HELSINKI UNIVERSITY OF TECHNOLOGY
Faculty of Information and Natural Sciences
Department of Information and Computer Science

Fast Variable Selection using Delta Test

Master's Thesis

Dušan Sovilj

Department of Information and Computer Science
Espoo 2009

HELSINKI UNIVERSITY OF TECHNOLOGY
Faculty of Information and Natural Sciences
Department of Information and Computer Science

ABSTRACT OF MASTER'S THESIS

Author: Dušan Sovilj

Title of thesis:
Fast Variable Selection using Delta Test

Date: August 30^{th} 2009 **Pages:** 12 + 68
Professorship: Computer and Information Science **Code:** T-115
Supervisor: Professor Olli Simula
Instructor: Docent Amaury Lendasse

Data nowadays are easy to acquire and measure, becoming increasingly high dimensional. These data are later used for prediction, clustering and other decision-making tasks. However, high-dimensional data often impose a burden by themselves to learning models, restricting them to poor generalization performance.

Variable selection is one way of fighting this problem. It consists of two parts: relevance criterion and search algorithm. In this thesis, Delta Test constitutes a foundation for variable selection as a relevance criterion, while three search algorithms are tested for optimization purposes in this problem: Forward-Backward Search, Tabu Search and Genetic Algorithm.

Different aspects of selection are also considered: scaling, projection, the combination of scaling plus projection, and the scaling with fixed number of variables. Each aspect of selection has different influence on the optimization performance and computation time.

Delta Test is the most important part of the whole methodology. All mentioned search algorithms and aspects of selection are tested on real-world data sets and their performance compared. Finally, we test the usefulness of variable selection with Delta Test in long-term time series prediction task using two competition data sets organized by European Symposium on Time Series Prediction.

Keywords: variable selection, delta test, time series prediction, forward-backward search, tabu search, genetic algorithm
Language: English

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ
Informaatio- ja luonnontieteiden tiedekunta
Tietojenkäsittelytieteen laitos

Tekijä:	Dušan Sovilj	
Työn nimi:		
Fast Variable Selection using Delta Test		
Päiväys:	30. elokuuta 2004	**Sivumäärä:** 12 + 68
Professuuri:	Informaatiotekniikka	**Koodi:** T-115
Työn valvoja:	professori Olli Simula	
Työn ohjaaja:	dosentiksi Amaury Lendasse	

Datan määrä kasvaa koko ajan. Helpommat mittausmenetelmät ja jatkuvasti kasvava tallennuskapasiteetti mahdollistaa entistä suuremmat ja moniulotteisemmat datajoukot. Monet päätöksentekoprosessit, datasta ennustaminen ja muu datasta oppiminen vaikeutuvat huomattavasti moniulotteisuuden takia, johtaen epätarkkoihin malleihin ja tuloksiin. Muuttujanvalinnalla pyritään helpottamaan näitä ongelmia.

Muuttujanvalinnassa on kaksi tärkeää osa-aluetta: hakualgoritmi muuttujien valintaan ja muuttujien relevanssin mittaaminen. Tässä työssä relevanssin mittaamiseen käytetään Deltatesti -menetelmää yhdessä kolmen eri hakualgoritmin kanssa: Forward-backward search, Tabu search sekä geneettiset algoritmit.

Suoraviivaisen valitsemisen ohella muuttujia voidaan myös painottaa eri kertoimilla tai projisoida uuteen muuttuja-avaruuteen. Myös erilaiset yhdistelmät muuttujien muokkaamiseen ovat mahdollisia ja vaikuttavat eri tavoin hakualgoritmin ja relevanssimittauksen toimintaan ja suoritusaikaan.

Tässä työssä vertaillaan edellä mainittuja hakualgoritmeja ja muuttujien muokkauskeinoja sekä niiden yhdistelmien tarkkuutta tosielämän mittaustuloksista koottujen datajoukkojen avulla. Samalla testataan Deltatestin toimivuutta vaativissa pitkän aikavälin aikasarjaennustamisessa. Käytetyt aikasarjat on poimittu kahdesta eri ennustuskilpailusta, jotka järjestettiin ESTSP-konferenssien yhteydessä vuosina 2007 ja 2008.

Avainsanat:	muuttujien valinta, deltatesti, aikasarjaennustaminen, forward-backward ja tabu search, geneettiset algoritmit
Kieli:	Englanti

Acknowledgements

This thesis would not be possible if it was not for a group of hard-working, easy-going, good-to-have-around individuals, always ready to assist and solve the problems of a master student. Sometimes these thank-yous are beyond reason and span pages and pages of unwanted text, but I will be reasonable and mention the most important individuals which I encountered during the my studies and work for the past 2 years.

First comes my supervisor Professor Olli Simula, with his assistance and understanding in dire times for a master student. The next person I want to mention is my instructor Docent Amaury Lendasse, but it is really hard to convey in words the amount of help, readiness, willingness, and patience he had for me. He will always have my endless gratitude for the time spent in his TSPCi Group. The Group itself with its members Francesco Corona, Federico Montesino Pouzols, Elia Liitiäinen, Yoan Miche, Antti Sorjamaa, Qi Yu, Mark van Heeswijk, and Emil Eirola also have my thanks. Special mention goes to Antti for outstanding job acting as a translator, driver, guide, and other small things related to LaTeX, Linux and faculty. Without Alberto Guillén and Fernando Mateo, the writing of the thesis would be a much more troublesome task. Last minute thanks goes to Mark for reading several versions of the thesis... several times.

Last but not least, many thanks to all my friends, especially those from back home. Finally, love and thanks to my parents, for always supporting me and believing in me.

Espoo August 30th 2009

Dušan Sovilj

Abbreviations and Acronyms

DT	Delta Test
NN	Nearest Neighbor
ANN	Approximate Nearest Neighbor
VS	Variable Selection
FBS	Forward-Backward Search
TS	Tabu Search
GA	Genetic Algorithm
BCGA	Binary Coded Genetic Algorithm
RCGA	Real Coded Genetic Algorithm
NSGA-II	Non-dominated Sorting Genetic Algorithm
EA	Evolutionary Algorithms
MSE	Mean Square Error
MO	Multi-objective
ESTSP	European Symposium on Time Series Prediction
ELM	Extreme Learning Machine
OP-ELM	Optimally-Pruned Extreme Learning Machine
LOO	Leave-One-Out error
HQ	Hannan-Quinn information criterion

Contents

Abbreviations and Acronyms **v**

1 Introduction **1**

1.1 Publications 4

2 Delta Test **6**

2.1 Complexity of Delta Test 7

3 Variable Selection, Scaling and Projection **9**

3.1 Variable Selection 10

3.2 Variable Scaling 10

3.2.1 Variable Scaling with Fixed Number of Variables . . . 12

3.3 Variable Projection 13

3.3.1 Automatic Selection of Projection Dimension 14

3.3.2 Combining Scaling and Projection 16

4 Search Algorithms **19**

4.1 Forward Search, Backward Search andForward-Backward Search 20

4.1.1 Forward-Backward Search in Variable Scaling 21

4.2 Tabu Search 22

4.2.1 Tabu Search for Variable Selection 24

4.2.2 Tabu Search for Variable Scaling 25

4.2.3 Setting the Tabu Conditions 25

4.3 Genetic Algorithm 26

4.3.1 Genetic Algorithm Basics 26
4.3.2 Setup for the Experiments 30
4.3.3 Parallel Implementation 32
4.3.4 Genetic Algorithm for Multi-Objective Optimization . 35

5 Experiments: Search Algorithms 36
5.1 Approximate Nearest Neighbor Influence 37
5.2 Performance of Search Algorithms 38
5.3 Custom Initialization of Population for Genetic Algorithm . . 42
5.4 Improvements on Parallel Architectures 44

6 Application to Time Series Prediction 47
6.1 Delta Test Minimization on Different Problems 47
6.1.1 DT Performance 48
6.1.2 Computational Time 49
6.1.3 Projection to many Dimensions 51
6.2 Long-Term Prediction using OP-ELM 53
6.2.1 ESTSP 2007 Competition Data 54
6.2.2 ESTSP 2008b Competition Data 55
6.2.3 Prediction Performance 56

7 Conclusions 60

Chapter 1

Introduction

Data are nowadays ubiquitous and plentiful, due to the fact that they have become rather easy to measure, acquire and store. The data originate from various sources (sensors, cameras, studies) and can be represented in variety of forms (image, sound, text, table, graph). Stored data are then used for pattern extraction, rule extraction, classification, visualization, prediction, and other machine learning tasks. In other words, the data are used to gain information and knowledge in order to make decisions. Since it has become so easy to obtain data, the resulting data sets are in most of the cases high dimensional, where dimensionality refers to the number of measured attributes, also called features or variables. Large number of dimensions characterizes following data: images, where attributes are pixels' RGB values; text documents in which features are the words themselves; sound data; spectral data, where each attribute corresponds to different wavelengths of light. However, high-dimensional data exhibit counter-intuitive properties that are understandable in low-dimensional spaces. In fact, these properties are commonly known as *curse of dimensionality* [7, 8]. One of the effects of the curse is the need for exponential number of samples when the dimension of data increases. This sparsity of samples in such cases often leads to poor generalization performance of all learning models.

Nevertheless, the severity of this problem can be decreased via variable selection, also called feature selection. Variable selection consist of choosing the relevant attributes in order to improve the prediction accuracy of the model. As a result, model trained with relevant set of variables will be able to outperform model trained with all input variables. Variable selection has two independent components: relevance criterion and search procedure. The relevance criterion measures the quality of certain subset of variables with

respect to the prediction accuracy, while the search procedure controls which subsets are to be examined in the upcoming iterations of the search.

The choice of good relevance criterion will depend on the data set and on the type of problem at hand: regression or classification. Criteria mainly used in machine learning community are correlation, mutual information, noise variance estimation, statistical tests. The focus of the thesis is on regression problems, with special treatment on time series prediction. For regression purposes, noise variance estimators give a reliable measure of how accurate models can be trained with available data. Recently, it has been shown that one such estimator, Delta Test, is a useful tool in deciding between relevant and irrelevant variables [9]. Delta Test is a nonparametric noise variance estimator that is based on nearest neighbor principle and is the foundation for variable processing throughout the thesis.

Considering search procedures, the commonly employed are Forward Selection, Backward Pruning and Forward-Backward Selection [10, 11, 12]. Although simple, fast, and easy to implement, these procedures do not necessarily return an optimal solution, known as local minima problem. Moreover, it is infeasible to evaluate all possible subsets of input variables in high-dimensional spaces, and greedy procedures examine only small portion of whole solution space. Search procedure should be designed to examine the solution space on a global scale as well, not just locally as is the case with greedy ones. Popular alternatives involve algorithms from different domains of optimization, such as Evolutionary Computation (Genetic Algorithm [13], Differential Evolution [14]), Swarm Intelligence (Particle Swarm Optimization [15], Ant Colony Optimization [16]), and other meta-heuristic approaches (Tabu Search [17], GRASP [18, 19]). In this thesis, we examine the optimization capabilities of two such algorithms in variable selection domain: Tabu Search and Genetic Algorithm, and show the improvement over standard Forward-(Backward) Search procedure.

Variable selection problem is only a special case of a scaling problem. In the scaling approach, the goal is to find the *weights* for all variables, where weights represent level of importance with respect to the output response. The weights usually take values from [0,1] range, with 0 meaning that variable has no predictive power, and 1 that variable is the most important for prediction accuracy. Weights of 0 and 1 correspond to excluded and included status in variable selection problem, respectively. However, the solution space in scaling approach grows much faster than that for selection, rendering the problem that more difficult. In the thesis, we also investigate a special case of scaling itself. Instead of finding the scaling weights for all variables, which

is very computationally demanding when data sets are high dimensional, we are restricting the search to only include weights for fixed number of variables. This problem has been designed as multi-objective optimization problem, and as such, requires modifications to search algorithms. This special case enables us to have quick insight into the most relevant variables for prediction.

Going further beyond the scaling is the projection, where a projection matrix is optimized according to the relevance criterion (Delta Test in this thesis). Projection includes scaling as a special case, and is more suitable if we want to explicitly alter the dimensionality of data, including all available samples. Thus, projection is more general than selection and should be able to reach better results. After the optimization is done, the complete data set is multiplied from right with the projection matrix to produce the projected data set, which lies in lower-dimensional space, which depends on the projection dimension, i.e. the number of columns in the projection matrix. We also present a special case of projection, which involves projecting the data to higher dimensional space. This might seem counter-productive from variable selection/projection perspective, but the main idea is to combine scaling and projection as separate matrices into a single projection matrix. With this form of projection, optimization unavoidably takes longer time, but the computational cost is slightly increased compared to scaling only. This method is called scaling plus projection.

For the rest of the thesis, we refer to different variable selection approaches (selection, scaling, projection, scaling with fixed number of variables and scaling plus projection) as problem types or as aspects, and these two terms are used interchangeably.

As the final step, we integrate our variable projection using Delta Test with predictive models into one global methodology for regression tasks. The special focus is on time series prediction. This type of prediction is a challenge in many fields: finance (stock exchange rates and indices); electricity production (load for the following days); data processing (flow of information over networks). The core of the problem is how to analyze and use the past to predict the future. Many techniques exist for the approximation of the underlying process of a time series: linear methods [20, 21] and nonlinear ones [22, 23]. Both types of models try to build a model of the process, which is used to predict future values based on present and current information. Predictions are made for immediate samples (short-term prediction) or give estimations for far-future samples (long-term prediction). Long-term prediction is more challenging because the accumulation of errors and inherent

uncertainties of time yields deteriorated estimates for future samples.

The remainder of the thesis is organized as follows. Chapter 2 provides background on Delta Test and discusses nearest neighbor computation needed for computing the Delta Test. Chapter 3 explains variable selection in more detail, and introduces two new problems connected to input selection: scaling with fixed number of variables and scaling plus projection. In Chapter 4 three search algorithms are explained in detail: Forward-Backward Search, Tabu Search and Genetic Algorithm. It also contains the setup of important parameters for the two latter algorithms, and an explanation about parallel implementation for Genetic Algorithm. Chapter 5 describes the data sets used for the experiments and the performance of the search algorithms. In Chapter 6 we propose a methodology for the time series prediction with variable selection as the first step. This methodology can also be applied to any regression problem. Finally, Chapter 7 gives conclusions on the work presented in the thesis and discusses some future ventures.

1.1 Publications

This section briefly reviews the publications related to the work presented in this thesis. Publications are sorted in chronological order:

Publication [1] is the first attempt at variable selection using meta-heuristic optimization method Tabu Search with Delta Test. It is used as a preprocessing step before the actual time series prediction using OP-KNN for the ESTSP 2008 competition data.

The next related work [2] takes a more in depth look at optimization of Delta Test, comparing three search algorithms: standard Forward-Backward Search, Tabu Search and Genetic Algorithm. Advantages and disadvantages of all three search algorithms are identified and a successful hybridization of Tabu Search and Genetic Algorithm is presented. The work is further adapted for parallel architectures, enabling better exploration and results in the same amount of time as the serial method. The work has been extended for a journal publication in [3].

Publication [4] presents the combination of scaling and projection as a single problem. This combination allows the search algorithm to reach better Delta Test values for all tested data sets. The choice of search algorithm falls on Genetic Algorithm for its better exploratory capabilities in scaling problem.

An extensive overview of application of Delta Test in different problem types

is given in [5]. A new type of problem, called scaling with fixed number of variables, is presented and solved with multi-objective optimization approach.

Finally, the variable selection with Delta Test and Genetic Algorithm is combined into a global methodology with OP-ELM/OP-KNN models for the task of time series prediction [6]. The proposed methodology is tested on one financial data set and two time series competition data sets.

Chapter 2

Delta Test

In this thesis, the Delta Test is used as a relevance criterion to optimize either the scaling weights or the projection matrix. The criterion comes from the study of noise variance estimation, where the problem consists of estimating the best possible generalization error given finite number of samples. More generally, in function approximation, the main goal is to design a function that represents given input points and their associated scalar outputs. That is, given N samples of input-output pairs $(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$, we wish to find a functional dependence between $\mathbf{x}$ and y with the following equation.

$$y_i = f(\mathbf{x}_i) + r_i, \quad 1 \leq i \leq N \tag{2.1}$$

where f is the unknown function and r_i is additive noise term. The function f is assumed to be smooth, and the noise terms r_i are independent and identically distributed with zero mean. Noise variance estimation is the study of how to give *a priori* estimate for $\mathrm{Var}[r]$ given some data, *without* considering any specific shape of f.

The Delta Test (DT), first introduced by Pi and Peterson for time series [24], is a technique to estimate the variance of the noise, or the mean squared error (MSE), that can be achieved without overfitting. It is a nonparametric noise estimator based on the nearest neighbor principle. The nearest neighbor of a point is defined as the unique point, which minimizes a distance metric to that point. Distance metric is usually the Euclidean distance, but other metrics can also be used. The DT is useful for evaluating the nonlinear correlation between input and output variables. It is based on hypothesis coming from the continuity of the regression function. If two points $\mathbf{x}$ and $\mathbf{x}'$ are close in the input variable space, their corresponding outputs $f(\mathbf{x})$ and

$f(\mathbf{x}')$ should be close in the output space. If this is not the case, this effect is due to the influence of the noise.

Let us denote the nearest neighbor of a point $\mathbf{x}_i \in \mathbb{R}^d$ as $\mathbf{x}_{NN(i)}$. The nearest neighbor formulation of the DT estimates $\mathrm{Var}[r]$ by

$$\mathrm{Var}[r] \approx \delta = \frac{1}{2N} \sum_{i=1}^{N} (y_i - y_{NN(i)})^2 \,, \tag{2.2}$$

where $y_{NN(i)}$ is the output of $\mathbf{x}_{NN(i)}$. This is a well-known and widely used estimator, and it has been shown [25] that this estimate converges to the true value of the noise variance when $N \to \infty$.

For variable selection problems, the goal is to *minimize* the value of DT, as this value represents the MSE that can be reached without overfitting. Thus, lower value of DT implies better selection of variables [9].

