FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Algorithm Selection and Runtime Prediction for the two Dimensional Bin Packing Problem

Analysis and Characterization of Instances

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Bernd-Peter Ivanschitz
Matrikelnummer 0603596

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Priv.-Doz. Dr. Nysret Musliu

Wien, 1. Oktober 2017

Bernd-Peter Ivanschitz Nysret Musliu

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Algorithm Selection and Runtime Prediction for the two Dimensional Bin Packing Problem

Analysis and Characterization of Instances

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computational Intelligence

by

Bernd-Peter Ivanschitz
Registration Number 0603596

to the Faculty of Informatics

at the TU Wien

Advisor: Priv.-Doz. Dr. Nysret Musliu

Vienna, 1st October, 2017

Bernd-Peter Ivanschitz   Nysret Musliu

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Bernd-Peter Ivanschitz
Schanzstraße 13/22, 1140 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Oktober 2017

Bernd-Peter Ivanschitz

Danksagung

Ich bedanke mich bei meinen Eltern und bei meiner Ditzy für ihre Liebe und ihre Geduld!

Kurzfassung

Das Behälterproblem (engl. Bin packing problem -BPP) ist eines der bekanntesten kombinatorischen Optimierungsprobleme. Trotz seiner einfachen Aufgabenstellung zählt es zu den NP-Schweren Probleme der Informatik. Seit ihrer Einführung in den 1960er Jahren wurden verschiedenste exakte und heuristische Algorithmen für diese Probleme vorgestellt. Das Berechnen einer exakten Lösung ist oftmals sehr aufwändig und benötigt unter Umständen eine exponentielle Laufzeit. Jedoch konnte in den letzten Jahren einen Vielzahl von heuristischen Algorithmen mit nahezu optimalen Lösungen für dieses Problem entwickelt werden. Allerdings scheint es, dass die Qualität der Ergebnisse oft von den konkreten Eigenschaften der Probleminstanz abhängen. Die Suche nach einem optimalen Verfahren dass das bestmögliche Ergebnis für die jeweilige Probleminstanz liefert ist eine, in Analogie zu dem No-free-Lunch-Theorem, sehr schwierige wenn nicht sogar unmöglich Aufgabenstellung.

Einen möglichen Lösungsansatz für dieses Problem bietet das Algorithm-Selection Verfahren. Hierbei wird mit Hilfe von optimal gelösten Probleminstanzen ein Vorhersagemodell erstellt und trainiert. Dieses Algorithm-Selection Vorhersagemodell ist danach im Stande, für neue Probleminstanzen ein möglichst optimales Verfahren vorherzusagen. Aus einem Set von Algorithmen wird jenes Verfahren ausgewählt, dass das beste prognostizierte Ergebnis liefern könnte.

Diese Arbeit befasst sich mit der Anwendung des Algorithm Selection Verfahren für das BPP unter Zuhilfenahme von Methoden des maschinellen Lernens. Damit wir das Vorhersagemodell erfolgreich nutzen können, werden mehrere Attribute von den Probleminstanzen erhoben und für die Trainingsphase und den Vorhersageprozess genutzt. Diese Merkmale, auch Features genannt, werden in polynomieller Zeit berechnet um eine gute Performance des Algortihm Selection Verfahren zu gewährleisten. Es werden des Weiteren neun state-of-the-Art Heuristiken verwendet, die mit 500 frei verfügbaren Probleminstanzen trainiert werden. Die Ergebnisse der unterschiedlichen Experimente zeigen, dass keine der Heuristiken im Allgemeinen besser als jede andere ist, wenn die Laufzeit und die Güte der Lösung betrachtet werden. Es wurden einzelne Untergruppen der Probleminstanzen analysiert, um die Attribute der Probleminstanzen besser definieren zu können. Des Weiteren wurde noch eine Laufzeitanalyse einzelner Heuristiken durchgeführt, um zusätzliche Informationen über die Probleminstanzen zu sammeln.

Der zweite Teil dieser Arbeit befasst sich mit den Machine Leraning Algorithm, die für die Erstellung des Vorhersagemodelles trainiert werden. Insgesamt wurden sieben unterschiedliche Klassifikations - sowie sechs Regeressionsalgorithmen trainiert. Mittels der Ergebnisse des Experiments konnten erfolgreiche wichtige Attribute der Probleminstanze identifiziert und evaluiert werden. Darüber hinaus untersuchen wir, wie ein verkleinertes Algorithmen-Set sich auf die Qualität der Vorhersage auswirkt.

Neben dem Algorithm-Selection Vorhersagemodell wurde ebenfalls eine Runtime-Prediction Verfahren implementiert. Diese wurden genutzt um die Laufzeiten der einzelnen Probleminstanzen vorhersagen zu können. Für die Auswertung der Runtime-Prediction wurden zusätzliche Probelminstanzen generiert um ein breiteres Spektrum an Instanzengrößen abdecken zu können.

Im letzten Teil dieser Arbeiten werden die trainierten Modelle mit den einzelnen Heuristiken verglichen. Es wird gezeigt ob die Nutzung der vorgestellten Verfahren für das BPP sinnvoll genutzt werden können, und welche Vorteile sich dadurch ergeben.

Abstract

The bin packing problem (BPP) is one of the best-known combinatorial optimization problems. This problem is proven to be NP-hard despite its simple task setting. Since its first introduction in the 1960s, different exact and heuristic algorithms have been proposed for this problem. A variety of algorithms that obtain nearly optimal solutions have been presented in the last decade. However, the proposed methods have advantages and disadvantages, which depend on the specific instance to which they are applied. Designing an algorithm which finds the best possible solution for every possible instance is hard or, by analogy to the No-free-Lunch-Theorem, even impossible. Therefore, selecting the optimal solver for a specific problem can be used in industrial areas of the BPP to reduce the costs and resources needed for real-life problems.

Many approaches have been proposed for the NP-hard problems to achieve better results on all available instances. One of them is the algorithm selection approach. The method predicts for each instance the algorithm which achieves the best performance. The procedure uses a set of intrinsic features computed from the problem instances, and a set of algorithms to predict the best algorithm for the particular instance.

This thesis investigates the algorithm selection approach and the runtime prediction approach for the BPP. We considered two variations of the BPP, in the first case the items are oriented (O) and guillotine cuts are required (G), and in the second case the items can be rotated by 90 degrees (R) and guillotine cuts are also required (G). To use this method, we first introduce a set of features for the problem instances, which can be computed in polynomial time. Then we evaluate the performance of nine state of the art heuristics for the BPP. In order to evaluate these algorithms, we use 500 problem instances from two publicly available instance sets. We analyse the behavior of the algorithms on classes of problem instances with different attributes. Furthermore, we investigate which attributes define hard instances and whether an algorithm exist, which clearly outperforms the other algorithms. We analyse the behaviour of the algorithms on classes of problem instances with different attributes. Furthermore, we investigate which characteristics define hard instances and whether algorithms exist, which clearly preforms better than others.

In the next step, we use the knowledge about the most appropriate algorithm and the newly developed features for each instance to train seven classification algorithms.

The obtained models use supervised learning methods to predict the most appropriate algorithm for new and unseen instances. For each classifier, we test multiple parameter settings. We analyse the impact of preprocessing and data preparation techniques on the quality of the performance for the prediction models. Furthermore, we use a cross-validation method as standard machine learning strategies to compare the prediction models. Our experiments show that the selection of an algorithm based on machine learning is able to beat all the single solver heuristics in terms of time - and solution quality.

For the run time prediction approach we use additionally a newly generated dataset with a higher variety of instance sizes. The experimental results with different regression models show that the designed features can be used for the runtime prediction of the BPP-algorithms.

Contents

CHAPTER 1

Introduction

Computer scientists have been studying NP-complete and NP-hard problems for decades. One of them is known as the bin packaging problem (BPP), which is a variety of Karp's [Kar72] well-studied NP-complete problems.

The problem has its origin in the 1960s and variants of the BPP find their use in many practical applications, such as cutting of materials (wood, glass, stone, cloth industries), as well as newspaper paging. Many different varieties of the BPP have been described in literature [LMV99, LMV02] with respective solving methods since its first introduction over forty-two years ago. The classical BPP is defined as follows: We are given a set of items $I = \{1, \ldots, n\}$, where each item $i \in I$ has size $s_i \in (0, 1]$ and a set $B = (1, \ldots, n)$ of Bins with capacity one. The goal is to find a valid assignment $a : I \rightarrow B$ such that the number of bins used is minimal. This does not sound hard at all, but it is indeed hard to find an optimal arrangement of items in the bins to achieve an optimal solution. Moreover, with the fact that the problem is NP-hard, it is very unlikely that an exact strategy exists, which needs less than exponential time to find a minimal packing of the items, unless $P = NP$. As a logical consequence, the research focused on (meta)heuristics to find good solutions for the BPP. These methods are often able to produce good solutions faster but do not guarantee optimality.

Popular developments were presented for the BPP in the last decade, such as justification improvement heuristic by [Fle13], ant colony optimization heuristics by [LD04] as well as genetic algorithms introduced by [FD92]. Other metaheuristic techniques [VMOR12, HT01, BHK06] or hyper heuristics [RSMBH02] were also introduced in the past ten years. The current state of the art solvers are based on two-stage or three-stage heuristics with tree structures for the insertion process [CCT15, CF11, Fle13].

The various algorithms show varying performance for particular problems, such as the BPP or other optimization tasks. To find the most suitable algorithms for the given task, we have to decide which method is suited best for the practical use with our problem cases. This defines a new problem since finding the best method is not trivial, and in analogy to the No Free Lunch Theorem[WM$^+$95] , it is highly unlikely that there exists one heuristic that performs on all our instances better than the others. For an optimal solution of a particular instance, it would be rather advantageous to know in advance which method is most appropriate.

The issue of the selection of the most appropriate algorithm for a specific instance is also known as Rices algorithm selection problem [Ric76]. The algorithm selection approach allows us to prevent a worst case scenario and helps to produce good solutions. This is important, especially for practical questions, as it allows us to save some money in the best case. On the other hand, it is also very interesting from a theoretical perspective to gain insight and knowledge under which circumstances an algorithm performs well or badly. That way we can choose the most suitable algorithm and develop new strategies to optimize their usage. In addition, it allows us to gather important information about difficult cases.

In the last decade, several new applications and methodologies have been developed for selecting the best algorithms for a problem. One of these approaches is the use of machine learning techniques for the algorithm selection problem. These methods are able to learn from given examples important patterns and help to predict the best algorithms based on the learned model.

Another interesting research topic is the algorithm runtime prediction. This method is not new, but with new powerful machine learning techniques, the predictions are more accurate and scalable than ever. The issue with NP-hard problems is that some state-of-the-art algorithms can take an enormous amount of time to solve large problems. Most of the problems are solved in a reasonable time, but some need an increased quantity of time and there is only little theoretical understanding why some algorithms need more time than other on a particular problem instance [GSCK00].

In this thesis, we will focus on the algorithm selection problem and the run time prediction problem for the 2-dimensional BPP with machine learning techniques. We will use a set of features computed from the problem instances of the BPP and investigate if the algorithm selection approach and the algorithm run time prediction is practicable for the BPP. Therefore we will use a set of eight state of the art bin packing algorithms and two publicly available instance sets for an empirical investigation on the performance of these algorithms. Furthermore, we will investigate the characteristic attributes for the instances of the BPP and use the gained knowledge to train the prediction models for the algorithm selection and run time prediction.
The final part of this thesis evaluates the performance of the approach based on automated algorithm selection and the single solver algorithms.

1.1 Objectives

The objectives for this thesis are:

- Identification of state-of-the-art algorithms used for BP and evaluation of their performance on a representative set of instances.
- Identification of important features of BPP instances that have an impact on the performance of algorithms.
- Investigation of applications of different machine learning techniques for automated algorithm selection based on features of specific instances.
- Comparison of the performance of an overall solver based on algorithm selection with other algorithms for the BPP.

1.2 Main Results

The main results of this thesis are:

- We identified a set of 23 features that characterise the BPP problem instances.
- We investigate the performance of the algorithm selection method with up to nine state-of-the-art heuristics and 500 instances of publicly available data sets for the BPP. The experimental findings show that some algorithms perform better, with respect to quality and runtime of the solution, on instances with a particular set of features.
- We investigate the performance of different machine learning techniques for automated algorithm selection based on features of specific instances and trained seven different classifiers to predict the best algorithm for new, unseen problem instances. Furthermore, we explored the effects of data preparation and apply a feature selection to identify the most important features.
- We train and test the runtime prediction models for three state-of-the-art bin packing algorithms. For the prediction task, a set of six different machine learning procedures has been used.
- We compare the overall solver based on algorithm selection with single solver heuristics for the BPP. The results show that our approach achieves a better performance than any single algorithm for the BPP.

1.3 Organization

This thesis is organized as follows. The thesis continues in Chapter 2 with the relevant background information on heuristics, complexity, experimental aspects and machine learning techniques. The third chapter explains the algorithm selection method and the relation to the No Free Lunch theorem. Chapter 4 explains the BPP in detail and gives an overview of popular heuristics to solve the problem. In addition, we describe the features used to characterize the problem instances. In Chapter 5 we describe the experimental setup and give a detailed overview of the used heuristics, instances and the machine learning techniques. Chapter 6 and 7 show the results of the experiments and compares them to heuristics in the BPP. Furthermore, we show the performance of the overall solver based on the algorithm selection method and discuss the impact of different preprocessing techniques and parameter settings. The last chapter concludes the results and gives an outlook on further work.

CHAPTER 2

Background

2.1 NP-Problems

Computer science encompasses many different challenges and fields of expertise. One core aspect is the efficiency of algorithms for different areas, like sorting, solving complex equations or finding the shortest path. All the aforementioned topics define sets of constraints, which have to be solved in order to retrieve valid answers.

In the context of computer science, problems are usually divided into so-called complexity classes. The problems are grouped according to their complexity as a function of the input parameter n. We use the well-known big O notation, or Landau notation, introduced by Paul Bachmann and Edmund Landau to classify the algorithms according to the runtime of an algorithm or the space needed to solve them [Knu76]. Therefore, a problem can be solved in:

- Constant time, then we write $O(1)$
- In linear time on the input variable n, then we write $O(n)$
- In polynomial time, then we write $O(p(n))$, where p is a polynomial
- In exponential time, then we write $O(x^n)$

2.2 The complexity of P and NP

Let P be the complexity class of the set of problems that can be solved in polynomial time using a Deterministic Turing Machine, in general, $O(1)$, $O(n)$, or $O(p)$. In practice, we say that the class P represents all the problems that are solvable in reasonable computational time.

The theory of NP gives us a framework for showing that it is very doubtful that a polynomial algorithm exists for a specific problem in this class. This means that the class NP, which stands for nondeterministic polynomial, contains all problems that can be solved in polynomial time only by a Nondeterministic Turing Machine. Another interesting fact is that all problems that are in P are also in NP. Furthermore, we are able to verify the correctness of a solution in the class NP in deterministic polynomial time [Coo00].

One of the most important theoretical questions according to the complexity classes in computer science is not yet solved. Are we able to solve all the problems that are computed in nondeterministic polynomial time also by using a Deterministic Turing Machine with a smart algorithm in polynomial time [WW05]? The $P = NP$ question is essential and the answer is still unknown but it is believed that $P \neq NP$.

In addition to the class P and NP, we also distinguish between $NP - Hard$ and $NP - Complete$ problems. We say that a problem is $NP - Hard$ if any NP problem can be converted into it in polynomial time [Coo00]. Informally, if a problem is $NP - Hard$ it means that it is at least as hard as any NP problem [Aar05]. A problem is $NP-Complete$, if and only if it is $NP-Hard$ and if its in NP (verifiable in non-deterministic polynomial time) [Coo00]. Informally, the $NP - Complete$ problems are defined as the hardest problems in NP.

Clearly, not all computer science problems are equally hard to solve. Often we are not only interested in the worst case runtime of an algorithm as mentioned earlier. In detail, we are also eager to define the best and the average runtime of an algorithm for a problem. We define the performance of an algorithm $f(n)$ with respect to n based on the worst $\mathcal{O}$, the best Ω and the average Θ case as follows from[Knu76]:

$$\mathcal{O}(g(n)) = \{f(n) \mid \exists c \in \mathbb{R}, n_0 \in \mathbb{N} : \forall n \geq n_0 : |f(n)| \leq c * |g(n)|\}$$

$$\Omega(g(n)) = \{f(n) \mid \exists c \in \mathbb{R}, n_0 \in \mathbb{N} : \forall n \geq n_0 : |f(n)| \geq c * |g(n)|\}$$

$$\Theta(g(n)) = \{f(n) \mid \exists c, c' \in \mathbb{R}, n_0 \in \mathbb{N} : \forall n \geq n_0 : c * |g(n)| \leq f(n)| \leq c' * |g(n)|\}$$

We define n as the problem specific variable and the other variables as constant factors c. For example, if an algorithm $f(n)$ is in $O(n^2)$ it means that any instance of size n can be solved with the algorithm $f(n)$ in at most $c * n^2$ time.

As already mentioned, the O notation only specifies an asymptotic upper bound for the algorithm. The information gained from this worst-case scenario helps to define the runtime of an algorithm and can be used the define if the problem is in P or NP as explained earlier in this chapter. [Sch12b].

2.3 Heuristics

Exact algorithms have usually one major disadvantage compared to heuristics. Depending on the complexity of a problem and its size, the exact algorithms may need an enormous amount of time and resources to solve the problem especially if the problem is big. Of course, such procedures are also researched and offered but their field of application is limited [Chi05].

For the main problem in this thesis, it has been firmly established [LMM02] that the 2 dim BPP is $NP - hard$. Therefore we can conclude that an optimal solution to the problem is rather unlikely, at least not in reasonable time. In some cases, it is possible that an instance can be solved with an exact method in moderate time, but this is rather uncommon [LMV02].

In recent years, heuristics have become popular and various new methods were introduced to solve $NP - Hard$ problems in reasonable time with good but mostly not optimal solutions.

Heuristics use different techniques to scan certain areas where good configurations are expected instead of searching the whole problem space. They start with a "random" starting point given from the heuristics and explore the area around for valuable configurations. They may encounter local optima which they can or cannot overcome, depending on the chosen technique. After a certain amount of time or iterations, the algorithm stops and returns the most promising solution it encounters. Reruns with different starting points using information of previous runs could lead to better overall solutions.

For this thesis, we only focus on the 2 dim BPP heuristics. These methods are normally distinguished by one-phase or two-phase approaches [BS13]. While the one-phase approaches tries to pack the items immediately in a bin, the two-phase heuristics pack the items into levels which are later packed into the bins efficiently. A level is a horizontal packing of items that do not exceed the bin width.

2.4 Experimental Aspects

One of the most important underlying assumptions of this thesis is that algorithms show different performances on the same datasets. Therefore, to show that this knowledge can be used to solve new, unseen problem instances more effectively is the goal of this thesis. This goal can be achieved by predicting the most suitable algorithm for the problem. It is important to explore the advantages and disadvantages of the algorithm used with some instance-specific features. This information is used in the training process to build the prediction models.

For this purpose, we analyze the performance of the algorithms on the two publicly available datasets. Unfortunately, it was impossible to gather the original implementations of the presented algorithms for a detailed analysis of newly generated dataset, due to license problems and contracts. For this reason, a cross-validation approach is used to ensure a valid analysis of the results and to prevent overfitting of the prediction model to the given datasets.

2.4.1 Runtime Comparison

A similar motivation is used for the algorithm runtime approach as for the algorithm prediction problem. In principle, it is possible to calculate the run times of individual algorithms, for previously unseen problem instances, to a certain extent at least for classic problems like the propositional satisfiability (SAT), the traveling salesperson (TSP) or the mixed integer programming (MIP) problem [HXHLB14]. One underlying assumption for this thesis is, that the runtime of different algorithms is somehow affected by the features of the problem instances. A goal would be to show that the algorithm runtime prediction approach can also be used for the BPP.

The tested three state of the art BPP algorithms with the original implementations. We extended the two publicly available datasets with 100 new problem instances, which should mainly model the run times for problems instances that pack more than 200 items. To evaluate the quality of the predictions we used the root mean square error $RMSE$ as our benchmark value.

2.5 Machine Learning

Over the last two decades, machine learning [GH88] has become one of the most popular technologies in the area of artificial intelligence with a multitude of use cases. The impact of machine learning should not be underestimated due to the fact of its vast range of applications that range from fraud detection, pattern/image recognition, and Email Spam filters are only a few of the practical use cases. The machine learning methods are capable of learning from historical data and use this knowledge to recognize and solve similar cases.

Machine learning techniques allow us to get valuable insight in the problem cases. Furthermore, the methods are not only able to choose a solution given a specific set of rules, they also learn form the old cases and advance their prediction. To predict which algorithm solves a new, unseen instance best, is clearly a learning problem. Despite its simple task setting, it is not easy at all, but with these machine learning techniques, we are able to build a robust system.

We concentrate our efforts in this thesis on the so-called supervised learning methods. These methods use a training data, where the best solutions for each problem instance is given. With this informations, the machine learning algorithms are able to build a prediction model and classify new instances. The following subsections of this thesis describe the used algorithms.

2.5.1 Classification

The task to classify objects into groups or classes is a very interesting field of research for computer scientists and other researchers [Wol96]. For the classification task, some features about the problem instance are needed, but to calculate or even to develop these features that describe a problem representative is a complex task and needs a lot of domain experience. For example, to describe a car we could use features like tires, doors and a windshield since every car has these characteristics. With that information, it is possible to classify new objects into a group of cars and other vehicles. However, if we really use this model also aircrafts would be classified as cars since they show the required characteristics. Clearly, that is a bad classification model since aircrafts should not be classified as cars. Machine learning techniques help us to define good classification methods through their ability to learn. Naturally, a classification task heavily depends on the quality of the features of the problem cases, which are provided for the training phase. If the data is insufficient or irrelevant, then the results of the classification reflect this, which leads to a misclassification of new unseen data. For the classification tasks, we used the supervised learning methods already mentioned in the last section.

The classification algorithms show some similarities to the already discussed heuristics. Not every classification method is suitable for each problem case. The No-Free-Lunch theorem [WM+95] also applies to classification which implies that no classification algorithm is superior to all other algorithms [Wol96]. Different techniques have been developed over the years to solve this problem. A detailed analysis of this methods would go beyond the goal of this thesis but can be found in [AS06] and [Bro93].

This work focus on eight well-known classification algorithms that have been used successfully in similar environments as the BPP. The used algorithms for the algorithm selection tasks are k-nearest neighbor (kNN), decision trees (DT), random forest(RF), naive Bayes (NB), support vector machines (SVM), Generalized Linear Models (GLMNET), and neural networks (NN). For the runtime prediction approached we used additionally the loess regression (LOESS) method since not all of the methods used for the algorithm selection are capable of regression tasks. The selected methods offer a good composition of modern and successful machine learning methods for the algorithm selection problem as well as the algorithm runtime prediction and are briefly described in the following section. For a detailed overview of the chosen methods and an introduction to machine learning, we point to the original publications and [Alp14, WFHP16].

Data Representation

The typical application of a machine learning algorithm requires a training and a test set. The effect of splitting the data into a training and test set prevents overfitting of the machine learning algorithms to the whole dataset. Additionally, we define a new dataset for our evaluation. Typically, the data is presented as a table for each instance. Here we must distinguish between a problem instance as described in Chapter 2.1 and the machine learning instances with the computed features of a problem instance. The former is the concerted problem case, and for this thesis, consists of the problem id, height, length of the bins and the dimensions of the single items with which we want to pack the bins. The machine learning instances consist of a fixed number of features $X = \{x_1, x_2, \ldots, x_n\}$ and some result variables $Y = \{y_1, y_2\}$ for the runtime and the best

solution.
A classification is then to assign the most likely class from Y to the instances on the testing set based on the computed attribute values $X_k = \{x_{1k}, x_{2k}, \ldots, x_{nk}\}$.

As already mentioned we use a cross-validation method to ensure the quality of our predictions [K$^+$95].

k-Nearest Neighbor

The KNN approach is a rather simple classification method and an extension of the classical nearest neighbor method [CH67]. In general, the (k)-nearest neighbor method is an instance-based classifier [AKA91] , which is based on the assumption that instances which have similar feature attributes should be classified in the same class. Therefore, new unseen data is classified using a distance/similarity function on the stored data from correctly classified cases. This means for the nearest neighbor method, a new instance is compared based on their characteristics with the other classified cases. The point with the minimal distance is called the nearest neighbor and the new instance takes as the prediction the class of this point. The advantage of this method is that no training session is needed because all the observations for the classification are stored. This, however, also leads us to the downside of this class of algorithms, which is an increased space requirement. The nearest neighbor method uses only one neighbor and classifies an instance based on the class of its nearest neighbor.

The extension of the nearest neighbor method, kNN, uses k-neighbors and takes the class from the majority of the neighbors which makes the method more robust against outliers. A too large value of k, on the other hand, takes to many neighbors from other classes which could lead to an incorrect classification [WKQ$^+$08]. Further modifications of the KNN method were presented in the last decade using different distance functions with scaling for $k > 1$, and voting methods with distance-depended weights

Decision Tree

Decision Tree (DT) [Qui86] is one of the most successful supervised learning method for classification and regression. It models the decision process in a graphical representation of a tree. This tree-based representation of the information includes the concepts of nodes, branches, and evaluation performance measures. One of the most used methods is the C4.5 decision tree [Qui93].

A training set is used to generate a DT using each level of the tree to split the data according to different attributes into smaller subsets. This concept is known as the divide and conquer method. Using this method the whole training data set is split into decreasingly smaller subsets using single attributes in the nodes until either an abort criterion is satisfied or all data instances in a terminal node belong to the same class. A key component of this algorithm is the selection of the observed splitting attributes for the splitting-node, as this affects the depth of the search tree and quality of the prediction. After each split of the data different evaluation criteria are applied to inspect the quality of the split. The information gain [Qui86] and the gain ratio [Qui93] are the most used criteria in practice. Once the tree is completed, it can be trimmed back by cutting of nodes. This process is generally referred to as pruning and helps prevent against overfitting [WKQ$^+$08]. After each pruning step the algorithm computes the prediction accuracy of the new subtree and replaces the unnecessary nodes with a leaf-node. If the prediction accuracy does not deteriorate, the change is kept, otherwise, the old tree is restored.

After the tree is completed and all the desired criteria are met it can be used to predict the classes of new data. New datasets are processed throw the tree, starting with the root node and the algorithm evaluates the decision rule at each node and forwards the data to the next node with the next decision rule until a leaf node is reached. When a leaf node is reached the algorithm returns a classification for the new data. So we are able to check each step of the classification process in the tree at any time which results in an easy and intuitive system.

Random Forest

The random forest method (RF) [Bre01] is a versatile machine learning method which uses an ensemble approach of several individual DT. The main principle behind this method is to group several "weak" decision tree classifiers to a "stronger" classification model with higher generalization accuracy. For the RF method, t different decision trees with a random subset of the training set are computed. To classify a new dataset based on his features, the data is processed by each of the t single DT. Therefore, each single DT classifies the new dataset. The RF method counts all the "votes" and chooses the classification with the most votes.

RF are able to handle large data sets with a rather short training phase[Ho98]. They are capable of handling thousands of input variables and identify the most important ones. Additionally, the RF method is able to handle missing data, use bootstrap sampling and is rather robust against overfitting [CKY08, Ho95]. As a drawback of its higher complexity, the method is no longer as intuitive as a single DT and can feel like a black box approach, since only the parameter t can be used to tweak the model.

Support Vector Machine

Support vector machines (SVM) [BGV92, Vap13, CV95] is a machine learning algorithm which is mostly used for the classification of data but is also capable of regression tasks. The method proceeds as follows: the SVM "plots" each data point in an n-dimensional space (where n defines the number of features) with a coordinate for each feature value in the respective dimension. The algorithm tries to separate the data into two subsets with a hyperplane. These subsets ideally consist of only one class, which is assigned to the data. The SVM tries to maximize the margin between these two classes with a distance measure, which computes the shortest distance between the hyperplane and the closest data points per class to the line. These data points are the so-called support vectors. If it is not possible to separate the data into mutually exclusive sets, then a punishment function is used to calculate a "soft" margin for the separation. An alternative is to transform the features in a high dimensional feature space. This transformation may be non-linear, therefore a non-linear classification kernel function is used. These kernel functions are usually radial basis functions or high order polynomials. After the training phase of the SVM, new data can be classified by using the decision function. This decision function positions and classifies the new datasets into one of the two classes. If more than two classes are given, we merge multiple (binary-splitting) SVM to compute a hierarchical decision for the new data point.

The SVM needs less training data than similar methods, to produce good classifications and is rather robust against outliers. Another honorable mention is that its theoretical bases are well-founded [WKQ$^+$08].

Neural Networks

Neural Networks (NN) [Fun89, RNI95] are inspired by the brain, especially the stimulus. The method uses the concept of neurons, which is a single unit that is linked with outer neurons by a selection of input and output channels. The neuron gets an input signal and sums them up until they exceed a certain threshold value. If the limit is reached, the neurons activate and start to "fire" a signal to the other connected neurons, which can thereby are also activated if their respective threshold is exceeded. If not, the signal stops at this node.

A variation of the NN method is the multilayer perceptron (MLPE) [Vap13], which is a popular modification used for classification duties [SMP11]. A perceptron is, in its simplest form, a neuron with multiple inputs and a corresponding weighting function which defines the influence of each input variable. The neuron processes the information and calculates a weighted sum which is passed to the next level via its output function.

