79–111. Springer U.S, 2008.

[FS97] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

[FWA08] A. Freitas, D. Wieser, and R. Apweiler. On the importance of comprehensible classification models for protein function prediction. *To appear in IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2008.

[GF08] J. Gray and G. Fan. Classification tree analysis using TARGET. *Computational Statistics & Data Analysis*, 52:1362–1372, 2008.

[Gol89] D. Goldberg. *Genetic Algorithms in Search,Optimization and Machine Learning*. Addison Wesley, Boston, 1989.

[Gre86] J Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Trans. Syst. Man Cybern.*, 16(1):122–128, 1986.

[HK06] J. Han and M. Kamber. *Data Mining Concepts and Techniques*. Elsevier, 2006.

[Hol75] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, USA, 1975.

[Hor93] J. Horn. Finite markov chain analysis of genetic algorithms with niching. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 110–117, San Francisco, USA, 1993.

[JS91] K. De Jong and W. Spears. An analysis of the interacting roles of population size and crossover in genetic algorithms. In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pages 38–47, London, UK, 1991.

[Koz92] J. Koza. *Genetic programming: On the programming of computers by means of natural selection*. The MIT Press, Cambridge, MA, 1992.

[LLM07] F. Lobo, C. Lima, and Z. Michalewicz. *Parameter Setting in Evolutionary Algorithms*. Springer, 2007.

[Llo82] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[MS07] Z. Michalewicz and M. Schmidt. Parameter control in practice. In *Parameter Setting in Evolutionary Algorithms*, pages 277–294. 2007.

[NB03] C. Nadeau and Y. Bengio. Inference for the generalization error. *Machine Learning*, 52(3):239–281, 2003.

[OZ06] S. Olariu and A. Zomaya. *Handbook of Bioinspired Algorithms and Applications*. Chapman & Hall/CRC, 2006.

[PE07] G. Potgieter and A. Engelbrecht. Genetic algorithms for the structural optimisation of learned polynomial expressions. *Applied Mathematics and Computation*, 186(2):1441–1466, 2007.

[PE08] G. Potgieter and A. Engelbrecht. Evolving model trees for mining data sets with continuous-valued classes. *Expert Systems with Applications*, 35:1513–1532, 2008.

[PF09] G. Pappa and A. Freitas. Automatically evolving rule induction algorithms tailored to the prediction of postsynaptic activity in proteins. *Intelligent Data Analysis*, 13(2):243–259, 2009.

[Qui92] J. Quinlan. Learning with Continuous Classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992.

[RM05] L. Rokach and O. Maimon. Top-down induction of decision trees classifiers - A Survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 35(4):476–487, 2005.

[Sch78] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464, 1978.

[Sim96] J. Simonoff. *Smoothing Methods in Statistics*. Springer-Verlag, 1996.

[SLZ02] R. Setiono, W. Leow, and J. Zurada. Extraction of rules from artificial neural networks for nonlinear regression. *IEEE Transactions on Neural Networks*, 13(3):564–577, 2002.

[TSK06] P. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson Education, 2006.

[Wei09] T. Weise. *Global Optimization Algorithms - Theory and Application*. Self-Published, second edition, 2009. Online available at http://www.it-weise.de/. Access in September, 2009.

[WF05] I. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2005.

[Zha07] H. Zhao. A multi-objective genetic programming approach to developing pareto optimal decision trees. *Decision Support Systems*, 43(3):809–826, 2007.

Appendix A
K-Means

K-means is one of the oldest and most widely used clustering algorithms [TSK06]. It is prototype-based, which means that objects are clustered according to the distance (similarity) each object is to the prototype that defines each cluster. In K-Means, prototypes are known as *centroids*, i.e., the average of all objects in the cluster it represents.

The K-means basic algorithm is quite simple. First we have to choose k initial centroids, where k will be the number of clusters the data will be partitioned in (a user-specified parameter). Then each point is assigned to the closest (most similar) centroid, and each set of objects assigned to a same centroid is a cluster. The centroids are recalculated according to the objects assigned to them. This process is repeated until no object changes clusters or until the centroids remain the same. Algorithm .1 depicts this rationale.

Algorithm .1 Pseudo-code of basic K-means.

Select k points as centroids
repeat
  Form k clusteres by assigning each point to its closest centroid.
  Recompute the centroid of each cluster
until Centroids do not change or no object changes clusters

The assignment of each object to its closest centroid is done through a distance function that will clearly quantify the notion of proximity. Examples of distance functions are the Euclidean distance, Manhattan distance, cosine similarity, Bregman divergence, etc.

K-Means is relatively scalable, handling efficiently large data sets since the computational complexity of the algorithm is $\mathcal{O}(nkt)$, where n is the total number of data set instances, k is the number of clusters and t is the number of iterations. The necessity for users to specify the number of clusters in advance can be seen as a disadvantage, as well as its sensitivity to noise and outliers due to their influence in the calculation of the centroids [HK06].

Appendix B
Lexicographic analysis

Algorithm .2 Pseudo-code of the lexicographic fitness analysis.

{this code returns the best individual between I_A and I_B}
$diffRMSE := RMSE(I_A) - RMSE(I_B)$
$diffMAE := MAE(I_A) - MAE(I_B)$
$diffSize := Size(I_A) - Size(I_B)$
if $|diffRMSE| > t_{RMSE}$ **then**
  if $diffRMSE > 0$ **then**
    return I_B
  else
    return I_A
  end if
else if $|diffMAE| > t_{MAE}$ **then**
  if $diffMAE > 0$ **then**
    return I_B
  else
    return I_A
  end if
else if $|diffSize| > t_{Size}$ **then**
  if $diffSize > 0$ **then**
    return I_B
  else
    return I_A
  end if
else if $diffRMSE > 0$ **then**
  return I_B
else if $diffRMSE < 0$ **then**
  return I_A
else if $diffMAE > 0$ **then**
  return I_B
else if $diffMAE < 0$ **then**
  return I_A
else if $diffSize > 0$ **then**
  return I_B
else if $diffSize < 0$ **then**
  return I_A
else
  return $I_A \vee I_B$
end if