2.1 Complexity of Delta Test

Optimization of Delta Test can be computationally demanding if the data set has a lot of available instances or samples. When computing the DT for thousands of solutions, most of the optimization is spent on DT calculations, and little resources on altering the parameters of the search algorithm. This is due to the nearest neighbor search among the samples. Nearest neighbor search is an optimization technique for finding closest points in metric spaces. Specifically, given a set of N reference points R and query point q, both in the same metric space V, we are interested in finding the closest or nearest point $c \in R$ to q. Usually, V is a d-dimensional space $\mathbb{R}^d$, where the distances are measured using Minkowski metrics (e.g. Euclidean distance, Manhattan distance, max distance).

The simplest solution to this neighbor search problem is to compute the distance from the query point to every other point in the data set, while registering and updating the position of the nearest or k-nearest neighbors of every point. This algorithm, sometimes referred to as the naive approach or brute-force approach, works for small data sets, but quickly becomes intractable as either the size or the dimensionality of the problem becomes large. This is due to the $O(dN)$ running time for a single query point.

To overcome naive approach and its computational drawback, other methods have been proposed [26, 27] which use data structures based on decomposition of multi-dimensional spaces. One such structure used in nearest neighbor

search is *kd-tree* [28]. In 1997, Friedman *et al.* [29] showed that for a data set with N samples and d attributes, a kd-tree can be build in $O(dN \log N)$ time and $O(N)$ space, such that expected computation for a query takes $O(\log N)$ time. However, even this method suffers as dimension increases. The constant factors hidden in the asymptotic running time grow at least as fast as 2^d (depending on the metric).

In some applications it may be acceptable to retrieve a "good guess" of the nearest neighbor. In those cases one may use an algorithm which does not guarantee to return the actual nearest neighbor in every case, in return for improved speed or memory saving. Such an algorithm will find the nearest neighbor in the majority of cases, but this depends strongly on the data set being queried. It has been shown [30] that by computing nearest neighbors approximately, it is possible to achieve significantly faster running times (on the order of tens to hundreds), often with relatively small actual errors.

Arya *et al.* in [30] state that given any positive real ϵ, a data point q' is a $(1+\epsilon)$-approximate nearest neighbor of q if its distance from q is within a factor of $(1+\epsilon)$ of the distance to the true nearest neighbor. It is possible to preprocess a set of N points in $\mathbb{R}^d$ in $O(dN \log N)$ time and $O(dN)$ space, so that given a query point $q \in \mathbb{R}^d$, and $\epsilon > 0$, a $(1+\epsilon)$-approximate nearest neighbor (ANN) of q can be computed in $O(c_{d,\epsilon} \log N)$ time, where $c_{d,\epsilon} \leq d\lceil 1 + 6d/\epsilon \rceil^d$ is a factor depending only on dimension and ϵ. In general, it is shown that given an integer $k \geq 1$, $(1+\epsilon)$ approximations to the k-nearest neighbors of q can be computed in additional $O(kd \log N)$ time.

This faster neighbor search has been applied to the computation of the DT as expressed in Equation 2.2 with high computational savings. A C++ library is used for this purpose, which is available at [66].

Chapter 3

Variable Selection, Scaling and Projection

Suppose we are given a data set that consist of input $\mathbf{X} = (X_1, X_2, \ldots, X_d)$ with d variables and one output variable $\mathbf{Y}$. We are interested in building a function (model) f that approximates the mapping between $\mathbf{X}$ and $\mathbf{Y}$, as given by Equation 2.1.

Variable selection is a methodology that consist of finding the most useful subset of input variables X_i that has maximal predictive power. The aim is to reduce the number of input variables because not all models are capable of distinguishing between relevant and irrelevant variables. A Support Vector Machine and Radial-Basis Function Network with Gaussian kernels assign equal importance to all variables in the data. Second reason for variable selection lies in limited amount of data samples, which affects the training of the models in a negative way. If the models have to many hyper-parameters, the downside is what is known as overfitting of the model. In other words, the model perfectly fits the available data, but has poor generalization abilities on unseen samples.

Variable selection consists of two independent components: relevance criterion and search algorithm. The relevance criterion measures the quality of certain subset of variables with respect to the output variable. One such criterion is Delta Test, which constitutes the basis for all selection problems and search algorithms presented in the thesis. Search procedure or search algorithm is responsible for generating new subsets to be examined based on the information from the currently available subsets. The algorithms use for DT optimization in this work are explained in more detail in Chapter 4.

This chapter first mentions the difficulty of variable selection for data sets

with large number of dimensions. Section 3.2 introduces the problem of variable scaling and one interesting subproblem of scaling. Finally, Section 3.3 discusses variable projection and gives the algorithm for automatic selection of the projection dimensions when optimizing the DT.

3.1 Variable Selection

Considering the case when a data set has d variables, to find the optimal subset one has to examine all non-empty $2^d - 1$ subsets of variables (the excluded case is when all variables are removed from the data set, making the actual model impossible to build). When d is even moderately large, $d > 30$, exhaustive search on all possible subsets becomes too time consuming and in most cases infeasible. Thus, knowing the optimal subset cannot be guaranteed before building the actual model. In such cases all search procedures explore only some small portion of the whole solution space, and this fact is what makes the difference between various search algorithms — the way they explore this space.

The status of a variable in a subset {included, excluded} can be interpreted in a different way: each variable is assigned a weight factor $w_i \in \{0, 1\}, 1 \leq i \leq d$, with respect to the output variable. The value of 0 means that a variable has no predictive power, while 1 indicates that the variable is useful for predicting the output. In other words, each variable is multiplied with its corresponding weight and a new data set is formed in lower-dimensional space before building a model. The set $\{0, 1\}$ can be extended to a full $[0, 1]$ interval, leading to *variable scaling*.

3.2 Variable Scaling

In variable scaling, weights are interpreted as importance factors, where $w_{i_1} > w_{i_2}$ implies that variable i_1 has more predictive power than variable i_2. The point of view is different: instead of completely removing variables as in variable selection, it could be beneficial to retain them with small weight factors. Extending the problem to the full $[0, 1]$ interval increases the complexity of the problem, but allows for a more robust model. In this setting, search algorithms that are based on neighborhood techniques, also called stepwise algorithms (see Section 4.1 and 4.2), have to be adapted to include values from the extended interval. Since an explicit connection between solutions is needed in this case, one approach to solve this situation is to break the

interval into equally sized subintervals, resulting in a set with the equidistant values. For example, the division can be done to form a set $H = \{0, 0.5, 1\}$. Usually, there is a parameter h that controls the number of subintervals. For the given example of $H = \{0, 0.5, 1\}$ we have $h = 2$ and $H = \{0/h, 1/h, 2/h\}$. Given parameter h with positive integer value, the set of scales is formed by taking $H = \{i/h \mid i = 0, 1, \ldots, h\}$. As h grows, the interval is divided into finer parts, and it becomes increasingly difficult to construct neighborhood relations between the solutions. One approach to this relationship that is used in the experiments is given in Section 4.1.1. In this thesis, variable scaling with restricted set of values H is referred to as *discretized scaling*. What also makes discretized scaling more difficult than selection is even larger solution space, which now contains $h^d - 1$ solutions and making h larger leads to exponential increase in complexity of the problem. The set H will be referred to as *set of scales* or a *set of scaling weights*.

When using the interval $[0, 1]$, it is assumed that all variables have the same unit of measure, which is not the case for all real-world data sets. Thus, before performing variable scaling, appropriate normalization of the data should be carried out, for example, to zero mean and unit variance.

In case of scaling the data set, each dimension is modified according to its weight factor as given by Equation 3.1.

$$x_{ij}^S = w_j x_{ij}, \quad i = 1, \ldots, N,\ j = 1, \ldots, d\,. \tag{3.1}$$

This way, a new data set $\mathbf{X}^{\mathrm{S}}_{N\times d}$ is formed, which has at most d variables. The dimensionality depends on the scaling weights, and when all $w_i \neq 0$ the new data set X_S has the same dimensionality as the original data set $\mathbf{X}$. On the other hand, it is possible to transform the data set using a linear projection, with explicit specification of the dimensionality of newly created data set. This approach is called *variable projection* and is discussed in Section 3.3.

An interesting property of variable scaling is the interpretability of the variables. This is done by examining the weights factors, with the larger weights indicating more importance for the prediction of the output variable. This information is important in some fields, such as forecasting or prediction of stock market prices. Before explaining the variable projection case, we touch upon special case of scaling problem, which involves fixing or restricting most the of scaling weights w_i to have zero values.

3.2.1 Variable Scaling with Fixed Number of Variables

In many real world data sets, the number of samples is sometimes so large ($N > 10000$) that optimizing scaling weights takes a considerable amount of time. This is due to the high computational cost of the inherent nearest neighbor search in the DT formula. One approach to solve this would simply be to randomly discard some portion of the samples in order to speed up calculation time, but there is risk of losing valuable data and there is no clear method to select important samples. Instead of removing samples, a different strategy involves drastically reducing the number of variables by forcing most of the scaling weights to have zero value ($w_i = 0$). To achieve this goal, an additional constraint is added to the problem which requires that at most d_f scaling weights have non-zero values. Therefore, d_f variables are *fixed* to be included in final scaling vector and the remaining $d - d_f$ weights are forced to zero, effectively changing the dimensionality of the data set. The computation of nearest neighbor search is reduced in a lower d_f-dimensional space. Thus, the fixed method enables a quick insight into the d_f (or less) most relevant variables of the regression problem. For easier notation and understanding, we refer to standard scaling as *scaling* or *pure scaling*, while scaling with a fixed number of variables is referred to as *fixed scaling*.

The same setup of of any search algorithm can be used for both scaling problems, with the difference that in fixed scaling we take the d_f most relevant variables. Another approach would be to completely modify control operators of search algorithms such that they only consider solutions with at most d_f non-zero scaling weights. However, both approaches in conjunction with genetic algorithm showed extremely quick convergence times in just a couple of tens of generations. A different approach would be to consider this as a multi-objective (MO) optimization problem [31, 32, 33], where one objective is minimization of the DT, the main goal, and the other objective is the minimization of the absolute difference between the number of non-zero scaling weights and the desired value d_f, i.e.

$$F_1(\mathbf{w}) = \mathrm{Var}[r] \text{ on scaled data set } \mathbf{X}^{\mathrm{S}} \tag{3.2}$$

$$F_2(\mathbf{w}) = \left| d_f - |\{w_i \neq 0 \mid i = 1, \ldots, d\}| \right|. \tag{3.3}$$

MO optimization tries to find the Pareto-optimal front [34] (a set of non-dominated solutions) instead of a single solution. This set contains solutions where the values of the objective functions are in *conflict*, i.e. improving one objective leads to deterioration in the other objective(s). Therefore,

the result to a MO problem is a set of solutions on *different* pareto fronts, after which the user selects one (or more) based on his/her preference. The goal in an MO optimization is to find the global pareto-optimal front, which dominates all other fronts in the problem space. In this thesis, when the solutions are returned, we look for the one with the exact required number d_f of non-zero scaling weights and the smallest DT. If such a solution does not exist, the one with the lowest F_2 value is used, that is, we try to stay close to d_f variables. With the fixed scaling we introduce new parameter d_f to the problem. However, this parameter d_f should not be considered as the additional hyper-parameter for the optimization. By taking $d_f = d$ we expect to reach the lowest DT value for a given data set (this is pure scaling), while the $d_f < d$ cases *purposely exclude* some the variables for the gain in computational speed. With this restriction, search procedure is constrained to overlook the best possible weights for some number of variables, thus giving worse results than pure scaling.

The algorithm used for MO optimization will be explained alongside other search algorithms in Chapter 4.

3.3 Variable Projection

In a projection, a matrix $\mathbf{P} = [a_{ij}], i = 1, \ldots, d, j = 1, \ldots, k$ with size $d \times k$ is optimized according to a relevance criterion, and later used to obtain a new data set given by Equation 3.4.

$$\mathbf{X}^{\mathrm{P}}_{N\times k} = \mathbf{X}_{N\times d}\mathbf{P}_{d\times k}\,. \tag{3.4}$$

In this setting, scaling is a special case since it can be represented as a $d \times d$ matrix with weights w_i on the main diagonal of that matrix, i.e.

$$\mathbf{P}_{\mathrm{S}} = \begin{bmatrix} w_1 & 0 & \ldots & 0 \\ 0 & w_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & w_d \end{bmatrix}_{d\times d.} \tag{3.5}$$

A good property of projection is the ability to linearly transform the data set to lower dimensional space when the matrix $\mathbf{P}_{d\times k}$ has less columns than rows, i.e. $k < d$.

However, the number of parameters in $\mathbf{P}_{d\times k}$ is dk, and all have real values

from $\mathbb{R}$, thus the problem becomes even harder compared to the scaling problem with d parameters in a limited range. Furthermore, the correct value of k, i.e. the number of dimensions to project to, is an additional parameter that has to be optimized. The advantage is the manual choice of k, enabling full control of the dimensionality of the formed data set $\mathbf{X}^{\mathrm{P}}$. In the following Section 3.3.1 we propose a way to automatically select good value of k in projection problem when using the Delta Test as relevance criterion, while Section 3.3.2 investigates one interesting case of projection for $k > d$.

Although projection is more general than scaling, the interpretability is lost as projection matrix elements have real values (including negative ones), and nothing can be said by examining the values in the matrix.

3.3.1 Automatic Selection of Projection Dimension

Consider the case when projection matrix $\mathbf{P}_{d\times k}$ has one column, i.e. $k = 1$, and suppose that by optimizing the DT using matrix $\mathbf{P}_{d\times 1}$ we can obtain value dt_1. Same dt_1 value can be obtained with $k = 2$ by setting the second column of matrix $\mathbf{P}_{d\times 2}$ to zero values. This setting does not have any influence on the search process, resulting in optimization problem with only one column.

Since $\mathbf{P}_{d\times 2}$ contains real values, optimizing $\mathbf{P}_{d\times 2}$ should be able to reach value dt_2 that is at least as low as dt_1. However, adding new d parameters to $\mathbf{P}_{d\times 1}$ increases the complexity of the problem, adds new local minima and optimization of $\mathbf{P}_{d\times 2}$ becomes more challenging. This complexity is manifested through the value dt_2, which can be larger than dt_1, if both optimization problems are given the same amount of resources.

Same reasoning applies with higher values of k. Thus, as k is increasing, the value of the Delta Test should always decrease. In practice, huge number of parameters prevents $\mathbf{P}$ matrices with large k to reach same results of those cases with lower k. When optimizing the DT as a projection problem, there will be a value of $k = k_p$ after which the search procedure is unable to return lower DT values. Thus, we can conclude that matrix $\mathbf{P}_{d\times k_p}$ is our best estimate and considering $k > k_p$ values is a waste of resources.

Previous discussion stems the strategy for automatic selection of k and projection matrix $\mathbf{P}$. Start with $k = 1$ and optimize $\mathbf{P}_{d\times 1}$ to obtain DT estimate dt_1. Then, increase k by 1, optimize $\mathbf{P}_{d\times k}$ acquiring dt_k and compare dt_k with dt_{k-1}. If it holds that $dt_k < dt_{k-1}$ then continue increasing k, otherwise stop the process and return matrix $\mathbf{P}_{d\times(k-1)}$ as the final solution. This strategy is presented in Algorithm 3.1.

Algorithm 3.1 Automatic Selection of k

```
1: best = ∞
2: k = 1
3: while true do
4:    (dt, P_{d×k}) = optimize(X, Y, k)
5:    if dt ≥ best then
6:       break
7:    end if
8:    best = dt
9:    bestP = P_{d×k}
10:   k = k + 1
11: end while
12: return bestP
```

In the Algorithm 3.1, function *optimize* returns both the value dt and the corresponding projection matrix used to obtain that value. This function uses at least one search algorithm to minimize the Delta Test. The input parameters are the data set $(\mathbf{X}, \mathbf{Y})$ and the target projection dimension k. All algorithms explained in Chapter 4 depend on the initial solution(s) (Genetic Algorithm as stochastic algorithm, and Forward-Backward Search and Tabu Search influenced by starting position). To have reliable estimate of the DT, several calls of *optimize* function are necessary. In the experiments we have chosen to run the optimization function 10 times for each value of k. However, this approach can be too time consuming if the best value for k is very large. In such situation, all optimization steps in early iterations with smaller k are unnecessary. In order to speed up the computation, we slightly modify the Algorithm 3.1.

If the optimization for current projection dimension k (with value dt_k) improves the dt_{k-1} value then immediately move onto the next $k+1$ projection dimension. Otherwise, when the dt_k is larger than the previously found best value, try to improve it by running optimization several times until it does improve. If we cannot get lower DT value after T tries, stop increasing k. When the process stops at some k_s, the optimization of the previous values $k < k_s$ is not necessarily run T times. Then, we back up to two previous cases, $k_s - 1$ and $k_s - 2$, and finish optimizing both projection dimensions up to T times. Once the optimizations on $k_s - 1$ and $k_s - 2$ are done, the projection matrix producing a data set $\mathbf{X}^{\mathrm{P}}$ with the lowest DT is returned as the final result. In the experiments, number of tries T is set to 10. The whole procedure is given as Algorithm 3.2.

Algorithm 3.2 Faster Automatic Selection of k

```
T = 10 {set the maximum number of tries}
best = ∞
k = 1
while true do
   i = 1
   while i ≤ T do
      dt[k, i] = optimize(X, Y, k)
      if dt[k, i] < best then
         best = dt[k, i]
         break
      end if
      i = i + 1
   end while
   if i > T then
      break
   end if
   k = k + 1
end while
Finish optimization for dimensions k − 1 and k − 2 cases up to T tries
return P for minimum dt among dt[{k − 1, k − 2}, {1, . . . , T}]
```

At the end of these steps, a proper projection dimensionality k has been found and the data can be projected to a lower-dimensional space. One can then use this data for the actual regression task using any desired model.

3.3.2 Combining Scaling and Projection

Although the idea of projection is to reduce the dimensionality, in the following discussion we explore one interesting case when projection matrix $P_{d\times k}$ has more column than rows, i.e. when $k > d$. As explained in Section 3.3.1, increasing k leads to harder and harder problems which cannot obtain smaller DT values. The good value of k in our experiments was always less than d, so the question is why even consider cases when $k > d$.