The MLPE method arranges the perceptrons in a layer structure, where each node is connected to all the other nodes in the next level of the layer.
The MLPE is trained via a back-propagation approach using the error of a wrong classification to update the weight function of the neurons. To classify new data, the features are used as input variables. This information is processed through the input - and several hidden layers until an output layer is reached with the associated class.

The hidden layers of the MLPE work like a black box, which makes the method a little harder to trace. The other downside of this method is, that in comparison to other procedures, more time is needed to create a reliable classification model. The advantage of this method is their robustness and the possibility to model non-linear dependencies.

Generalized Linear Models

The Generalized Linear Models (GLM) [NB72] is an extension of the classical linear Models and was introduced by Wedderburn and Nelder in the year 1972. The advantage of GLM is that they allow the mean to depend on the explanatory variables through a link function. Furthermore, the GLM model allows us to use, besides the normal distribution, also the distributions from the exponential family for the response variables. [McC84]. This distribution class also includes Binomial, Poisson, Gamma and inverse Gaussian distribution. This method could, therefore, be used for the estimation and testing of classification and regression models for a variety of problems.

The GLM's have been used successfully for various fields of application like economics, education, engineering, environmental studies and pollution, geography, geology, history and medicine [Lin00]. Furthermore, the GLM's have been applied for the insurance industry to support critical decisions [DJH+08]. An extension of the GLM which has received special attention especially in the last years is the Generalized linear mixed model (GLMM) [MN13]. These models add some random effects into the linear prediction of the GLM's.

Naive Bayes

A Naive Bayes classifier is a probabilistic classifier (NB) [HY01] derived from the Bayes theorem [BPC63] named after the English mathematician Thomas Bayes and published by Richard Price after Bayes death. In the Naive Bayes classifier, each object, in our case, each bin packing instance, is assigned to a class or category to which it most likely belongs. The methods predicts therefore, the probability of a previously unseen problem instance belonging to each class and simply picks the most probable one. The underlying assumption in the Naive Bayes classifier is the independence of the problem features used. This "naive" assumption gives the name to the classifier.
This "naive" approach achieves for some applications good results in practice [Ris01, Mur06].

The NB method has already been successfully applied in various applications, including medical diagnoses support, text classification and systems performance management [DP97, M+97, HJR+00]. For example, the studies [TMM+81, MPW97] show that the NB is the best choice for their predictions. Other studies, which found that the NB method performed very well, sometimes even better than the alternatives [RKF83, TS94] and [FGG97].
On the other hand, there are seams to be some studies which show that the NB classifier performs rather poorly in comparison with the other used methods. Experience shows that the assumption of independence of all input variables is most of the time not useful (also in our use case). This fact leads to the consequence that the model was often rejected by researchers for the benefit of other procedures.

2.5.2 Regression

The machine learning procedures we discussed so far classify the given instances. If we try to predict a target value which is a real-valued number we have to use regression methods since classification methods, in general, are not able to do so. However, there are procedures that are capable of regression and classification tasks. If we are using regression, we try to approximate the underlying function of our training data to predict the value of the target variable for a new problem instance. Regression analysis [BH03, MT77] is a statistical tool for the investigation of relationships between some input variables $x1, x2, \ldots, x_n$ to one target variable y. The simplest variation of this technique is the linear regression which uses a linear model to find relations between the input variables $x1, x2, \ldots, x_n$ and y. The expected value $E(y_j)$ for the target variable y can be modelled as

$$E(y_j) = \beta_0 + \sum_{i=1}^{n} \beta_i * x_{ij}$$

The variable $i = 1, \ldots, n$ is the index for the feature values and β_k with $k = 0, \ldots, n$ determines the weighting values of the formula. The complexity of the model can be increased significantly with other functions like cubic or quadratic regression when no linear relation between the data is suspected. In this thesis, the regression analysis is used for the algorithm selection but especially for runtime prediction of the algorithm. The model can be trained to find a context between the computed features of the problem instance and some performance criteria (e.g. runtime). The regression predicts the performance of all algorithms which can be ranked afterward according to the results. Consequently, we compute the best algorithm for each instance which can then be used to find the optimal algorithm for a new problem instance.

As mentioned above, some of the machine learning methods are able to cope with classification tasks as well as regressions tasks. Therefore we used the k-nearest neighbor (kNN), decision trees (DT), support vector machines (SVM), random forests (RF), the generalized linear models (GLM) for the classification and regression experiments. We extended the methods with the loess regression model (LOESS) to bring more variability into the evaluation. In the next section, a short overview is given for the loess regression.

Loess

The LOESS regression procedure (LOESS), which stands for locally weighted scatterplot smoothing is a nonparametric method for estimating regression surfaces introduced by [CDG88, CG91] and [CGS92]. The LOESS regression is mostly used to model datasets, where the different variables clearly show a non-linear behavior. The non-parametric method uses a least squares regression that is performed on localized subsets of the data, which makes the method a suitable candidate for smoothing any numerical vector. The size of the subset or neighborhood is chosen such that a fixed percentage of the data points is covered. This value can be adjusted to fit the particular problem and is called the smoothening parameter. In addition, the individual data points in the local neighborhood are also weighted using a weight function depending on how far their position is from the center of the local neighborhood.

The loess regression method is used in practice for time series prediction [HN97] time tracking and predicting target motion [RFB07].

One of the biggest advantages of LOESS is that it only needs a smoothing parameter value and the desired sizes for the various neighborhoods to fir a model for the dataset. Second, the procedure is rather flexible and able to model complex data structures where standard regression models fail.

One disadvantage of the procedure is that it requires a relatively large data set with densely sample data to produce good models. This is due to the local neighborhood structure of the method which needs enough data points to build a comprehensible model.
The other disadvantage is that the LOESS regression model is, like all other least square models, also noticeably affected by outliers in the data. In addition, it is worth mentioning that the method is rather complex and computationally intensive in relation to the selected parameters described above.

2.5.3 Discretization

As already discussed, we calculate features from the problem instance to use them in a variety of machine learning methods. Usually, the values are represented as either ordinal, which represent the information in a finite set of possibilities. The numerical values can take an arbitrary value that is usually represented in a linear order. The problem is that some of the machine learning techniques are not able to handle continuous data. To bypass this problem a transformation of the continues data into nominal data with different discretization methods could be used. Some procedure like the $C4.5$ decision tree have this procedure per default already implemented which is very convenient for the users. The discretization techniques are categorized into supervised, unsupervised, global, local, static and dynamic methods [DKS$^+$95]. As an example for a supervised, global and dynamic method we could use the minimum descriptive length(MDL) method [FI93].

For this thesis, we use two different supervised discretization techniques. The Minimum Description Length Principle (MLP) [FI93] discretizes the continuous attributes of data matrix using entropy criterion with the Minimum Description Length [BRY98] as stopping rule.

The second discretization technique used is a top-down(TOP) method using the Class-Attribute Interdependence Maximization (CAIM) algorithm [GACVO09]. The CAIM criterion measures the dependency between the class variable and the discretization variable of a feature or attribute.

For a more detailed overview of discretization methods, we refer to [DKS+95].

A data transformation can also be taken into consideration even if it is not necessarily required by the machine learning algorithm. Some algorithms show a better performance if the data is converted from continuous variables into nominal ones [DKS+95]. Research [YW03] in this area clearly indicates that kNN and naive Bayes deliver better results if the data is previously transformed.

2.5.4 Feature Selection

Using a set of attributes to teach a model to predict new information is one of the core applications of machine learning procedures. One motivation of this thesis is to find high-quality features to model the significant properties of a problem instance. Inexperienced users could attempt to create more and more features trying to enforce better predictions. In some cases, meaningless features are able to distract the classifier and could lead to a poor performance[Hal99, TBB+91]. Bad features are distinguished by the fact that they provide only redundant information or that they are strongly correlated with other features and thus contribute no useful information to the prediction process. Other downsides of too many useless features is a longer computation time [LS97] and lower accuracy [LS97, Joh97].

To choose the right subsets $S \subseteq F$ from the set of all computed features F which provide the best performance for the algorithms is an independent optimization problem. Finding these subsets S is a hard task for every machine learning method and is generally referred to as the feature selection problem. The problem space for F is defined by all possible subsets of F. This means they are 2^n possibilities to split the features for our experiment. Even for a very small n, it is practically impossible to test all subsets to find the optimal one. A pre-selection of the possible features by hand could decrease the search space significantly, but requires a very good knowledge of the domain, some expertise for classification algorithms, and of course, time. A pre-sorting of the features, like in [JKP+94], into strongly relevant, weakly and irrelevant features can help to find the ideal composition. However, just using all the strongly relevant features often leads into suboptimal results [GE03]. A mix of all strongly relevant and some weakly relevant features usually leads to good results. Thankfully, there are several processes for automated feature selection that use two techniques as shown in [GE03]. One technique uses a search algorithm, similar to forward - or backward selection and genetic search procedures, to find a good bundle of features that perform well. The subset selection approach can use some filtering methods [JKP+94] with no more information needed for the execution of the selection task. These algorithms compute a "value" of the feature set considering certain criteria (e.g. correlation) and select the set with the highest "value". These filter algorithms are fast, flexible and rather versatility. The wrapper-based techniques need a predetermined classification method which they apply to the subsets and then evaluate performance as the selection criteria. Advantages, of the wrapper algorithms, are the better preference for a targeted classifier, but with the downside of losing generality.

Alternative methods to the subset selection approach, use the individual features and rank them to suit their individual performance on some specific measurement like the information gain [GE03]. This ranking is used in a second step to choose the best features for the task. For further information on the feature selection we recommend [DL97] and [GE03] to get a good overview about this procedure.

CHAPTER 3

Algorithm Selection

3.1 Define a good algorithm

In computer science, and especially for heuristics [BGK+95], we mostly deal with a set of three criteria which define the "best" algorithms: computational effort, quality of the solution and robustness.

The first criterion that we analyze is the computation complexity of algorithms. One of the most important issue, when we talk about the complexity of an algorithm, is the memory consumption. The memory usage strongly depends on the used machine and on the quality of the implementation (e.g. check if the code is optimized and delete redundant or dead variable). These factors may greatly differ and lead to inaccurate measurements. However, the hardware and software components can easily be extended and since most heuristics only use a smaller part of the search space, they are much less memory depended than the exact algorithms. Moreover, the lack of memory can be easily mitigated by more computational effort, which leads to an increased computation time, and therefore is one of most important argument against the memory criteria. That is why in most practical cases the researchers concentrate more on the runtime requirements and use the memory criteria only with reservations or they exclude it for the comparison.

The most important benchmark for a good algorithm is the time requirement. There is more than one method to compute the runtime of an algorithm. One way is to measure the CPU-execution time, which leads to easily comparable measurements, but the method is controversial among researchers because the results depend highly on the hardware and are difficult to reproduce [AO96]. Despite these problems, the method is still popular and widely used in practice.

An alternative is to count the program calls [AO96] and evaluate the algorithm based on the operation cycles needed. These give a hardware independent indicator for the quality of a solution.

As we already mentioned, in computer science we usually talk about optimization and decision problems. Depending on the problem we usually need different measures to describe the quality of the computed solution. When two algorithms produce the same results the time is usually used as the second criteria to judge the quality of the solution. The optimization problems, on the other hand, are a little more complex to describe. In this context, we speak of the solution quality, which highly correlates with the runtime in most cases. Better solutions can be achieved by a longer runtime which also allows the algorithm to search a bigger area of the search space, in an extreme case the algorithm could be able to check to whole search space and find the optimal solution. Clearly, that would result in a long runtime, which is in most cases not desired. Therefore, a balance has to be found between the current solution and the time needed to find it. The task to find a good termination condition (e.g number of iterations, max runtime), is also a very important.

The robustness of an algorithm gives some insight into the stability of a method. It tells the user how well the algorithm can adapt to different instances of the problem, and how well the parameter tuning can be used to solve problems better. The robustness describes, therefore, the behavior on a set of instances and not on a single one. The robustness criteria are less effective in this field of research and are therefore ignored in this thesis since we are mostly interested in instance-based decisions.

3.2 Choose the best algorithm

In some cases, where algorithms show similar performances, it is hard to say which algorithm performs better based on some criteria on a set of new instances. To just select one algorithm random from the set of algorithms which perform similarly, is a valid method but could lead unpleasant results.

In this thesis, we use a combination of the quality of the solution, i.g. the number of bins needed to pack all items, and the execution time of the algorithm to rank the procedures. This helps us to avoid more than one optimal algorithm per problem instance.

A further important factor for the quality of algorithms is the use of randomness of the procedure. Often state-of-the-art algorithms use random components to compute good solutions. This randomness factor is used in the progress of creating a starting solution to help to explore different promising areas of the search space. Of course, the randomness has an impact on the quality of the solutions and also influence the duration of a process [HLH97].

According to the information of the individual evaluation criteria for an algorithm we are now able to say that an algorithm A is, in general, better than an algorithm B, if we compute the evaluation criteria and A performs at least as good as algorithm B and for at least one criteria A must perform better than B. Then it should be save to say that A is clearly better than B for the tested instance[Ewa12].

3.3 No Free Lunch Theorem

The no free lunch (NFL) theorem was introduced by Wolpert and Macready [WM$^+$95, WM97]. It states that we are not able to find one optimal universal optimization algorithm for all problems. If we find one particular algorithm outperforming another in a particular situation, it is a consequence of its fit to the particular problem, not the general superiority of the algorithm [WM$^+$95].

Originally the theorem was developed for naturally inspired heuristics, but it is generally valid for all optimization algorithms that use a so-called black box[1] approach. The findings of Wolpert and Macready may indicate, that if we are interested solely in the generalization performance, there are no reasons to prefer one algorithm over another. After we have looked at the NFL theorem more closely, the question arises how we can find a "suitable" algorithm for our problem?

Fortunately, for a particular problem instance, the performance of two or more algorithms can be different, especially when a procedure is specifically designed for some particular classes of instances [WM$^+$95, DJW02]. Therefore, algorithms can outperform all other algorithms for a particular set of instances.

Using the concept of the NFL theorem, we are eager to find a hidden structure of a problem that describes how difficult an instance is. In general, we try to find out whether it is possible for BPP to create a set of features that influence the performance of algorithms. If this is possible, we are able to create some instance-specific predictions about the performance of an algorithm for a new instance.

We investigate the following question in this thesis:

- Does the assumption hold that no single algorithm outperforms all other algorithms on all instances of the BPP?
- Are we able to create some features of the problem instances, for the BPP, which express information about the performance of various algorithms?
- Are we able to predict the "best" algorithm for a new and unseen problem instance of the BPP, using these calculated attributes?

But how can we select the "best" algorithm for an instance and which features describe an instance in the best manner? To answer this question we will deal with the well-known Algorithm Selection Problem.

[1] The black box approach describes an optimization technique where the algorithm has no information about the problem and just tries to max a cost function.

3.4 The Algorithm Selection Problem

Many researchers use different approaches to solve a problem. The invented procedures have mostly advantages and disadvantages which can be used selectively to solve particular problem instance better. Finding the "best fitting" algorithm is a hard challenge, given the need to address the different problem areas of instances deliberately.

3.4.1 Algorithm Selection: The Concept

The Algorithm Selection Problem (ASP), first introduced by Rice [Ric76], solves the task of finding, among a set of algorithms, the best for a specific problem instance.

A schematic overview of the ASP by Rice [Ric76] is given in Figure 3.1 [2].

Giving this overview, term x describes an instance of the problem space P. The problem space P is the set of all possible problems, in our case the input data, on which the different algorithm should operate. Further, the term $f(x)$ is a representation of the features of $x \in P$ using a feature extraction function F. The criteria space is defined as a vector $w \in \mathbb{R}^n$, in our case the performance measures are defined by $n = 2$, the bins needed and the second being the time needed to pack the items. The algorithm space A defines the set of all algorithms from which we choose the best one for each instance. The ASP tries to find the best algorithm $a \in A$ from the set of all algorithms A, using a performance mapping which maps the algorithm $a \in A$ and a problem $x \in P$ to an element from the criteria space $\mathbb{R}^n$. This performance is finally mapped by a norm function $g(p, w)$ to the performance of the algorithm [Ewa12].

Over the years, the ASP has evolved and we can distinguish between static and dynamic approaches [Guo03]. The static methods select an algorithm using a predefined underlying model, while the dynamic method observes the performance of selected algorithms and, if needed, changes the selection of the algorithms in the progress. Especially the recursive ASP may be mentioned here, which checks the selected algorithm after each recursive call [LL00, LLP01, XHHLB08]. In contrast to the non-recursive algorithms, the recursive processes divide the decision process into subproblems with sequential decision tasks.

The next sub-chapters deals with important measurements of the ASP, namely the feature space and the algorithm space. The criteria space, a further possible measure for an algorithm, was already discussed in Section 3.1 and therefore will not be discussed further in this sub-chapters.

3.4.2 Algorithm Space

Algorithms with different performances on instances are necessary for the ASP. Theoretically any procedure, no matter how simple or complicated it is, can be used as a source for the selection process. Should we use some methods that prove to be unsuitable for

[2]The figure was taken form [Sch12b]

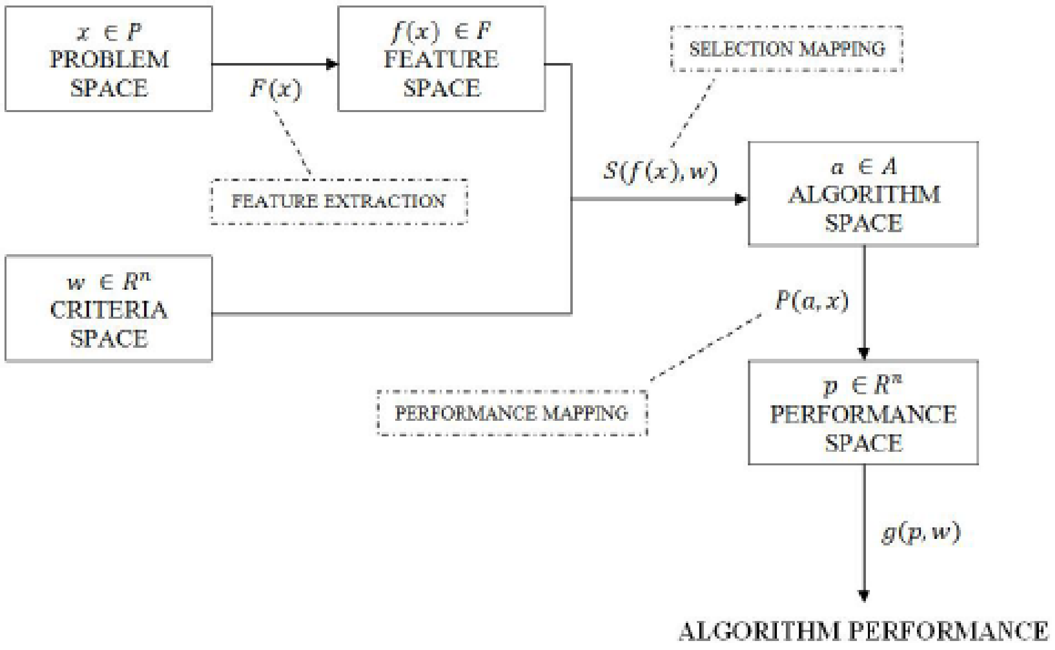


Figure 3.1: Model overview of the algorithm selection by Rice [Ric76]

the problem, an optimal selection strategy will avoid them anyway. So in theory, we are not bound to any specification regarding the methods to choose, but in practice, it is rather difficult to find the right mix of algorithms for a problem. For example, exact methods may solve some problems well but need to much time for other instances. A good selection of complementary algorithms for the set of instances is necessary for a useful ASP approach. To select this set of algorithms a manual selection process is sometimes used to identify good candidates for the problem instances. This process requires a depth knowledge of the problem domain.
Another way is to use the meta-information available provided by benchmarks and existing research or complexity features, which will be discussed in Section 3.4.4, to preselect the algorithms for the ASP.

3.4.3 Feature Space

The feature space describes the quantity of all features that describe a problem instance. It is one of the most important elements for the ASP since the features determine how well we are able to describe the problem instances. There are no generally valid rules for automatically choosing features [Nud05]. Therefore, expert knowledge about the domain and analytical intuition is needed. Another possibility is to use features from similar problems [SML12], that usually contain some basic information about the size of the instances and other metrics. More about features for optimisation problems can be found at [SML12].

Basically, every measure that is associated with an element of the problem instance can be used as a feature. We must also consider that too many features or redundant ones do not always produce better outcomes [Nud05]. However, in order to determine which features are useful and which not is often a very difficult task but the following two rules help us to select appropriate candidates: First, it is important, that the features do not need some prior knowledge about the instance to be computed since this information is usually not available. Second, the time needed to compute a feature is critical. If the features take too much time, e.g. longer than a low-order polynomial time period, they are usually impractical to be effectively used in practice. A similar problem is the meta-reasoning-partition problem [HB90], which describes the problem that, the more time a method needs to compute some features, the less time there is to actually solve the problem. So there is always a trade-off between the benefit of a good feature computation and the benefits of solving the underlying problem in reasonable time.

A method that is very often used is the so-called feature selection method which was discussed in Section 2.5.4 . Feature selection uses different machine learning techniques to pick the best available features.

3.4.4 Analytical Algorithm Selection

The use of analytical procedures and complexity theory to compare algorithms is another approach to find the best suitable algorithm for a particular instance. The complexity theory uses characteristics like best/worst/average performance to describe an algorithm asymptotically. Using this method for algorithm selection, we can schematically find the best algorithm for each instance based on its best/worst/average performance of all candidates.

These methods have some advantages and disadvantages with regard to the ASP. It allows us to compare two algorithms on a different level more clearly separated from the complexity of the implementation. The downside is, that the concerned instance and the implementation of the algorithm has a strong impact on the performance of an algorithm. This could lead to significantly different performances for asymptotically similar procedures [HCR$^+$00].

Due to the aforementioned problems, analytical analysis and complexity theory is rather inappropriate for an automatic algorithm selection approach [Ewa12]. Nevertheless, this method can be used in the manual pre-selection process for the ASP.

3.4.5 Machine Learning for Algorithm Selection

When we are using machine learning techniques we are mostly interested in the performance on different problem instances rather than some mathematical analysis of the algorithms. To do so we use some empirical procedures. We analyze the algorithms on their performance on different types of instances and compare the results. The gained information is used afterward to choose the best procedure for an instance with specific properties. This procedure refers to machine learning techniques because trying to learn a pattern from the performance of different algorithms to predict the best solution is the typical machine learning approach. Many of earlier introduced machine learning methods have been already used very successfully in the area of the algorithm selection problems [Ewa12].

Nowadays, machine learning methods are considered the most promising techniques for solving the ASP. Nevertheless, there are discussions which of the two fundamental concepts, classification or regression, is more suitable for the ASP. Recent research suggests a preference for regression methods, like in [XHHLB08, LBNS09, PM14] while the classical analysis of Rice [Ric76] encourages a procedure similar to decision-trees as well as other approaches like in [Aha92, Bro93] favor classification.

The two methods use a different approach to deal with the prediction of errors. Classification methods make no distinction between the different miss classifications. Whether the proposed algorithms give a nearly optimal solution or he performs not even close to a good solution. The punishment of an almost optimal miss-classification or a rather bad classification is similar for the classification methods. Worth mentioning is that advances in the classification methods also allow a differentiated penalization of mismatches.

The regression methods, on the other hand, penalize an almost optimal prediction only slightly. This advantage of regression methods is provided due to a better error metric handling. Thereby, regression procedures are able to minimize the risk of getting a rather poor algorithm for an instance. The downside of regression methods is the execution time and the complexity of the methods [MSSS11].

It is difficult to choose between the classification and regression methods as it greatly depends on the area of application. With the better error handling, regression methods seem to have an advantage, but using regression for optimization problems, forces us to predict the solution quality and the execution time [MSSS11] which includes more computational complexity. The good thing is that both methods have a rather active community to provide us with state-of-the-art algorithms. Additional results for the comparison between regression and classification models can be found in [MDC11]. A comparison of different machine learning methods for the ASP can be found in [KGM12].

The next subsection gives a brief insight into the application of machine learning methods for algorithm selection. For more information, please refer to the cited sources.

Representatives of machine learning techniques for the algorithm selection problem are the portfolio-based regression methods like SATzilla [XHHLB08] and its descendants [XHS^{+}12]. This state-of-the-art algorithm selection method successfully predicts the best suited SAT-solver for particular problem cases and was able to win different categories in the 2007 and 2009 sat competition [XHHLB12]. Another method for SAT that achieves better results than the SATzilla, using a kNN classifier, was introduced in [MSSS11].

Other achievements in this area to go back to static and dynamic scheduling strategies by [KMS+11] as well as greedy selection strategies from [NMJ13].

The ASlib library [BKK+16] is a benchmark library trying to standardized formats of algorithm selection problems and scenarios. This formats are applicable to a wide variety of algorithm selection scenarios and helps to compare the results from different approaches.

Other areas where machine learning techniques are successfully used for algorithm prediction is answer-set-programming (ASPr) [GL91], with the CLASPFOLIO methode [GKK+11] and AQME [PT07] a similar framework for quantified Boolean formulas. Also worth mentioning is the classification- regression-based procedure by [Sch12a] and the multi-level approach by [MPR15] for ASP. Furthermore, in 2016 a benchmark library has been introduced by [BKK+16] to better compare different approaches.

For the traveling salesman problem (TSP), a multilayer perceptron method is introduced by [VG08]. DT, kNN, SVM and Naive Bayes networks were used by [KCHS11] to predict the best algorithm for the TSP. Other approaches using a regression-based system and other strategies can be found in [LBNA+03, LBNS09, PM14].

The traveling thief problem (TTP) was introduced by [BMB13] as a combination of the TSP and the knapsack problem (KP) [KPP04]. For this rather new optimization problem, an algorithm selection approach has been presented by [WLM+17]. They evaluated the performance of 21 TTP algorithms and introduced the first algorithm portfolios for the TTP.

The algorithm selection technique has also been successfully used for the graph coloring problem (GCP) by [MS13]. They evaluate the performance of six state-of-the-art (meta) heuristics for the GCP and were able to achieve better performance with the machine learning algorithms.

For learning problems, which are not directly correlated with our use case we refer to [SM09] with the approach to select the best learning algorithm for a problem by [Aha92, Bro93, BSDC03, LBV12, AS06].

Besides these "classical" machine learning approaches meta-learning [WFHP16] methods have been introduced in the last decade. Procedures for Markov decision processes using meta-learning methods for a subproblem in SAT were introduced by [LL01] or the OSSP [LL00]. A statistical model using regression methods and a plain set of rules for a method selection approach is given by [Fin98]. Using runtime prediction for branch and bound algorithm is documented in [LL+98].

Method for backtracking search using a portfolio based procedure can be found in [WvB07] and in [YMTS09].

An example of a combination of a portfolio based algorithm selection approach and an automated algorithm configuration is HYDRA [XHLB10]. For more information of new technologies and historical remarks we recommend [SM09], [Kot12] as well as [Kot14, MSKH15].

CHAPTER 4

Algorithm Runtime prediction

4.1 What is Runtime prediction

Most of the $NP - complete$ problems are solvable these days with good results in a reasonable amount of time with a good heuristic. The downside is, depending on the algorithm, the run times could vary greatly [GSCK00]. In the last decade, the algorithm runtime prediction (ARP) method was introduced, which provides information about the approximate runtime given some criteria, to verify if an algorithm is usable for a specific task or not.

The method is quite similar to the ASP, since both procedures use features to predict the result on an unseen problem instance, using prediction models. Even more, the ARP is used in different applications like the automatic configuration of parametrized algorithms or portfolio-based algorithm selection. It follows, that the more information we are able to gather of the problem the more we can improve the quality of our solution.

The goal of the ARP is to predict the runtime of an algorithm on a previously unseen input using machine learning techniques. The forecast is done using a prediction model for the algorithm's runtime. The instance features are used as the input of the prediction model to produce the forecast for the unseen problem instance [HXHLB14].

In this thesis, we use the ARP to gather additional information about the problem instances for future work and to predict the run times of three state of the art bin packing algorithms.

4.2 Algorithm Runtime Prediction: The Concept

The Algorithm Runtime Prediction (ARP) has been researched in various fields of computer science. Inspired by the ASP [Ric76] different machine learning techniques are used for the ARP. One method uses statistical linear regression techniques to predict the expected runtime of new unseen problem instances. The first models using this method date back at least to the mid-1990s [Bre94, Bre95].

The problem instances are described by the term x , with $f(x)$ defining the features of x. The performance criteria w defines the runtime needed to solve a particular instance x on a reference machine. The ARP tries to find the best algorithm a, from the set of all algorithms $a \in A$, using a selection mapping S that minimizes w. The function $P(a, x)$ then predicts the quality of the solution p of a on the instance x. This performance is finally mapped to a function $g(p, w)$ to the performance of the algorithm [HXHLB14].

Some data transformations are able to simplify the problem, which helps the modeling process. A log-transformation of the runtime is often used to effectively predict the log runtime due to the limitations of the accuracy of CPU timers to measure time below a certain threshold [HXHLB14].

4.3 Machine Learning for Algorithm Runtime Prediction

By similar reasoning as discussed in Section 3.4 we also use machine learning methods for the ARP. However, please note that the ARP is, in contrast to the ASP, a regression problem. Not all of the already proposed methods in Section 2.5 can be used.

The next section briefly describes some popular machine learning methods for the ARP. For more information please refer to the given sources.

One popular and simple linear regression method for the ARP is the ridge regression as featured in [Bis06]. This method is used in many different ARP approaches due to its simplicity and interoperability [Fin98, HDH$^+$99, LBNS02, LBNS09, NLBH$^+$04, HHHLB06, XHLB07]. Variations of the ridge regression, e.g. Ridge Regression Variant RR [XHHLB07, XHHLB08] and [HJY$^+$10], have been developed over time allowing the use of iterative modeling methods and forward/backward feature selection methods to reduce the cross-validation error in the models.

The neural networks are used by Smith-Miles and van Hemert [SMvH11] to predict the runtime of local search algorithms for solving timetabling instances. Popular resources for such methods are the Neuroshell software and the Matlab neural network package NETLAB [Nab02].

Another machine learning technique is the Gaussian Process Regression [Ras06] first applied by Hutter [HHHLB06] for the ARP. This method has its roots in geostatistics [Kri51] and uses a kernel-based function computing the similarities between the pairs of elements.

Similar to the classification trees of the ASP, the ARP uses regression trees [BFOS84] for the prediction. This method was first applied by Bartz-Beielstein and Markon [BBM04] for the ARP and is known to handle discrete inputs very well. The pruning procedures of the regression trees paired with the cross-validation method allow a cost-complexity pruning of the tree to define a good trade-off between the prediction quality and the complexity of the tree. Additional information about regression trees can be found in [DF00, ELH08].

CHAPTER 5

Application of the Algorithm Selection and Algorithm Runtime Prediction for the BPP

In the following chapter, we discuss our approach of the Algorithm Selection Problem (ASP) and the Algorithm Runtime Prediction (ARP) for the bin packing problem. We give an overview of the heuristics used describe the bin packing features as well as how to compute them to apply the ASP and ARP for the BPP.

In the last section, we demonstrate our algorithm selection and algorithm runtime approach for the BPP.

5.1 The Bin Packing Problem

5.1.1 Definition

One of the most studied computer science problem is the bin packing problems (BPP), which is a variation of Karp's [Kar72] well studied NP complete problems. The problem has its origin in the 1960‘s and variants of the BPP find their use in many practical applications, such as cutting of materials (wood, glass, stone, cloth industries), as well as newspaper paging. Many different variations of the BPP have been presented in literature and a variety of solving heuristics have been proposed since its first introduction over forty-two years ago. The classical BPP is defined as follows: We are given a set of items $I = \{1, \dots, n\}$, where each item $i \in I$ has a size $s_i \in (0, 1]$ and a set $B = (1, \dots, n)$ of Bins with capacity of one. The goal is to find a valid assignment $a : I \rightarrow B$ such that the number of bins used is minimal without overlapping any item and exceeding the capacity of the bin.

The problem description is simple, but it is indeed hard to find a good arrangement of items in the bins to achieve an optimal solution. Moreover, with the fact that the problem is $NP - hard$, it is very unlikely that there exists an exact strategy that needs less than exponential time to find a minimal packing of the items unless $P = NP$. The BPP is part of the so-called Cutting and Packing category of problems [Dyc90].

This thesis deals with the 2 dimensional (2d) bin packing problem, which is an extension of the classic 1-dimensional packing. For the 2d BPP a set of n items $i \in I$ is defined by his height $h_i \in (0, m]$ and by its width $w_i \in (0, m]$. The bins of the 2d BPP are also defined by a height and width. There exist some varieties of the two-dimensional bin packing problem which are defined by the (1) orientation of the items and (2) the guillotine cutting pattern[1]. As an example, for a newspaper article rotation is not desired but for wood cutting, it is normally no problem. The guillotine cutting pattern is mostly needed for automated cutting machines and the complexity of programming them. [BLS05].

According to these settings, the following variations of the 2d BPP exist:

1. 2BPOG: the items are oriented (O), and a guillotine cutting (G) is required;
2. 2BPRG: the items can be rotated by 90°(R), and a guillotine cutting (G) is required;
3. 2BPOF: the items are oriented (O), and the cutting is free (F);
4. 2BPRF: the items can be rotated by 90° (R), and the cutting is free (F);

It is important to know that some of the proposed methods can deal with more the one variation of the 2d BPP, others are suitable only for one of the four cases.

Our approach in this thesis deals with the first two cases of the 2d BPP variations, due to organizational reasons and licensing problems with the needed procedures and algorithms. In the next section, we discuss recent advances in heuristics and exact algorithms for the 2d BPP.

5.1.2 Exact Algorithms for the BPP

One property of $NP - hard$ problems is that the process of finding the optimal solution for exact algorithm could be very time-consuming. There are some exact procedures for the 2d BPP, which were particularly popular during the 1990s. The most recent work for exact algorithms for the BPP is given by Delorme, Iore and Martello [DIM16]. An exact algorithm was presented by Martello and Vigo [MV98], for which the items were sorted in a decreasing order of their size. In the next step, the method tries to pack the

[1]The guillotine cutting pattern specifies that the items must form a sequence of edge-to-edge cuts. This cuts must be parallel to the edges of the bin

first bins with the items in an optimal order by reducing the size of the instance. This process leads to a first intermediate solution z^* which is used in the next step. By using this intermediate solution a decision tree is constructed.

Next, a two-level branching scheme is used by the algorithm:

- outer branch-decision tree: for each branch in the decision tree an item is appointed to a bin without finalizing its position;
- inner branch-decision tree: a valid packing is searched for the items assigned to the bins.

The outer search tree uses a depth-first search to select the items. At a defined level $k = (1, \ldots, n)$ the item is assigned to an open bin, or to a new one if the number of bins needed by the current solution is less than $z^* - 1$.

In the next phase, the feasibility of the assigned items to the bin is checked heuristically in two ways:

- (i) a lower bound is computed for the instance. If the value of the lower bound is greater than 1 the packing is not feasible.
- (ii) an upper bound is computed for the same instance if the value is 1, a feasible packing is found.

Should both methods fail to find an acceptable solution, all possible packings of the items are enumerated by the inner branching scheme. After a feasible packing is found for the instance, the algorithm returns to the outer enumeration tree. Otherwise, the outer backtracking routine is performed.

Alternatively, another exact method introduced by Fekete and Schepers [FSVdV07] is based on an enumerative approach.

5.1.3 Heuristics for the BPP

The next section gives a brief overview of different heuristics for the BPP, explains there principles, shows the similarities and differences as well as their relation to each other. We focus mostly on the state-of-the-art algorithms used in the experimental part of this work. We distinguish between the previously discussed variations of the BPP, concentrating on the 2BPRG and 2BPOG case, and provided a listing of the best methods currently available. For more information about recent approaches on the 2d BPP the reader is referred to [WOZL13], [LH13] and [HBZS13].

Approaches for 2BPRG and for 2BPOG

Constructive heuristics

Some of the most famous constructive heuristics are the first-fit insertion (**FFIHOGJ04**), best-fit insertion (**BFIHOGJ04**) and the critical fit insertion (**CFIHOGJ04**) heuristic introduced by Fleszar in 2013 [Fle13].

These three procedures use a tree-based structure to perform guillotine cutting patterns. The items are inserted in a partial solution one at a time based on a procedure for enumerating possible insertions. The insertions are based on a fitness criterion which specifies the best placement for the item. The first two heuristics have a quadratic worst-case computational complexity, only the critical-fit insertion procedure has a cubic worst-case complexity. A detailed analysis of the performance and effectiveness of the three methods by comparing their empirical performance against other heuristics using popular benchmarks can be found in [Fle13].

Another heuristic introduced by Charalambous and Fleszer [CF11] using a principal of average-area sufficiency to choose the best fitting item for a bin is the constructive heuristic (CH). Three variations of this method have been developed, the constructive heuristic (**CH**), CH with bias (**CHB**), and CHB with post-processing (**CHBP**). The efficiency of this method is tested by a set of benchmark problem instances which can be found in the given source.

Neighbourhood heuristics

The Stochastic Neighbourhood Structures (SNS) solves the 2dBPP either with a two-stage or three-stage procedure. This heuristic was introduced by [CASDC11]. The method uses previously defined packing criteria to pack the items in existing or new generated bins. A solution is stored as a sequence of items which are packed into the bins. Three different neighborhood structures (cut-and-paste, split, swap blocks) are used by the SNS to modify the current sequence of items to improve the solution. Computational results show that the heuristic provides results within a small range of the optimal values of the instances. Generally, this method makes improvements to the other neighborhood heuristics like the Variable Neighbourhood Descent (VND) meta-heuristic [PAVOT10, HMP10].

Agent based heuristics

The next heuristic, proposed by Polyakovsky and MHallah (2009) [PM09] , uses an agent-based (**A-B**) implementation to solve the 2dBPP. The A-B systems use different agents, that try to fill the bins dynamically. The agents work together, though each agent is driven by its own decision processes, parameters, and fitness criteria. The basis of each agent is the guillotine bottom left (GBL) constructive heuristic which places the items in the first available most bottom left position of the chosen bin and defines the direction of the first cut of the strip containing the positioned item. The heuristics constantly updates the unoccupied areas in the chosen bin and tries to fill unpacked items into the bin until it tried all unpacked items. If there are still unpacked items, a new bin is opened until all the remaining items are packed.

Sequential value correction heuristics

The last algorithm introduced in this section, the **SVC2BPRG** proposed by Yi-Ping Cui and Yaodong Cui [CCT15], is based on the sequential value correction procedure that computes a specified number of cutting patterns, from which the most promising is selected as the current solution. Each of the patterns is generated sequentially which allows it to restrict some cuttings/packings if needed. Compared to other procedures, the heuristic performs very effectively by improving the quality of the solution, in most of the benchmark instances. This method is one of the best current state of the art procedures. The SVC2BPRG is the only heuristic used in this thesis which cannot be applied for the 2BPOG variation of the 2d BPP.

Other approaches

Of course, other methods were also used for the 2d BPP such as ant colony optimization [GTM+05], simulated annealing [LCT03] or a space defragmentation heuristic [ZGZ+11]. The quality of these methods, however, does not reach the level of heuristics already described. Interested readers will find more information in the referenced sources.

5.2 Our Contribution

In the following subsection, we describe the created features that describe the bin packing problem instances and present our approach for the ASP and the ARP for the BPP. We show which procedures and methods were used to build the prediction models. Furthermore, we list which preprocessing, feature selection and parameter setting we introduced.

5.3 Features

One of the central hypothesis of this thesis is that it is possible to create a representative set of features that characterize well the 2d bin packing problem instance. Using this set of features we want to find the best algorithm to solve the instance and define its hardness. Basically, features must be calculated quickly because otherwise they can not be effectively used for the ASP or the ARP. A set of good features covers different constraints, characteristics and properties of the problem instance.

However, "good" features alone aren’t of much use. A good learning algorithm is essential in order to make sense of the obtained features.

5.3.1 Feature Overview

In this section we will introduce and explain each feature using the syntax defined in Chapter 5.1.1. The features are computed for each of the 600 problem instance used in this thesis and stored as a matrix.

- Max Item size / (Elementgröße)
  - Calculate the size for each item by multiplying the height h_i with the width w_i of each item i of the problem instance and choosing the maximum size of the items.
- Total item size / (Gesamtfläche Items)
  - This features Sums up the size for each item $i \in I$ of the problem instance: $\sum_{i=1}^{n} h_i * w_i$
- Mean item size / (Mittelwert der Itemsgrößen)
  - This features calculates the mean of the item size: $\frac{\sum_{i=1}^{n}(h_i * w_i)}{n}$
- Median item size / (Median der Itemsgrößen)
  - This features calculates the median of the item size.
- Variance item size / (Varianz der Itemsgrößen)
  - This features computes the variance of the item size.
- Mean height / (Mittelwert der Items Höhen)
  - This features computes the mean of the item heights h.
- Mean width / (Mittelwert der Items Breiten)
  - This features computes the mean of the item widths w.
- Variance height / (Varianz der Item Höhen)
  - This features computes the variance of the item heights h.
- Variance width / (Varianz der Items Breiten)
  - This features computes the variance of the item widths w.
- Bin size / (Bin Kapazität)
  - This features computes the capacity of the bins B of the problem instance.
- Big items / (Grosse Teile)

  - This features computes the number of items i that have an $ItemSize \geq 50\%$ of the capacity of the Bins B.

- Small items / (Kleine Teile)
  - This features computes the number of items i that have an $ItemSize \leq 25\%$ of the capacity of the Bins B.

- Ratio Small Big / (Ratio Gross Klein)
  - This features computes the computes the ratio between the smallest and the largest item : $\frac{min(ItemSize)}{max(ItemSize)}$

- Ratio items / (Flaechenverhaeltnisse)
  - This features computes for each item i the ratio between the heigth h_i and width w_i. An $RatioItem$ value of 1 indicates a square : $\frac{h_i}{w_i}$

- Square items / (Quadrat Anteil)
  - This features computes the number of items i that are square or almost square : $RatioItems \leq 0.80$ or $RatioItems \geq 1.20$

- Odd items / (Unfoermigen Anteil)
  - This features computes the number of items i that have Uneven height and widths ratios : $RatioItems \leq 0.25$ or $RatioItems \geq 1.85$

- Lower bound bins / (Theoretische min Bin Anzahl)
  - This features computes the theoretical lower bound, the min number of bins B to pack all the items.

- Filler items / (Filler Items)
  - This features computes the number of items i that have $ItemSize \leq 10\%$ of the $BinSize$.

- One bin items / (Singell bin items)
  - This features computes the number of items i that have $ItemSize \geq 87.5\%$ of the $BinSize$.

- Ratio height item / (Hoehen Range)
  - This features computes the ratio between the min and max height of the items i of the problem instance: $\frac{min(h)}{max(h)}$

- Ratio width item / (Breite Range)
  - This features computes the ratio between the *min* and *max* width of the items i of the problem instance: $\frac{min(w)}{max(w)}$

Table 5.1 gives also an overview of all the features with a short description.

5.4 Experimental Setup and Environment

The following section explains the setup and the environment which was used for our algorithm selection and runtime prediction approach. The first part explains the evaluation of the chosen heuristics for the BPP, describes the different procedures and their parameter configurations. After that, a detailed overview of the setup for the machine learning task is given.

5.4.1 Definition and Algorithm setup

The classical algorithm selection approach, introduced by Rice [Ric76] consists of the following components as explained in Chapter 3.4 :

- the problem space P
- the feature space F
- the algorithm space A
- the criteria space W

Using this standard notation for the BPP, we define the BPP instances (P) as the problem space, the 24 extracted attributes belong to the feature space (F) as presented in Section 3.4.1, the algorithm space (A) is defined by the state-of-the-art methods introduced in section 5.1.3. The criteria space (W) is given by the combination of the bins and the time needed by the algorithms to solve the problem instances. The classification task/decision procedure is handled by different machine learning methods using various empirical approaches. Using a training set and the information gained from the extracted features, a classifier is built. This classifier is trying to learn the underlying patterns/structures of the problem instances. In the training set, the best algorithm is defined according to the performance criteria from their criteria space W for each instance.

The algorithm prediction task calculates the features for new unseen problem instances, feeds the information to the previously calculated classifier which finds the best-suited algorithm based on the learned information.

The algorithm runtime prediction uses a similar procedure but predicts the expected runtime of a new unseen problem instance based on the information learned from the prediction model.

Name	**Shortcut**	**Description**
Max item size	itemSize	Calculates the max area of the items
Total item size	totalItemSize	Calculates the sum of all item areas
Mean item size	meanItem	Calculates the mean of the sum of all item areas
Median item size	medianItem	Calculate the median of the sum of all item areas
Variance item size	varItem	Calculates the variance of the sum of all item areas
Item height	heightItem	A vector with the height of the items
Item width	widthItem	A vector with the width of the items
mean height	meanHItem	Calculates the mean height of the items
mean width	meaWItem	Calculates the mean width of the items
Variance height	varHItem	Calculates the variance height of the items
Variance width	varWItem	Calculates the variance width of the items
Bin size	binSize	Calculates the area of the bins
Big items	bigItems	Number of items that have an area larger than half of the bin size
Small items	smallItems	Number of Items that have an area less or equal than one quarter of the bin size
Ratio Small Big	ratioSBItems	Computes the ratio between the smallest and the largest item
Ratio items	ratioHWitems	Computes the ratio between height and width of the individual items
Square items	squareItems	Number of items that are square / almost square
Odd items	oddItems	Number of items that have an unequal high / wide ratio
Lower bound bins	lbBins	The theoretical lower bound, so many bins must at least be used
Filler items	fillerItems	Calculates the Number of very small items that can be used to fill open spaces
One bin items	oneBinItems	Calculates the number of items that require a new bin with high probability
Ratio height item	ratioHItems	Calculates the ratio between the min and max height of the items
Ratio width item	ratioWItems	Calculates the ratio between the min and max width of the items

Table 5.1: List of features from 5.3.1

5.4.2 Algorithms for the BPP

To create a sophisticated algorithm selection model for the BPP we need a large selection of algorithms with different performances on the problem instances. We have chosen current state-of-the-art algorithms to simulate a practical use case of the experiment. In total, we have nine different heuristics for the 2BPRG and the 2BPOG cases. In collaboration with the authors of these algorithms, we were able to get access to three of the original implementations with the associated data set. These three algorithms were used for the ARP. Unfortunately, due to copyright law and other problems we were not able to gather the other heuristics despite the comprehensive support of the authors. Nevertheless, we were able to collect detailed results of the most popular data set for each of the heuristics which we used for our experiment.

In detail we have chosen the following state-of-the-art heuristics:

- **FFIHOGJ04**
- **BFiHOGJ04**
- **CFIHOGJ04**
- **CH**
- **CHB**
- **CHBP**
- **A-B**
- **A-B.New**
- **SVC2BPRG**

As already mentioned in Section 5.1.3 the **SVC2BRG** algorithm can't be used for the 2BPOG bin packing problem case. Therefore, the algorithm is excluded from the set of these experiments.

Furthermore, we excluded the **A-B** and **A-B.New** heuristic for both cases, since the performance of the algorithms, could not compete with the other heuristics.

We ended up using 7 different heuristics, 7 for the 2BPRG and 6 for the 2BPOG. The result of these 7 algorithms on the 500 benchmark problem instances was used for our algorithm selection approach.

For the algorithm run-time prediction approach, the original implementation of the **CH**, **CHB**, and **CHBP** heuristics was used in our experiment. In addition, we extended the data sets with a greater variety of problem instances to build a more robust prediction model. The experiment was performed with 3 heuristics and 600 benchmark instances in total. The heuristics were compiled with Windows OS using Microsoft Visual Studio.

Algorithm rank scheme

For the BPP algorithms are usually compared on the basis of the number of containers needed for the particular problem instance. The first evaluations of the algorithms have shown that a comparison based only on this one criterion gives a relatively inaccurate result, due to the fact that no information of the spare space in the container is stored, nor how much time was needed to compute the solution.

In order to evaluate the process better, we use a combination of computation time and containers required to solve a particular instance. This allows us to rank the algorithms according to their performance on a single problem instance. The best algorithm for an instance x is the one which needed the fewest number of bins in the shortest amount of time. The chosen state-of-the-art heuristics are very fast and specialized for the bin packing task, therefore there are able to solve "normal" sized problem instances in a very short amount of time.

In our experiment, the most important criterion is the required number of bins to pack all items. If two or more algorithms perform equally well then the time needed to compute the solution is used to rank them.

Variations of the Algorithm Space

Another aspect we tested in our experiment is the performance of the classification algorithms with a subset of the chosen heuristics, to see if the number of possibilities affects the performance of the algorithm selection experiment. To test this aspect, we reduce the set of heuristics step by step and compare the performance to the previous complete model. The subsets were indicated by h_x where the x defines the chosen heuristics. The algorithms are chosen on their average performance rank. In each step, we eliminate the heuristic with the lowest average rank.

Algorithm Section Classifier Evaluation

The solutions of the optimization problems can immediately be used to train a machine learning prediction model to predict the best class/runtime/solution for a new unseen problem instance. However, in some cases there exist more algorithms that obtain the best solution for a certain problem instance [DZ02]. The existence of multiple algorithms that compute the best solution can occur when for example the metric to compare the results is too vague or a to generous solution space is permitted.

Of course, two algorithms can also calculate the same solution randomly or both methods are able to solve the problem optimally. In order to deal with the multiple best algorithms, we have to ensure that the metric chosen to compare the performance of the algorithm selection is capable of dealing with more the one optimal solution.

In our experiments, no instance was solved by multiple algorithms best. Since we used the rank schema proposed in Section 5.4.2 each problem instance was solved optimally by only one algorithm. We counted the number of times how often the *best* algorithm was correctly predicted for the problem instances. This metric helps us to evaluate the quality of the classification models. In detail the success metric $s(c, I, A)$ for the classifiers c, the instances I, and $c(i)$ the predicted algorithm for the current instance i is defined as:

$$s(c, I, A) = \frac{\{|i \in I : c(i) \in B^i|\}}{|I|}$$

The variable B indicates the best algorithm for each problem instances $i \in I$ which the prediction $c(i)$ is compared to. The algorithm space A defines the chosen set of algorithms for each of the experiments.

5.5 Benchmark Graphs

5.5.1 Training and Test Data

The base of our algorithm prediction and runtime prediction approach is are versatile data sets. We choose the two most popular publicly available data sets for 2-dimensional bin packing and extended them with 100 more instances for our runtime prediction approach. These data sets consist of 10 classes with 50 instances each. Each of this instances contain 10 items of the sizes $n \in 20, 40, 60, 80, 100$. The first set, consisting of classes $1 - 6$, was introduced by Berkey and Wang [BW87], while the second set of benchmark problems, classes $7 - 10$ were provided by Lodi et al. [LMV99]. The bin sizes vary for the 10 classes, ranging from 10×10 to 300×300. These intervals are also used to define the item height and width of the problem instances.

For the runtime prediction model, we expanded the data sets with 2 more classes. These 100 problem instances are characterized by a larger variety of n. The number of items ranged from 200 to 1000 items as well as a greater variety of bin sizes.

For testing and the final evaluation of our prediction models, we used a cross-validation method. In Section 5.6.5 we give a detailed overview of the cross-validation and preprocessing methods used to evaluate the best model.

5.6 Test Methodology and Experimental Setup

5.6.1 Evaluation System

All our experiments were performed on the same system. We used a PC running Windows 10 OS with an Intel i5 2,55 GHz processor with 16 GB ram. Each of the prediction models was executed as a single process/thread and real computation times were recorded for the runtime prediction procedure.

5.6.2 Algorithm Evaluation

As earlier mentioned we used 3 algorithms for our algorithm run-time prediction approach. For each of the algorithms, we applied an instance set 10 times with random seeds and computed the median time needed to pack the items. The information computed for each of the instances and all algorithms is used in the next step to build the prediction model.

5.6.3 Classifier Evaluation

For the different machine learning classier methods, a set of success criteria was needed to compare the performance of our algorithm selection and algorithm runtime approach. The most popular method is to success-rate, as introduced in 5.4.2, which counts how often the "*best*" algorithm is predicted for the set dived by the total number of problem instances. In detail, we calculate the accuracy of the classier determined as the percentage of correct classifications for the set.

Due to the fact that we also tested different subsets of the classifiers we introduce a second performance measure. If we would use the success-rate for the subsets of classier we would lose some information since each instance has only one *best* algorithm. If we exclude this algorithm, we would not be able to solve the instance with the optimal solution. Whenever an algorithm is excluded from the set of classifiers, we update the list of the best algorithms for each instance using the ranking scheme introduced in Chapter5.4.2.

For each problem instance, where the excluded algorithm was defined as the best one, we choose the second best choices with the next best rank. This process is repeated after each subsampling step. Following this strategy to ensure that the prediction model returns the best algorithms among the set of available heuristics for every single problem instance.

5.6.4 Chosen Classification Algorithms

The main body of this thesis focuses on the performance differences of the 2d BPP algorithms. Additionally, the various machine learning methods also play an important role in the quality of the solution. The task of finding the best-suited machine learning algorithms for a problem instance is the main topic of many different research projects e.g. [Aha92, Bro93, BS00, BSDC03, AS06, LBV12]. This subject, of finding the optimal machine learning algorithm, is by itself an algorithm selection problem. Several well-known methods have been used in our experiments with different configurations, however, the main focus of this thesis stays on the BPP algorithms.

To find the best-suited algorithm for each problem instance we used 6 different machine learning methods, namely:

- Naive Bayes (NB)

- C4.5 decision trees (DT)
- Random forests (RF)
- K-nearest neighborhood (kNN)
- Neural Networks (NN)
- Support vector machines (SVM)

These methods were both used for the algorithm selection task as well as the algorithm runtime prediction, except the Bayesian Networks, since not all the machine learning techniques are capable of classification and regression predictions.

5.6.5 Setup and Tools

For the evaluation itself we used that programming language *R* [2] with the caret package.

The caret package (short for Classification And REgression Training) is an extension for *R* which combines different functions for the purpose of creating predictive models. The package consists of many different *R* prediction and data management packages and combines them comfortably in one meta package. It provides a uniform interface for the functions, as well as a standardized way for parameter tuning, feature selection, and other tasks.

Caret contains tools for:

- data splitting
- pre-processing
- feature selection
- model tuning using re-sampling
- variable importance estimation

The current release version can be found on CRAN[3] and the project is hosted on github[4].

Many algorithms featured in the caret package offer a wide range of parameter tuning options, which might influence the performance and the solution of the learning phase. Consequently, we tested different configurations to find the most promising setup for each method. The following paragraphs describe the variables we used and the parameter setting for each method.

[2] R version 3.3.2 https://www.r-project.org/
[3] https://cran.r-project.org/web/packages/caret/
[4] https://github.com/topepo/caret

Methode	Compelexity parameter
DT1	0.000000
DT2	0.007702182
DT3	0.015404365
DT4	0.023106547
DT5	0.030808729
DT6	0.038510911
DT7	0.046213094
DT8	0.053915276
DT9	0.061617458
DT10	0.069319641
DT11	0.077021823
DT12	0.084724005
DT13	0.092426187
DT14	0.100128370
DT15	0.107830552
DT16	0.115532734
DT17	0.123234917
DT18	0.130937099
DT19	0.138639281
DT20	0.146341463

Table 5.2: Parameter settings for the DT classifier

Another important factor for the performance of the machine learning techniques are the different preprocessing methods. The used preprocessing methods are featured in section 5.6.6.

All of the models presented here were designed using a 10-fold cross validation, with the default parameters as introduced in Section 5.6.5.

Starting with the **Decision Trees** we used the following configurations. We modified the pruning rate of the trees and defined different levels of minimal elements per leave node. The parameters define the complexity of the tree and his extent. We used a set of twenty possible values for the complexity parameter *cp*. A detailed overview of the *cp* values used are given in the Table 5.2 for the model with all algorithms available, the items are oriented (O) and guillotine cuts are required (G). The *cp* values are slightly different for the various models, however the are always in a range of $cp = 0, \dots, 0.5$.

An extension of the Decision Trees is the **Random forest** (*RF*) method. To test the *RF* method, we experimented with a different number of trees *ntree*, which should not be set to a too small limit, to ensure that every input row gets predicted at least a few times. After the first test attempts, we set the number of trees to constant 500. The second parameter defines the number of variables randomly sampled as candidates at

Setting	mtry	ntree
RF1	2	500
RF2	3	500
RF3	5	500
RF4	7	500
RF5	9	500
RF6	11	500
RF7	13	500
RF8	15	500
RF9	17	500
RF10	19	500

Table 5.3: Parameter settings for the RF classifier

each split (*mtry*).
Even though the method can be used with arbitrarily large ranges, the computational time available to spend on modeling forms a practical upper bound on for the *RF* method. The selected values are shown in table 5.3 for the RF method.

One of the most promising machine learning technique is the **support vector machine** (SVM). For the first run, we always used the SVM start configuration optimization, integrated in the R package *caret*. After the first cold start optimization, we tested different complexity parameter values c and two kernel functions which define the performance of SVM. For the complexity parameter c we used values between $c \in \{0.1, \ldots, 12\}$. We tested a linear as well as a radial kernel for our experiment.

Usually, the decision is whether to use linear or a non-linear kernel is defined by two main factors.

- Speed: Solving the optimisation problem for a linear kernel is much faster.
- Quality: Typically, the best possible predictive performance is better for a non-linear kernel (or at least as good as the linear one).

We tested for both kernel the same values of the complexity parameter c. Table 5.4 shows the different settings for the SVM.

The **Naive Bayes** method (NB), has three options to tune a model. The fL values stand for a Laplace correction. We tested different values but decided to choose the default factor 0, i.e. no correction at the end. The *usekernel* variable was held constant at a value of TRUE. For the *adjust* tuning parameter values between $\{1, 2, 3, \ldots, 10\}$ were used.