The case that is of interest is the combination of the scaling and the projection. This combination aims to shape the data to have the following form:

$$\mathbf{X}^{\mathrm{SP}}_{N\times(d+k)} = [\mathbf{X}^{\mathrm{S}}_{N\times d}, \mathbf{X}^{\mathrm{P}}_{N\times k}] \,, \tag{3.6}$$

where $\mathbf{X}^{\mathrm{S}}$ is the scaled version of $\mathbf{X}$ (Equation 3.1), $\mathbf{X}^{\mathrm{P}}$ is the projected version of $\mathbf{X}$ (Equation 3.4) and $\mathbf{X}^{\mathrm{SP}}$ is the new scaled+projected input matrix. The new matrix that needs to be optimized has the form:

$$\mathbf{P}_{\mathrm{SP}} = \underbrace{\begin{bmatrix} w_1 & 0 & \cdots & 0 & a_{11} & a_{12} & \cdots & a_{1k} \\ 0 & w_2 & \cdots & 0 & a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_d & a_{d1} & a_{d2} & \cdots & a_{dk} \end{bmatrix}}_{\mathbf{P}_{\mathrm{S}} \qquad\qquad \mathbf{P}_{\mathrm{P}}}{}_{d\times(d+k)}\,, \tag{3.7}$$

where $\mathbf{P}_{\mathrm{S}}$ is the same as in Equation 3.5 and is responsible for scaling the data, while $\mathbf{P}_{\mathrm{P}}$ is "classic" projection given by Equation (3.4). For the rest of the thesis we call this approach *scaling plus projection*, and denote it as *scaling + projection*.

With a combination of both scaling and projection, the optimization problem should be able to reach a DT value that is not larger than the value obtained for scaling or projection alone. Consider the following two special cases. In the first special case, projection columns are set to zero values, i.e. $\mathbf{P}_{\mathrm{P}} = \mathbf{0}$, and we have $\mathbf{X}^{\mathrm{SP}} = [\mathbf{X}^{\mathrm{S}}, \mathbf{0}_{N\times k}]$. This special case is just a scaling problem with additional zero columns that do not influence the search process, but only increase computational time. The second special case is similar, with all scaling weights set to zero $\mathbf{P}_{\mathrm{S}} = \mathbf{0}$, and we have new data set $\mathbf{X}^{\mathrm{SP}} = [\mathbf{0}_{N\times d}, \mathbf{X}^{\mathrm{P}}]$, which is a pure projection problem with extra computational cost. These two extreme cases suggest that by allowing both scaling weights in $\mathbf{P}_{\mathrm{S}}$ and elements in $\mathbf{P}_{\mathrm{P}}$ to have real values, it becomes possible to find solutions that are at least as good as solutions for either scaling or projection problem. In the experiments we will clearly see the benefit of this merger.

Comparing projection and scaling + projection in terms of number of parameters to optimize, in projection of $\mathbf{P}_{d\times k}$ there are dk numbers, while in scaling + projection we have $d + dk = d(k + 1)$. The combined approach has only d extra parameters, which are from limited $[0, 1]$ range, making it not that more challenging than projection alone. The same difference of d parameters also exists between two projection cases $\mathbf{P}_{d\times k}$ and $\mathbf{P}_{d\times k+1}$, and instead of projecting to more and more dimensions it might be fruitful to include scaling to replace that one extra dimension of projection. On the other hand, in terms of computational speed, combined method takes much longer since we are increasing the dimensionality (from k dimensions in projection to $k + d$ dimensions in scaling + projection) which increases the nearest neighbor computations.

Since we are using projection in the combined method, which requires parameter k to be specified before optimization of $\mathbf{P}_{\mathrm{SP}}$, same algorithm that is used in projection alone (Section 3.3.1) can be applied here as well. The difference is only in the extra dimensions responsible for scaling $\mathbf{P}_{\mathrm{S}}$ that are added to form bigger $d \times (d + k)$ matrix $\mathbf{P}_{\mathrm{SP}}$.

The scaling + projection problem includes scaling, and thus we can also employ the fixed variant (Section 3.2.1) into the combined method. The combination of scaling with a fixed number of variables and projection will be referred to as *fixed scaling + projection*. The projection in this problem is not modified, only the scaling is replaced with the fixed version.

Chapter 4

Search Algorithms

Search algorithms in the context of variable selection are responsible for exploring through solution space in order to find an optimal solution with respect to the relevance criterion (the relevance criterion acts as an objective function for the process of optimization). There are many choices of to consider, and each of the available algorithms has its own advantages and drawbacks. Among the popular search algorithms used in machine learning are Forward Search [10], Backward Search [10] and Forward-Backward Search [11, 35]. All three methods are greedy in nature, and thus do not guarantee to return an optimal solution to the problem. The following sections briefly explain the workings of these three algorithms with special attention on the Forward-Backward Search.

As mentioned in Chapter 3, for a data set with d variables there are total of $2^d - 1$ possible subsets to consider. To guarantee optimality, an exhaustive search must be performed on all subsets. Since for large d this is infeasible, simple methods such as Forward and Backward Search explore only small portion of solution space, that is, they sacrifice optimality for faster execution time. Both methods take at most $d(d-1)/2$ steps explained in Section 4.1. Section 4.2 explains Tabu Search and the setup of its parameters, and Section 4.3 presents Genetic Algorithm, its operators, and the setup established for experiments.

4.1 Forward Search, Backward Search and Forward-Backward Search

Forward Search starts with an empty set S of variables and progressively adds new variables to S as long as the relavance criterion is improving over previous set with one less variable. In the first iteration d subsets are examined and the one with the best value of relevance criterion is selected, that is, one variable X_i is added permanently to the set S. In the next iteration, all subsets with two variables out of which one is X_i are explored. There are $d-1$ such subsets. Following the scheme in iteration j, a total of $d-j+1$ subsets are examined and the best variable is added to S. Finally, Forward Search explores at most $d(d-1)/2$ subsets since the algorithm may converge to a solution where $|S| < d$, i.e. the best subset has less than d variables.

Backward Search is similar to Forward Search, except that it starts with all variables included in the set S, and then removes variables one by one if the relevance criterion improves. With same calculations, Backward Search also explores at most $d(d-1)/2$ subsets.

Forward-Backward Search (FBS) is the combination of both previous methods. It can start from any subset of variables, and then considers new solutions which either add or drop one variable. This adding and dropping of variables is continued until the criterion no longer improves. Forward Search and Backward Search always return the same solution since the starting point is always the same. On the other hand, Forward-Backward Search can start from any subset of variables, which can result in different returned solutions. In order to get good selection of variables, FBS is run couple of tens or hundreds of times (depending on the problem) with different starting positions, and the best solution is returned as the final estimation of the criterion that is being optimized. The complete algorithm is summarized in the following 3 steps:

In the previously given steps, it is possible to generate an empty subset, which halts the execution of the optimization. If an empty subset is generated, we just remove it from the consideration set and enable the search to continue without interruption. This explicit check for the presence of an empty subset is left out from the algorithms presented in the thesis.

1. Initialization:

   Let S be the selected input variable set, which can contain any input variables, and F the unselected input variable set, which contains the variables not present in S. Compute Var[r] using Delta Test on the set S.

2. Forward-Backward selection step:

   Find the variable X_S to include or remove from the set S to minimize Var[r]

   $$X_S = \arg\, min_{X_i,X_j}\{\text{Var}[r] \mid \{S \cup X_j\} \cup \{S \setminus X_i\}, X^i \in S, X^j \in F\}$$

3. If the old value of Var[r] on the set S is lower than the new result, stop; otherwise, update set S and save the new Var[r]. Go to step 2.

4. Return set S as a solution.

4.1.1 Forward-Backward Search in Variable Scaling

The first modification that requires the adaptation to the scaling problem is the definition of the adding and dropping of variables. Because the weights are no longer binary $\{0, 1\}$ indicators, but rather values from discretized $[0, 1]$ range, adding can be constructed as increasing the scaling weight, and dropping as decreasing the scaling weight for a *single variable*. In the experiments, the distinction between increase and decrease is not explicit, that is, we only consider the change of a value for a single variable. Suppose that the set of scales has values $H = \{0, 0.5, 1\}$, corresponding to $h = 2$. Then, for a solution $v = (0, 1, 0.5)$ with $d = 3$ variables, the following set of solutions

$$\begin{array}{lll} (0.5, 0.5, & 1), & (\ 0, \ 0, \ 1), \quad (\ 0, 0.5, \ 0), \\ (\ 1, 0.5, & 1), & (\ 0, \ 1, \ 1), \quad (\ 0, 0.5, 0.5), \end{array}$$

are examined as the *closest* solutions to v. This closeness is encountered again in terms of neighbors for the Tabu Search in next section. In this setting, the number of solutions for examination is exactly hd. The solution from the examined set with the best relevance criterion is the selected solution for the next iteration. That is, we perform a change, also called *move* or *step*, from v to the best solution. This is the reason FBS is sometimes called a stepwise algorithm.

4.2 Tabu Search

Tabu Search (TS) is a meta-heuristic method designed to guide local search methods to explore the solution space beyond local optimality. The first most successful usage was by Glover [36, 37, 38] for combinatorial optimization. Later TS was successfully used in scheduling [39, 40, 41], design [42, 43], routing [44, 45] and general optimization problems [46, 47, 48]. The TS has become a powerful method with different components tied together, that is able to obtain excellent results in different problem domains.

Suppose we have a optimization problem in the form of an objective or cost function $f(v)$, and that solutions are members of a discrete set, i.e. $v \in V$.

The term meta-heuristic refers to the underlying idea of TS – it uses other technique, lets denote it L, for the search through the solution space. During the search, TS uses internal memory structures to modify the way L visits solutions. The memory is used to prevent the reversal of recent moves, and also to reinforce the exploration of promising ares of space. The memory designed for the first task is called short-term memory, while the second type is called long-term memory. The idea behind TS is to use technique L until it reaches an optimum, in which case the search is allowed to visit solutions with worse objective values. The memory structures keep track of local optima and the technique L is forbidden to revisit these in upcoming iterations. The concept of accepting worse solutions is also present in Simulated Annealing [49], where the acceptance is stochastic and based on a cooling scheme.

The basic elements behind TS are the definition of the search space and the neighborhood structure. Definition of search space is mentioned again in the context of genetic algorithms. For variable selection problem, the search space V is easily defined as a set of vectors of length d with binary $\{0, 1\}$ values. As mentioned in the previous section, the case when all variables are excluded is impossible and thus removed from the search space. This is classic encoding of solutions in variable selection problem, where 1 represents that the variable is selected and 0 that the variable is not selected. The scaling case has been adapted in the same manner as scaling for FBS, with $[0, 1]$ interval transformed into a set of equidistant values.

Next important issue is the definition of the neighborhood of a solution v. The neighborhood, denoted $Ne(v)$, is defined as the set of solutions that are *reachable* from the solution v. Reachability is defined through moves or local transformations applied to v in order to produce solutions in $Ne(v)$. In later section we define the moves for both problems: selection and scaling. There are the same as in FBS, but with more formal TS notation.

The short-term memory responsible for preventing cycling effects, also called tabu list, keeps track of recently used moves. Once an (sub)optimal solution has been found, tabu list forbids the search to revisit this solution by restricting the use of a move with reversing effect. Moves stored in tabu list are called tabu, and thus forbidden to use for a fixed number of iterations. Storing only moves does not guarantee prevention of cyclic effects, as not all information is kept when the optimum has been found. To stop revisiting, one can store *complete* solutions in the memory. However, this approach becomes impractical as the complexity of the problem increases, and substantial amount of execution time is spent on comparing new solutions to those in memory. This is the reason for storing smaller pieces of information, such as moves, segments or other attributes of solutions.

One important parameter of TS is the *tenure*, which is defined as the number of iterations a single move is considered tabu. In some implementations this corresponds to the length of the tabu list, usually coded as cyclic list. The tenure value is fixed throughout the whole search for most of the problems, but other approaches are possible: varying tenure value or randomly choosing the value for each move. In some problems this might help the search process, but in this thesis we only consider fixed tenures.

During the search, tabu list can prevent the moves to solutions which have not been encountered before (assuming no storage of complete solutions). This leads to a situation when TS discards a move to a solution with better objective value than the currently best one. To enable such moves, another level is added to TS, which allows the search to override tabu list and *aspire* to the new solution. This is known as aspiration criterion. The simplest aspiration criterion is to allow the move in the case as described, when the best solution has been found. Other criteria can be defined to revoke the tabu status, but they are seldom used. In the implementation of TS for the experiments, no aspiration criteria are used as they involve computing the actual objective value of a solution. By dropping the aspiration criteria, these saved DT evaluations are then used for new solutions, allowing more exploration of the search space in the same amount of time.

Due to the fact that this thesis considers the variable selection and the scaling problem, two different algorithms are designed. Both algorithms use only short-term recency based memory to store *reverse moves* instead of solutions to speedup the exploration of the search space. This setup is known as Simple Tabu Search [17], and the pseudocode is given as Algorithm 4.1.

In the Algorithm 4.1, a new term is introduced – stopping condition. The stopping condition, or termination criterion, prevents the TS to run indef-

Algorithm 4.1 Simple Tabu Search storing reverse moves in list

1: $TabuList = []$
2: choose starting solution $v \in V$
3: **while** not *stopping condition* **do**
4: $T = \{u \mid u \in Ne(v), \text{move}(v, u) \in TabuList\}$
5: $N = Ne(v) \setminus T$
6: choose $u \in N$ with minimum $f(u)$
7: add move(u, v) to $TabuList$
8: $v = u$
9: remove from $TabuList$ moves added *tenure* iterations ago
10: **if** $f(v) < bestF$ **then**
11: $bestF = f(v)$
12: $bestV = v$
13: **end if**
14: **end while**
15: **return** $bestV$

initely. This condition can be set to one of the following possibilities: the amount of time spent on optimization, the number of calls of the objective function f, the amount of memory used, and other choices. The first two conditions are the usually employed in the domain of optimization.

There are other parts of the TS which make it a powerful method, such as probabilistic TS, candidate list generation, intensification and diversification strategies, auxiliary objectives. These are not considered here, but for a detailed explanations on the topics see [17].

The following sections give the definition of moves and neighborhood structures for variable selection and scaling. The structures are exactly the same as the ones given for FBS. As TS is a meta-heuristic method, the final algorithm can be seen as FBS with tabu conditions.

4.2.1 Tabu Search for Variable Selection

In the case of variable selection, a move is defined as a flip of the status of exactly one variable in the data set. The status is excluded (0) or included (1) from the selection. For a data set of dimensionality d, a solution is then a vector of zeros and ones $v = (v_1, v_2, \ldots, v_d)$, where $v_i \in \{0, 1\}, i = 1, ..., d$, are indicator variables representing the selection status of k-th dimension.

The neighborhood of a selection (solution) v is a set of selections u which

have exactly one variable that has different status. This can be written as

$$Ne(v) = \{u \mid \exists_1 q \in \{1, \ldots, d\} \; v_q \neq u_q \wedge v_i = u_i, i \neq q\} \tag{4.1}$$

With this setup, each solution has exactly the same amount of neighbors, which is equal to d.

4.2.2 Tabu Search for Variable Scaling

TS for the optimization of scaling weights is defined in the same manner as FBS for the same problem in Section 4.1.1. The difference is the notion of neighbors of a solution in TS. In TS, a solution v is now a vector with scaling values from a discretized set $v_h \in H = \{0, 1/h, 2/h, \ldots, 1\}$, where h is discretization parameter. Two solutions are neighbors if they *disagree* on exactly one variable. For example, for $h = 10$, $d = 3$ and solution $v_1 = (0.4, 0.2, 0.8)$, a solution $v_2 = (0.7, 0.2, 0.8)$ would be a neighbor, but not the solution $v_3 = (0.1, 0.5, 0.8)$. The move between solutions is defined as a change of value for one dimension, which can be written as a vector (dimension, old value, new value).

4.2.3 Setting the Tabu Conditions

As mentioned, the *tenure* for a move is defined as the number of iterations that it is considered tabu. This value is determined empirically when the TS is applied to solve a concrete problem. For the variable selection problem, the thesis proposes a value which is dependent on the number of dimensions so it can be applied to several problems. In the experiments, two tabu lists, and thus two tenures, are used. The first list is responsible for preventing the change along certain dimension for $d/4$ iterations. The second one prevents the change along the same dimension and for specified scaling value for $d/4+2$ iterations. The combination of these two lists gives better results than when each of the conditions is used alone.

For example, for $h = 10$, if a move is performed along dimension 3 from value 0.1 to 0.5, which can be written as a vector $m = (3; 0.1, 0.5)$, then the value 3 is stored in the first list and the reverse move $m^{-1} = (3; 0.5, 0.1)$ is stored in the second list. The search will be forbidden to use any move along dimension 3 for $d/4$ iterations (first condition), and after that time, it will be further 2 iterations restricted to use the move m^{-1} (second condition), or in other words to go back from 0.5 to 0.1.

With these settings, in the case of variable selection, two conditions are then implicitly merged into one condition: restrict a flip of the variable for $d/4+2$ iterations. This is because there are only two values $\{0, 1\}$ as possible choices.

4.3 Genetic Algorithm

Genetic Algorithm (GA) is one of the algorithms of the larger family of optimization techniques known as Evolutionary Algorithms (EAs) [50]. All of these algorithms share concepts found in biological processes, such as natural selection and survival of the fittest principle. The algorithms are population based, meaning that there is a set of solutions present at all stages of the optimization. The solutions are chosen randomly from the search space of the problem, and all EAs are considered *stochastic* search algorithms. The process of searching through the solution space is influenced by several mechanisms common to all EAs: how are the solutions encoded as the chromosomes, initialization of the population, selection operators, and reproduction operators. We briefly give the representation and workings of the GA on a general level, while numerous publications give more formal definitions and theory behind genetic algorithms [51, 52, 53].