Regarding the **Neural Networks** (NN) we used two options to tune the model. The *size* parameter defines the number of units in the hidden layer. The second parameter is

Setting	c	Kernel
SVM1	0.05	linear/radial
SVM2	0.15	linear/radial
SVM3	0.25	linear/radial
SVM4	0.50	linear/radial
SVM5	1.00	linear/radial
SVM6	2.00	linear/radial
SVM7	4.00	linear/radial
SVM8	8.00	linear/radial
SVM9	10.00	linear/radial
SVM10	12.00	linear/radial

Table 5.4: Parameter settings for the SVM classifier

Setting	size	decay
NN1	1	0.00
NN2	1	0.05
NN3	1	0.10
NN4	5	0.00
NN5	5	0.05
NN6	5	0.10
NN7	10	0.00
NN8	10	0.05
NN9	10	0.10

Table 5.5: Parameter settings for the NN classifier

the *decay* value, which is a regularization to avoid over-fitting. This method is used as a penalty for the sum of squares of the weights of the units in the hidden layer [VR13]. A detailed overview of the used values can be found in table 5.5.

For the **Generalized Linear Model** (*GLM*) we can achieve a better performance by tuning the *alpha* and *lambda* parameters. We used the tuning method included in the *caret* package to find the best settings for our purpose. The method set the tuning parameter *alpha* constant at a value of one and tested for the *lambda* parameter the values $\lambda = \{0.001, 0.002, 0.011, 0.020, 0.030, 0.040\}$.

For the last method, the **k-Nearest Neighbor** (*kNN*), we experimented with a different number of neighbors k. We experimented with mostly odd values for $k = \{2, 5, 7, 9, 11, \ldots, 41, 43\}$ for this method.

Cross Validation

Cross Validation (CV) helps, in general, to not overfit the prediction model to a selected part of the dataset (i.g. the training set). It supports the search process to find the best fitting model based on the given datasets, with the lowest possible error. Cross-validation is a way of measuring the predictive performance of the models. Without validating the prediction models, we are not able to say anything about the performance on an unseen set of data. It is possible to fit a model to almost 100% on a training set (overfitting) by simple tuning the model until every problem case is optimally solved. But if we use this model on a new and unseen set of data, it would produce poor results with a very high probability.

Using this motivation all feature selection processes were performed using a repeated k-fold cross-validation. The k-fold cross validation method splits the dataset into k-subsets $(S_1, \dots, S_k)$ (we tested differrent values for k and decided on the value $k = 10$). In the next step the model is trained an all other subsets excluding one, for example $(S_1 \cup S_2 \cup \dots \cup S_{i-1} \cup S_{i+1} \cup \dots \cup S_k)$. This approach is repeated for all subsets, computing an accuracy each step, and an overall mean accuracy estimation at the end. After all subsets had been excluded the model with the lowest estimated generalization error is chosen for further computations.

If the process of splitting the data into k subsets (folds) is repeated a number of times, and then processed as above described, we perform a so-called Repeated k-fold Cross-Validation. For our experiment we used a 10-fold cross-validation which was 3 times repeated.

5.6.6 Feature Selection and Preprocessing

In order to exploit the full potential of the machine learning algorithms, it is advisable to prepare the data. Furthermore, some machine learning algorithms require the data to be in a specific form to be applied, but not all preprocessing methods are suitable for all machine learning techniques. However, basically, the more we are able to expose the underlying structure and relationships of the data the better are the predictions of the machine learning algorithms.

So pre-processing the raw data is part of most machine learning applications. In this section, we will explain the different preprocessing methods used in this thesis in order to expose the data to the machine learning algorithms. We used two discretization techniques which were already introduced in Section 2.5.3.

Different methods were used for the feature selection and the preprocessing techniques featured in the caret package for R. The package offers several types of techniques including the centering, scale and principal component analysis (PCA) which we used for our experiment.

Scale and Center

The first two preprocessing methods are used for the standardization of the data. This requirement is very common for machine learning estimators. If the individual features do not more or less look like standard normally distributed data, it could seriously affect the performance of the machine learning algorithms.

For example, the SVM assumes (at least for the RBF kernel) that all features follow a standard normal distribution, which means that they are centered around zero with a standard deviation of one. Is this property not given, it could interrupt the learning process and therefore lead to an unexpected or bad performance.

The scale transform calculates the standard deviation for an attribute and divides each value by that standard deviation.

The center transform calculates the mean for an attribute and subtracts it from each value.

We tested the machine learning methods with each preprocessing step separately, but we also combined these two methods to standardize the dataset to a mean value of zero and a standard deviation of one.

PCA

The Principal Component Analysis (PCA) [Jol02] is a modern statistical preprocessing method that has been successfully used for face recognition, image compression and other fields of application. This method helps us to find only the important features for our prediction model. Depending on the number of features n there are 2^n possible feature subsets and the PCA supports us to find the most promising one.

The method tries to find some patterns in high dimensional data. The PCA transforms the features in so-called principal components using a variance threshold function. This technique leads to a set of features that are uncorrelated and that contain important information about the dataset which are useful for machine learning algorithms. All other features which do not exceed the threshold function are not used in the training process for the prediction model and therefore dropped.

Regarding our experiments, we first tested the performance of the machine learning methods without any preparation. To generate better performance we added the three preprocessing methods step by step and compared the individual results until the best configuration for each machine learning method was found.

CHAPTER 6

Evaluation for the Algorithm Selection

The following chapter presents the results and the evaluation of the algorithms for 2D Bin packing case, the algorithm selection approach. For our testing proposes we used the two free available data sets.

The main goal is to define the best performing algorithm for each problem instance and to analyze the performance on subsets of instances with different characteristics. The accuracy measure is used to compare the performance of the algorithms to each other. We also investigate different parameter settings for the procedures used and how they affect the quality of the solution.

After the general analysis of the heuristics, the second part of this chapter examines the algorithm selection approach for the 2BPOG and 2BPRG BPP. With the already introduced machine learning methods several prediction models were trained and tested using different parameter configurations and data-preprocessing methods. In addition, we analyzed the effects of reducing the algorithm portfolio and compare the collected results.

The last part of these chapter evaluates the trained prediction models using a test set.

6.1 Heuristic Evaluation

The first task of the heuristic evaluation analyses the behavior and results of the algorithms using the given dataset.

The dataset contains 500 problem instances with $10, 20, 40, 50$ and 100 items. To compare the performance of the algorithms first only the number of bins needed (to pack all the items) is used.

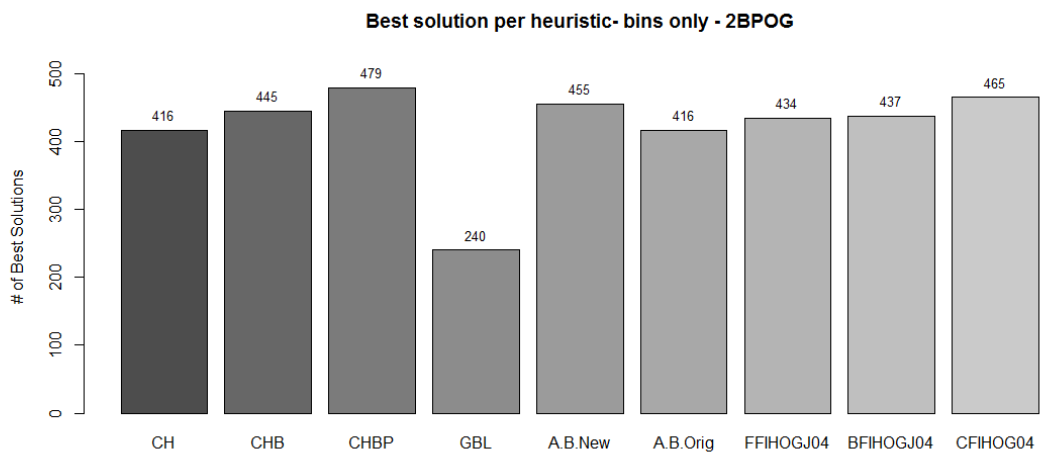


Figure 6.1: Number of instances of the test set on which the algorithms obtain the best solution for the 2BPOG case using only the bin - criteria

After this first analysis of the 2BPOG and 2BPRG case another metric is used for the following experiments. The most important performance metric is the combination of the runtime and the needed bins to pack all items of the problem instance. With our main benchmark criteria, the combination of bins and runtime, there is a possibility that more than one algorithm solves an instance best, but since we used a precise time measurement unite till the 3rd decimal place it is very unlikely. In fact, for our experiment, each instance was solved best only by one method using the main benchmark criteria.

To get a first overview of the algorithms used we rank all methods according to our metric and count how often a heuristic is identified as the best one.

6.1.1 2BPOG Metric Bins

Figure 6.1 shows a graphical representation of how often each method was able to solve an instance best. More than one algorithm solves an instance optimal using only the bins a quality metric. This is very common since the bins are a rather big unit to use as a quality metric for a problem.

As shown in figure 6.1 especially the $CHBP$ algorithms performs best for 479 of the 500 bin packing instances, followed by the $CFIHOGJ04$ with 465 and the $A.B.New$ method with 455.

To get a more representative overview of the algorithms we investigated the results on different subsets of the data.
First we separated the data in $Easy$, $Medium$ and $Hard$ sets using two lower bounds ($DevMV$ and $DevBst$). The DevMV lower bounds are based on the work of Martello

Subgroup	**Description**
EASY	Both lower bounds are reached by all algorithms
MEDIUM	One of the two lower bounds is reached by all algorithms
HARD	No algorithm was able to reach one of the two lower bounds

Table 6.1: Subsets defined by the hardness of the test set

Subgroup	**Description**
SMALL	$n \leq 40$
INTERMEDIATE	$n > 40 \wedge n \leq 80$
BIG	$n > 80$

Table 6.2: Subsets defined by the number of items to pack

and Vigo [MV98] and the DevBst are collected from the web page of the Operations Research Group of the University of Bologna based on there research [1] [Fle13]. The definitions of the subsets are given in table 6.1.

Furthermore, we separated the test set using the number of items to pack n. We created also three sets *Small*, *Intermediat* and *Big* based on the rules of table 6.2.

Figure 6.2 shows the result of the algorithms according to chosen subsets. We assumed, that the methods show different performances in dependence of the problem-size and hardness. All in all, we can say that for the 2BPOG case of the BPP no very large difference in the number of best-solved problem instances per algorithm can be found. In detail, only the GBL algorithm underperforms for all nine subgroups of the test set. Furthermore, we see that the $CHBP$ outperforms the other algorithms in almost all subgroups, except in the medium hardness classes.

6.1.2 2BPRG Metric Bins

For the second variant of the BPP, the $2BPRG$ case, we also separated the data set into nine different subsets according to the size and the hardness of the instances. The same set of rules was used as shown in Table 6.1 and 6.2. Only different lower bounds were used, the $DevDA$ lower bounds based on the work of Dell'Amico et al. [DMV02] and the DevC lower bounds are based on the research of Clautiaux et al. [CJEH07].

Keep in mind that the $2BPRG$ case contains one more algorithm than the $2BPOG$ case, the $SCV2BPRG$ method. Figure 6.3 gives an overview on the number of best solutions each single algorithm was able to achieve. Similar to the $2BPOG$ case a lot of the procedures could solve the bulk of the problem instances well. The best performance is shown by the $SCV2BPRG$ method by solving $96,2\%$ optimal according to the bin quality criteria. The second and third best results are achieved by the $A.B.NEW$ with

[1] http://www.or.deis.unibo.it

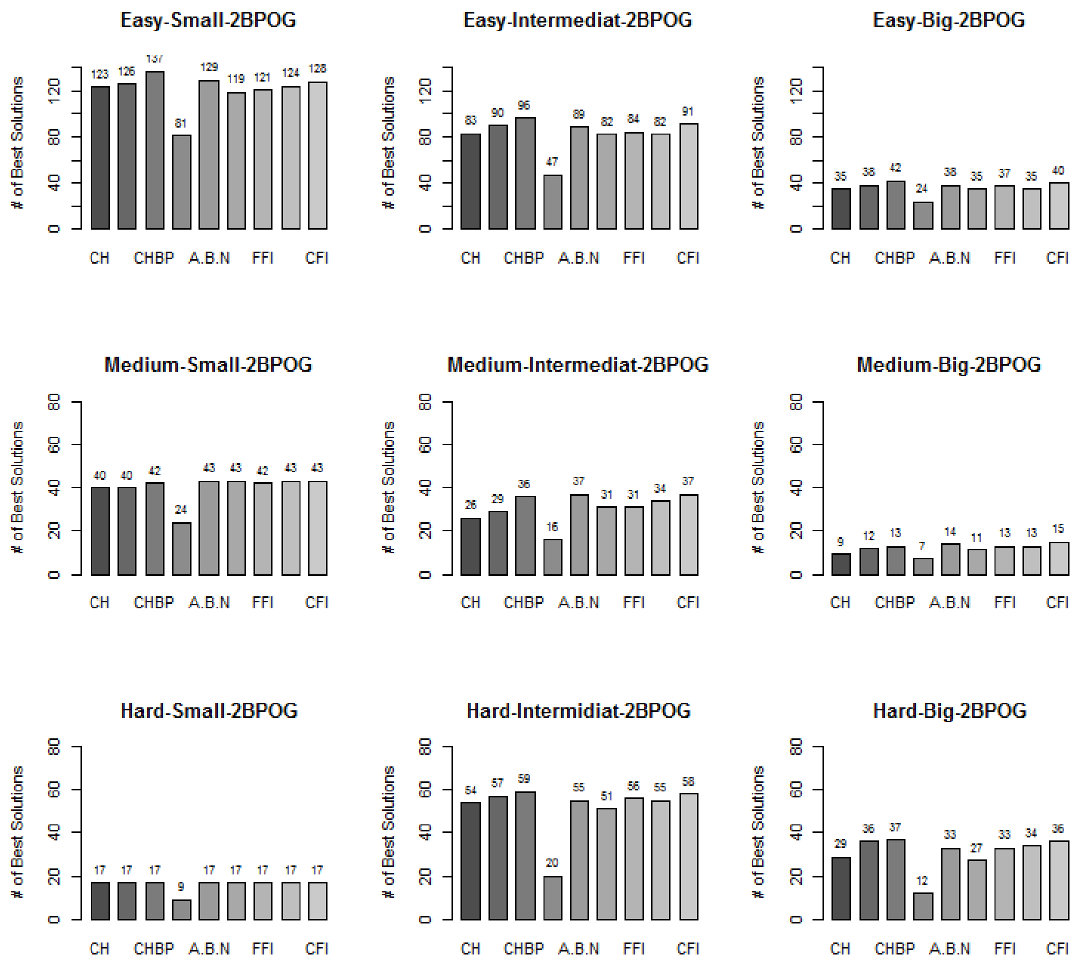


Figure 6.2: Number of instances for the 2BPOG case on which the algorithms obtain the best solution for each subgroup using only bins as criteria

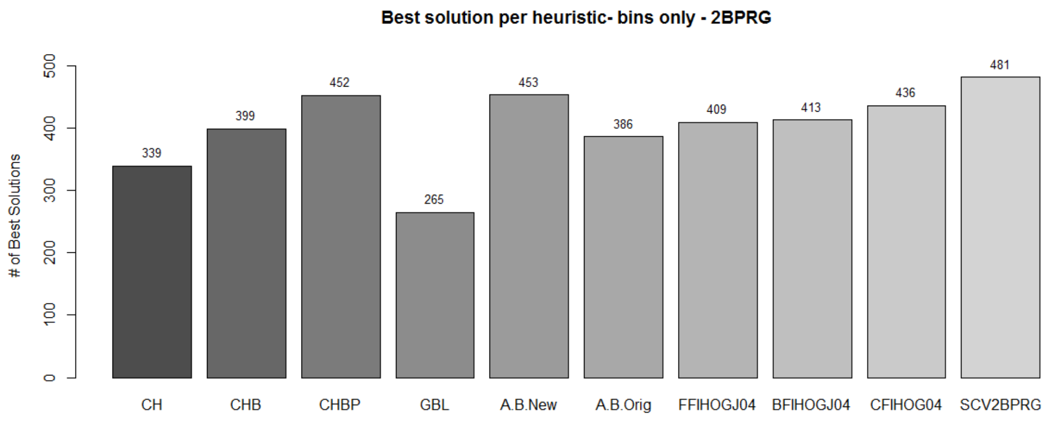


Figure 6.3: Number of instances of the test set on which the algorithms obtain the best solution for the 2BPOG case using only the bin - criteria

90, 6% as well as the *CHBP* algorithm with 90, 4%. Mentionable is that the other algorithms are less frequently able to achieve the best-known solution compared to the $2BPOG$ case. The worst performance returns once again the *GBL* algorithm with 53, 0% followed by the *CH* method with 67, 8%.

Consider the individual subsets, as shown in Figure 6.4, the $SCV2BPRG$ method is able to solve almost all subgroups at least as good as the next best algorithm. Only the *CHBP* is able to outperform the $SCV2BPRG$ method in the $Easy - Big$ subgroups and ties with them in two more subgroups. Furthermore, we are not able to recognize any trend. The strong methods solve both the easy and the hard instances better than the other procedures.

6.1.3 Conclusion - Metric Bins

The first experiments clearly show that if we use only the number of bins as a performance measure, most of the 500 instances are solved best by the at least three of the presented algorithms. These three algorithms are superior to the others. Therefore, one of the essential requirements for the algorithm selection approach is not given.

The second observation shows that we are not able to identify a trend that influences the performance of the different BPP heuristics.

These two observations lead to the conclusion that the algorithm selection approach is not helpful for the 2D BPP using only the bins as the performance measure. The goal for the following experiments is to evaluate if an advanced performance measure is suitable for the purpose of applying the algorithm selection approach for the BPP.

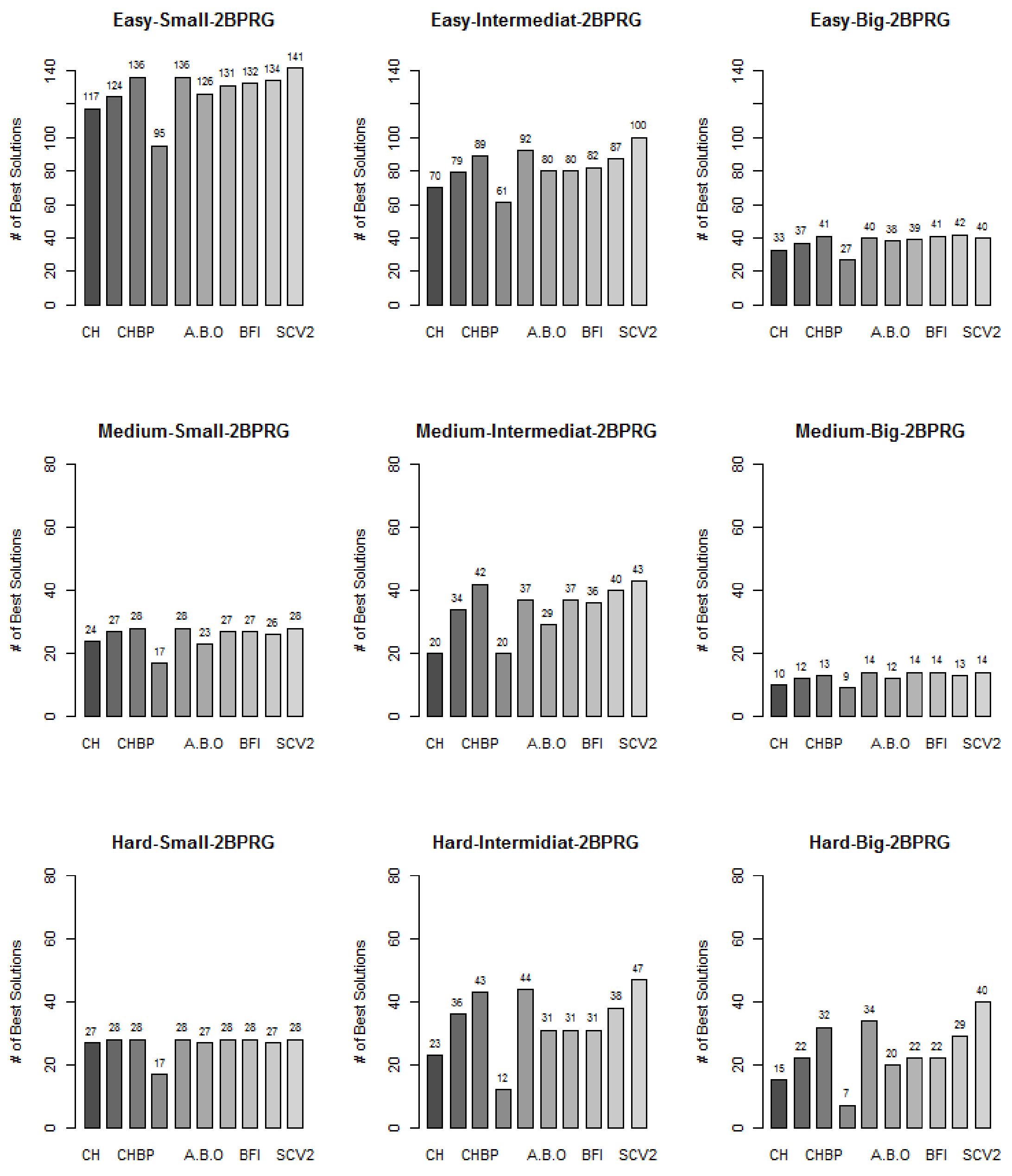


Figure 6.4: Number of instances for the 2BPOG case on which the algorithms obtain the best solution for each subgroup using only bins as criteria

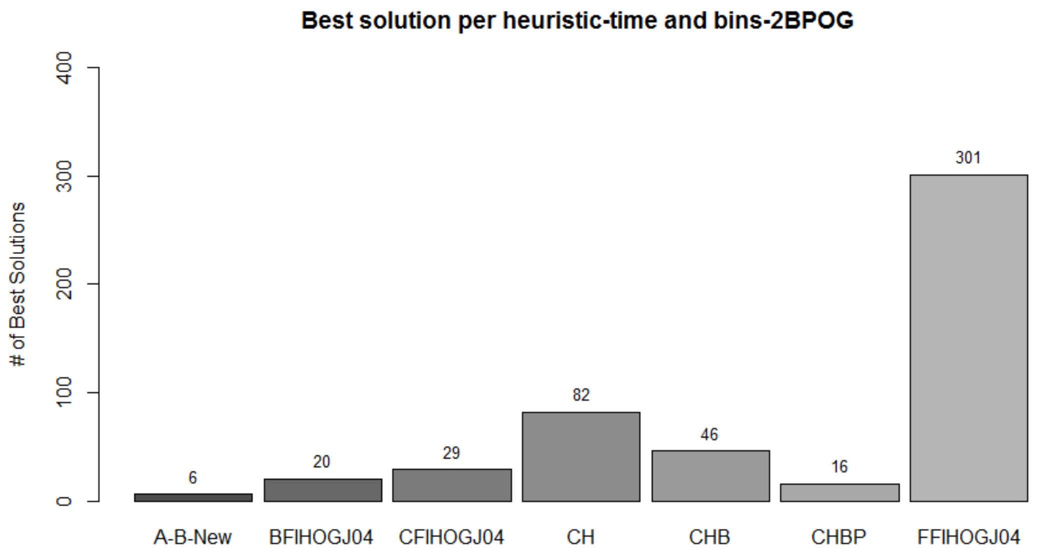


Figure 6.5: Number of instances of the test set on which the algorithms obtain the best solution for the 2BPOG case using the main criteria

6.1.4 2BPOG Metric Bins-Time

The second phase of the experiment uses the main benchmark criteria, bins needed to pack all items in combination with the time needed to compute the result. The 500 benchmark instances were also used for the second part of this experiment.

Figure 6.5 gives a first overview of the $2BPOG$ case of the BPP. The $FFIHOGJ04$ solves clearly the most problem instances best using the main benchmark criteria. This method is in 60.2% better than the other BPP heuristics used.
This value serves as a benchmark for our further experiments for the $2BPOG$ case. This means without an algorithm selection approach and any further information, we could solve 60.2% of the benchmark instances optimally by simply picking the $FFIHOGJ04$ for all instances. Therefore we will try to improve this value using machine learning techniques introduced earlier. Mentionable is also that the GBL and the $A.B.Orig$ are not able to solve one single instance best using the main criteria. Therefore, we exclude this two heuristics from future experiments in this thesis.

Figure 6.6 and Figure 6.7 give a more detailed overview of the performance of the heuristics using the main benchmark criteria and the earlier introduced subgroups as well as only the hardness subgroups. The graphics show that especially for the small and medium instances the CH algorithms and its variations are able to solve some instances better than the leading $FFIHOGJ04$ algorithm. The reasons for this could be that the

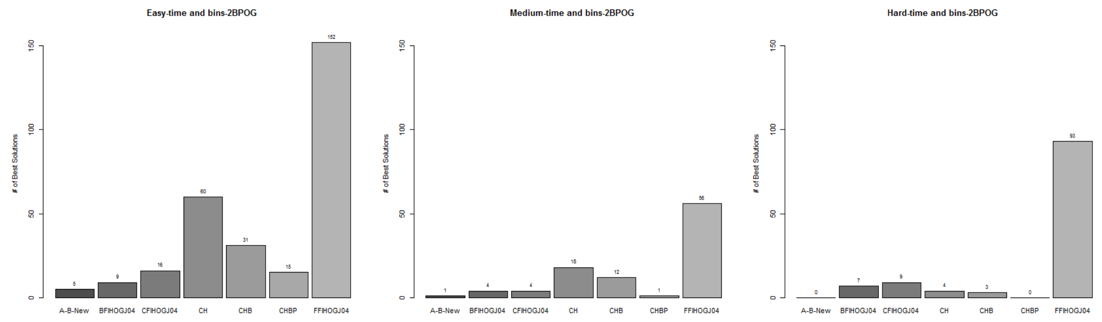


Figure 6.6: Number of instances for the 2BPOG case on which the algorithms obtain the best solution for the hardness subgroups using the main criteria

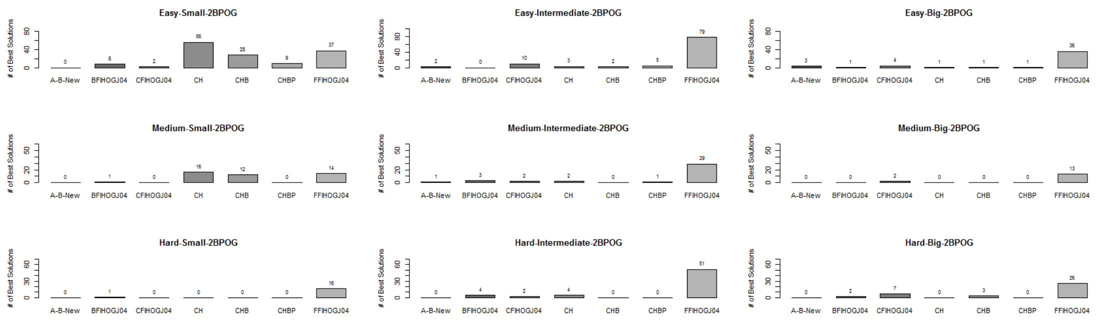


Figure 6.7: Number of instances for the 2BPOG case on which the algorithms obtain the best solution for the subgroups using the main criteria

CH, CHB and $CHBP$ heuristics are faster in placing the items into the bins. Further investigations are needed to analyze why these algorithms are suited for this subgroup of data.

The ranks of the different subsets are summarized in Figure 6.8 show the performance of each algorithm. The areas of interest are especially the subgroups where the algorithms were able to beat the mean ranks of the $FFIHOGJ04$ algorithm. This is the case in the $Easy - Small$ and in the $Medium - Small$ subgroups. On this sets, we can clearly see that the mean rank of the $CHBP$ and CHB algorithms perform better than the other methods.

6.1.5 2BPRG Metric Bins-Time

For the second 2DBPP case we separated the 500 benchmark instances the same way as in the previous experiment. The results obtained using the main benchmark criterion are shown in Figure 6.9. Similar to the 2BPOG chase, the $FFIHOGJ04$ algorithm is able to solve the most benchmark instances optimal with 59%. Once again this value

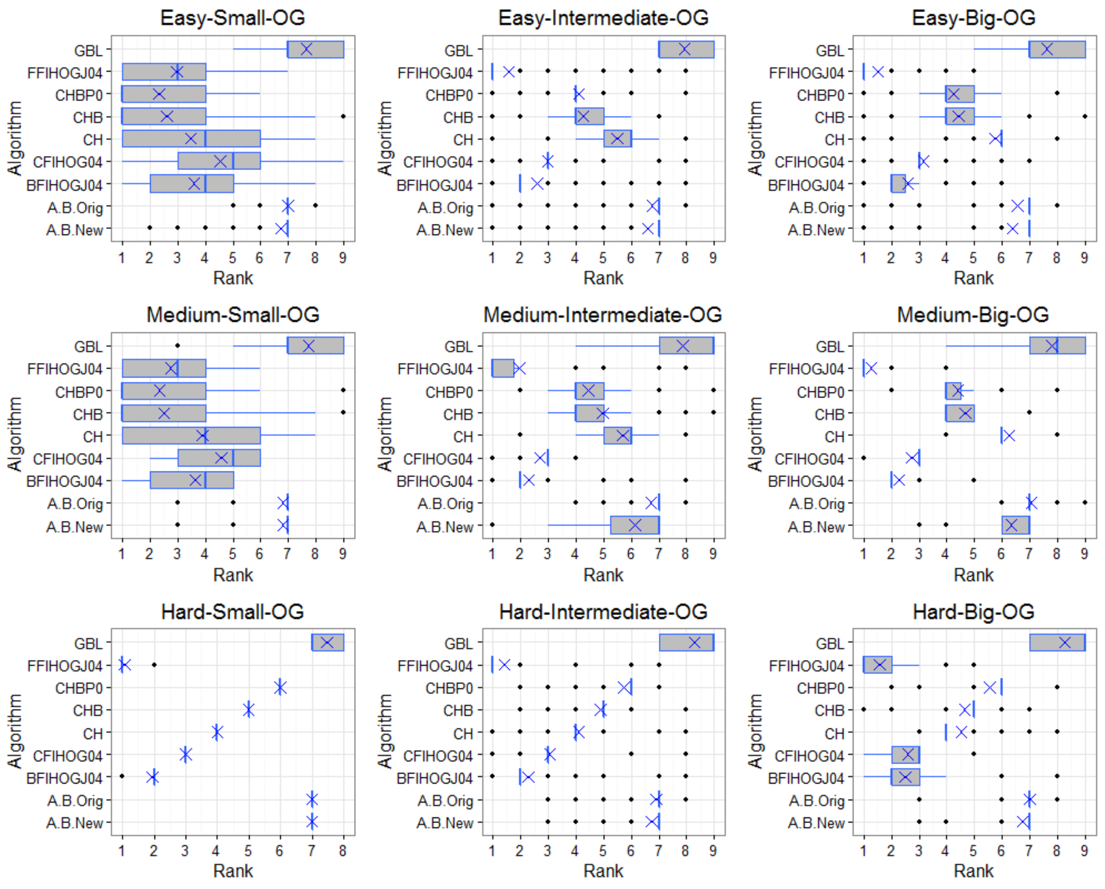


Figure 6.8: Boxplot diagram showing the ranking of the algorithms on the different subgroups of the training set - 2BPOG

can be used as a benchmark value for further research.