To understand better GA (and other EAs), this algorithm should be compared to the classical optimization methods. Regarding the search process, classic methods uses deterministic rules to move from one solution to the next in the search space, while EAs use probabilistic rules. In EA, the starting position is a set of solution, as in classic approach it is only a single point which is improved upon in sequential steps. Classical methods also use derivative information (first-order, second-order) to guide the search, while EA uses only information about the *fitness* of the individuals. That is, GA uses only information about the surface of the space to decide on new directions of the search.

Genetic algorithms have been widely used in machine learning community for variable selection [2, 54, 55, 56, 57, 58], with most of the work devoted to classification tasks.

4.3.1 Genetic Algorithm Basics

The Algorithm 4.2 shows the outline of GA, which with small modifications can be made into any other EA paradigm. The functionality of each of the operators is explained in the following sections.

Algorithm 4.2 Outline of Simple Genetic Algorithm

1: select *selection* operators σ_1 and σ_2
2: select *reproduction* operators ρ
3: P = create initial population
4: **while** not *stopping condition* **do**
5: $e = \text{fitness}(P)$ {evaluate population}
6: $P_1 = \sigma_1(e)$ {select parents}
7: $P_2 = \rho(P_1)$ {reproduction – generate offspring}
8: $P = \sigma_2(P, P_2)$ {select new generation}
9: **end while**
10: **return** p the fittest individual in P

Representing Solutions

In the context of the GA, each individual in the population represents a solution to an optimization problem. The *characteristics* of an individual are represented by *a chromosome*. These characteristics refer to the variables of the problem, and in the context of the EA, a variable is called *a gene*. For variable selection problems, each gene corresponds to: a selection status for variable selection; scaling weight for variable scaling; real number in the matrix for projection problem. For the rest of the thesis, we interchange the terms solution, individual and chromosome, since values (genes) in a chromosome fully explain the individual, which is in fact a solution for the problem. This is also done with the terms variable and gene. The nature of a problem also has the impact on the coding scheme of individuals.

Initial Population

GA is population-based algorithm, thus requires a pool of solutions in order to apply the operators. The first step in GA (and all other EAs) is the creation of this population. Most of the time this is done by randomly sampling the solution space. The goal of random selection is to ensure that most of the search space is *covered*, that is, the solutions should be uniformly spread across the space. If some of the regions are left out, there is a possibility that the search will neglect this regions. The size of the population is one of the parameters to be decided before optimization. There is a trade-off between the size of the population and the convergence speed of the GA. The larger the population, the better it is spread across the space (more diversity), and less iterations are needed to find a good solution. On the other hand, smaller

populations need more generations to reach an acceptable solution. One has also to consider the computation time per iteration: the more individuals in the population, the more time it takes to compute one generation.

Evaluation

Evaluation is a simple step, which involves computing the objective function value for each individual in the population. As discussed later, some selection operators use this information in order to select the individuals for the reproduction step. It is usually the practice to *scale* the values of the objective values to a more representative range. This is accomplished with a fitness function, which can be linear or non-linear. This function transforms the objective values into the fitness values.

Selection

One of the main operators in GA is the selection operator. Its main purpose is to emphasize better solutions. Selection takes part in two phases of the algorithm:

- **Selecting individuals for reproduction** σ_1: New individuals, called offspring, are created by applying *crossover* and/or *mutation*. The selection for the crossover phase should favor fit individuals, ensuring that their genes are passed onto the next generation. In the case of mutation, selection mechanisms should focus on "weak" individuals. Introducing new genetic material into weaker chromosomes might improve their fitness, enabling them to compete with fitter individuals.
- **Selecting individuals for the new population** σ_2: When the offspring are generated, the decision is on how to select the individuals for the next generation based on the current generation and offspring. This can be done using only offspring or the combination of both sets. The selection operator should ensure that good individuals are present in the next population.

Selection operators are often characterized by their *selective pressure*, which is defined as the speed at which the best solution will occupy entire population by repeated application of the selection operator alone. High selective pressure makes the population lose diversity and degrades the exploration

abilities of GA, while low selective pressure might take more time to converge.

The selection is mostly based on the value of the objective function, leading to proportional selection. In this setting, individuals with better objective value are selected more often. To prevent the values of the objective function from dominating the selection process, all values are transformed by fitness function, which scales the objective values into *fitness values*. Fitness values form more suitable range of values to prevent the dominating effect of better individuals.

One operator that does not need fitness scaling is the tournament selection, which is used in the experiments. Tournament selection selects a group of $n_t > 1$ individuals randomly from the population. The objective values of these individuals are then compared, and the best one is returned by the operator. For the crossover with two parents, the selection is performed twice, once for each parent. If the tournament size n_t is not too large, this type of selection prevents the best individual from dominating. In the case of small tournament size, the chances that bad individuals are selected increase.

Selection operator called *elitism* is one of the most used selection operators. With elitism, a number e_n of the best individuals of the current population is just copied into the new population. This approach guarantees that the fitness of the population never deteriorates. The remaining part of the new population is filled with individuals decided by the selection operator σ_2.

Reproduction

Reproduction is the process of generating offspring from the selected parents by applying crossover and/or mutation. Crossover is responsible for creating new individuals by recombining the genes of the two or more parents. Mutation works by randomly changing the values of the genes in a chromosome. The purpose of mutation is to introduce new genetic material and bring diversity in the population. Care must be taken in order to *not destroy* the genes of the best individuals.

Crossover operators are categorized based on the representation scheme: binary or real valued problem. For each of the problems there exist specific crossovers. An important part of GA is that the crossover is *probabilistic*, that is, once the parents have been selected, they exchange their genes with certain probability. Usually, a high crossover probability (also called crossover rate) is used to favor creation of new individuals.

Stopping Conditions

The mentioned operators are applied in each generation until a stopping criterion is satisfied. The simplest stopping condition is to limit the search based on the number of generations that GA is allowed to execute. Other simple solution is to limit on elapsed time since the start of the algorithm. More elaborate criteria exist as well, that are based on convergence of the algorithm: terminate when there is no change in the population, terminate when no improvement is made over a number of consecutive generations, terminate if an acceptable solution is found. These are all loose definitions of the convergence of GA.

4.3.2 Setup for the Experiments

In this section, we give a list of operators and encoding schemes that are used in the experiments.

Encoding of the Individuals

As is the case in TS, the individuals of the GA are vectors with binary values. Since this thesis also considers variable scaling, other encoding must be chosen to accommodate new approach. If instead of using 0 and 1, the algorithm uses real numbers to determine the weigh of a variable, the GA could fall into the category of Real Coded Genetic Algorithms (RCGA). However, the number of scaling weights has been discretized in order to easily compare the performance of GA to those of FBS and TS. This discretization makes the algorithm a classical GA where the cardinality of the alphabet increases to $h + 1$ values (with h being the number of subintervals). Both approaches, classical BCGA with extended alphabet and RCGA, are tested in the experiments. For RCGA, the scaling set is not discretized, i.e. weights take values from the whole scaling interval $[0, 1]$.

Initial Population

Regarding the initial population for BCGA, some individuals are included in the population deterministically to ensure that each scaling value for each variable exists in the population. These individuals are required if the classical GA crossover operators (one/two-points, uniform) are applied in order to reach all possible combinations (assuming no mutation). For example, for

$h = 3$ (producing $H = \{0, 1/3, 2/3, 1\}$) and 3-dimensional problem ($d = 3$), the following individuals are always included in the population: (0, 0, 0), (1/3, 1/3, 1/3), (2/3, 2/3, 2/3) and (1, 1, 1).

Selection, Crossover and Mutation Operators

The GA algorithm is designed to be as fast as possible so when several designs options appeared, the fastest one (in terms of computation time) was selected. The selection operator chosen is the binary tournament selection by Goldberg [51], instead of the roulette wheel operator [59] or other complex operators [60]. Tournament selection does not require computation of any probabilities, saving a considerable amount of operations in each iteration. This is especially important for large populations. Binary tournament has very low selective pressure, allowing less fit individuals to be selected. Nevertheless, this selection mechanism allows better exploration capabilities which are important in the case of variable scaling (discretized or unrestricted). However, the algorithm also incorporates the elitism mechanism, keeping the 10% of the best individuals of the population, so the convergence is still feasible.

For the initial test, the classical operators (one-point, two-point, uniform) for binary problems were tested. The performance of these operators was similar and acceptable. Nonetheless, since the algorithm could be included into the Real Coded GA class, an adaptation of the BLX-α [61] was implemented as well. This operator is designed for continuous problems, and works as follows: given two individuals $I_1 = (i_1^1, i_2^1, ...i_d^1)$ and $I_2 = (i_1^2, i_2^2, ...i_d^2)$ with $(i \in \mathbb{R})$, a new offspring $O = (o_1, ..., o_j, ..., o_d)$ can be generated where $o_j, j = 1...d$ is a random value chosen from a uniform distribution within the interval $[i_{min} - \alpha \cdot B, i_{max} + \alpha \cdot B]$ where $i_{min} = \min(i_j^1, i_j^2)$, $i_{max} = \max(i_j^1, i_j^2)$, $B = i_{max} - i_{min}$ and $\alpha \in \mathbb{R}$. The adaptation to discrete alphabet requires rounding of gene values to match the scaling weights. The working of the BLX-α is depicted in Figure 4.1 for the case $d = 2$.

The mutation operates at a gene level, so a gene has the chance to get any value of the alphabet.

The setup of the GA is summarized in the following list:

- Selection operator: Binary tournament ($n_t = 2$)
- Crossover operator: BLX-α ($\alpha = 0.5$)
- Crossover rate: 0.85

- Mutation operator: Random uniform on gene level
- Mutation rate: 0.1
- Elitism: 10% of population size
- Replacement: Complete, i.e. generational approach

With the complete replacement, the selection operator σ_2 chooses only the offspring, i.e., in line 8 of the Algorithm 4.2 we have $\sigma_2(P, P_2) = P_2$.

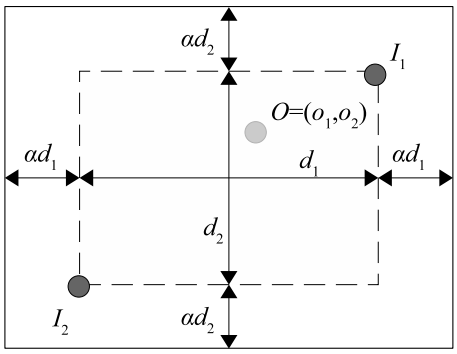


Figure 4.1: BLX-α crossover in 2-dimensional space. Offspring O is chosen randomly from extended rectangle (solid line) defined by two parents I_1, I_2 (dashed line).

4.3.3 Parallel Implementation

Genetic algorithms work with a population of independent solutions, which makes it easy to distribute the workload from one processor to several, speeding the execution time. It is very easy to adapt them to parallel architectures.

Going into more detail, the first step is to decide upon the distribution of the population. The GA can still remain with a single population, or the population can be divided into several populations. The latter approach is also called GA with multiple populations. When considering the second case, subpopulations can remain separated or communicate between themselves. Communication involves extra cost, as well as decisions about the pattern of communications (policy), the number of individuals for exchange (also called migration), and the rate of exchange. Careful decisions must be taken in order not to burden the execution with excessive communication patterns.

The idea behind parallel GAs is to divide larger problem into smaller ones, and use separate processors to solve these problems simultaneously. There are many possibilities for this division, and the the common classification of parallel GA is in following caterogies: single-population master-slave GAs [62], multiple-population GAs, fine-grained GAs, and hierarchical GAs. We briefly explain the first two categories.

Master-slave topology is defined by having *a master* processor, or node, to perform sequential part of the GA – selection and reproduction, while all the other processors, *slaves*, evaluate the individuals of the population. The communication consists of sending the individuals from the master node to the slaves (first direction), and after evaluation, the slaves send the fitness values for the individuals they received to the master node. Master-slave approach uses only single population, thus there is no difference in results between parallel master-slave GA and serial GA, assuming same operators.

Multi-population GAs divide the population into several smaller groups, called subpopulations. Subpopulations exchange information by migrating some of the individuals. The migration is control by several parameters, such as the sizes of the subpopulations, rate of migration, the number of migrating individuals, migration pattern, and the selection of which individuals should migrate. Multiple-population parallel GAs are also known as the *island model*, since subpopulation are split into groups (islands) with migration patterns between any two subpopulations.

Smaller populations lower the diversity of solutions, and this decreases the expected convergence time. When a population is spread across multiple processors, we may expect faster results due to this convergence. However, the final solution returned should be of the same quality as that of serial GA, suggesting that the division should be done carefully to balance this trade-off.

Unlike master-slave version, multiple-population approach does not return the same results as the serial GA, since operators do not take into account the whole population, only one subpopulation at a time.

Another important aspect in parallel implementations is the synchronization scheme. There are two types: synchronous and asynchronous, and both can be adapted to each of the parallel categories of the GAs. In the synchronous implementation, all processors always have the same population at disposal and the communication exist to synchronize the processes. This effectively means that faster processors have to wait for the slower ones. When all processors execute their jobs, the algorithm continues onto the next generation. On the other hand, in asynchronous case, all processors execute the code

without any delay, resulting is much less idle time. However, the interpretation and comparison of the results with synchronous GAs is difficult, since the communications occur at random times.

Implementation for Delta Test Optimization

As discussed in Section 4.3.2, the algorithm is designed to be as fast as possible. Nonetheless, the fitness function (DT) still remains expensive in comparison with the other stages of the algorithm (selection, crossover, mutation). At first, all the stages of the GA were parallelized, but the results showed that the communication and synchronization operations could be more expensive than performing the stages synchronously and separately on each processor (we consider that a processor executes one process of the algorithm). Hence, only the computation of the DT for each individual is distributed between the different processors. This corresponds to the master-slave topology that is easily implemented. Some questions might arise at this point like: Are the processors homogeneous?; How many individuals are sent at a time?; Is the fitness computation time constant?

The algorithm assumes that all processors are equal with the same amount of memory and speed. If they are not, it should be considered to send the individuals iteratively to each processors as soon as they have finished with the computation of the fitness of an individual. This is equivalent to the case when the fitness function computational time might change from one individual to another. However, the computation of the DT does not significantly vary from one individual to the another in a larger population. Thus, using homogeneous processors and constant time consuming fitness function, the amount of individuals that each processor should evaluate is $size\ of\ population/number\ of\ processors$.

The algorithm has been implemented so that the amount of communication (and the number of packets) is minimized. To achieve this, all the processors execute exactly the same code so, when each one of them has to evaluate its part of the population, it does not require to get the data from the master because it already has the current population. The only communication during execution of GA are receiving and sending of values of the DT, but not the individuals themselves. The exchange of DT values is done after the evaluation of the individuals. To ensure that all processors have the same population all the time (considering the presence of random values), at the beginning of the algorithm the master processor sends the seed for the random number generator to all the slaves. Having same random seed implies

that they all produce the same values when calling the function to obtain a random number. This enables quicker communication between processors as the only information sent is computed DT value of the individual. This is especially important since some problems require large individuals, increasing the amount of traffic in the network and slowing the execution of the algorithm.

4.3.4 Genetic Algorithm for Multi-Objective Optimization

The main idea in MO problems is to find the global pareto-optimal front. Of course, this cannot be guaranteed, but the algorithms designed for this problem must have two properties: generating solutions along other pareto-optimal fronts and finding new fronts. In the reproduction phase, it is common to generate solutions that are dominated by other individuals in the population. These have to be discarded since they are of no interest. The algorithm used for multi-objective optimization in the fixed scaling problem is the Elitist Non-Dominated Sorting Genetic Algorithm (NSGA-II) proposed in [34].

NSGA-II algorithm works in two basic steps: sorting of solutions based on dominance and elitism selection to keep the best fronts encountered. Since the population size does not change, the last front for inclusion has to be divided in two parts. The division is done to keep the most diverse solutions for the next generations. This diversity in NSGA-II is calculated using the *crowding distance*, which measures the distance between solutions in the objective function space. With this approach, the algorithm excludes the *sharing parameter*, which is responsible for calculating the proximity between population members and has to be defined by user. Sorting of solutions is done by careful book-keeping in order to speed up the execution time. For details refer to [34]. The overall complexity of the algorithm is $O(mp^2)$, where m is the number of objectives (in our case $m = 2$) and p is the size of the population. One thing worth mentioning is the constant factor in the mentioned asymptotic running time. The algorithm works by sorting on the set of *both* the current population and the offspring, doubling the size of the set. With this taken into account, the more precise complexity is $O(m(2p)^2)$.

Chapter 5

Experiments: Search Algorithms

The experiments were carried out on a variety of computer architectures and different setups, but MATLAB was used as the main environment to run the experiments. In this chapter we compare the performance of the search algorithms explained in Chapter 4 on several regression data sets.

A number of data sets with varying number of samples and dimensionality was used to test the quality of the composition of Delta Test with the three mentioned search algorithms. The following data sets were used for comparing the performance of FBS, TS and GA, with Table 5.1 summarizing the sizes of all data sets.

1. Housing data set [67]: The housing data set is related to the estimation of housing values in suburbs of Boston. The value to predict is the median value of owner-occupied homes in $1000's. The data set contains 506 instances, with 13 input variables and one output.

2. Tecator data set [68]: The Tecator data set aims at performing the task of predicting the fat content of a meat sample on the basis of its near infrared absorbance spectrum. The data set contains 215 useful instances for interpolation problems, with 100 input channels, 22 principal components (which remain unused) and 3 outputs, although only one is going to be used (fat content).

3. Anthrokids data set [69]: This data set represents the results of a three-year study on 3900 infants and children representative of the U.S. population of year 1977, ranging in age from newborn to 12 years of age. The data set comprises 121 variables with the weight of a child being the target variable. As this data set presented many missing values, a

prior sample and variable discrimination had to be performed to build a robust and reliable data set. The final set without missing values contains 1019 instances, 53 input variables and one output (weight). More information on this data set reduction methodology can be found in [63].

4. The Santa Fe time series competition data set [70]: The Santa Fe data set is a time series recorded from laboratory measurements of a Far-Infrared-Laser in a chaotic state, and proposed for a time series competition in 1994. The set contains 1000 samples, and it was reshaped for its application to time series prediction using regressors of 12 samples. Thus, the set used in this work contains 987 instances, 12 inputs and one output.

5. ESTSP 2007 competition data set [71]: This time series was proposed for the European Symposium on Time Series Prediction 2007. It is an univariate set containing 875 samples, while the regressor size for this series varied for different set of experiments as explained in this chapter and the next one.

Dataset	Samples	Input variables
Boston Housing	506	13
Anthrokids	1019	53
Tecator	215	100
Santa Fe	987	12
ESTSP 2007	819	55

Table 5.1: Data sets used for testing the performance of search algorithms.

5.1 Approximate Nearest Neighbor Influence

First we show the importance of using faster nearest neighbor search when optimizing the DT. Table 5.2 shows the average running times for the Genetic Algorithm for Santa Fe, ESTSP 2007 and Anthrokids data sets. As can be seen, the computational savings from using underlying data structure in ANN is substantial, with improvement of 80% for Santa Fe and roughly 90% for both ESTSP 2007 and Anthrokids. The results are averaged over 10 runs, while the GA uses 200 complete generations as the stopping criterion.

Data set	Naive search	Approximate k-NN
Santa Fe	620	124
ESTSP 2007	2573	283
Anthrokids	2938	314

Table 5.2: Average running time in seconds for DT optimization using naive NN approach and approximate k-NN search.

5.2 Performance of Search Algorithms

Next experiment compares different search algorithms used for searching through solutions space, for selection and discretized scaling. The scaling weights are set up to take values from $H = \{0, 0.1, 0.2, \ldots, 0.9, 1\}$ set. This experiments involves standard Forward-Backward Search, Tabu Search and Genetic Algorithm. Table 5.3 summarizes the statistics of DT values obtained with all three methods on 5 data sets. The stopping condition for all methods was set to 10000 DT evaluations. Since FBS is a greedy method, and converges in less than 10000 evaluations, it is reinitialized from another random starting solution. This procedure is repeated until the algorithm examines specified 10000 solutions. TS is only initialized *once* from a single solution and evaluates the subsequent solutions until it reaches the mentioned stopping criterion. Each of the methods is run 10 times since all of them are influenced by random initial solution(s). The size of the population of the GA was set to 150 [3, 4].

All data sets were normalized to zero mean and unit variance, including the output variable. Therefore, all DT values shown in this section are normalized by the variance of the output. The normalization was done variable-wise for all data sets except for Tecator, in which variable selection works better with sample-wise normalization.

As can be seen from Table 5.3, FBS performs quite well over TS. This is due to couple of initializations of FBS as soon as it has converged. On the other hand, TS is only initialized from one point and explores the solution space until it reaches 10000 evaluations. This shows that there are many local minima for all data sets, and that reinitialization is beneficial for local methods. For two data sets with the least number of variables, Housing and Santa Fe with 13 and 12 variables respectively, there are less than 10000 possible solutions in total for the selection problem, and search algorithms should return the global minimum for these data sets. However, only FBS is able to do so on both of them in all 10 runs, and GA only on Santa Fe. TS has trouble finding these in some runs, mostly due to high tenure value

Dataset	DT value	Selection			Scaling		
		FBS	TS	GA	FBS	TS	GA
Anthrokids	Mean	**0.00872**	0.00878	0.01316	0.01419	0.01432	**0.00856**
	Std	0.00073	0.00030	0.00148	0.00102	0.00186	0.00049
	Min	0.00840	0.00844	0.01183	0.01273	0.01233	0.00799
	Max	0.01078	0.00918	0.01512	0.01556	0.01681	0.00907
ESTSP 2007	Mean	0.01368	**0.01312**	0.01480	0.01452	0.01427	**0.01254**
	Std	0.00024	0.00026	0.00033	0.00072	0.00088	0.00019
	Min	0.01339	0.01257	0.01425	0.01326	0.01309	0.01217
	Max	0.01410	0.01345	0.01515	0.01571	0.01596	0.01290
Housing	Mean	**0.07104**	0.07123	0.07113	0.05816	0.05871	**0.05690**
	Std	0.00000	0.00040	0.00030	0.00142	0.00610	0.00119
	Min	0.07104	0.07104	0.07104	0.05518	0.05558	0.05578
	Max	0.07104	0.07199	0.07199	0.06073	0.07591	0.05948
Santa Fe	Mean	**0.01647**	0.01783	**0.01647**	0.01112	0.01163	**0.01053**
	Std	0.00000	0.00145	0.00000	0.00070	0.00086	0.00073
	Min	0.01647	0.01647	0.01647	0.00954	0.01095	0.00942
	Max	0.01647	0.02097	0.01647	0.01221	0.01334	0.01115
Tecator	Mean	0.01485	**0.01310**	0.01702	0.02372	0.02588	**0.01388**
	Std	0.00175	0.00098	0.00064	0.00222	0.00382	0.00025
	Min	0.01327	0.01114	0.01624	0.02104	0.02175	0.01365
	Max	0.01944	0.01387	0.01844	0.02758	0.03495	0.01442

Table 5.3: Performance comparison of Forward-Backward Search, Tabu Search and Genetic Algorithm for selection and discretized scaling

of 5 for both. Nevertheless, TS has much better results in data sets with large number of variables, particularly for Tecator, for which it found the best minimum value, even outperforming GA for scaling.

The results of GA for selection are disappointing, which suggest that GA has problems converging to some local minima. However, its performance in scaling problem is noticeably superior to those of FBS and TS, and even surpasses the selection results of FBS/TS (except for Tecator). In scaling, the exploration capabilities of GA are more evident, while FBS and TS both highly depend on initial solution.

Table 5.4 shows the computation time of the three algorithms for Anthrokids, ESTSP 2007 and Tecator data sets. Also shown is the percentage of this time used for generating new solutions. It is clear that most of the optimization is spent on DT evaluations. As expected, the GA has the highest time spent on generating new populations, due to the complexity of its operator which also involves generating random numbers. Random number generation is absent in both FBS and TS. The surprising result was the time spent by FBS and TS on Anthrokids data, with TS having roughly 40% faster running time on the *same* number of DT evaluations. Figure 5.1 explains this observation. TS starting from one initial position favors moves toward those regions of space

Data	FBS		TS		GA	
	time	%	time	%	time	%
Anthrokids	361.5	0.16	217.5	0.04	442.4	1.64
ESTSP 2007	98.9	0.22	100.5	0.04	99.5	1.62
Tecator	17.5	0.18	16.3	0.10	24.8	2.42
Santa Fe	30.1	1.68	29.7	0.24	29.6	1.81
Housing	22.8	1.68	24.6	0.31	20.5	1.79

Table 5.4: Average running time in seconds and percentage of that time spent of generating new solutions. Given values are for the selection problem.

that have less variables selected in the solution. Combining this results with DT values from Table 5.3, TS is able to find solutions with DT values on the same level as FBS, but with smaller number of variables. This effectively reduces the computation time which goes up to 40% for Anthrokids, while for the other data sets there is no clear distinction.

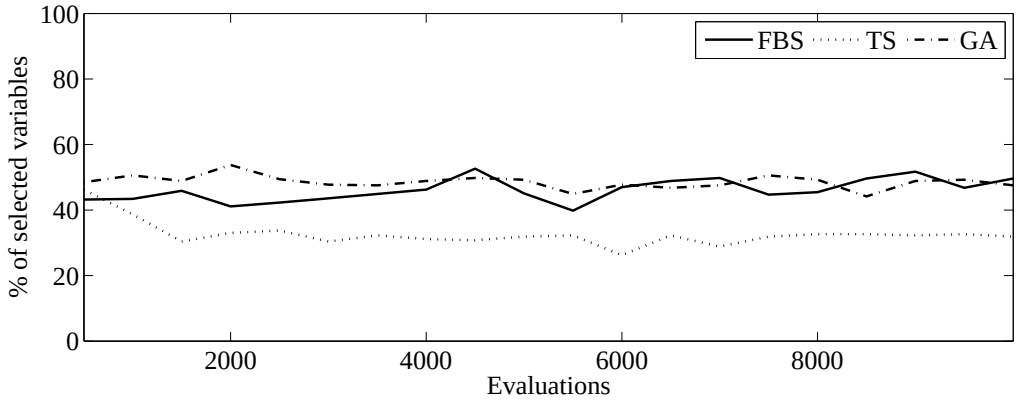
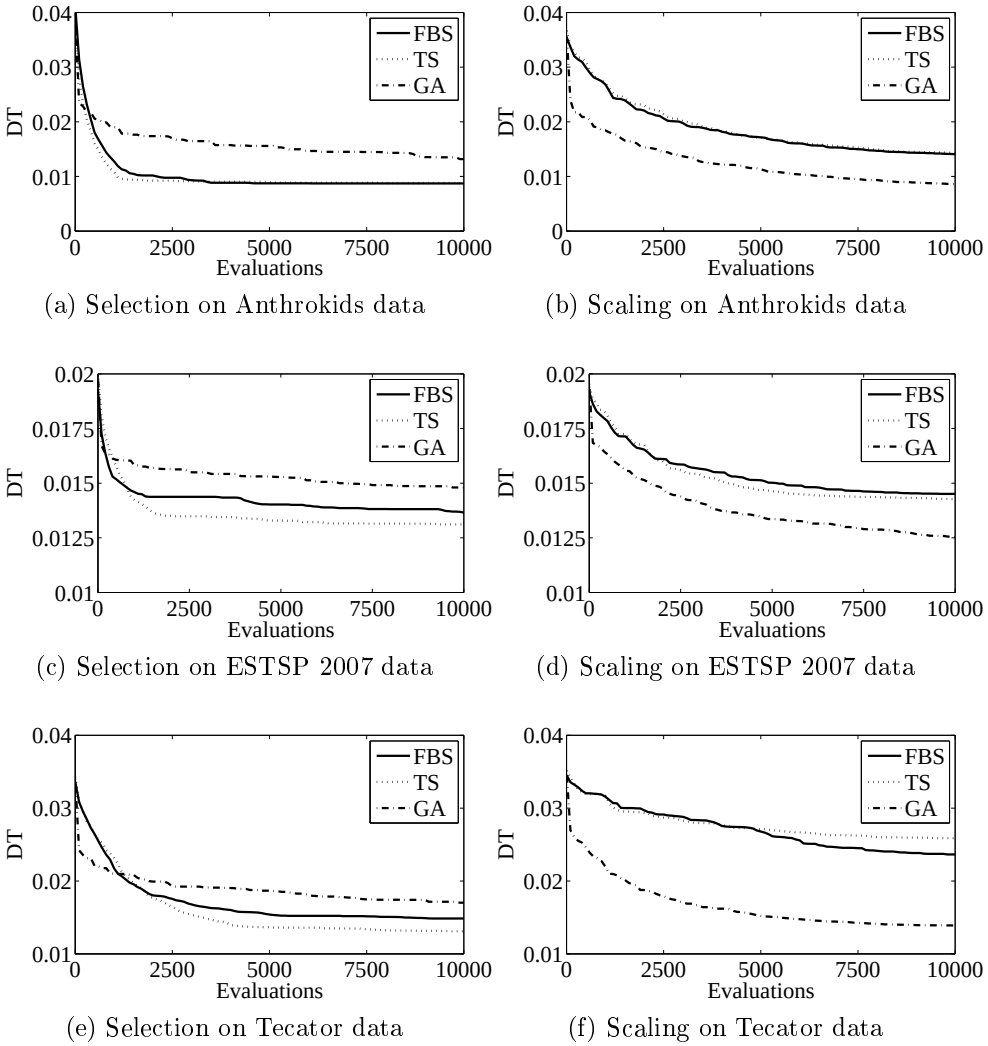


Figure 5.1: Percentage of selected variables throughout the execution of Forward-Backward Search, Tabu Search and Genetic Algorithm for Anthrokids data.

Figure 5.2 shows the evolution of the DT value as a function of DT evaluations. As discussed, GA does poorly in selection, while in scaling its noticeably better. TS has better performance on average over FBS in later iterations since it searches from a single initial solution, while FBS is better in first 1000-2000 DT evaluations benefiting from several starting points. In selection case, we see that GA is able to find promising regions of solution space in the first couple of evaluations, but is unable to converge. Having in mind exploratory capabilities of GA and local convergence of TS, a hybrid

approach was developed in [2] with superior result than all three algorithms presented here.



(a) Selection on Anthrokids data

(b) Scaling on Anthrokids data

(c) Selection on ESTSP 2007 data

(d) Scaling on ESTSP 2007 data

(e) Selection on Tecator data

(f) Scaling on Tecator data

Figure 5.2: Performance of Forward-Backward Search, Tabu Search and Genetic Algorithm as a function of the number of DT evaluations.

For all previous experiments, GA is set up as binary coded GA, both for selection and for scaling. In the case of scaling, the alphabet consists of scaling weights from a discretized $[0, 1]$ range. The next experiment investigates whether this restriction affects the results of the GA. Table 5.5 shows the difference between DT values obtained with discretized scaling (BCGA) and pure scaling (RCGA).

The performance of BCGA and RCGA in the scaling problem are quite

Data	BCGA	RCGA
Anthrokids	0.00856	0.00894
ESTSP 2007	0.01254	0.01253
Housing	0.05690	0.05524
Santa Fe	0.01053	0.00966
Tecator	0.01388	0.01385

Table 5.5: Average DT values for Binary Coded and Real Coded Genetic Algorithm in variable scaling.

similar and neither is clearly the best choice in terms of average DT value. The advantage of RCGA over BCGA is the absence of rounding of genes to match the scaling weights in set H, which saves time in each generations. Therefore, we use Real-Coded GA in the rest of the experiments, which also relieves the burden of choosing the right set of scaling weights H. Following section explains how can we further lower the value of the DT by smartly initializing the population.

5.3 Custom Initialization of Population for Genetic Algorithm

As shown in Figure 5.1, majority of solutions had less than 50% of selected variables. With this in mind, the initialization of population of GA was modified to take this into account, that is, the population is created to have a lot of zeros among the individuals.

This new approach involves having a percentage of the initial population with lots of zero values, while the rest is created in standard way by sampling from uniform distribution. Table 5.6 shows the effect of having many zeros in the initial population for the scaling problem. In this experiment, the stopping condition for GA is 50 generations, while retaining the same population size of 150. The table only shows the splitting of the population into two parts and their respective percentages, while the actual creation of the individual consists of distributing 90% of zero genes randomly along d dimensions, and the 10% rest of the genes take values from uniform distribution over $[0, 1]$ range. The custom creation can also be applied to projection, where elements in projection matrix are set to zero in a similar manner. The only difference is the sampling of values for the non-zero genes, which in the projection case is done by taking random samples from $[-1, 1]$ range.

Custom/Uniform	Housing	Tecator	Anthrokids	Santa Fe	ESTSP
0%/100%	0.0554	0.0134	0.0083	0.0101	0.0123
10%/90%	0.0551	0.0133	0.0079	0.0099	**0.0122**
20%/80%	0.0553	0.0127	0.0077	0.0096	0.0123
30%/70%	0.0554	0.0124	0.0076	0.0099	0.0123
40%/60%	0.0550	0.0118	0.0075	0.0091	0.0123
50%/50%	0.0552	0.0109	0.0073	0.0091	0.0123
60%/40%	0.05491	0.0110	0.0073	0.0091	**0.0122**
70%/30%	**0.0548**	0.0105	0.0073	0.0087	0.0124
80%/20%	0.0553	0.0098	0.0072	0.0085	0.0124
90%/10%	0.0549	0.0092	**0.0071**	**0.0080**	0.0124
100%/0%	0.0605	**0.0087**	0.0073	**0.0080**	0.0126

Table 5.6: Average DT values calculated for several initialization ratios for the Genetic Algorithm.

From Table 5.6 we see the advantage of introducing many zero genes for the scaling problem. The best mean values are marked in bold, and the only data set not benefiting from this zero insertion is ESTSP. However, the results for this data set are quite similar for any custom/uniform initialization and having many zero genes does not degrade performance too much. The best overall improvement is found for Tecator (more than 68% in some cases), which can be attributed to very high dimensional data, out of which only few tens of variables are needed for lower DT values. Comparing the best results from Table 5.6 to those in Table 5.3, with custom initialization GA is able to reach lower DT than in 10000 evalutions without it (50 generations with population size of 150 accounts for 7500 DT calls).

For the rest of the thesis we use this custom initialization for GA, but we slightly modify percentages of zero genes in individuals. The setup is as follows: 80% of the individuals have many zeros, while the remaining 20% are created in a standard way by choosing randomly from a uniform distribution. The custom individuals are further divided into three equally sized parts in which individuals have different number of zero genes. First part consists of individuals with 90% of zero genes placed randomly over d dimensions, while the rest of the genes are randomly initialized. Second part consists of individuals with 80% of zero genes, while the third part has 70% of zero

genes in its individuals. Table 5.7 presents this custom initialization.

Standard	Custom Initialization, 80 % of the population		
20 %	Part 1	Part 2	Part 3
100 % Random	90 % zeros 10 % Random	80 % zeros 20 % Random	70 % zeros 30 % Random

Table 5.7: Summary of the custom initialization of the GA population. Genes to be set to zero are selected randomly.

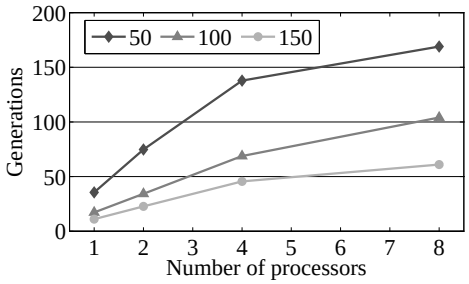
The splitting of the custom part into three segments should be able to make this process suitable for data sets with different number of variables. For small $d \leq 20$, 10% is only 2 variables, which is sometimes not enough to for prediction of the output variable. The custom part is set to a value less than 100% of the population to initialize the rest of the population as diverse as possible for all genes. This is particularly noticeable for Housing data in Table 5.6, where the DT values degrade with this absence of diversity.