Figure 6.10 and Figure 6.11 give some interesting insights of the performance of the algorithms. We can see that the $FFIHOGJ04$ algorithm is able to solve the most benchmark instances optimal in all subgroups. However, especially in the $easy - small$ and in the $hard - big$ subgroups, we can see that the gap between the $FFIHOGJ04$ and all other algorithms is relatively small. Furthermore, we can see that the CH, CHB and $CHBP$ perform best for the rather small instances.

For a more insight illustration, Figure 6.12 shows the ranking results for the 500 benchmark instances. As expected the mean and median rank of the $FFIHOGJ04$ algorithm is better than the other alternatives. However also the CH , CHB , $CHBP$ and the $SVC2BRG$ algorithms show good performance for a great part of the benchmark instances. Another observation worth mentioning is that the $A.B.Old$, the $A.B.New$ and the GBL algorithms are not able to solve a single instance as good as the other algorithms. Therefore, these procedures are no longer taken into account for further analysis of the 2BPRG case.

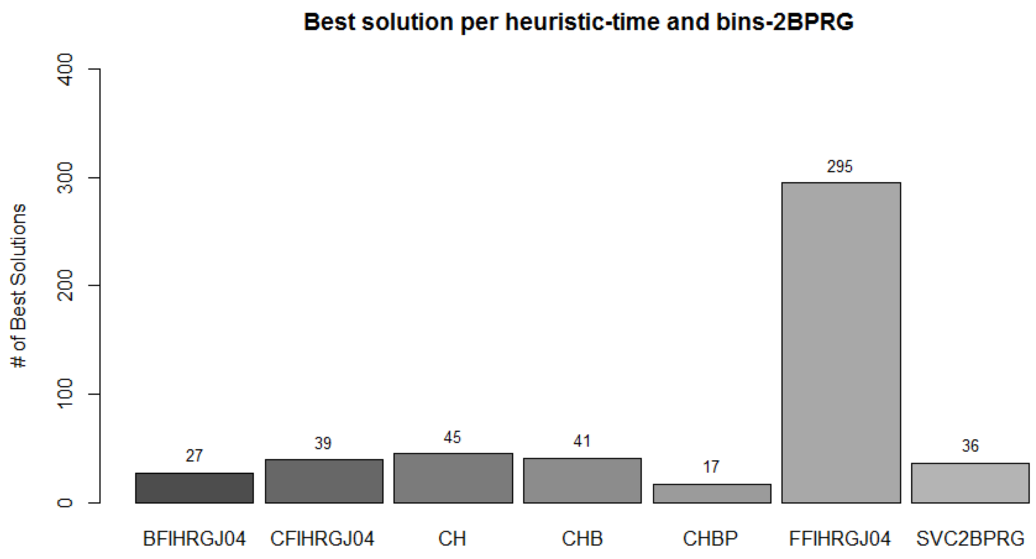


Figure 6.9: Number of instances of the test set on which the algorithms obtain the best solution for the 2BPRG case using the main criteria

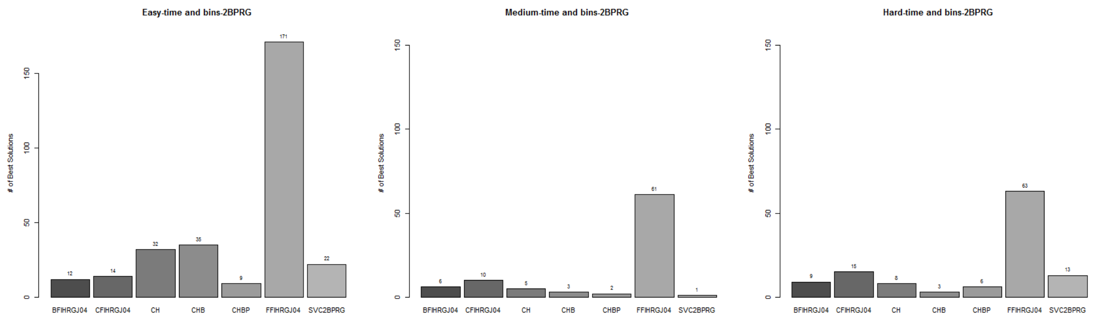


Figure 6.10: Number of instances for the 2BPRG case on which the algorithms obtain the best solution for the hardness subgroups using the main criteria

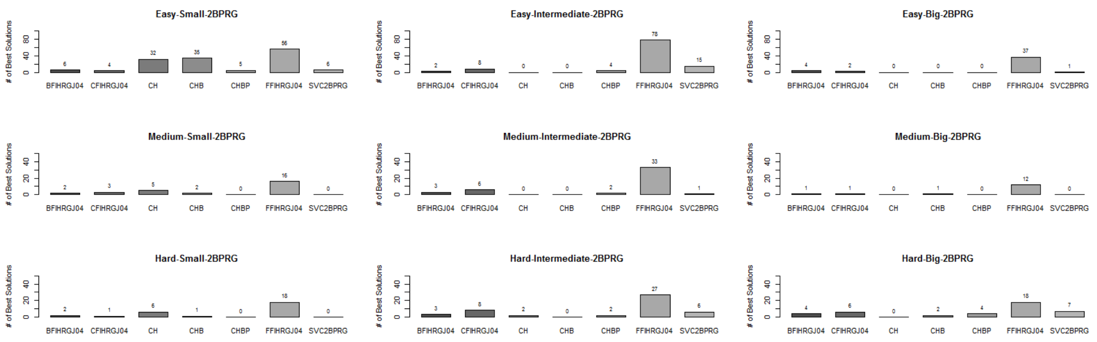


Figure 6.11: Number of instances for the 2BPRG case on which the algorithms obtain the best solution for the subgroups using the main criteria

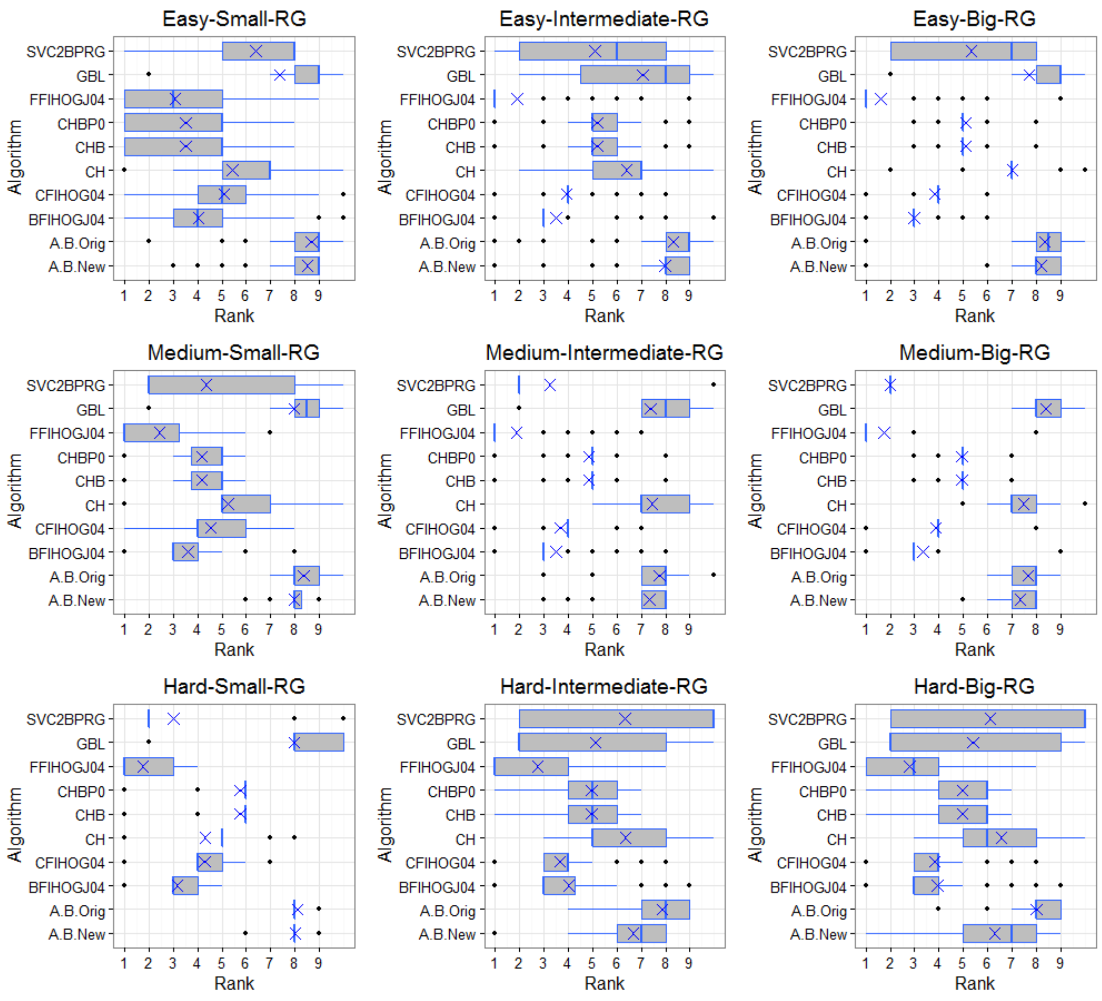


Figure 6.12: Boxplot diagram showing the ranking of the algorithms on the different subgroups of the training set- 2BPRG

6.1.6 Conclusion - Metric Bins- Time

Our second experiment shows that the results of the individual algorithms clearly provide heterogeneous solutions for the benchmark set using the main criteria for both cases. The first evaluation shows that one algorithm is able to solve a major part of the benchmark instances. However, the algorithm is not able to dominate the other procedures since two or more heuristics perform better than the rest for some subclasses of the instances. Furthermore, many algorithms show a better performance on some subgroups than on the others, which could indicate some hidden structure that influences the performance of the algorithms.

Another observation is that using the algorithm selection approach, a high proportion of the problem instances could be optimally solved as with a single algorithm solution.
The goal for the following experiments is to evaluate for both BPP cases if the simple extracted problem instance features are able to train a prediction model to choose the best-suited algorithm for each problem instance.

6.2 Solvers based on Algorithm Selection

6.2.1 Terminology

For our experiment, we used some specific terminology and nomenclature to simplify our attempts as follows.

As mentioned earlier in this thesis, we tested 500 bin packing instances from the well-known benchmark instances. We created a test and a training set by splitting this 500 instances. The training set contains 400 randomly picked problem instances and the test set all others. The additional instances for the algorithm runtime prediction approach were also divided using the same schema 80% training and 20% test set. For the creation of the prediction models we excursively used the training set and therefore the test set was only used for the model evaluation proposes.

In our experiments, we used different instance sets, preparation steps and featured subsets. We created a naming schema in order to distinguish the individual models better. The sets were labeled as $MET - Time - Disc - Case - Set$ where MET stands for one of the machine learning methods $\{NN, kNN, GLM, SVM, RPART, RF, NB\}$, the attribute $Time$ is optional and only defines the algorithm runtime prediction models. The attribute

$Disc \in \{No - Discretization, NB, TOP\}$ describes the applied discretization method, the value $Set \in \{All, Set6, Set5, Set4\}$ stands for the number of used heuristics and the value $Case \in \{OG, RG\}$ describes the BPP case as already explained earlier.

For example, the $kNN - MLP - OG - All$ stands for the kNN k-nearest-neighbourhood algorithm selection classifier, with the MLP discretization method. Furthermore, the OG value stands for the 2BPOG case of the BPP and the All variable hints that all available algorithms are used for this model.

6.3 Classifier Parameter Evaluation

The next section analyzes the varying parameter settings for the different machine learning classifier. We evaluate the different tuning parameters using the accuracy of the prediction models.

k-Nearest Neighbour

Starting with the kNN machine learning method our experiment includes different parameters for the value k. Figure 6.13 shows the results for the 2BPOG case. The results show that large k have almost no effects while using discretized features. Only for neighborhoods between 2 and 15, we can identify slight improvements in accuracy. For the no-discretized data, we are able to see a bigger effect when changing the size of the neighborhoods.

The results for the 2BPRG use case of the BPP are shown in Figure 6.14. Here can especially be noted that for a large k the accuracy decreases using no discretization technique. The best results can be achieved with a value between 2 and 6. Similar results can be seen using the TOP preprocessing method. Only the MLP method seems to be able to achieve better results with a larger neighborhood.

For the 2BPOG case, a big neighborhood seems to lead to better results and almost never to a drop of the accuracy, while for the other BPP case the exact opposite seems to lead to the best possible prediction model.

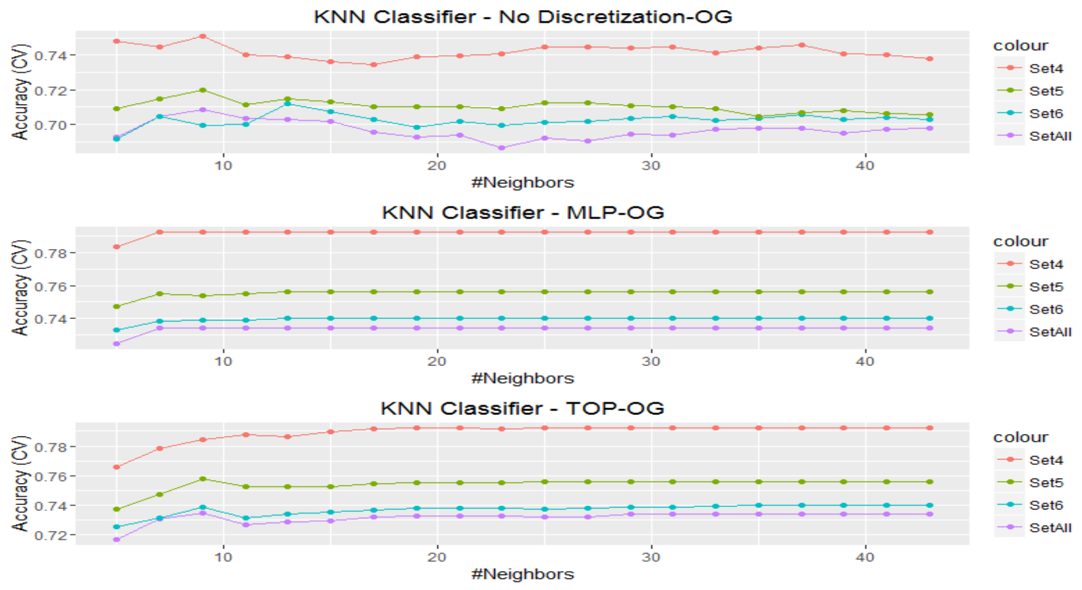


Figure 6.13: Accuracy of the KNN classifier on the training set using different parameter settings for the 2BPOG case. The results are grouped using the different dicretization methods and the algorithm sets. The dot represent the mean value using cross validation for each parameter setting.

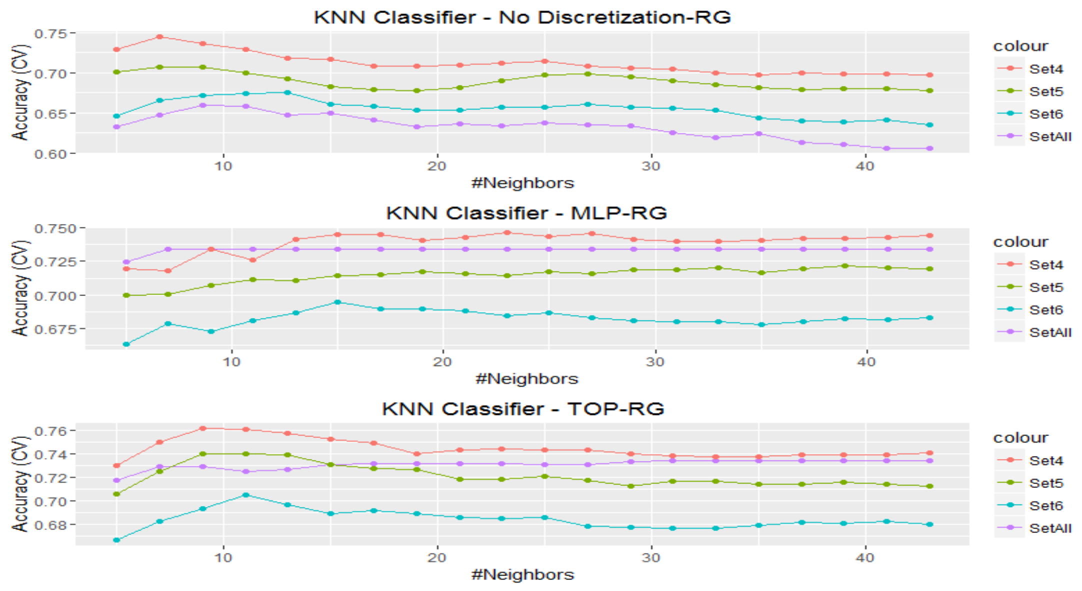


Figure 6.14: Accuracy of the KNN classifier on the training set using different parameter settings for the 2BPRG case. The results are grouped using the different dicretization methods and the algorithm sets. The dot represent the mean value using cross validation for each parameter setting.

Naive Bayes

Figure 6.15 shows the results for the varying Bandwidth adjustment for the NB algorithm for the 2BPOG case. It is clearly shown that for all three models the accuracy drops for an adjustment value. A small increase of the accuracy can be seen for the no discretization and the TOP preprocessed models. The best results can be achieved for all models with an adjustment value around two.

The results for the 2BPRG case as shown in 6.16. With the increase of the value of the performance of the model is stagnating for all three preprocessing models of the NB method. However, as in the 2BPOG case of the BPP, we are able to see an increase of the accuracy for the no discretization and the TOP preprocessed models with a higher adjust value before the model falters.

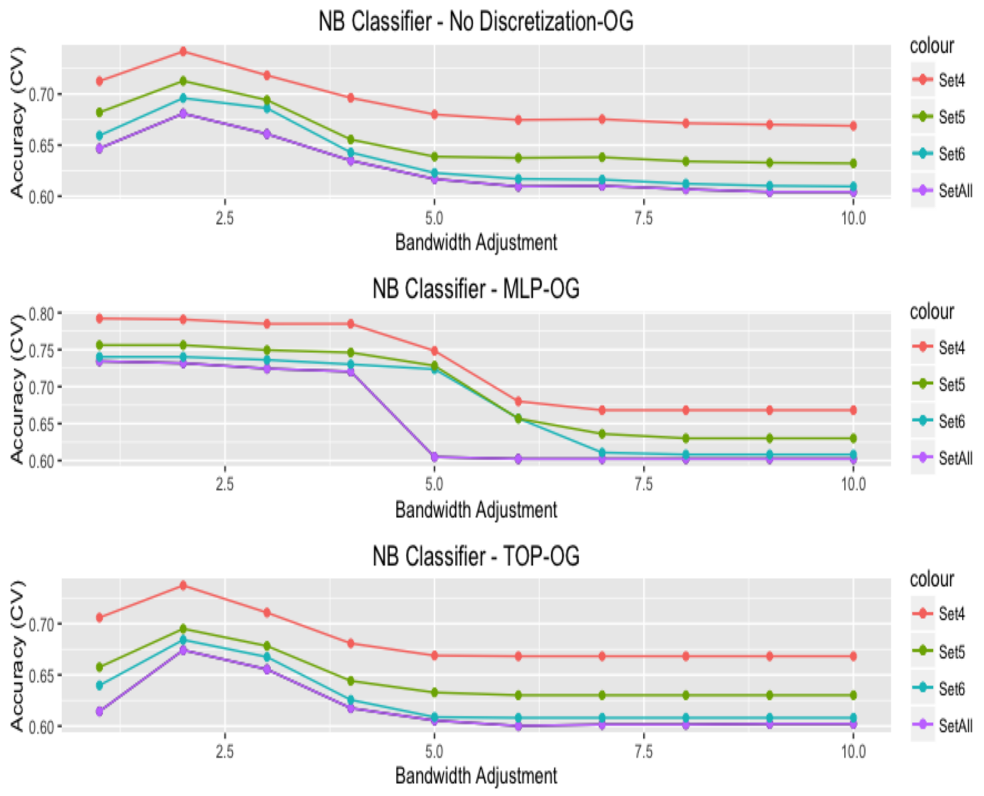


Figure 6.15: Accuracy of the NB classifier on the training set using different parameter settings for the 2BPOG case. The results are grouped using the different dicretization methods and the algorithm sets. The dot represent the mean value using cross validation for each parameter setting.

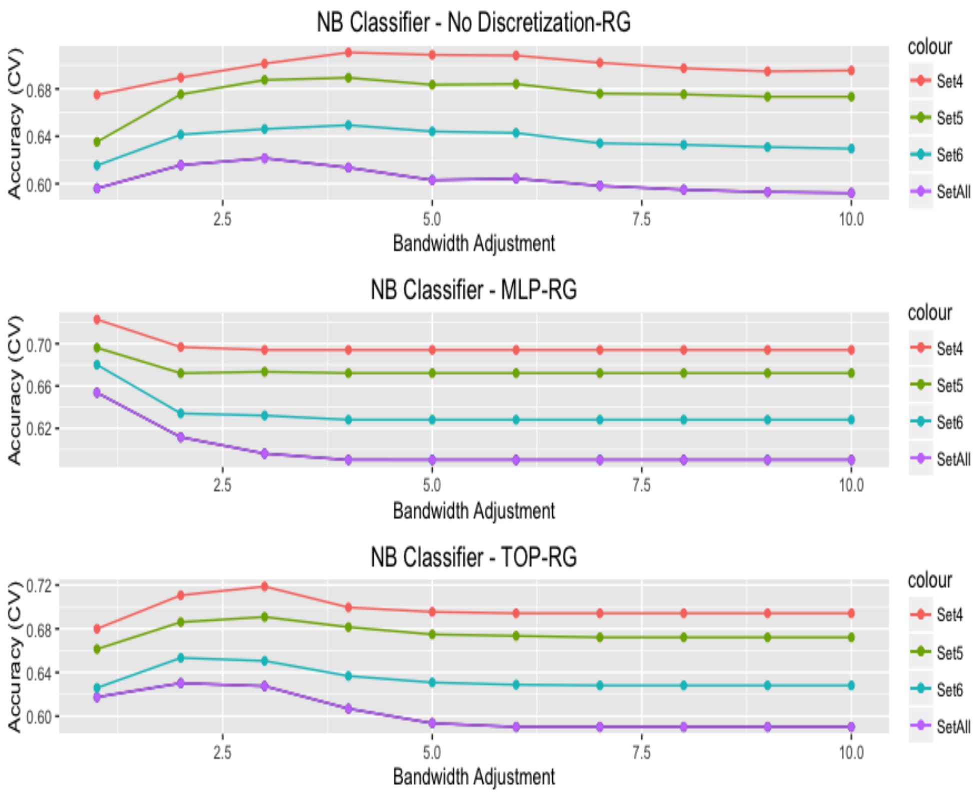


Figure 6.16: Accuracy of the NB classifier on the training set using different parameter settings for the 2BPRG case. The results are grouped using the different dicretization methods and the algorithm sets. The dot represent the mean value using cross validation for each parameter setting.

Decision Tree

For our decision trees (DT) classifier, we tested complexity parameters as shown in the Figures 6.17 and 6.18. For the 2BPOG case, we see that for the non-discretized data, the complexity parameter has a significant impact on the performance of the classification models. All three models are characterized by the fact that the higher the complexity parameter is chosen, the worse the performance gets. However, the value seems to have the least influence on the MLP model. Furthermore, we are able to recognize that the overall performance of the methods using some discretization methods is on average up to 5% more accurate.

In Figure 6.18 we are also able to see that the performance of the methods using discretization methods lead to better results. The best model can be achieved with the TOP method using a rather small complexity parameter around 0 and 0.05.

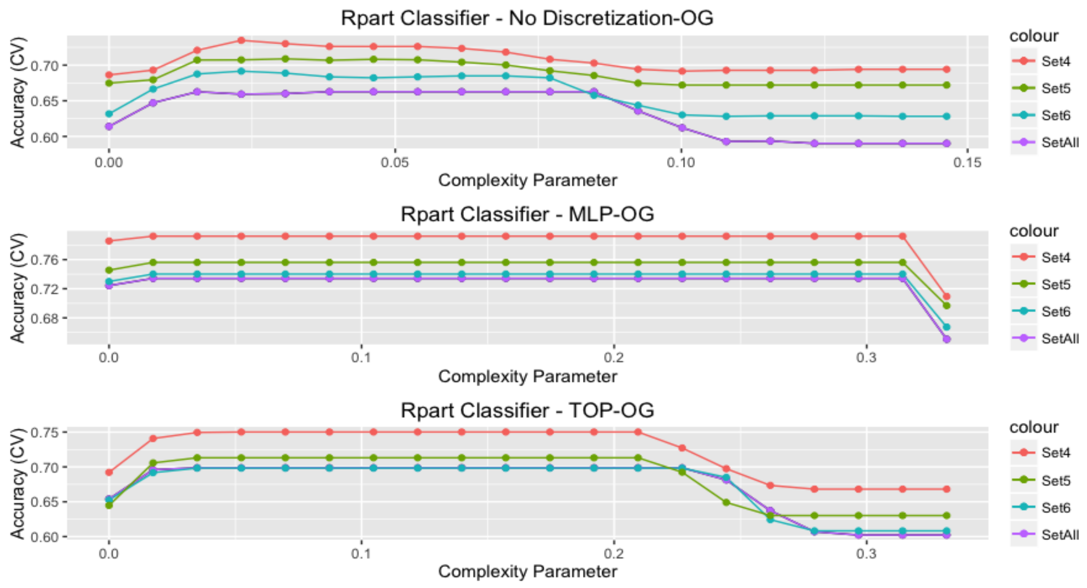


Figure 6.17: Accuracy of the decision trees classifier on the training set using different parameter settings for the 2BPOG case. The results are grouped using the different dicretization methods and the algorithm sets. The dot represent the mean value using cross validation for each parameter setting.

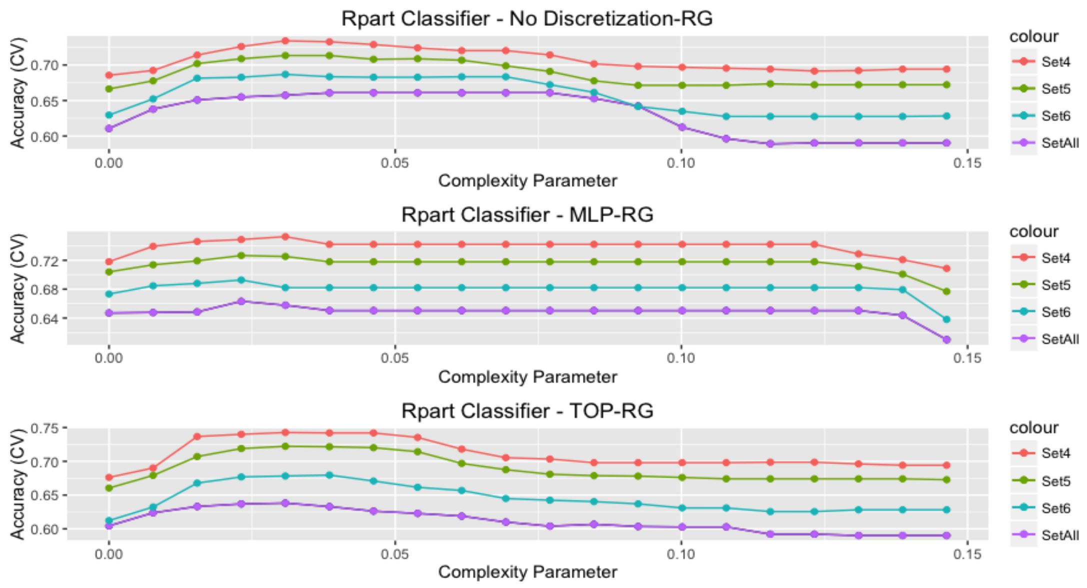


Figure 6.18: Accuracy of the decision trees classifier on the training set using different parameter settings for the 2BPRG case. The results are grouped using the different dicretization methods and the algorithm sets. The dot represent the mean value using cross validation for each parameter setting.