5.4 Improvements on Parallel Architectures

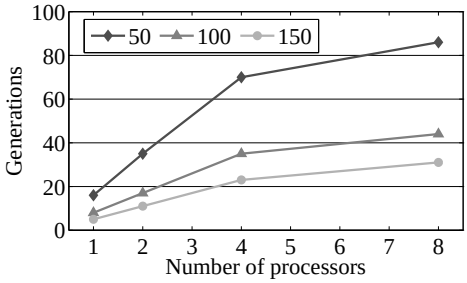
This section shows the benefits that are obtained by adding parallel programming to the serial GA. The serial version was designed as described in Section 4.3.2 and that setup was used for the experiments in the previous section. However, the evaluation of the individuals was performed on a single processor.

For the tests with parallel programming, the GA parameters are the same as described, except for the two things: custom initialization is replaced with the standard random, and the crossover type was one-point crossover instead of the better performing BLX-α. The stopping condition was set to 600 seconds to easily compare serial and parallel implementations. The setup was tested on three data sets (Anthrokids, ESTSP 2007, Tecator) and two problems (selection and discretized scaling). The Tecator data set was normalized variable-wise for this experiment instead of the sample-wise normalization as before. In this section, we are more interested in the the effect of increasing the number of processor to the optimization of the DT, and less on the actual minimization. The performance is presented in Table 5.8 for different number of processors (np) used, including a statistical analysis of the value of DT and the number of generations evaluated.

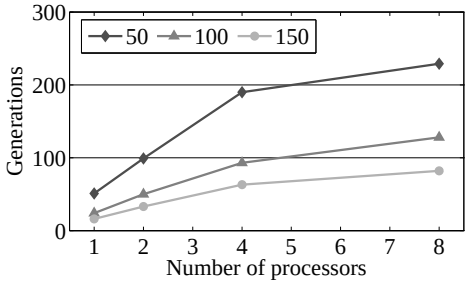
Figures 5.3 show the effect of increasing the number of processor in the number of generations done by the algorithm for a constant number of individuals. As it was expected, if the number of individuals increase, the number of generations is smaller. This effect is compensated with the introduction of more processors that increase almost linearly the number of generations completed. The linearity is not that clear for small population with 50 individuals since the communication overhead starts to be significant. However, large population sizes guarantee good scalability for the algorithm.



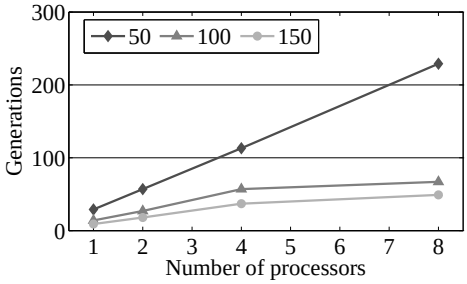
(a) Selection on Anthrokids data



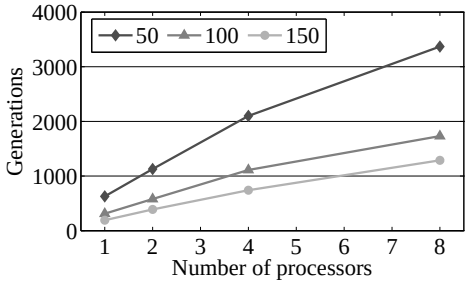
(b) Scaling on Anthrokids data



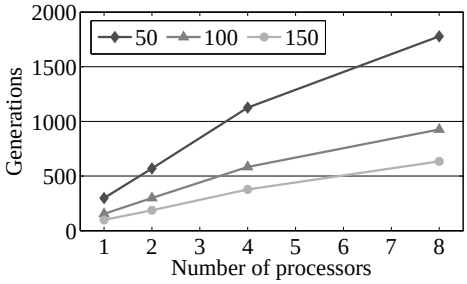
(c) Selection on ESTSP 2007 data



(d) Scaling on ESTSP 2007 data



(e) Selection on Tecator data



(f) Scaling on Tecator data

Figure 5.3: Number of completed generations for different population sizes (50,100,150).

Data set	Population	Measurement	Selection				Scaling			
			np=1	np=2	np=4	np=8	np=1	np=2	np=4	np=8
Anthrokids	50	Mean (DT)	0.01278	0.01269	0.01204	0.01347	0.01527	0.01425	0.01408	0.0142
		Mean (Generations)	35.5	74.8	137.8	169.3	16.7	35.3	70	86
	100	Mean (DT)	0.01351	0.01266	0.01202	0.0111	0.01705	0.01449	0.0127	0.01285
		Mean (Generations)	17.2	35.4	68.8	104	8.5	17.3	35	44.5
	150	Mean (DT)	0.01475	0.01318	0.01148	0.01105	0.01743	0.0151	0.01328	0.01375
		Mean (Generations)	11	22.7	45.6	61	5.7	11.2	23.2	31
Tecator	50	Mean (DT)	0.13158	0.14297	0.13976	0.1365	0.14151	0.147	0.14558	0.1525
		Mean (Generations)	627	1129.4	2099.2	3369.5	298.1	569.5	1126.6	1778.5
	100	Mean (DT)	0.13321	0.13587	0.13914	0.13525	0.14507	0.14926	0.14542	0.1466
		Mean (Generations)	310.8	579.6	1110.4	1731	154.4	299.9	583	926.5
	150	Mean (DT)	0.13146	0.1345	0.13522	0.1323	0.14089	0.15065	0.14456	0.1404
		Mean (Generations)	195	388.1	741.2	1288	98.3	197.8	377	634.5
ESTSP 2007	50	Mean (DT)	0.01422	0.01452	0.01444	0.01403	0.01401	0.01413	0.014	0.0142
		Mean (Generations)	51	99.2	190.8	229	29.1	57.6	113.8	126.7
	100	Mean (DT)	0.01457	0.01419	0.01406	0.01393	0.01445	0.01414	0.01382	0.01393
		Mean (Generations)	24.8	50.5	93	128.7	14	27.9	57.8	67.7
	150	Mean (DT)	0.01464	0.01429	0.01402	0.0141	0.01467	0.01409	0.01382	0.01325
		Mean (Generations)	16.6	33.6	63.2	82.5	9.1	18.7	37.6	49.5

Table 5.8: Performance of serial and parallel implementations on three data sets.

Chapter 6

Application to Time Series Prediction

In this chapter, we closely investigate DT minimization on different problem types (scaling, projection, scaling + projection, fixed variants) in the domain of time series analysis and prediction. For these experiments, only GA is used since it is the best performing algorithm in the scaling problem. The version without discretization is chosen. In the following sections, we are also considering projection problems, where a projection matrix contains real values. Thus, with the use of RCGA, we are avoiding the setup of discretized values for both the scaling and projection.

6.1 Delta Test Minimization on Different Problems

First, we investigate how DT behaves on different problem types mentioned in Chapter 3. These types are: selection, scaling, projection, scaling + projection, fixed scaling and fixed scaling + projection. Four time series are used as a benchmark for the performance of DT optimization: Santa Fe, Mackey Glass 30 [72], ESTSP 2008a [73] and Darwin SPL [74, 75], with Table 6.1 giving the number of values in each series and the chosen regressor sizes. Out of these four series, only ESTSP 2008a has been preprocessed in order to remove the trend. The final series was obtained by taking the first order difference.

As was the case with previous experiments, all data sets are normalized to zero mean and unit variance, including the output variable. For this

Name	Values	Regressor size
Santa Fe	1000	12
Mackey Glass 30	1500	20
ESTSP 2008a	354	20
Darwin SPL	1400	15

Table 6.1: Time series used for different variable selection problems.

experiment, no splitting of data sets was performed (i.e. training and test sets), because we are interested in the minimization of the DT without model building. Optimizations are done only for the first horizon of prediction (one-step ahead), while the stopping condition for the GA is set to 200 generations allowing the algorithm to find solutions with high fitness. The actual model building for long-term prediction is done in Section 6.2.

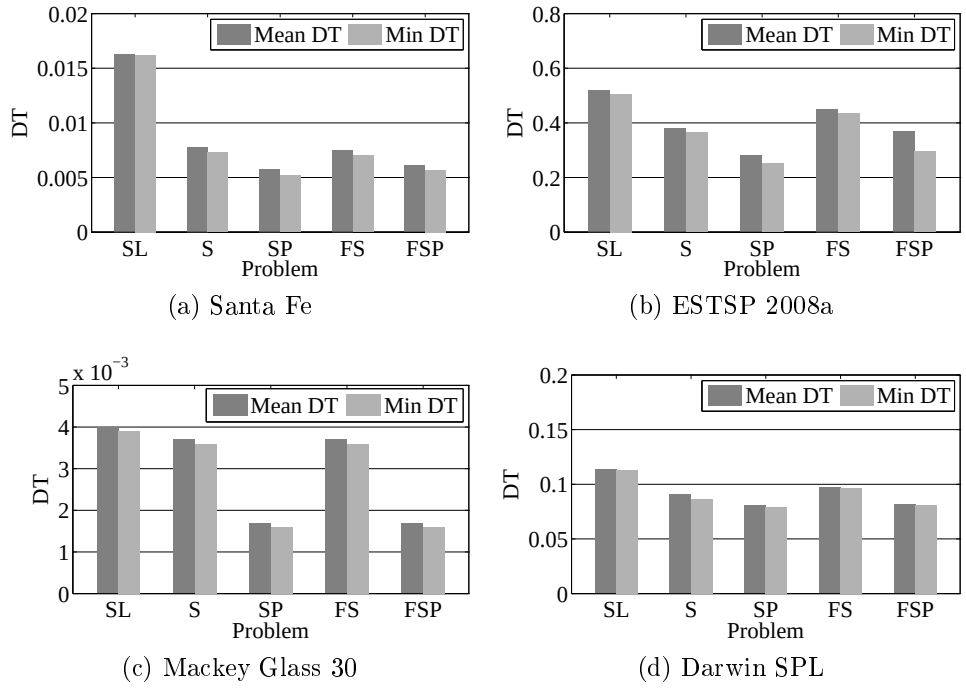
For easier notation in this section, we use SL as an abbreviation for minimization of DT in variable selection, S for variable scaling, SP for scaling + projection (SP-k when projection dimension is k), FS for fixed scaling (FS-d_f when d_f variables are fixed), and FSP for fixed scaling + projection (FSP-d_f-k for the problem with d_f fixed variables and k projection dimensions). For all the setups of parameters, the optimization process is done 10 times for each selection problem.

6.1.1 DT Performance

The average DT values computed for each problem and for each data set are plotted in Figure 6.1. For these experiments, we set $k = 1$ and $d_f = \lceil d/2 \rceil$, resulting in SP-1, FS-$\lceil d/2 \rceil$ and FSP-$\lceil d/2 \rceil$-1.

Looking at the results in Figure 6.1, we are able to reach lower DT values for all problems compared to pure variable selection. The best result is obtained for SP-1 in all cases. Inclusion of projection leverages the possibility of using a newly created variable to gain an advantage over scaling alone. We see that with one additional variable included to the data, we are able to lower DT by a large margin in some data sets (almost 100% for Mackey Glass 30).

In general, the fixed variations provide slightly worse DT values than their standard counterparts (e.g. Santa Fe, Mackey Glass 30), meaning that learning models will be able to give similar performance on the halved data set. Since only half of the variables are used, the training times of models will greatly benefit from this reduction. The fixed version also gives an insight into the most relevant variables, and in these experiments the $\lceil d/2 \rceil$ most



(a) Santa Fe

(b) ESTSP 2008a

(c) Mackey Glass 30

(d) Darwin SPL

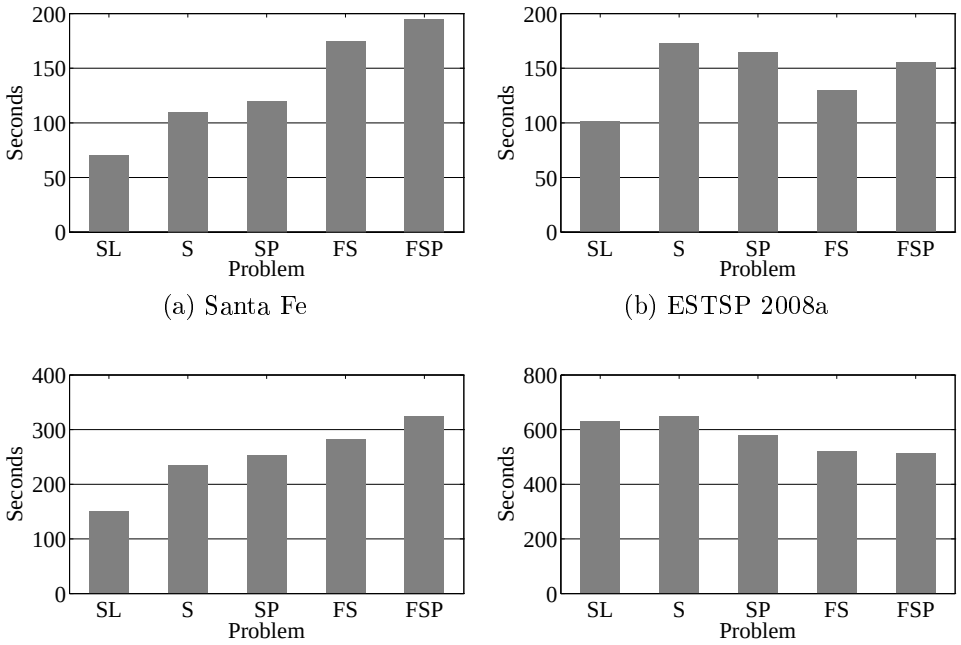
Figure 6.1: DT performance (average and minimum values) for four time series.

important dimensions for prediction. For the ESTSP 2008a data set, values of fixed scaling are not on the same level as those of pure scaling, suggesting that more than $d/2$ variables are required for better prediction.

6.1.2 Computational Time

A computational time comparison of selection problems is shown in Figure 6.2.

It is noticeable that SP-1 sometimes requires less computational time to perform the 200 generations than scaling, and even getting close to or slightly improving (Darwin SLP) times computed for pure selection. This might seem contradictory, as the size of the individuals is twice the size of those used for scaling in GA setup. Although the computational time for GA doubles when moving from scaling to SP-1, the running time of DT optimization is dominated by nearest neighbor search (Section 2.1). The faster calculation time for DTSP-1 could be attributed to the construction of the underlying data structure of approximate nearest neighbors, which uses a hierarchical space decomposition tree called balanced-box decomposition (BBD) tree [30].



(a) Santa Fe (b) ESTSP 2008a

(c) Mackey Glass 30 (d) Darwin SPL

Figure 6.2: Average running times obtained for four time series.

Additional dimensions might lead to favorable splitting of the points/samples into leafs of the tree, eventually improving response time for query searches.

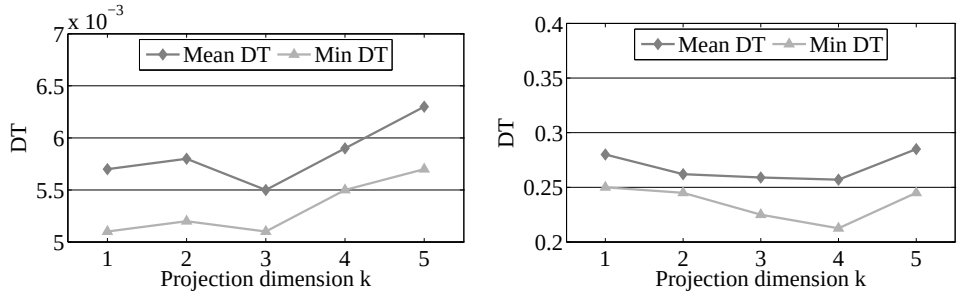
The fixed versions gave good results in terms of DT values for some data sets, but their computational times are generally higher than their non-fixed versions. When using multi-objective optimization there is an additional cost inherent in NSGA-II method, which sorts solutions based on the dominance notion. The additional $O(mp^2)$ complexity of NSGA-II for $p = 150$ slightly increases the running time for most data sets. The exceptions are ESTSP 2008a and Darwin SPL series, where the computational time for fixed methods is lower than for their non-fixed counterparts.

Casting the scaling problem with a fixed number of variables into multi-objective setting increases the run time on the tested data sets. In order to achieve lower running times, the number of individuals in the population has to be reduced, which influences the exploration capabilities of the GA in a negative way. The additional computational time of NSGA-II prevents it from being used in this type of problem. Therefore, faster and simpler techniques should be employed to lower the running times of the fixed scaling (plus projection) problem. One such possibility is island GA with migration policies which do not have such high complexity.

In the next section, the computational time for scaling + projection is further analyzed for several values of k.

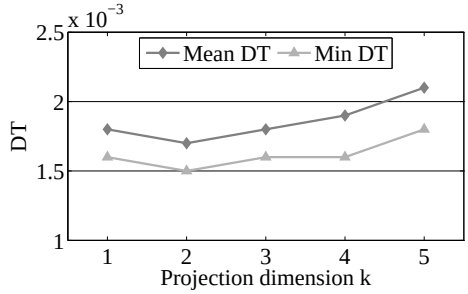
6.1.3 Projection to many Dimensions

From the previous results one can extract the conclusion that SP-1 has clearly outperformed the rest of the methods while still keeping reasonably low computational times in many scenarios. Following experiment compares DT values and computational time when projection is done to $k = \{1, 2, 3, 4, 5\}$ dimensions. Figure 6.3 illustrates the DT results obtained and Figure 6.4 represents the computational time evolution.