Random Forest

For the random forest (*RF*) classifier, we tested the different numbers of variables for each split. The results for 2BPOG are shown in Figure 6.19. The outcome shows that the three models perform almost similar. The best model is once again the one using the *TOP* discretization method.

For the 2BPRG case gives Figure 6.20 an overview of the accuracy using the different settings. Here the classifier training with the non-discretized data could achieve the best results. Furthermore, the number of randomly selected predictors seem hardly to have an influence.

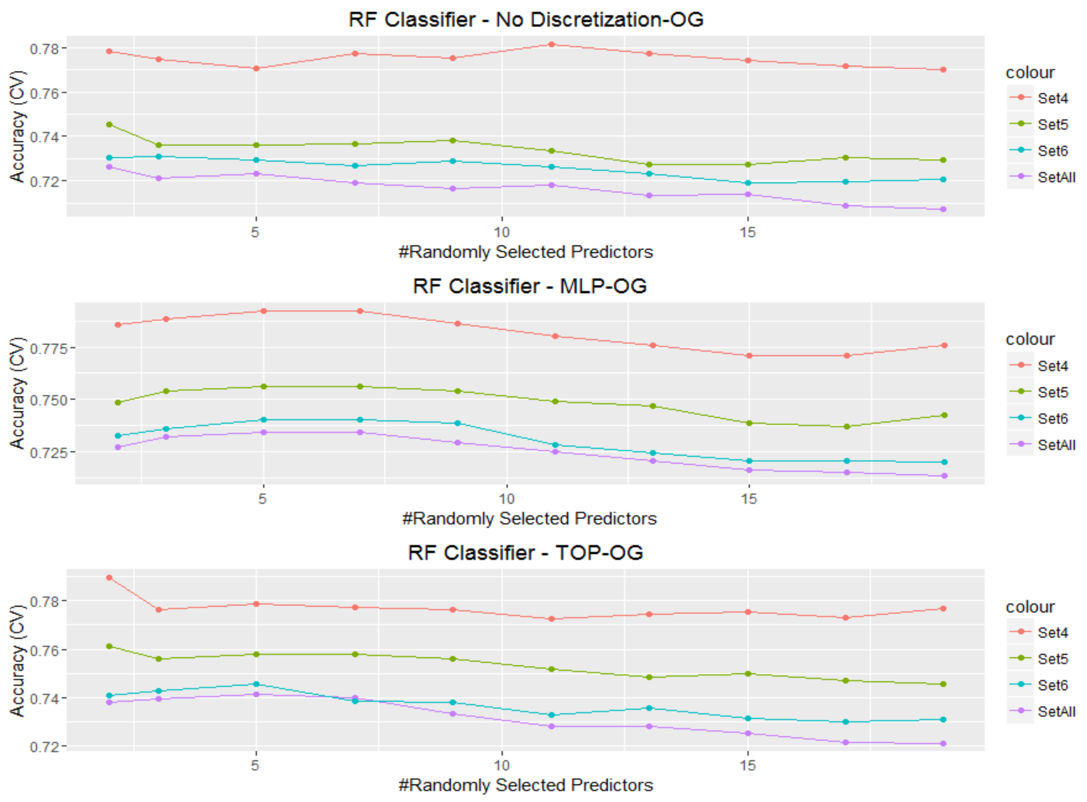


Figure 6.19: Accuracy of the random forest classifier on the training set using different parameter settings for the 2BPOG case. The results are grouped using the different dicretization methods and the algorithm sets. The dot represent the mean value using cross validation for each parameter setting.

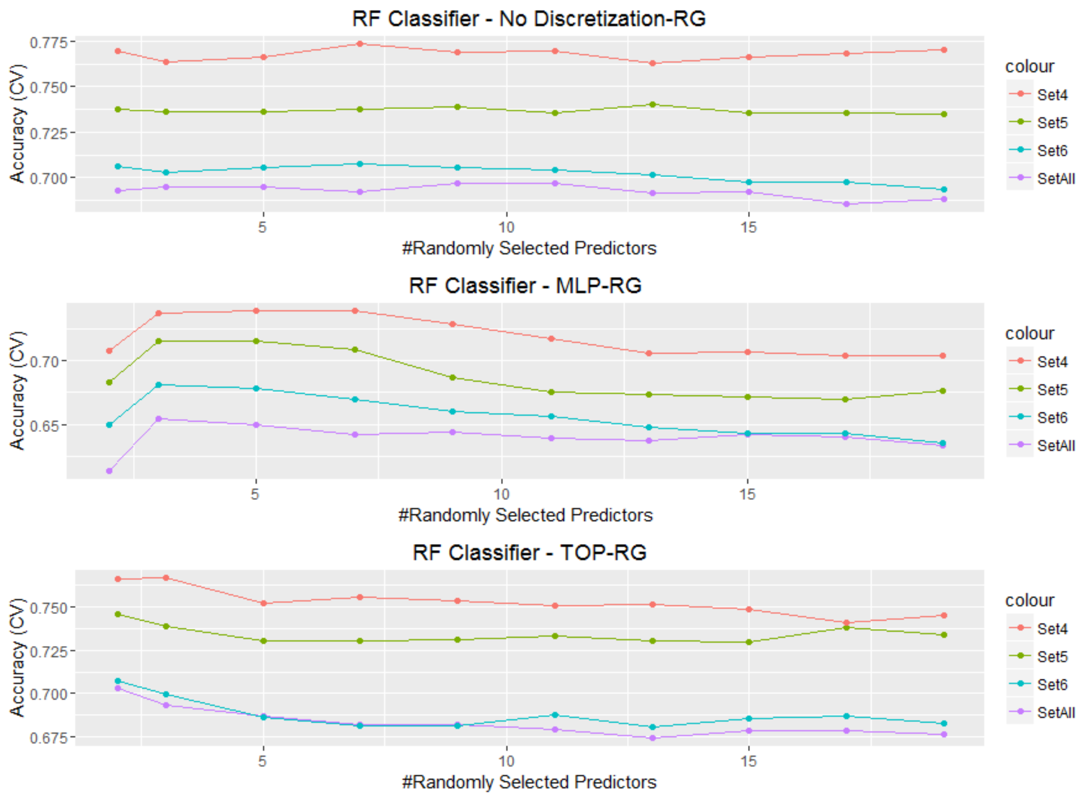


Figure 6.20: Accuracy of the random forest classifier on the training set using different parameter settings for the 2BPRG case. The results are grouped using the different dicretization methods and the algorithm sets. The dot represent the mean value using cross validation for each parameter setting.

Support Vector Machines

Concerning the Support Vector Machines (SVM), we tested two different kernels (linear and radial) and different complexity parameters as shown in Table 5.4. First, we analyze the results for the 2BPOG case as shown in Figure 6.21 and 6.22. For the linear SVM, the complexity parameters only show a small influence on the performance of our experiment. Further, the SVM using a radial kernel is able to outperform almost all three models using the linear one.

As shown in Figure 6.23 and 6.24 for the 2BPRG case of the BPP we are able to recognize that all models are positively influenced by an increased complexity parameter. The only notable exception is the model using the MLP discretization method with the radial kernel. The performance of the models using different kernels is almost identical.

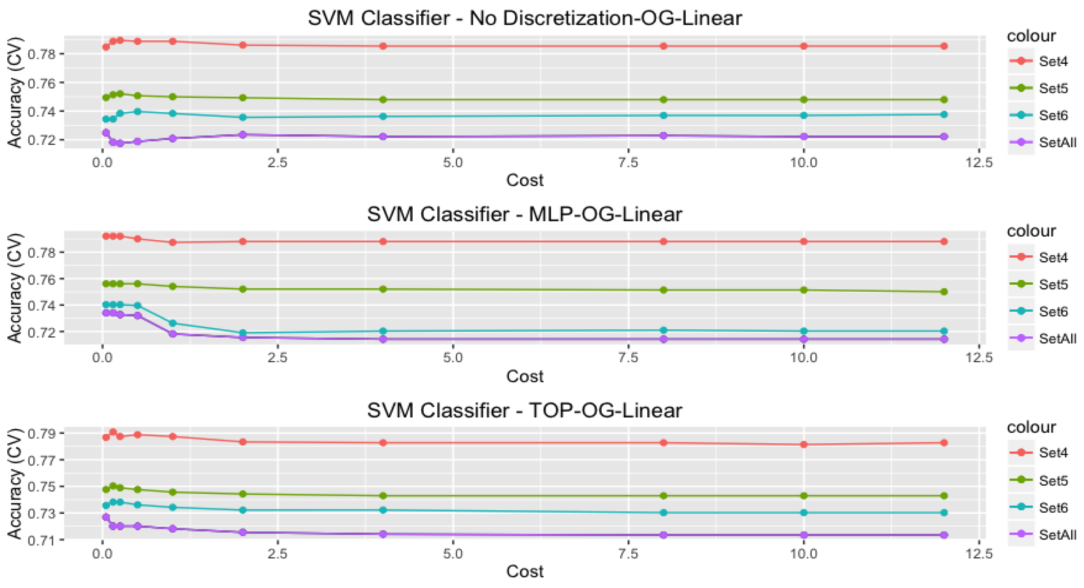


Figure 6.21: Accuracy of the SVM classifier with a linear kernel on the training set using different parameter settings for the 2BPOG case. The results are grouped using the different dicretization methods and the algorithm sets. The dot represent the mean value using cross validation for each parameter setting.

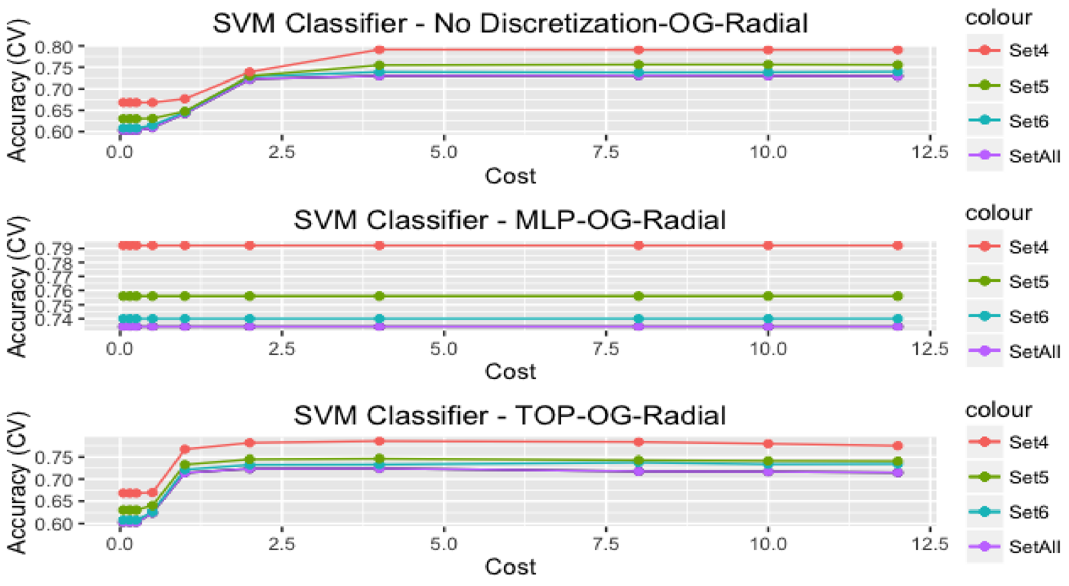


Figure 6.22: Accuracy of the SVM classifier with a radial kernel on the training set using different parameter settings for the 2BPOG case. The results are grouped using the different dicretization methods and the algorithm sets. The dot represent the mean value using cross validation for each parameter setting.

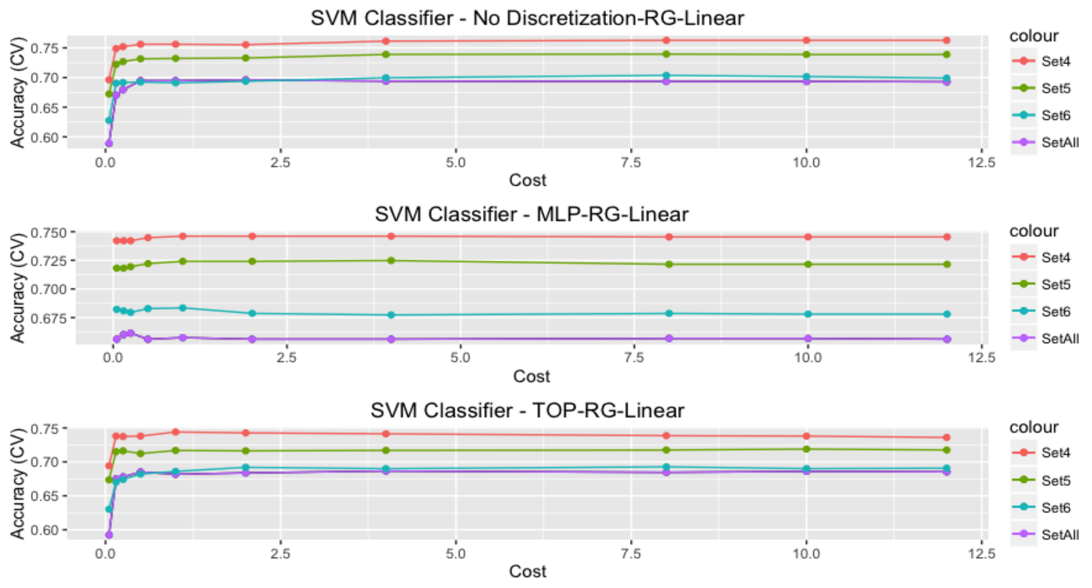


Figure 6.23: Accuracy of the SVM classifier with a linear kernel on the training set using different parameter settings for the 2BPOG case. The results are grouped using the different dicretization methods and the algorithm sets. The dot represent the mean value using cross validation for each parameter setting.

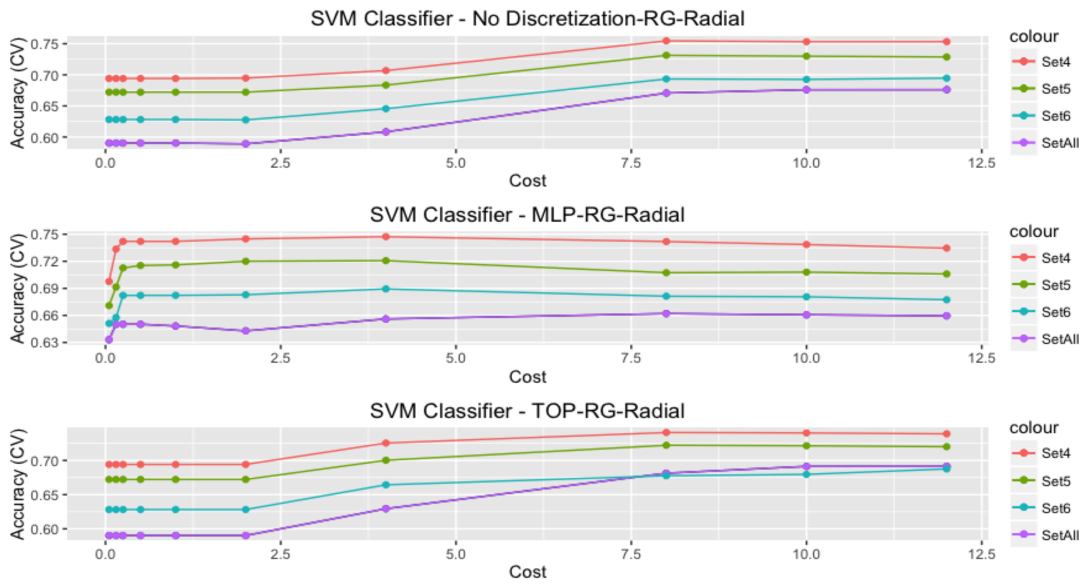


Figure 6.24: Accuracy of the SVM classifier with a radial kernel on the training set using different parameter settings for the 2BPRG case. The results are grouped using the different dicretization methods and the algorithm sets. The dot represent the mean value using cross validation for each parameter setting.

Neural Networks

As mentioned earlier, we tested three different numbers of hidden units $(1, 5, 10)$ and different values for the weight decay function. The Figures 6.25, 6.26 and 6.27 show the results for the 2BPOG case. The best accuracy can be achieved with one hidden layer and a relatively high value for the weight decay function. The NN model with one hidden layer profits most from a higher value of the weight decay function.

For the 2BPRG case, we are able to achieve the best accuracy using a one or five hidden layer model. Basically, for both cases, the models with the ten hidden layers are not able to achieve any good results. That could be because the models are rather complicated and need a lot of time and resources to be computed. More precise results can be taken from the Figures 6.28, 6.29 and 6.30

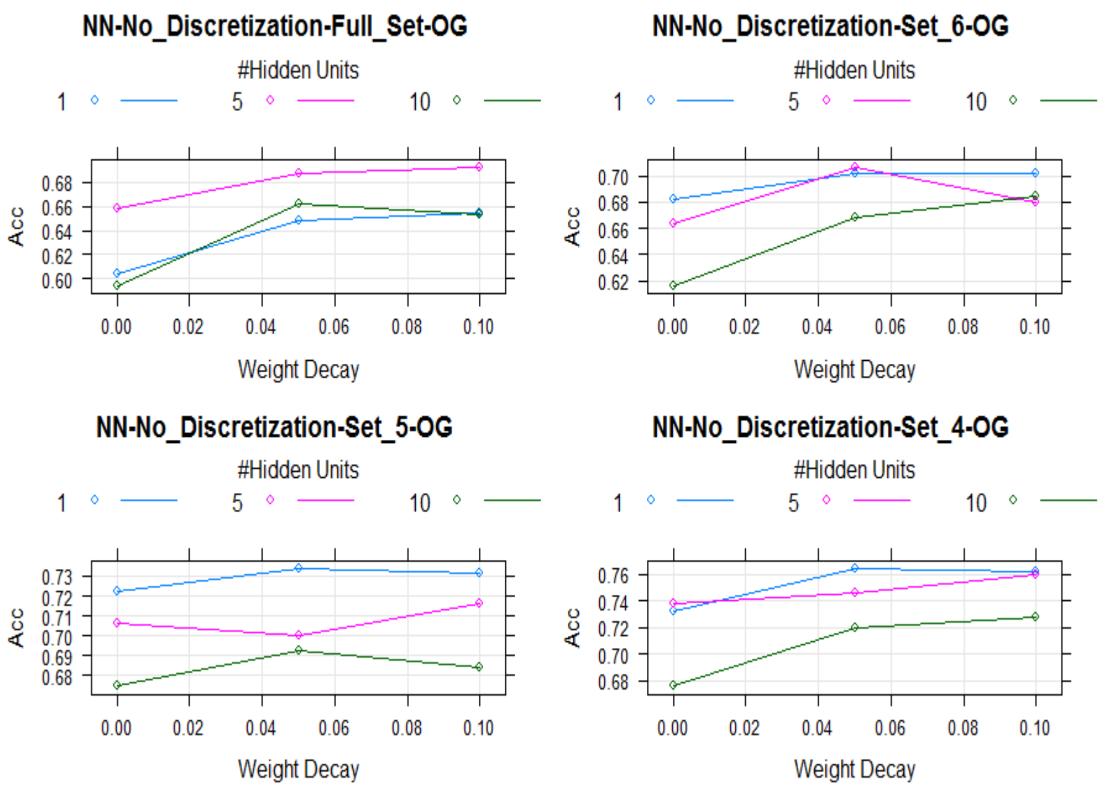


Figure 6.25: Accuracy of the neural network classifier on the training set using different parameter settings for the 2BPOG case. The dot represent the mean value using cross validation for each parameter setting.

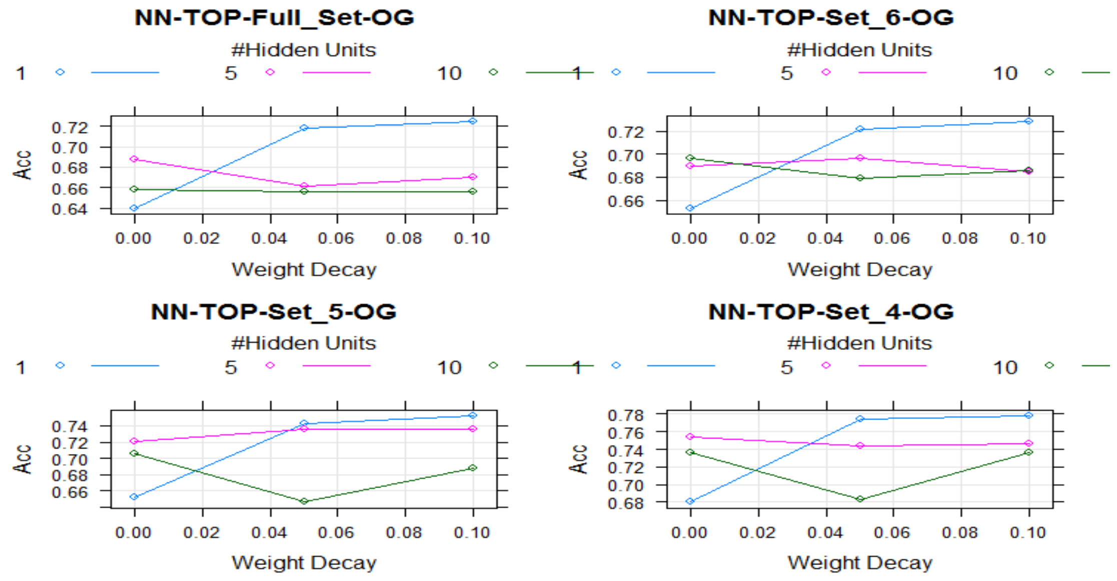


Figure 6.26: Accuracy of the random forest classifier with the TOP dicretization on the training set using different parameter settings for the 2BPOG case. The dot represent the mean value using cross validation for each parameter setting.

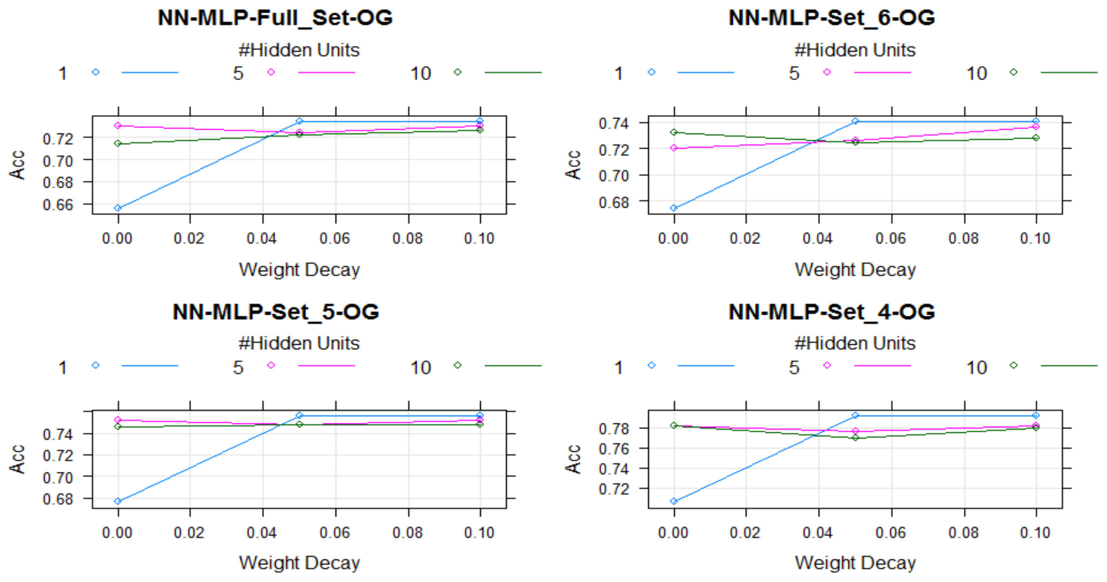


Figure 6.27: Accuracy of the neural network classifier with the MPL dicretization on the training set using different parameter settings for the 2BPOG case. The dot represent the mean value using cross validation for each parameter setting.

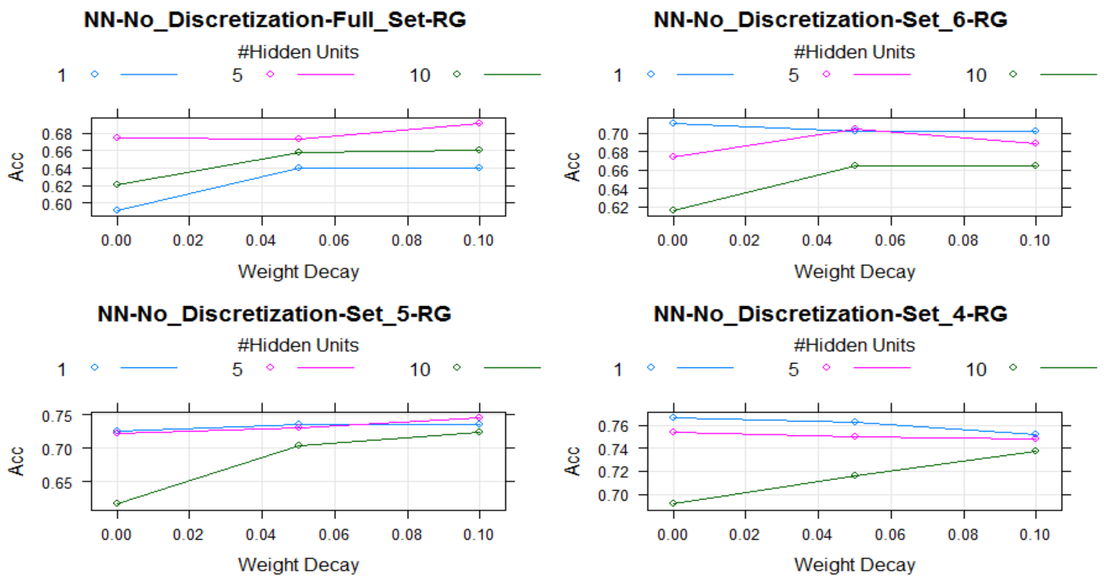


Figure 6.28: Accuracy of the neural network classifier on the training set using different parameter settings for the 2BPRG case. The dot represent the mean value using cross validation for each parameter setting.

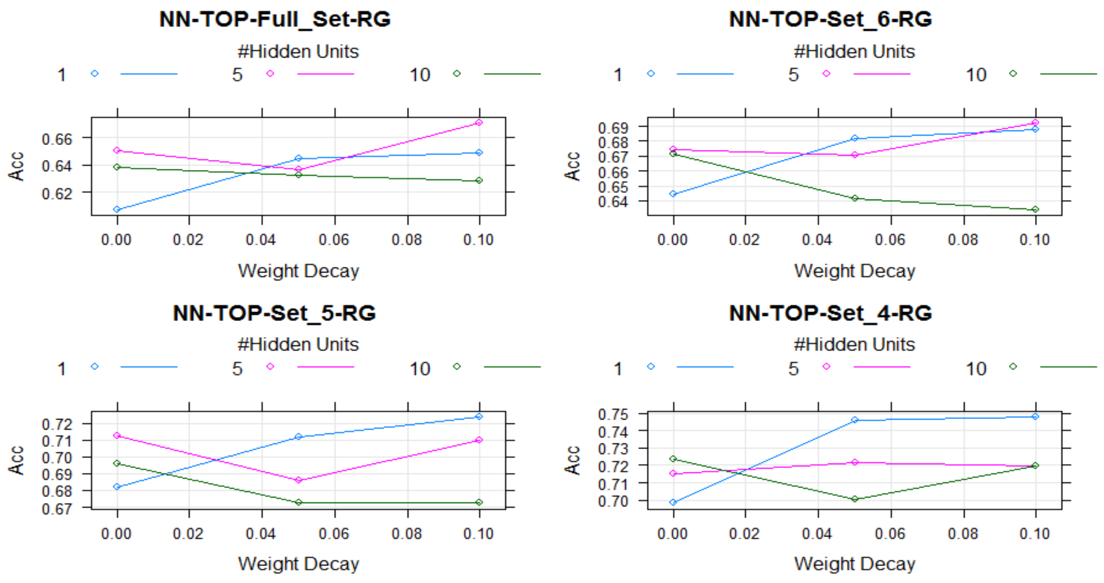
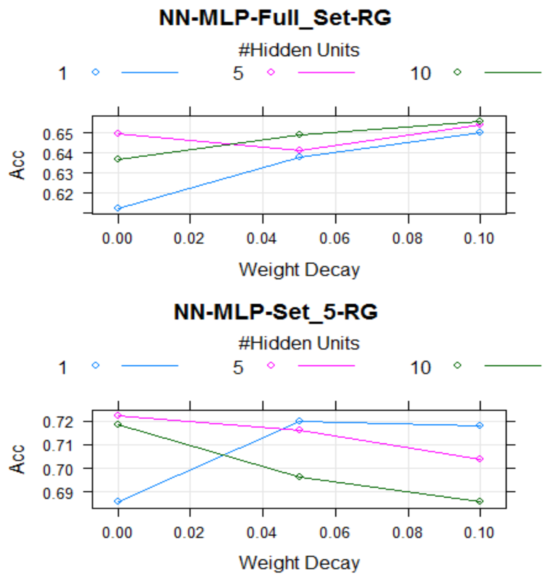


Figure 6.29: Accuracy of the random forest classifier with the TOP dicretization on the training set using different parameter settings for the 2BPRG case. The dot represent the mean value using cross validation for each parameter setting.