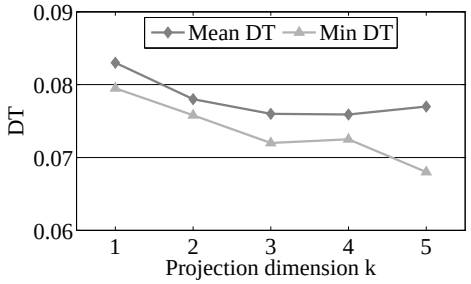


(a) Santa Fe

(b) ESTSP 2008a



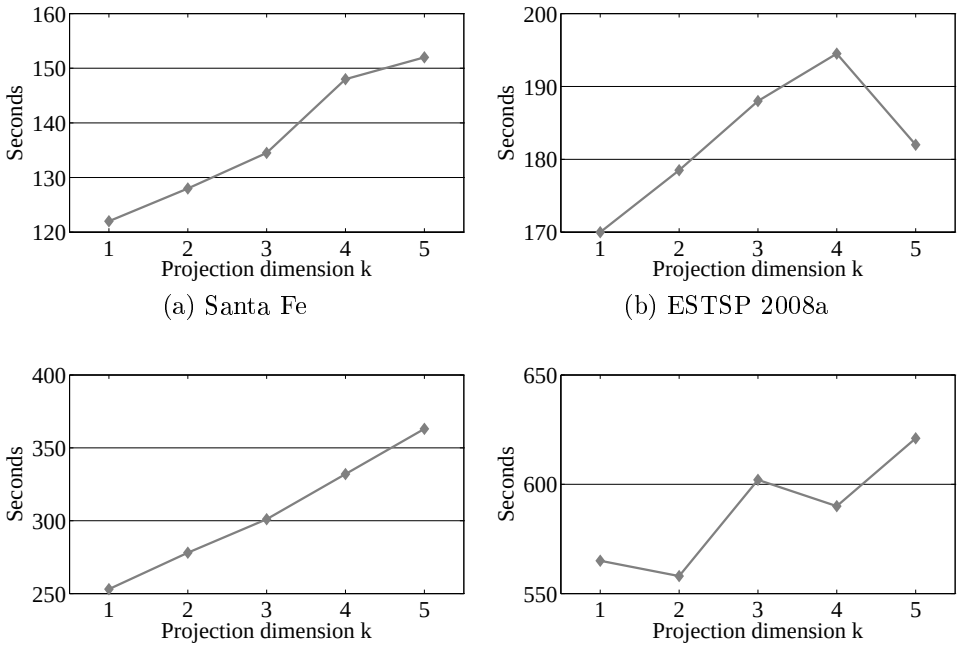
(c) Mackey Glass 30



(d) Darwin SPL

Figure 6.3: SP-k results using projection to $k = \{1, 2, 3, 4, 5\}$ dimensions for four time series.

By looking at the results it is easy to observe that, for all data sets, the value of DT has an optimum value after which it starts to rise again when adding more projections. With Algorithm 3.1 this value is reached if we start from $k = 1$. The downside of this approach is huge computational cost, as for each projection dimension one has to run GA several times in order to get reliable DT estimate. Algorithm 3.1 will miss the correct value of k for Santa Fe and Darwin SPL series. However, the improvement in these cases is negligible



(a) Santa Fe (b) ESTSP 2008a

(c) Mackey Glass 30 (d) Darwin SPL

Figure 6.4: Average running times obtained for SP-$\{1, 2, 3, 4, 5\}$ for four time series.

if we take into account the additional number of parameters needed for the projection matrix (in both cases there are extra $2d$ parameters).

The adjustable number of projections aids in obtaining lower DT values. The progression of the DT curves as a function of k shows a minimum where the optimum DT has been registered. As we only tested projections to $k \leq 5$ dimensions, better values may be found for higher values of k, at the expense of computational time. The increase in computational time as a function of k is sometimes irregular (Darwin SPL and ESTSP 2008a), while for the other two series it shows increasing tendency as expected due to the increased number of parameters.

Finally, the lowest DT values achieved for each data set and the corresponding problem are listed in Table 6.2.

Data set	Problem	Minimum DT
Santa Fe	SP-1 / SP-3	0.0051 (11.22)
ESTSP 2008a	SP-4	0.2122 (2.13)
Darwin SLP	SP-5	0.0686 (0.466)
Mackey Glass 30	SP-2	0.0015 (1.2E-4)

Table 6.2: Minimum DT values obtained for tested time series (denormalized values in brackets).

6.2 Long-Term Prediction using OP-ELM

Finally, we integrate variable selection aspects with the model building step into one global methodology, depicted in Figure 6.5. The high-dimensional data is first projected based on input selection carried out with Delta Test and Genetic Algorithm. After this projection, the OP-ELM method is used for the actual prediction of future values. The term projection encompasses all selection problems as explained in Section 3.3, and in the experiments we test the following problems: scaling, projection and scaling + projection.

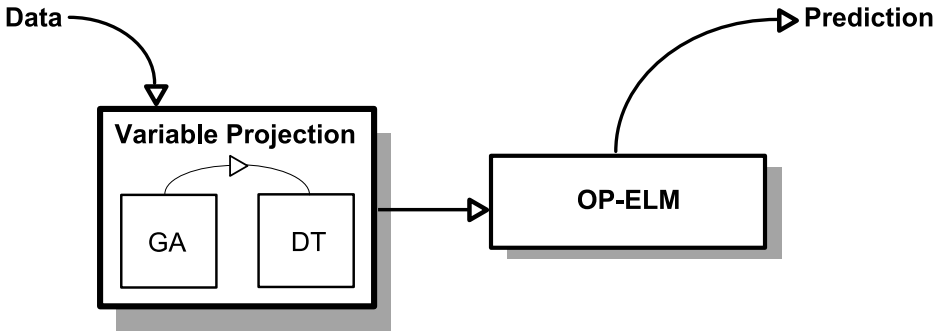


Figure 6.5: Global methodology for long-term times series prediction.

The predictions are done by inspiring new methodology – Optimally-Pruned Extreme Learning Machine [64] (OP-ELM). OP-ELM has its roots from the ELM [65] principle of fast training of a Single Layer Feed-forward Neural Network. The OP-ELM improves the novel ELM concept by pruning out the unnecessary neurons, thus making the estimation more reliable and stable. The OP-ELM does not require any extra parameters compared to the ELM.

The proposed methodology is tested on two time series: ESTSP 2007 and ESTSP 2008b competition data. Before applying the first step (variable projection) of the methodology, both time series are preprocessed in order

to remove trend and/or seasonality. These preprocessing steps are explained for both series in the following sections.

6.2.1 ESTSP 2007 Competition Data

This dataset is from a prediction competition organized in the European Symposium of Time Series Prediction conference (ESTSP) in 2007. The dataset has 875 samples and it is shown in Figure 6.6.

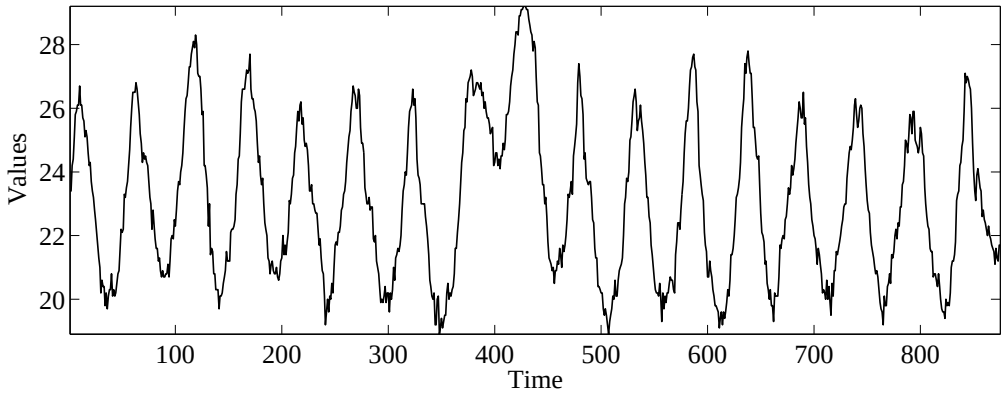


Figure 6.6: ESTSP 2007 competition data.

There is a clear seasonality present throughout the data, except around time point 400. Since the data seems to be corrupted or otherwise completely different from the rest of the data, some portion of the data is removed prior to any other preprocessing. In order to keep the phase correct, two full sequences of 52 values were removed.

After removing some data points, the modified series was separated into learning and test sets. First two thirds of the data was used for the learning, while the remaining third served as a test set. The regressor size of 60[1] was chosen to capture the seasonality plus some extra values. The learning and test sets contain 406 and 148 samples, respectively. The goal is to predict next 50 values of the series.

The next step after splitting involved fitting a sawtooth wave into the learning data, then removing this wave from both data sets, in order to get rid of the seasonality. In other words, the preprocessing for the test set is done only based on the information available in the learning set.

[1]This series had a different regressor size (55) in the previous experiments.

After these steps, we have the preprocessed learning data shown in Figure 6.7.

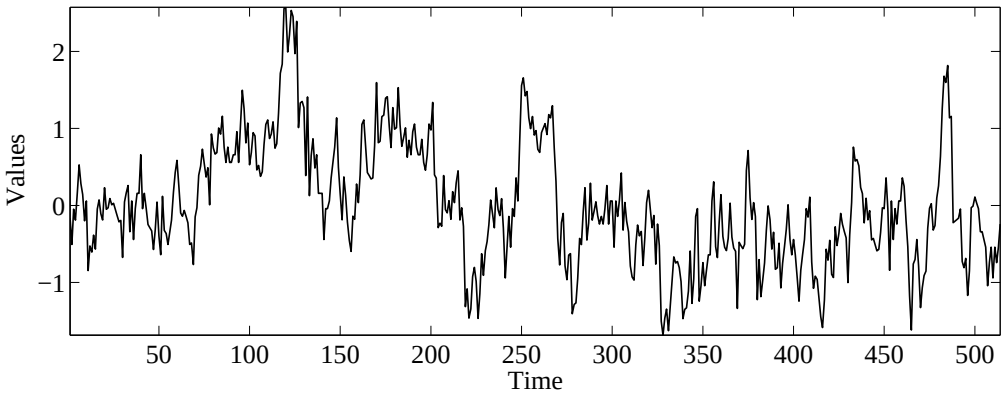


Figure 6.7: ESTSP 2007 learning data after preprocessing.

6.2.2 ESTSP 2008b Competition Data

Like the previous dataset, this one is also from a prediction competition, except this one is taken from the ESTSP conference organized in 2008. The series is a second series out of three given in the competition, and consists of 1300 values shown in Figure 6.8. Before any preprocessing steps, the data set was divided into learning and test sets containing the first two thirds and the last third, respectively. After initial tests with OP-ELM with different regressor sizes, the chosen size was set to 50. Thus, the number of samples is 717 and 285 for the learning and test set respectively. The goal for this competition data is prediction of the next 100 values.

For this data set, two preprocessing steps were used. First one takes care of the clear upward jump around time point 600 and the second one deals with the seasonality of the series. Both steps are done using only the information available in the learning set, even when preprocessing the test set.

Step function fitting found the exact place of the jump at time point 588. The large scale seasonality was removed using a double square wave. Finally, the standard deviation of the data was removed according to the fitted double square wave, in the high parts and in the low parts separately. The result of preprocessing gives the data shown in Figure 6.9.

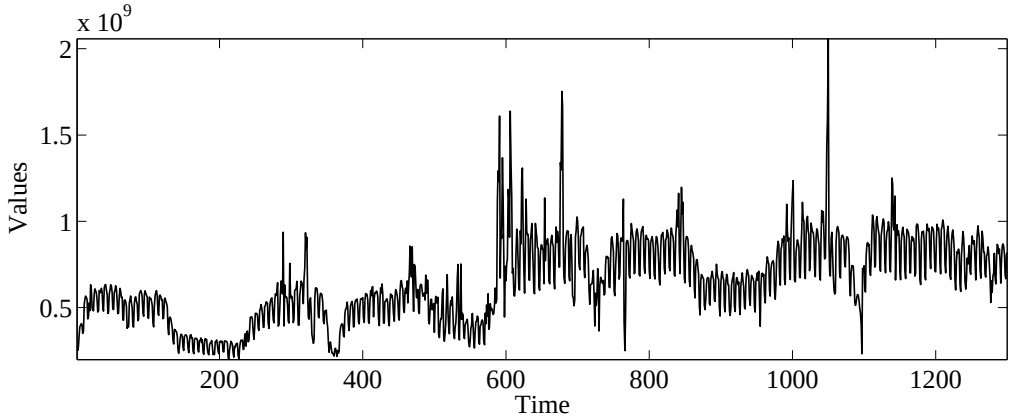


Figure 6.8: ESTSP 2008b competition data.

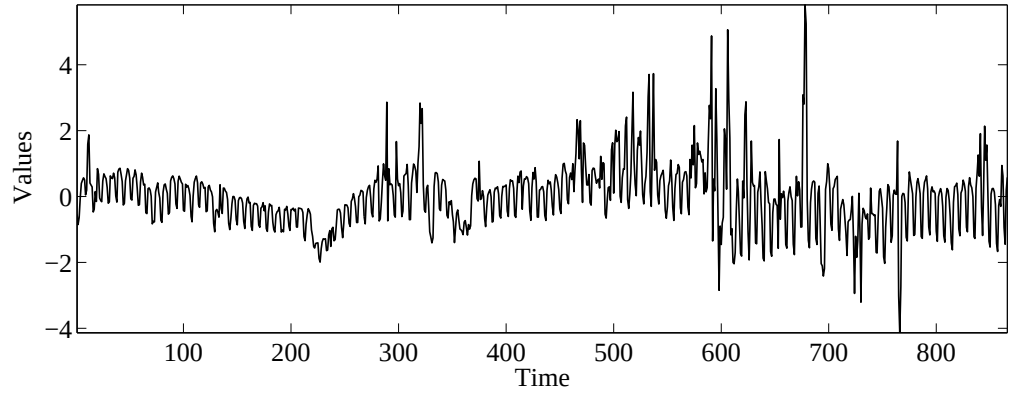


Figure 6.9: ESTSP 2008b learning data after preprocessing.

6.2.3 Prediction Performance

For the long-term prediction, we use the Direct Strategy, which has proven to be accurate and easily implementable choice [35]. This means that 1) a projection matrix is optimized and 2) a model is build on top of projected data for each prediction horizon. Once the data is projected, the OP-ELM is trained, and later used to predict the future samples. Table 6.3 summarizes the performance of OP-ELM) for both series on the our test set and the actual test. Our test consists of samples obtained by splitting the series (denoted T_1 in the table) and the actual test set are the values from the competition (denoted T_2). The performance on the data sets with all variables included is also given for comparison (denoted "original" in the table).

In both data sets, the predictions on projected data are always the highest,

Data set	Problem	MSE(T_1)	MSE(T_2)
ESTSP 2007	original	0.6376	**0.8033**
	scaling	**0.6231**	0.8525
	projection	0.7824	1.1829
	s+p	0.6413	0.8419
ESTSP 2008b *	original	2.5996	1.8181
	scaling	**2.5963**	**1.7381**
	projection	2.8181	2.1515
	s+p	2.6108	1.8114

* error in scale of $\times 10^{16}$

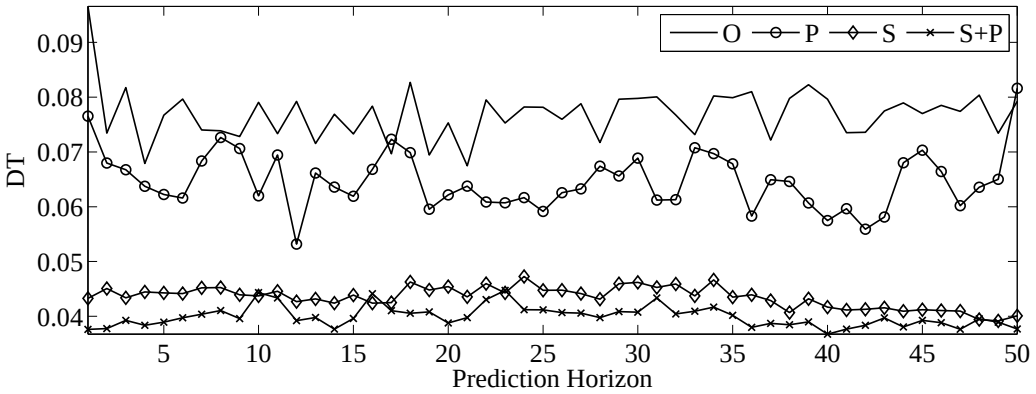
Table 6.3: Summary of the average test errors over all prediction horizons for all methods.

suggesting that linear projection does not suit OP-ELM model. On the other hand, projection is a difficult task with large number of parameters, and the final projection dimension may be much lower than desired for time series prediction. The new approach with scaling + projection produces mixed results, with better results in ESTSP 2007 series. Finally, the ranking of approaches with test set T_1 roughly corresponds to the ranking on the test set T_2. For ESTSP 2008b scaling is the best on both test sets, with original and scaling + projection switching places. In general, it is surprising that no input selection approach has such good performance, considering the regressor sizes of 60 (ESTSP 2007) and 50 (ESTSP 2008b).

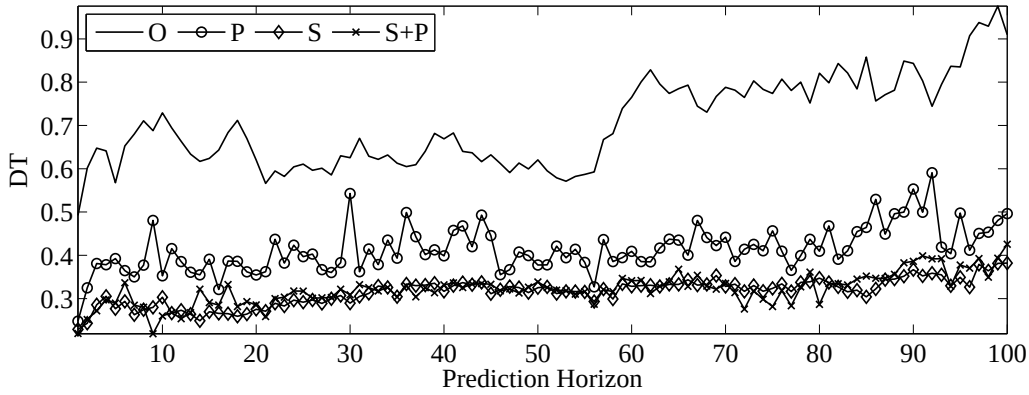
Figure 6.10 shows the values of the Delta Test for all prediction horizons for three types of problems, plus the original data is included for comparison purposes.

As we can see, the scaling+projection obtains the lowest DT values, while and the original data sets using all unprocessed inputs have the highest values. Comparing this ranking with the results presented in Table 6.3, the OP-ELM achieves roughly the same performance with all selection aspects, even though the scaling+projection obtains the lowest DT. The improvement is roughly 40% and 50% for ESTSP 2007 and ESTSP 2008b series respectively. The correspondence between the DT values and the actual performance of the prediction methodologies warrants further study.