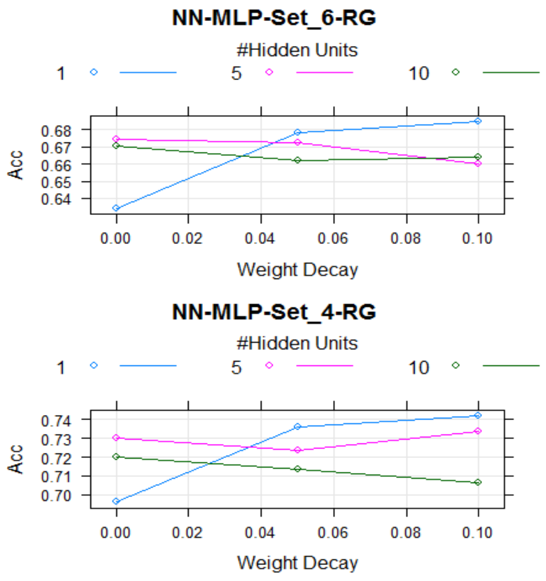


Figure 6.30: Accuracy of the neural network classifier with the MPL dicretization on the training set using different parameter settings for the 2BPRG case. The dot represent the mean value using cross validation for each parameter setting.

Generalized Linear Models

The last classification method used is the Generalized Linear Model (*GLM*). Figure 6.31 shows the results for the different parameter settings of the 2BPOG case. The best accuracy can be achieved with the *MLP* discretized data. This is also the group where the tuning parameter has the least influence on the accuracy.

Another result can be observed on Figure 6.32 for the 2BPRG case. For all three models, the regularisation parameter seems to have more influence on performance. It can be observed that an increase of the parameter leads to poorer results overall. Furthermore, we can see that the best accuracy can be achieved with the non-discretized data.

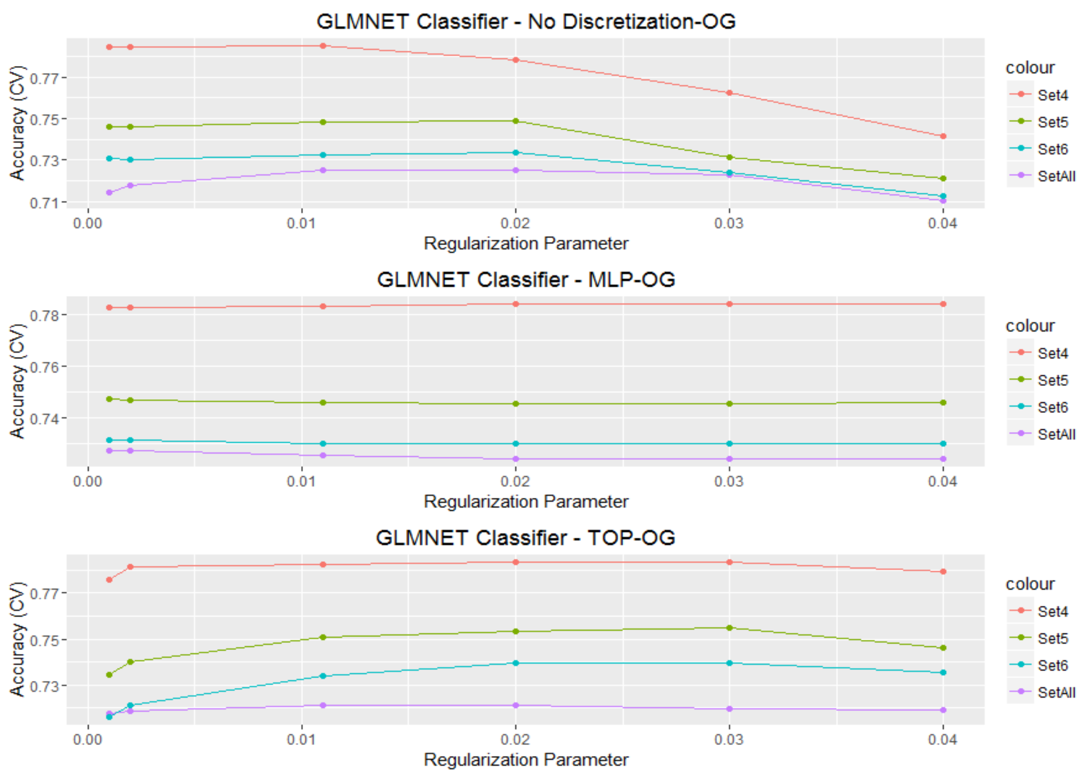


Figure 6.31: Accuracy of the *GLM* classifier on the training set using different parameter settings for the 2BPOG case. The results are grouped using the different dicretization methods and the algorithm sets. The dot represent the mean value using cross validation for each parameter setting.

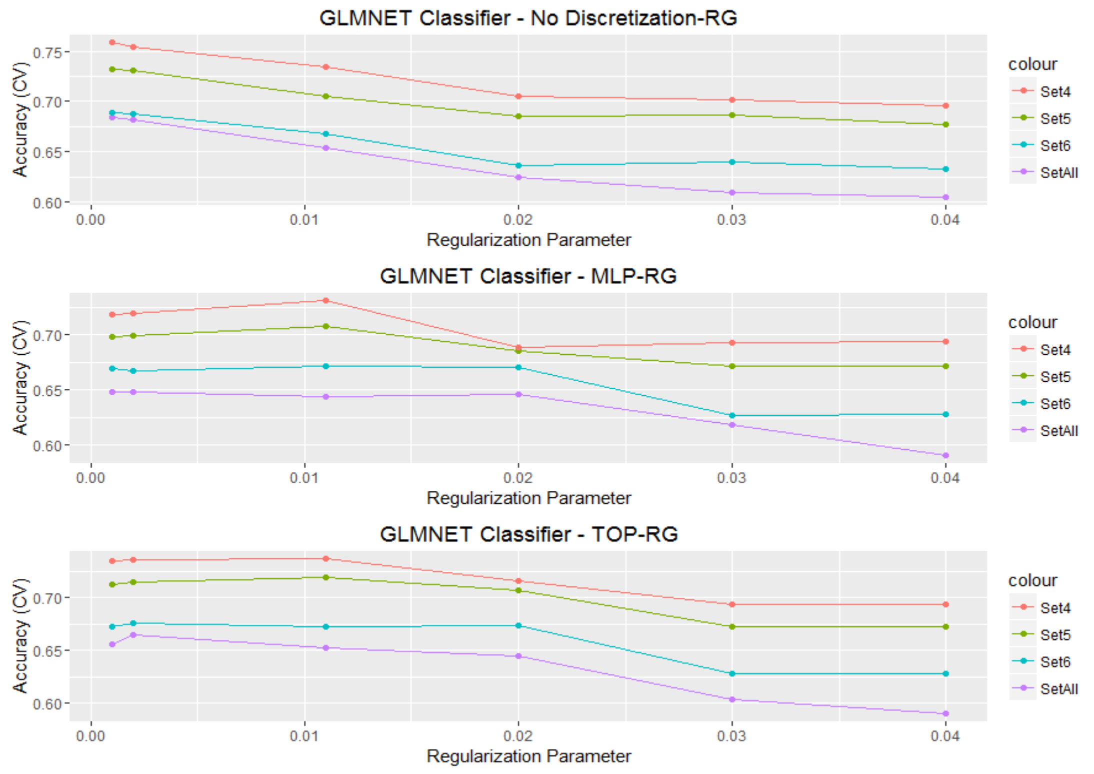


Figure 6.32: Accuracy of the *GLM* classifier on the training set using different parameter settings for the 2BPRG case. The results are grouped using the different dicretization methods and the algorithm sets. The dot represent the mean value using cross validation for each parameter setting.

6.4 Variation in the Algorithm Space

Another experiment that we want to test in this thesis is the effect of the algorithm space on the performance of the classifier. In theory, the fewer options the algorithm selection method has to choose from, the more likely it is that he will choose the right algorithm. This means for our experiment that by leaving out some algorithm options the better the performance of the machine learning procedures gets. To do so we applied all our test with subsets of the algorithms using only the best $x \in \{4, 5, 6\}$ algorithms.

Figure 6.33 displays the results for the 2BPOG case using the accuracy of the different machine learning methods for each of the different discretization methods. The accuracy is once again calculated using the ratio between the number of instances solved optimal and the total number of instances. For each algorithm that was taken from the set, the instances that were optimally solved using that particular algorithm had been split on the remaining algorithm with the help of the algorithm rank schema. Therefore, the metric gives an overview of the performance of the classifier always in relation to the set of algorithms. We can clearly identify a trend that the smaller the set of algorithms available, the better the performance of machine learning methods gets. Only in one case does the performance of the algorithm selection process improve with a larger algorithm set, such as the *RF* method using the *TOP* discretization preprocessing method. The algorithm selection models show the biggest performance boost between the accuracy of the *B*4 and *B*5 algorithm set.

Similar results can be seen for the 2BPOG case of the *BPP*. The results are shown in Figure 6.34. The effects of a smaller algorithm set seem to be similar as for the 2BPOG case but not quite as strongly. The accuracy of the algorithm set *B*5 to *B*4 increases especially for the *GLM* and the *RF* machine learning methods. Furthermore, for all cases, an increase of the accuracy can be recognized with each smaller algorithm subset.

These results can be derived from the fact that there are some algorithms that only solve a few instances optimally and therefore add some noise in all other cases.

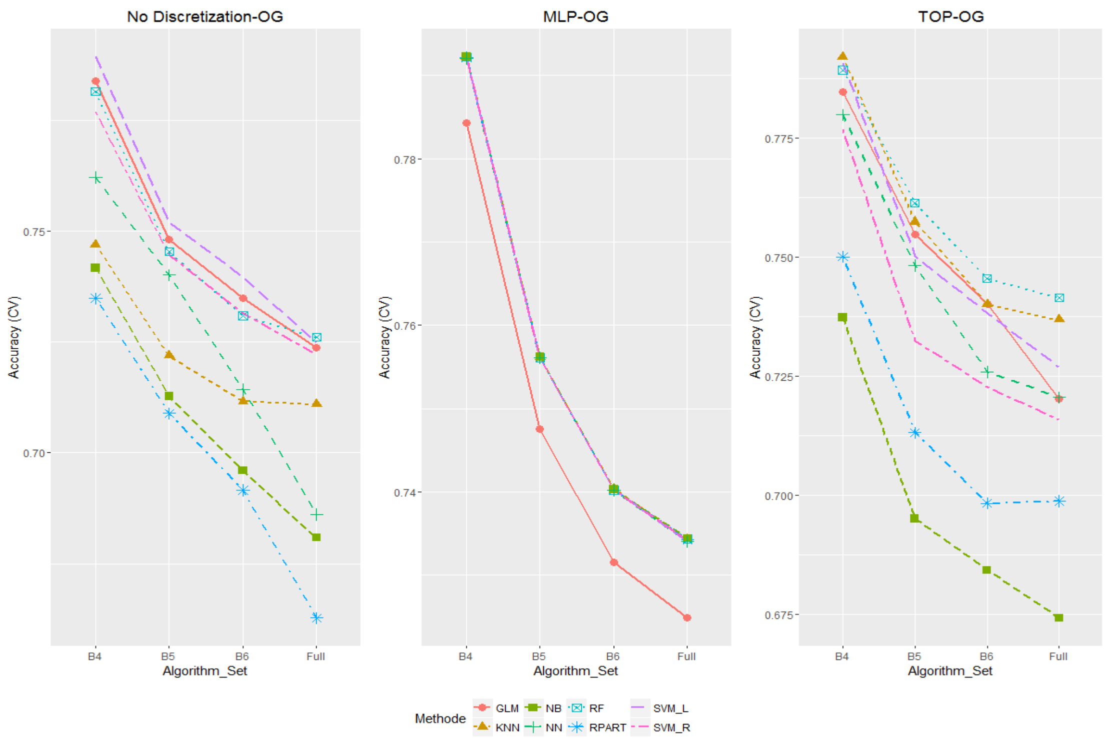


Figure 6.33: Accuracy of the tested classifiers using the best $x \in \{4, 5, 6\}$ heuristics (described as B4, B5, B6 and Full) for the 2BPOG case. The dots represent the best (highest) value achieved with the most promising parameter configurations for each machine learning method.

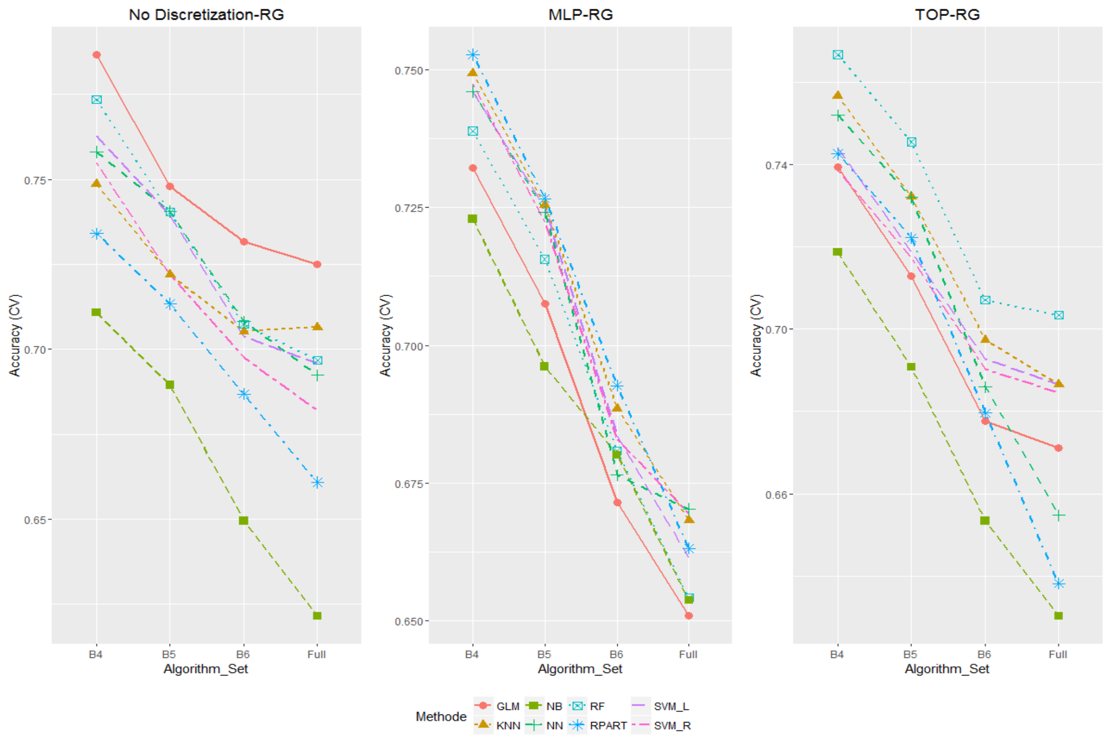


Figure 6.34: Accuracy of the tested classifiers using the best $x \in \{4, 5, 6\}$ heuristics (described as B4, B5, B6 and Full) for the 2BPRG case. The dots represent the best (highest) value achieved with the most promising parameter configurations for each machine learning method.

6.5 Comparison of Classifier

In addition to the effects of different algorithm sets, we are also interested in performances of the different machine learning classifier. Each of the methods presented uses different ways to compute the classification which can significantly influence the quality of the solution. To find the best method for our propose, we compared the accuracy of the classifiers using their best parameter settings.

The results of the evaluation of the 2BPOG case are shown in Figure 6.33. Once again we can see that the best results can be achieved with the $B4$ algorithm set. Almost all the models using the MLP discretized data, except the GLM classifier, are able to produce the best results for all three discretization methods. On the contrary, poor results have been achieved by the NB and the DT classifier. This method seems to be inadequate for the classification task of our BPP. Another interesting fact is that the kNN is only able to achieve a higher accuracy with the discretized data.

Concerning the classifier performance of the 2BPRG case Figure 6.34 shows an overview of the results. It can be observed that especially the kNN classifier is able to perform well in all three cases. Furthermore, the kNN classifier seems to benefit most from the reduction of the algorithm set. Compared to the 2BPOG case the DT classifier is able to perform rather good for the discretized data but the accuracy is only average for the not processed data. The other classifier perform very similarly to the 2BPOG case of the BPP.

6.5.1 Effects of Discretization

Besides the effects of the parameter configurations and the algorithms sets, we are also interested in the insights for the effects of the two different discretization methods compared to the not modified data. The effects of transforming numerical values into nominal ones can have a significant impact on the performances of the different classifier.

Figure 6.33 reveals some results using the discretization methods for the 2BPOG case. Concerning the data processed with the MLP discretization, the graphic shows that classifiers are able to achieve the best accuracy of the experiment with the $B4$ algorithm set. In general, better results can be achieved with the MLP data as shown in detail in 6.3. The *avg* column displays the average accuracy of the best parameter settings for each data set compared to the non-discretized data. The *best* column shows the comparison between the best value of the discretized and the non-discretized data.

On the data where the TOP discretization was used, it is observable that the performance is varying concerning the accuracy of the classifier. The kNN benefits most from this preprocessing, with 4.34% but not as much as for the MLP prepared data. A detailed overview of the results can also be found in Table 6.3.

Method	MLP		TOP	
	avg (%)	best (%)	avg (%)	best (%)
GLM	+ 0.87	- 0.07	- 0.56	- 0.20
KNN	+ 5.92	+ 4.55	+ 4.34	+ 4.08
NB	+ 5.17	+ 6.77	- 2.90	- 2.55
DT	+ 11.83	+ 8.02	+ 1.02	+ 1.43
RF	+ 1.15	+ 1.31	+ 0.79	+ 1.40
NN	+ 6.65	+ 4.13	+ 0.54	+ 2.16
SVM-L	- 0.01	+ 0.55	- 1.73	- 0.94
SVM-R	+ 5.82	+ 1.58	- 3.52	- 2.10

Table 6.3: Impact of the accuracy (in percent) for the 2BPOG case when using discretized data in relation to the results achieved with dicretization

For the 2BPRG case of the BPP Figure 6.34 and Table 6.4 show the achieved accuracy and the comparisons between the results. It is easy to see that for the 2BPRG case the best accuracy is achieved using the non-discretized data for five classifiers. Only the DT

Method	MLP		TOP	
	avg (%)	best (%)	avg (%)	best (%)
GLM	- 9.15	- 7.66	- 9.32	- 6.07
KNN	- 2.10	- 1.76	- 2.43	0.04
NB	- 0.31	+ 3.06	- 0.98	+ 1.19
DT	+ 3.70	+ 1.44	- 1.80	+ 0.35
RF	- 6.79	- 4.40	- 2.04	+ 0.76
NN	- 1.66	- 2.83	- 3.09	- 1.66
SVM-L	- 1.55	- 2.96	- 2.46	- 2.03
SVM-R	+ 1.78	- 1.21	- 2.53	- 0.69

Table 6.4: Impact of the accuracy (in percent) for the 2BPRG case when using discretized data in relation to the results achieved with dicretization

classifier seems to benefit from the processing methods. These results confirm the first assumption for the 2BPRG case.

6.6 Analysis of the Best Configuration per Classifier

To summarise the results of the different discretization methods and parameter settings, the best configurations for each classifier with the highest accuracy have been selected and summarized in Table 6.5 for the 2BPOG case and in Table 6.6 for the 2BPRG case.

First, we decided to choose the $B4$ algorithms set for all further analysis since all classifier show their best accuracy using this set. For the 2BPOG case, we can see that the discretization has a positive effect on the accuracy of all classifier. The highest accuracy is achieved using the NN but the RF, SVM and DT are also able to achieve nearly similar results. In general, the algorithm shows a good performance by classifying around 79% of the problem instances correct. Compared to the best single heuristic, which has an accuracy of 60.2%, we are able to improve the prediction by 19% .

The positive effects of the discretization methods cannot be displayed so clearly for the 2BPRG case. Only the NB, kNN, and DT algorithms are able to produce better results with the edited data. Furthermore, the best performance, achieved from the GLM classifier, is able to classify 79% of the test set correctly using the non-discretized data. The performances of the other procedures are again very similar, only the NB classifier delivers a slightly lower accuracy. Compared to the single solver solution we are able to boost the accuracy by almost 20%.

Algorithm	Algorithm Set & Discretization	Accuracy (%)	Configuration
NB	B4_MLP_full_OG	0.72	fL = 0, usekernel = T, adjust = 1
GLM	B4_TOP_full_OG	0.78	alpha = 1, lambda = 0.02
kNN	B4_MLP_full_OG	0.79	k = 9
RF	B4_MLP_full_OG	0.791	mtry = 5
NN	B4_MLP_full_OG	0,794	size = 1 and decay = 0.1
RPART	B4_MLP_full_OG	0.792	cp = 0.3538364
SVM-L	B4_MLP_full_OG	0.792	C = 0.05
SVM-R	B4_MLP_full_OG	0.792	sigma = 0.2500781, C = 0.25

Table 6.5: Summary of the best-performing parameter settings for the 2BPOG case with respect to the accuracy of the different classifier on the training set

Algorithm	Algorithm Set & Discretization	Accuracy (%)	Configuration
NB	B4_MLP_full_RG	0.72	fL = 1, usekernel = TRUE , adjust = 1
GLM	B4_full_RG	0.79	alpha = 1, lambda = 0.002
kNN	B4_TOP_full_RG	0.76	k = 9
RF	B4_full_RG	0.75	mtry = 7
NN	B4_full_RG	0,76	size = 1, decay = 0.05
RPART	B4_MLP_full_RG	0.75	cp = 0.03302374
SVM-L	B4_full_RG	0.76	C = 10
SVM-R	B4_full_RG	0.75	sigma = 0.1779091, C = 2

Table 6.6: Summary of the best-performing parameter settings for the 2BPRG case with respect to the accuracy of the different classifier on the training set

6.7 Analysis of the Performance of the Classifier on the Test Set

The experiment so far was always performed using cross-validation the training data, which was used for the learning phase and the evaluation part. We split the data into two different sets, the test 20% of the data and training set 80% of the data, to get a more realistic view of the performance of our prediction model. This allows a more unbiased comparison between the algorithm selection solvers and the single solver solutions. The test set was, until now, not used in any part of the experiment to simulate the performance of the classifier on a newly unseen problem instance. If we omit this step, we could theoretically adapt our models to the data until all cases are processed correctly. This model would then perform perfectly for our data, but with high probability, it would be useless for any real BPP application.

Concerning the test set, we separated the dataset randomly using the key 80% training and 20% test set. Therefore, the test set consists of 100 problem instances. Figure 6.35 gives a first overview of the number of problems solved optimally by each classifier for the 2BPOG case. As already mentioned in the section before we tested our trained prediction models using the $B4$ algorithm set. The figure reveals that the $FFIHOGJ04$ method is able to solve 69 of the 100 test set instances optimal. This means to show that the algorithm selection approach is useful for the BPP at least 70% of the problem instances have to be solved optimally by the algorithm selection solver.

Concerning the 2BPRG case, Figure 6.36 shows the performance of the four algorithms from the $B4$ algorithm set. Once again, the $FFIHOGJ04$ performs best on the test set by solving 60% of the instances optimal. Therefore, a single solver solution is able to solve 60% without any training.

To review how good the algorithm selection procedure is suited for the BPP, as described previously, we trained each classifier using the training set with the parameter settings that showed the most promising results on the training set. The used configurations are shown for both cases in the Table 6.5 and 6.6.

The most important outcome of this experiment is certainly the number of problem instances on which the trained solver show the best performances. For this propose, Figure 6.37 shows the results of the 2BPOG case compared to the single solver performances. As illustrated we clearly see that the almost all classifier are able to beat the single solver benchmark of 67%. A noteworthy observation is that the $SVM-L$ and the NB classifier are the only two classifiers that are not able to outperform the single heuristic. On the contrary, the RF solver perform very well on the test set with an overall accuracy of 91%. The confusion matrix for the best learning algorithm RF is shown in Table 6.7. The table shows that the RF solvers confuse mostly the $CFIHOGJ04$ with the $FFIHOGJ04$. The only other misclassification is done for the CH, which is once wrongly classified as CHB and twice as $FFIHOGJ04$. More details for the algorithm selection approach is given in Table 6.9. The best algorithm selection approach is able to beat the best single solver solution by 24%.

To verify the results of the experiment for the 2BPRG using the training set we also evaluated the best configurations with the test set. Figure 6.38 presents the results of the machine learning solver compared to the single solver solutions. Similar to the 2BPOG case, the RF was able to classify even more instances correctly to the corresponding BPP heuristics. This procedure was able to solve 93 of the 100 problem instances correctly, followed by the NN and the $SVM-R$ solver with 81 correctly classified instances. The NB was once again not able to beat the single solver solution. The $SVM-L$ method is able to perform better for the 2BPOG case with 80 correct classified problem instances. Table 6.8 shows the results of the confusion matrix for the 2BPRG case. Again, it can be observed that especially the $CFIHOGJ04$ heuristic is incorrectly classified as is $FFIHOGJ04$. The results for the algorithm selection experience of the 2BPRG case are summarized in Table 6.10.

Classified as	**CFIHOGJ04**	**CH**	**CHB**	**FFIHOGJ04**
CFIHOGJ04	4	0	0	6
CH	0	12	1	2
CHB	0	0	8	0
FFIHOGJ04	0	0	0	67

Table 6.7: Confusion martix for the classifications of the *RF* solver using the test set with the 2BPOG case.

Classified as	**CFIHOGJ04**	**CH**	**CHB**	**FFIHOGJ04**
CFIHOGJ04	4	0	0	6
CH	0	13	0	1
CHB	0	0	10	0
FFIHOGJ04	0	0	0	66

Table 6.8: Confusion martix for the classifications of the *RF* solver using the test set with the 2BPRG case.

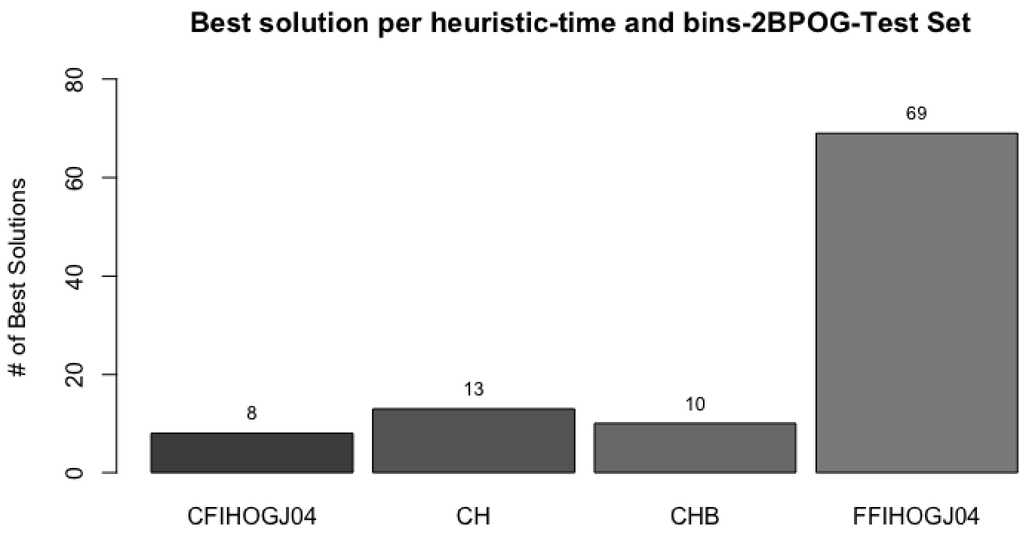


Figure 6.35: Number of instances form the test set on which the algorithms from the $B4$ set show best performance for the 2BPOG case.

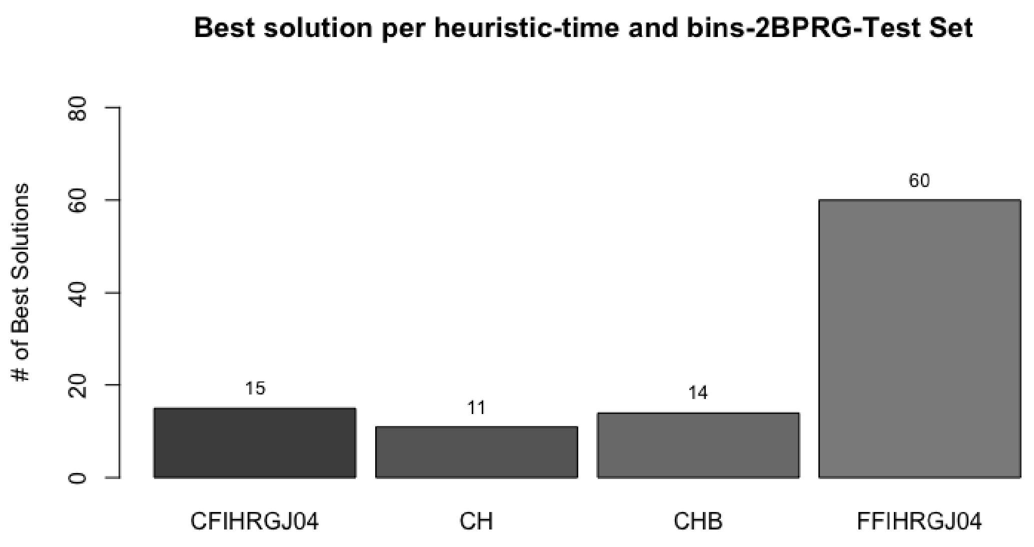


Figure 6.36: Number of instances form the test set on which the algorithms from the $B4$ set show best performance for the 2BPRG case.