The final prediction for both series are shown in Figures 6.11 and 6.12.



(a) ESTSP 2007 data.



(b) ESTSP 2008b data.

Figure 6.10: Delta Test values for original (O), projected (P), scaled (S) and scaled+projected (S+P) data for all prediction horizons.

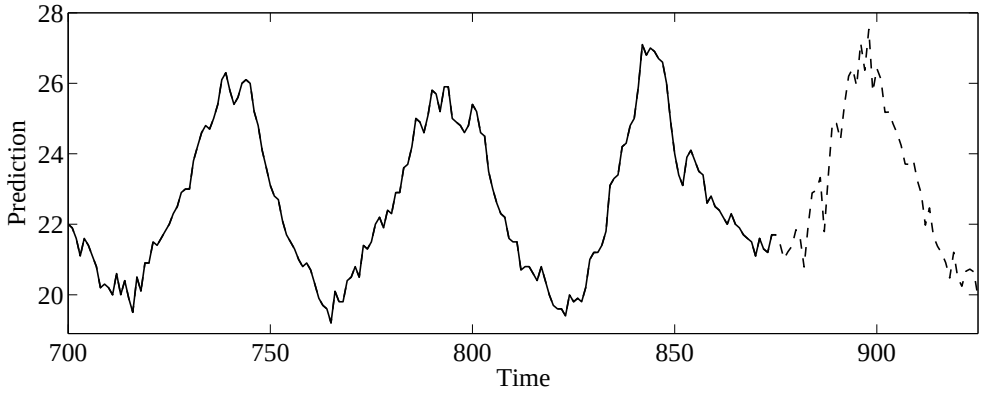


Figure 6.11: ESTSP 2007 competition data. Prediction for 50 steps ahead. Solid line represents the real data and dashed one the prediction.

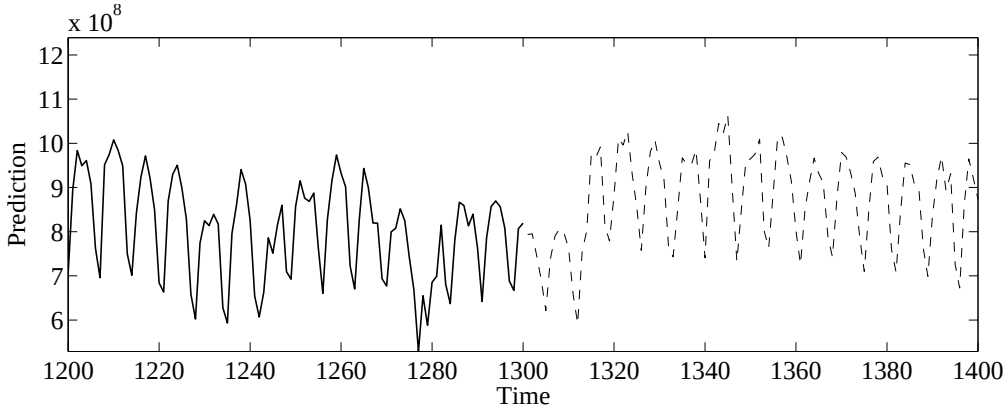


Figure 6.12: ESTSP 2008b competition data. Prediction for 100 steps ahead. Solid line represents the real data and dashed one the prediction.

Chapter 7

Conclusions

In this thesis, we present several aspects of variable selection using Delta Test as relevance criterion. New aspect or problem types of, i.e. fixed scaling and scaling + projection, are introduced and their influence on DT optimization compared to the standard approaches (selection, scaling and projection). Overall, the best values are obtained with the combination of scaling and projection, while the optimization cost is slightly increased, both in the number of parameters of the projection matrix and nearest neighbor calculation time.

The scaling with fixed number of variables provides efficient way of finding the most useful small subset of variables for the actual regression task. Unfortunately, casting this type of problem into a multi-objective setting increases the run time on the tested data sets. The algorithm, NSGA-II, with non-dominated sorting algorithm prevents it from being used in this type of problem. Therefore, faster and simpler techniques should be employed to lower the running times of the fixed scaling (plus projection) problem. One such possibility is island GA with migration policies that do not have such high complexity.

Three search algorithms (Forward-Backward Search, Tabu Search, Genetic Algorithm) are tested in two different problem domains: selection and discretized scaling. Tabu Search has proven to be a better choice for higher-dimensional data in variable selection, while retaining the simplicity in the form of Simple Tabu Search, which uses only short-term recency based memory. This algorithm is easily extended from Forward-Backward Search with additional checks for tabu status. Genetic Algorithm does not provide satisfactory results in selection, but its mechanism is more suitable for complex problems (projection), and its exploration capabilities have proved to overcome both tested neighborhood techniques (FBS, TS). The setup of GA is

extended to include the full interval $[0, 1]$ of scaling weights, which makes the problem fully continuous. For future work, other population based algorithms that are designed for continuous solution spaces, such as Differential Evolution and Particle Swarm Optimization, remain to be tested and compared to the performance of GA. Replacing serial GA with a parallel implementation allows the algorithm to explore more solutions in the same amount of time, enabling faster results for all selection problems.

Finally, variable selection using DT and GA has been included into global methodology for long-term time series prediction. For prediction purposes, we used OP-ELM, a fast and accurate methodology suitable for the task of long-term predictions using Direct strategy. The methodology is tested on two competition time series using different approaches to variable selection: no input selection, scaling, projection, and the combination of both. The results are not as expected, with the first approach (no input selection) being on the same level as scaling and scaling + projection, even though the DT values are substantially reduced in all cases. Although projection approach did not provide satisfactory results, it still provides the user with projected lower-dimensional data, which can be crucial for methods that cannot effectively handle data sets with large number of dimensions. However, the additional dimensions added to the scaling problem can improve the predictions with small extra computational cost.

For further work, this relationship between the DT values and the actual model performance will be studied. Also, the obtained projection performance will be evaluated in even more high-dimensional cases, to ensure the validity of the global search ability of the GA and to study the limits of the search.

Bibliography

[1] D. Sovilj, A. Sorjamaa, and Y. Miche, "Tabu search with delta test for time series prediction using OP-KNN", in *ESTSP, European Symposium on Time Series Prediction* (A. Lendasse, ed.), pp. 187–196, Multiprint Oy / Otamedia , Espoo, Finland, September 17-19 2008.

[2] A. Guillén, D. Sovilj, F. Mateo, I. Rojas, and A. Lendasse, "New methodologies based on delta test for variable selection in regression problems", in *Workshop on Parallel Architectures and Bioinspired Algorithms*, (Toronto, Canada), October 25-29 2008.

[3] A. Guillén, D. Sovilj, F. Mateo, I. Rojas, and A. Lendasse, "Minimizing the delta test for variable selection in regression problems", *International Journal on High Performance Systems Architecture*, vol. 1, no. 4, pp. 269–281, 2008.

[4] F. Mateo, D. Sovilj, R. Gadea, and A. Lendasse, "RCGA-S/RCGA-SP methods to minimize the delta test for regression tasks", in *10th International Work-Conference on Artificial Neural Networks, IWANN 2009, Salamanca, Spain, June 10-12, 2009. Proceedings, Part I* (J. C. at el., ed.), vol. 5517 of *Lecture Notes in Computer Science*, pp. 359–366, Springer, 2009.

[5] F. Mateo, D. Sovilj, and R. Gadea, "Approximate k-NN delta test minimization method using genetic algorithms: Application to time series," *Neurocomputing*, August 2009. Submitted.

[6] D. Sovilj, A. Sorjmaa, Q. Yu, Y. Miche, and E. Séverin, "OP-ELM and OP-KNN in long-term prediction of time series using projected input data", *Neurocomputing*, August 2009. Submitted.

[7] M. Verleysen and D. François, *The curse of dimensionality in data mining and time series prediction*, vol. 3512 of *Lecture Notes in Computer Science*, pp. 758–770. Heidelberg: Springer, 2005.

[8] J. A. Lee and M. Verleysen, *Nonlinear Dimensionality Reduction*. Springer Publishing Company, Incorporated, 2007.

[9] E. Eirola, E. Liitiäinen, A. Lendasse, F. Corona, and M. Verleysen, "Using the delta test for variable selection", in *European Symposium on Artificial Neural Networks 2008, Bruges (Belgium)*, pp. 25–30, April 2008.

[10] P. Pudil, J. Novovičová, and J. Kittler, "Floating search methods in feature selection", *Pattern Recognition Letters*, vol. 15, no. 11, pp. 1119–1125, 1994.

[11] P. Somol, P. Pudil, J. Novovičová, and P. Paclík, "Adaptive floating search methods in feature selection", *Pattern Recogn. Lett.*, vol. 20, no. 11-13, pp. 1157–1163, 1999.

[12] E. Alpaydin, *Introduction to Machine Learning*. MIT Press, 2004.

[13] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[14] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution A Practical Approach to Global Optimization*. Natural Computing Series, Berlin, Germany: Springer-Verlag, 2005.

[15] J. Kennedy and R. Eberhart, "Particle swarm optimization", in *In Proceedings of the IEEE International Joint Conference on Neural Networks*, pp. 1942Ű–1948, 1995.

[16] M. Dorigo and G. D. Caro, "Ant colony optimization: A new meta-heuristic", in *In Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 2, p. 1477, July 1999.

[17] F. Glover and F. Laguna, *Tabu Search*. Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[18] T. A. Feo and M. G. C. Resende, "A probabilistic heuristic for a computationally difficult set covering problem", vol. 8, pp. 67–71, 1989.

[19] T. A. Feo and M. G. C. Resende, "Greedy randomized adaptive search procedures", *Journal of Global Optimization*, vol. 6, pp. 109–133, 1995.

[20] G. Box, G. M. Jenkins, and G. Reinsel, *Time Series Analysis: Forecasting & Control*. Prentice Hall, 3rd ed., February 1994.

[21] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*. Springer, March 2002.

[22] M. Casdagli, "Nonlinear prediction of chaotic time series", *Physica D*, vol. 35, pp. 335–356, 1989.

[23] A. Lendasse, E. de Bodt, V. Wertz, and M. Verleysen, "Nonlinear financial time series forecasting - application to the bel 20 stock market index", *European Journal of Economic and Social Systems*, vol. 14, pp. 81–92, February 2001.

[24] H. Pi and C. Peterson, "Finding the embedding dimension and variable dependencies in time series", *Neural Computation*, vol. 6, no. 3, pp. 509–520, 1994.

[25] E. Liitiäinen, F. Corona, and A. Lendasse, "Nearest neighbor distributions and noise variance estimation", in *ESANN 2007, European Symposium on Artificial Neural Networks, Bruges (Belgium)*, pp. 67–72, April 25-27 2007.

[26] S. Berchtold, D. A. Keim, and H.-P. Kriegel, "The x-tree: An index structure for high-dimensional data", in *Proceedings of 22nd VLDB Conference*, pp. 28–39, 1996.

[27] P. B. Callahan and S. R. Kosaraju, "A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields," *Journal of the ACM*, vol. 42, no. 1, pp. 67–90, 1995.

[28] J. L. Bentley, "Multidimensional binary search trees used for associative searching", *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[29] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time", *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 209–226, 1977.

[30] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions", *Journal of the ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[31] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Chichester, UK: Wiley, 2001.

[32] K. Miettinen, *Nonlinear Multiobjective Optimization*, vol. 12 of *International Series in Operations Research and Management Science*. Kluwer Academic Publishers, Dordrecht, 1999.

[33] R. E. Steuer, *Multiple Criteria Optimization : Theory, Computation, and Application*. New York, Toronto: Wiley, 1986.

[34] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[35] A. Sorjamaa, J. Hao, N. Reyhani, Y. Ji, and A. Lendasse, "Methodology for long-term prediction of time series", *Neurocomputing*, vol. 70, no. 16-18, pp. 2861–2869, 2007.

[36] F. Glover, "Future paths for integer programming and links to artificial intelligence", *Computers & Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.

[37] F. Glover, "Tabu search part i", *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989.

[38] F. Glover, "Tabu search part ii", *ORSA Journal on Computing*, vol. 2, pp. 4–32, 1990.

[39] M. DellÁmico and M. Trubian, "Applying tabu search to the job-shop scheduling problem", *Annals of Operations Research*, vol. 41, no. 1-4, pp. 231–252, 1993.

[40] A. H. Mantawy, S. A. Soliman, and M. E. El-Hawary, "A new tabu search algorithm for the long-term hydro scheduling problem", pp. 29–34, 2002.

[41] C. Zhang, P. Li, Z. Guan, and Y. Y. Rao, "A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem", *Computers & Operations Research*, vol. 34, pp. 3229–3242, November 2007.

[42] J. Xu, S. Chiu, and F. Glover, "Probabilistic tabu search for telecommunications network design", *Journal of Combinatorial Optimization, Special Issue on Topological Network Design*, vol. 1, pp. 69–94, 1996.

[43] J. Xu, S. Chiu, and F. Glover, "Using tabu search to solve steiner tree-star problem in telecommunications network design", *Telecommunication Systems*, vol. 6, pp. 117–125, 1996.

[44] J. Brandao, "A tabu search algorithm for the open vehicle routing problem," *European Journal of Operational Research*, vol. 157, pp. 552–564, September 2004.

[45] S. Scheuerer, "A tabu search heuristic for the truck and trailer routing problem", *Computers & Operations Research*, vol. 33, no. 4, pp. 894–909, 2006.

[46] F. Glover, "Parametric tabu-search for mixed integer programs", *Computers & Operations Research*, vol. 33, no. 9, pp. 2449–2494, 2006.

[47] A.-R. Hedar and M. Fukushima, "Tabu search directed by direct search methods for nonlinear global optimization", *European Journal of Operational Research*, vol. 170, pp. 329–349, April 2006.

[48] K. S. Al-Sultan and M. A. Al-Fawzan, "A tabu search hooke and jeeves algorithm for unconstrained optimization", *European Journal of Operational Research*, vol. 103, pp. 198–208, November 1997.

[49] S. Kirkpatrick, G. C.D., and M. Vecchi, "Optimization by simulated annealing," pp. 671–680.

[50] A. P. Engelbrecht, *Computational Intelligence: An Introduction.* John Wiley & Sons, Ltd, 2nd ed., 2002.

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

[52] M. D. Vose, *The Simple Genetic Algorithm: Foundations and Theory.* MIT Press, 1999.

[53] M. Mitchell, *An Introduction to Genetic Algorithms.* MIT Press, 1996.

[54] I. Oh, J. Lee, and B. Moon, "Local search-embedded genetic algorithm for feature selection", in *ICPR*, pp. II:148–151, 2002.

[55] I. Oh, J. Lee, and B. Moon, "Hybrid genetic algorithms for feature selection", *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 26, no. 11, pp. 1424–1437, 2004.

[56] W. F. Punch, E. D. Goodman, M. Pei, L. Chia-Shun, P. Hovland, and R. Enbody, "Further research on feature selection and classification using genetic algorithms", in *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA'93)* (S. Forrest, ed.), (San Mateo, California), pp. 557–564, Morgan Kaufmann Publishers, 1993.

[57] M. Raymer, W. Punch, E. Goodman, L. Kuhn, and A. Jain, "Dimensionality reduction using genetic algorithms", *IEEETEC: IEEE Transactions on Evolutionary Computation, A Publication of the IEEE Neural Networks Council*, vol. 4, 2000.

[58] Y. Saeys, I. Inza, and P. Larrañaga, "A review of feature selection techniques in bioinformatics", *Bioinformatics*, vol. 23, no. 19, pp. 2507–2517, 2007.

[59] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm", in *Proceedings of the Second International Conference on Genetic algorithms and their application*, (Hillsdale, NJ, USA), pp. 14–21, L. Erlbaum Associates Inc., 1987.

[60] U. K. Chakraborty, K. Deb, and M. Chakraborty, "Analysis of selection algorithms: A markov chain approach", *Evolutionary Computation*, vol. 4, no. 2, pp. 133–167, 1996.

[61] L. Eshelman and J. Schaffer, "Real-coded genetic algorithms and interval schemata", in *Foundation of Genetic Algorithms 2* (L. Darrell Whitley, ed.), pp. 187–202, Morgan-Kauffman Publishers, Inc., 1993.

[62] J. J. Grefenstette, "Parallel adaptive algorithms for function optimization," Technical Report TCGA CS-81-19, Department of Engineering Mechanics, University of Alabama, Vanderbilt University, 1981.

[63] F. Mateo and A. Lendasse, "A variable selection approach based on the delta test for extreme learning machine models", in *Proceedings of the European Symposium on Time Series Prediction* (M. Verleysen, ed.), pp. 57–66, d-side publ. (Evere, Belgium), September 2008.

[64] Y. Miche, A. Sorjamaa, and A. Lendasse, "OP-ELM: Theory, experiments and a toolbox", in *LNCS - Artificial Neural Networks - ICANN 2008 - Part I* (R. N. Vera Kurková and J. Koutník, eds.), vol. 5163/2008 of *Lecture Notes in Computer Science*, pp. 145–154, Springer Berlin / Heidelberg, September 2008.

[65] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications", *Neurocomputing*, vol. 70, pp. 489–501, December 2006.

[66] Approximate Nearest Neighbour library homepage. http://www.cs.umd.edu/~mount/ANN/

[67] Housing data set.
http://archive.ics.uci.edu/ml/datasets/Housing

[68] Tecator data set.
http://lib.stat.cmu.edu/data sets/tecator

[69] Anthrokids data set.
http://ovrt.nist.gov/projects/anthrokids

[70] Santa Fe Competition data sets.
http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe.html

[71] ESTSP 2007 competition data set.
http://www.cis.hut.fi/projects/tsp/index.php?page=timeseries

[72] Mackey-glass time series.
http://www.bme.ogi.edu/~ericwan/data.html

[73] ESTSP 2008 competition data sets.
http://www.cis.hut.fi/projects/tsp/index.php?page=timeseries

[74] Darwin Sea Level Pressure (period 1882-1998).
http://www.stat.duke.edu/~mw/ts_data_sets.html

[75] Climate prediction center homepage.
http://www.cpc.noaa.gov/