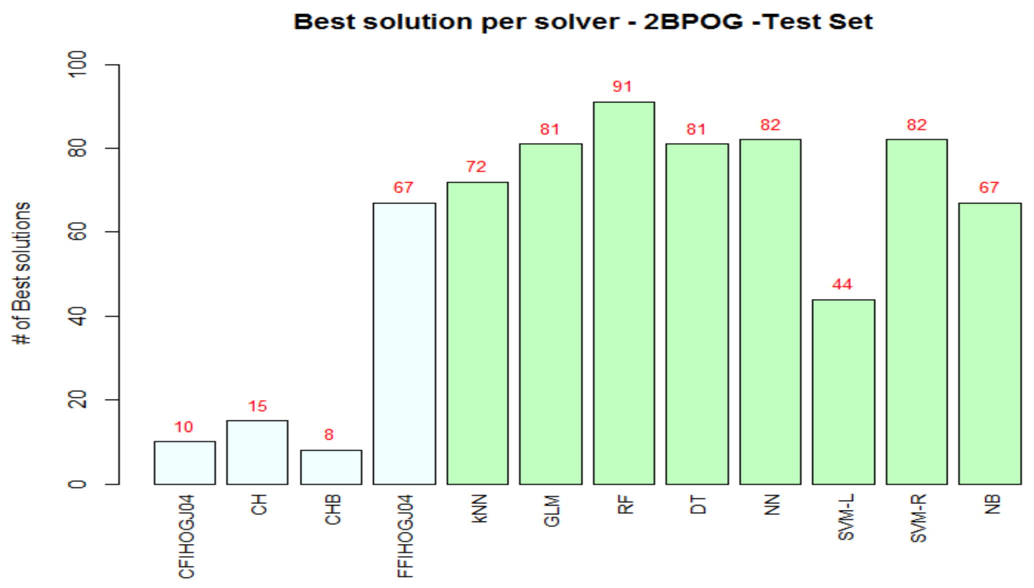


Figure 6.37: Comparison between the single solver heuristics and the algorithm selection classifier for the 2BPOG case using the test set.

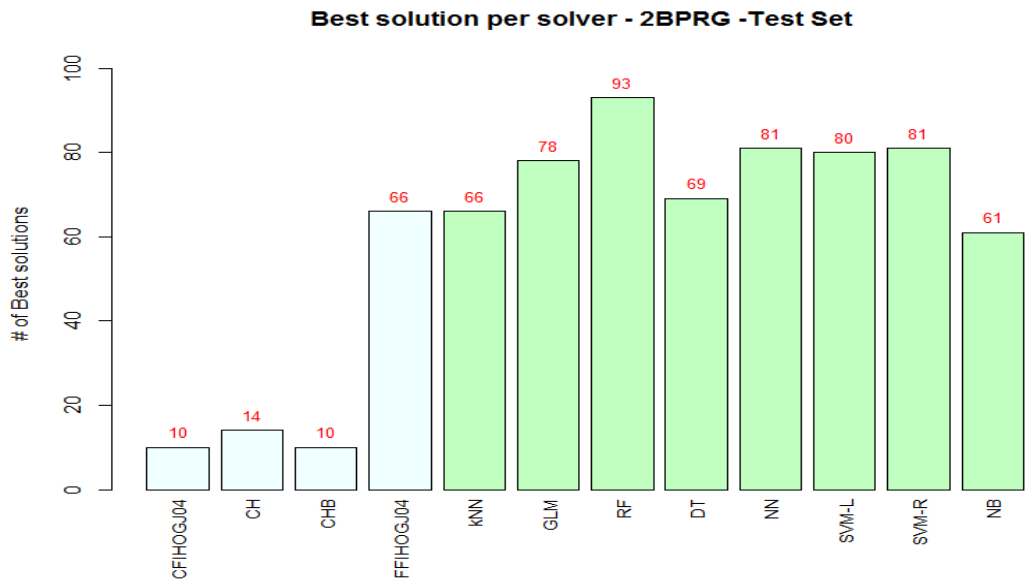


Figure 6.38: Comparison between the single solver heuristics and the algorithm selection classifier for the 2BPRG case using the test set.

Solver	**No. Best Solution**	**Accuracy**
Heuristics		
CFIHRGJ04	10	10 %
CH	15	15 %
CHB	8	8 %
FFIHOGJ04	67	67 %
Algorithm Selection		
kNN	72	72 %
GLM	81	81 %
RF	**91**	**91 %**
DT	81	81 %
NN	82	82 %
SVM-L	44	44 %
SVM-R	82	82 %
NB	67	67 %

Table 6.9: Performance metrics of the algorithm selection and the underlying heuristics on the test set for the 2BPOG case.

Solver	**No. Best Solution**	**Accuracy**
Heuristics		
CFIHRGJ04	10	10 %
CH	14	14 %
CHB	10	10 %
FFIHOGJ04	66	66 %
Algorithm Selection		
kNN	66	66 %
GLM	78	78 %
RF	**93**	**93** %
DT	69	69 %
NN	81	81 %
SVM-L	80	80 %
SVM-R	81	81 %
NB	61	61 %

Table 6.10: Performance metrics of the algorithm selection and the underlying heuristics on the test set for the 2BPRG case.

CHAPTER 7

Evaluation for the Algorithm Runtime Prediction

Based on the approach presented in Chapter 6, the following paragraphs demonstrate the results of the algorithm runtime prediction. In addition to the publicly available data sets, we extend the instances with a newly created data set for the algorithm runtime prediction.

We tested the algorithm runtime prediction model for the BPP using three already introduced state-of-the-art algorithms, namely the constructive heuristic CH, the constructive heuristic with bias CHB, and the constructive heuristic with bias and post-processing $CHBP$. Different configurations are used to build a robust prediction model for the runtime of the algorithms. Finally, we tested our approach and define which features have the biggest impact on the computation time needed.

7.1 Regression Parameter Evaluation

The next section describes the results of the different parameter settings for the regression methods used for the algorithm runtime prediction. As already mentioned, for the algorithm runtime prediction three algorithms were used, the CH, CHB and $CHBP$. Besides the already presented procedures, we will also analyze the $LOESS$ regression method. We use the Root-mean-squared error to evaluate and compare the performance of the different regression models with the discretization methods for the three algorithms.

7.1.1 k-Nearest Neighbour

The first regression method which we evaluate with different parameter settings is the kNN procedure. The results for the 2BPOG case can be taken from Figure 7.1. By increasing the number of neighbors, no decreasing $RMSE$ could be detected in all three models.

This observation also applies to the 2BPRG case of the BPP shown in figure 7.2. Furthermore, the models using the non-discretized data performs poor for both cases.

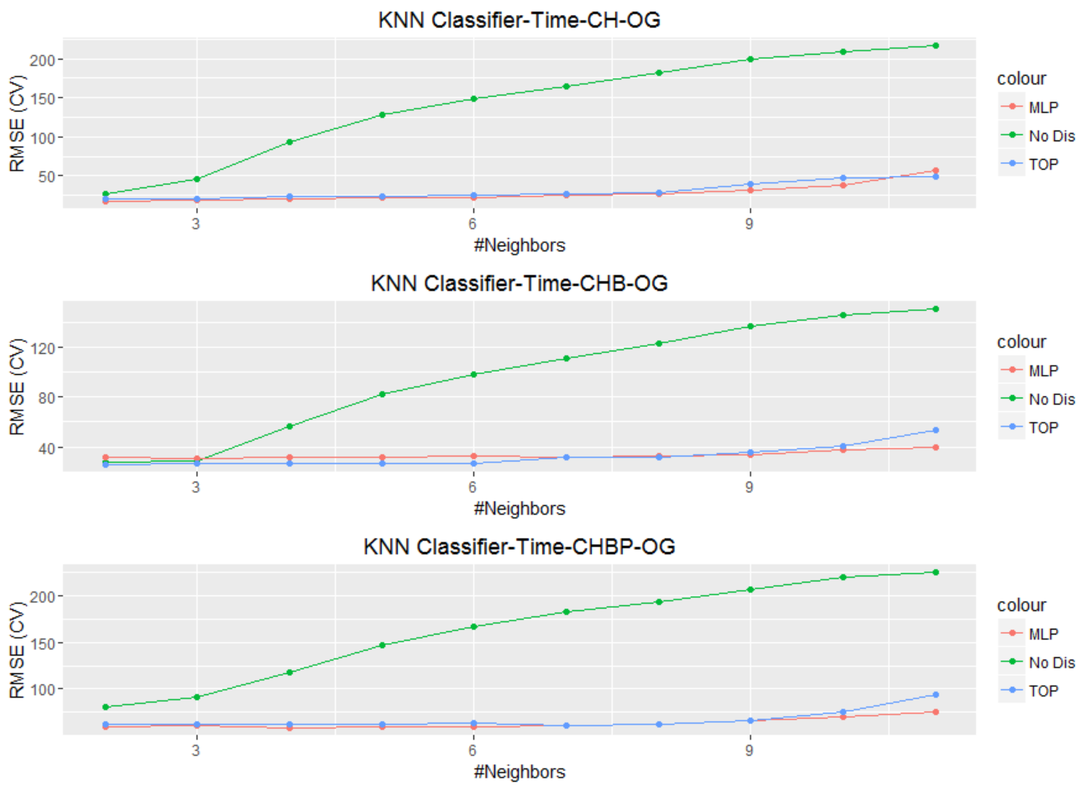


Figure 7.1: Root-mean-squared error of the kNN regression method on the training set using different parameter settings for the 2BPOG case. The results are shown for each algorithm using the different dicretization methods. The dot represent the mean value using cross validation for each parameter setting.

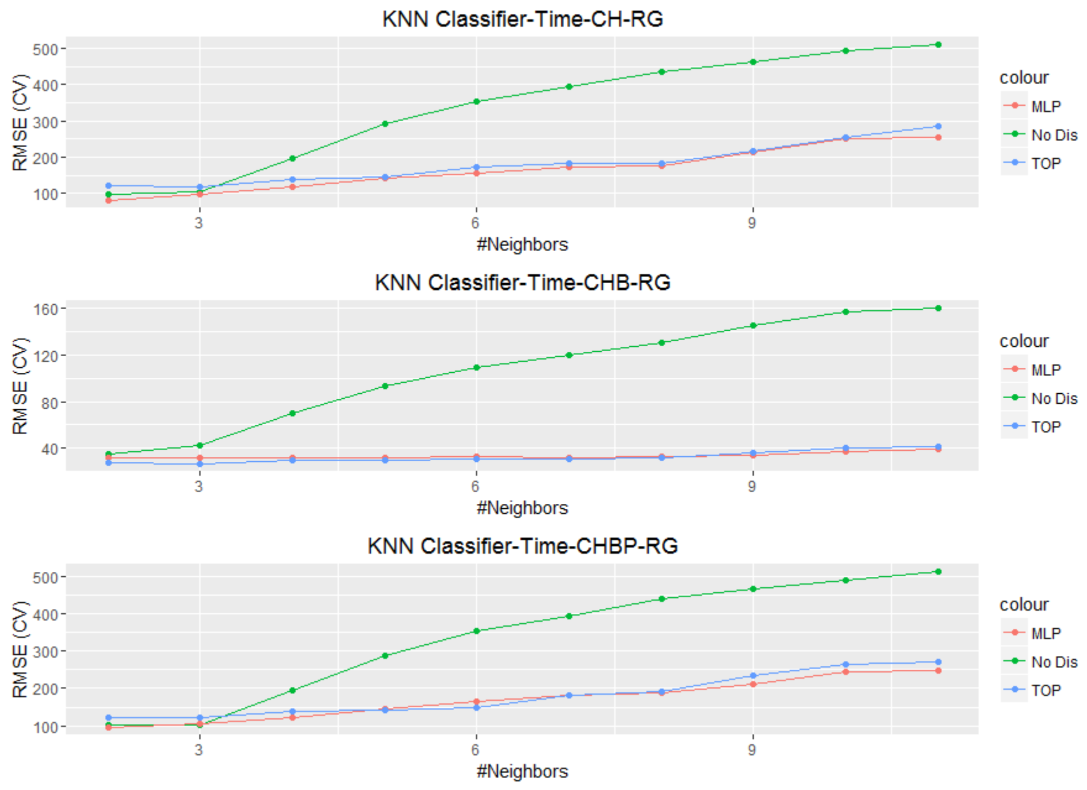


Figure 7.2: Root-mean-squared error of the kNN regression method on the training set using different parameter settings for the 2BPRG case. The results are shown for each algorithm using the different dicretization methods. The dot represent the mean value using cross validation for each parameter setting.

7.1.2 Regression Tree

Figure 7.3 and Figure 7.4 show the effects of the complexity parameters for the regression trees (RT). In both cases, we can observe that the higher the complexity parameter is chosen the higher the $RMSE$ gets. The most promising configuration for the 2BPOG case consists of a complexity parameter of zero and TOP discretized data for all three algorithms.

On the contrary, for the 2BPRG case, the MLP preprocessing method was able to achieve twice the best solution.

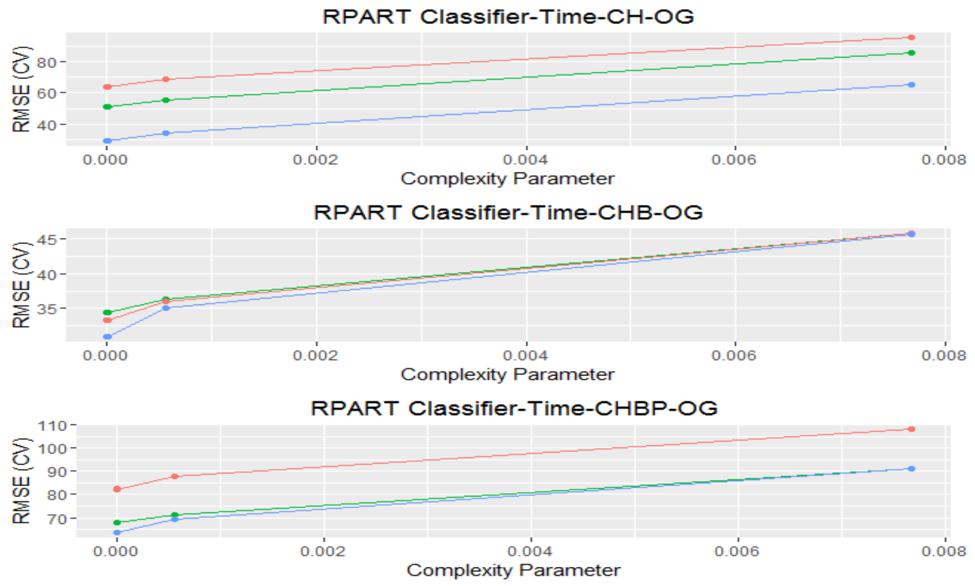


Figure 7.3: Root-mean-squared error of the regression trees on the training set using different parameter settings for the 2BPOG case. The results are shown for each algorithm using the different dicretization methods. The dot represent the mean value using cross validation for each parameter setting.

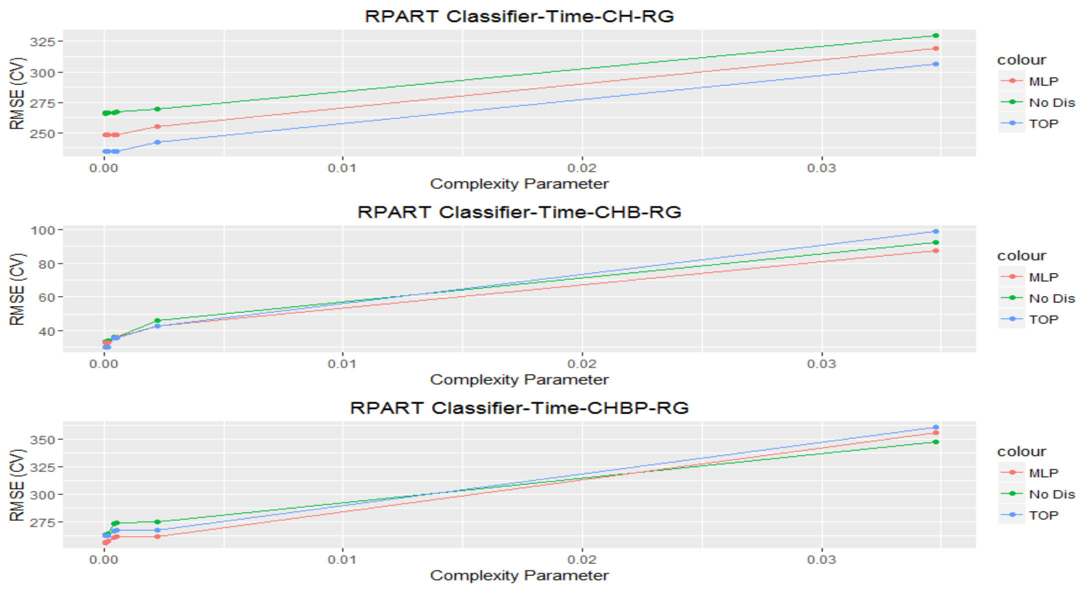


Figure 7.4: Root-mean-squared error of the regression trees on the training set using different parameter settings for the 2BPRG case. The results are shown for each algorithm using the different dicretization methods. The dot represent the mean value using cross validation for each parameter setting.

7.1.3 Random Forest

The results for the random forest (RF) show that the more predictors we have to split the data, the better the prediction gets for both cases. Further, we can see in Figure 7.5 and Figure 7.6 that all models perform almost the same with a big enough number of predictors. However, it is worth mentioning that the model using the TOP preprocessing method has some major problems if the number of predictors is rather small.

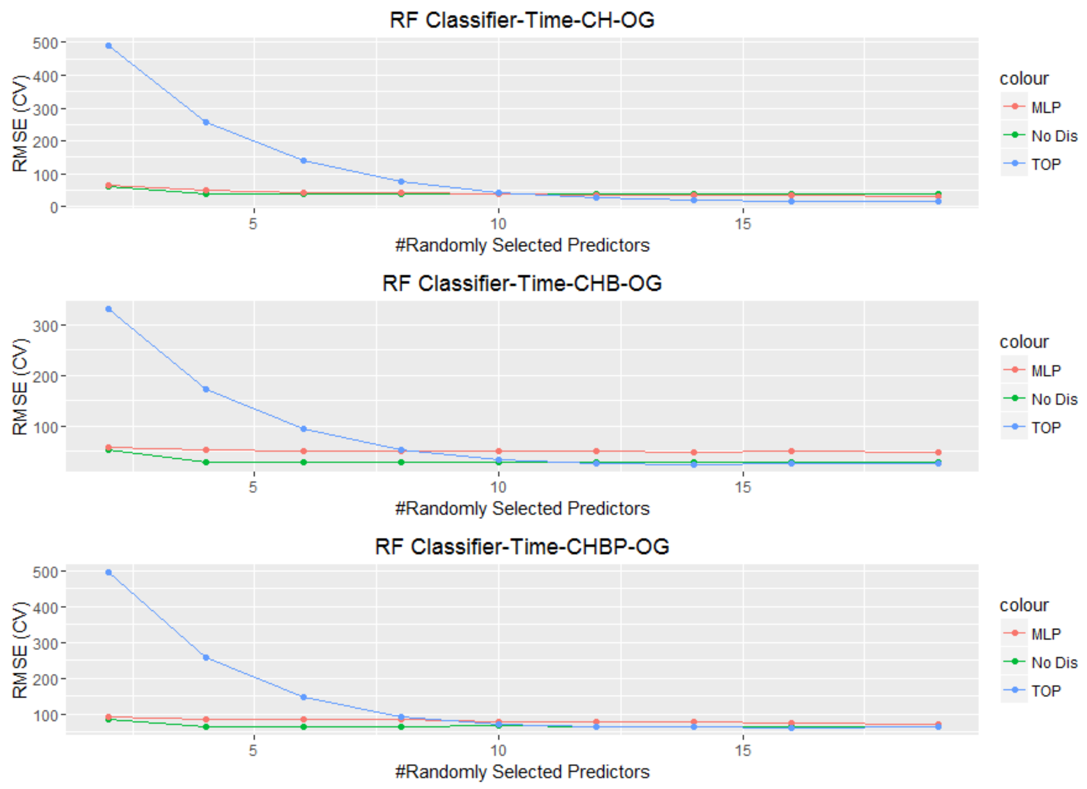


Figure 7.5: Root-mean-squared error of the random forest regression method on the training set using different parameter settings for the 2BPOG case. The results are shown for each algorithm using the different dicretization methods. The dot represent the mean value using cross validation for each parameter setting.

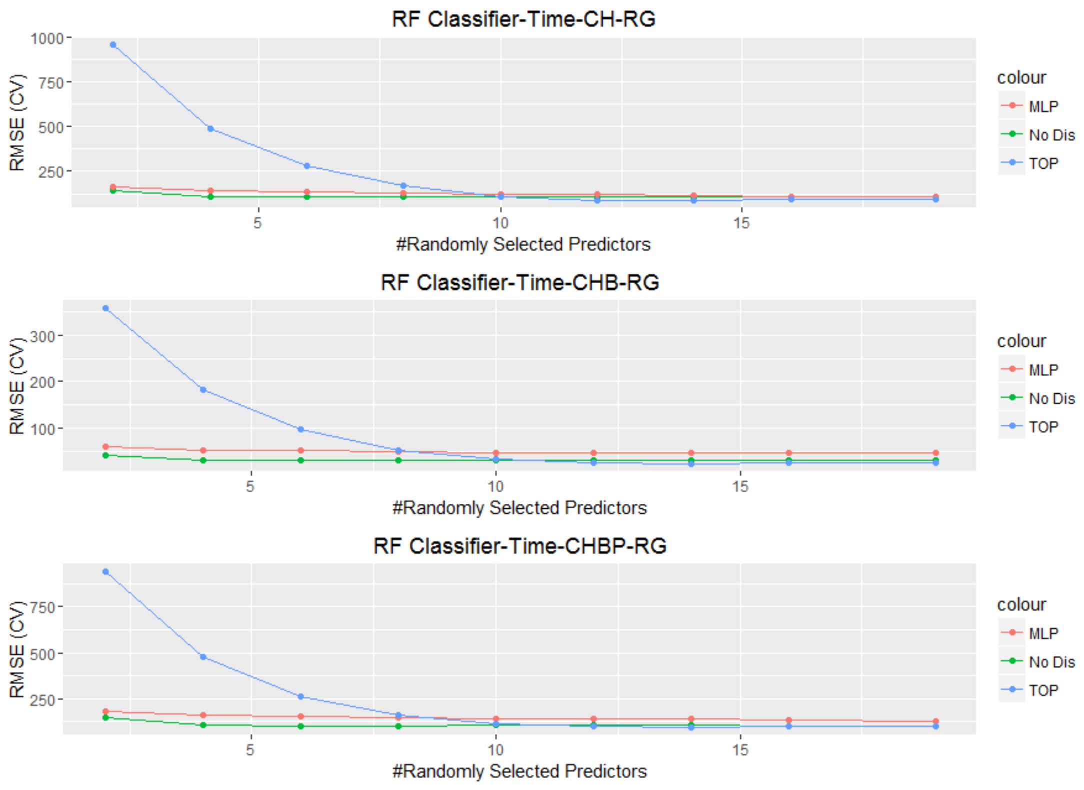


Figure 7.6: Root-mean-squared error of the random forest regression method on the training set using different parameter settings for the 2BPRG case. The results are shown for each algorithm using the different discretization methods. The dot represents the mean value using cross-validation for each parameter setting.

7.1.4 Support Vector Machine

The next regression method, the Support Vector Machines (SVM), shows some interesting behaviors for the two discretization methods. The experiment clearly illustrates that we are not able to apply the TOP model in all cases and the MLP model for the 2BPRG case usefully. The most consistent results are obtained using the non-discretized data in both cases. Figure 7.7 and Figure 7.8 shows the $RMSE$ values for the different configurations. Unfortunately, none of the configurations show to perform satisfactorily but a higher cost function seems to have a positive effect on the prediction error.

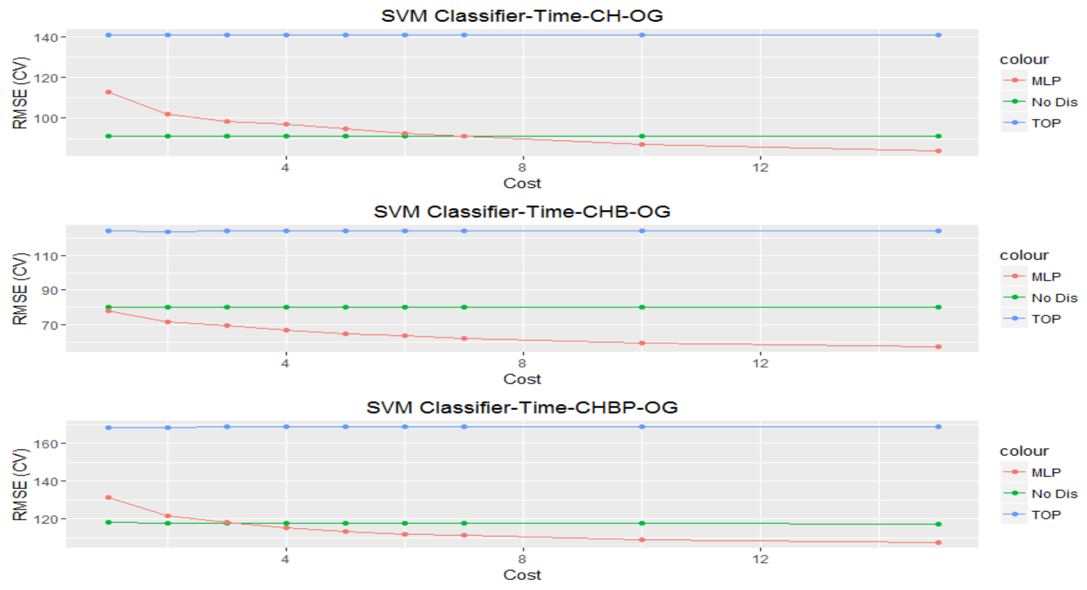


Figure 7.7: Root-mean-squared error of the Support Vector Machine regression method on the training set using different parameter settings for the 2BPOG case. The results are shown for each algorithm using the different dicretization methods. The dot represent the mean value using cross validation for each parameter setting.

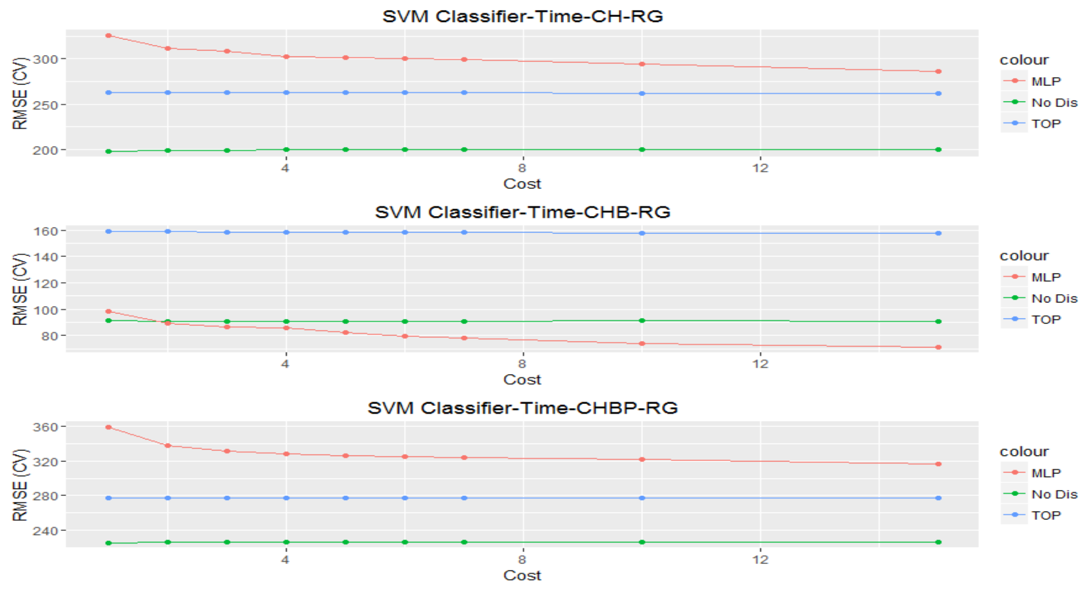


Figure 7.8: Root-mean-squared error of the Support Vector Machine regression method on the training set using different parameter settings for the 2BPRG case. The results are shown for each algorithm using the different dicretization methods. The dot represent the mean value using cross validation for each parameter setting.

7.1.5 Generalized Linear Model

Concerning the results for Generalized Linear Model (GLM) regression method, we tested different regularization parameters and values for alpha. The 2BPOG and the 2BPRG case show once again a similar basic behavior for the regression tasks. Both models perform best using the smallest possible regularization value with an alpha value of 0.1 as shown in Figure 7.9 and Figure 7.10. The graphics reveal that for the 2BPOG case significantly better results can be achieved than for the other case.

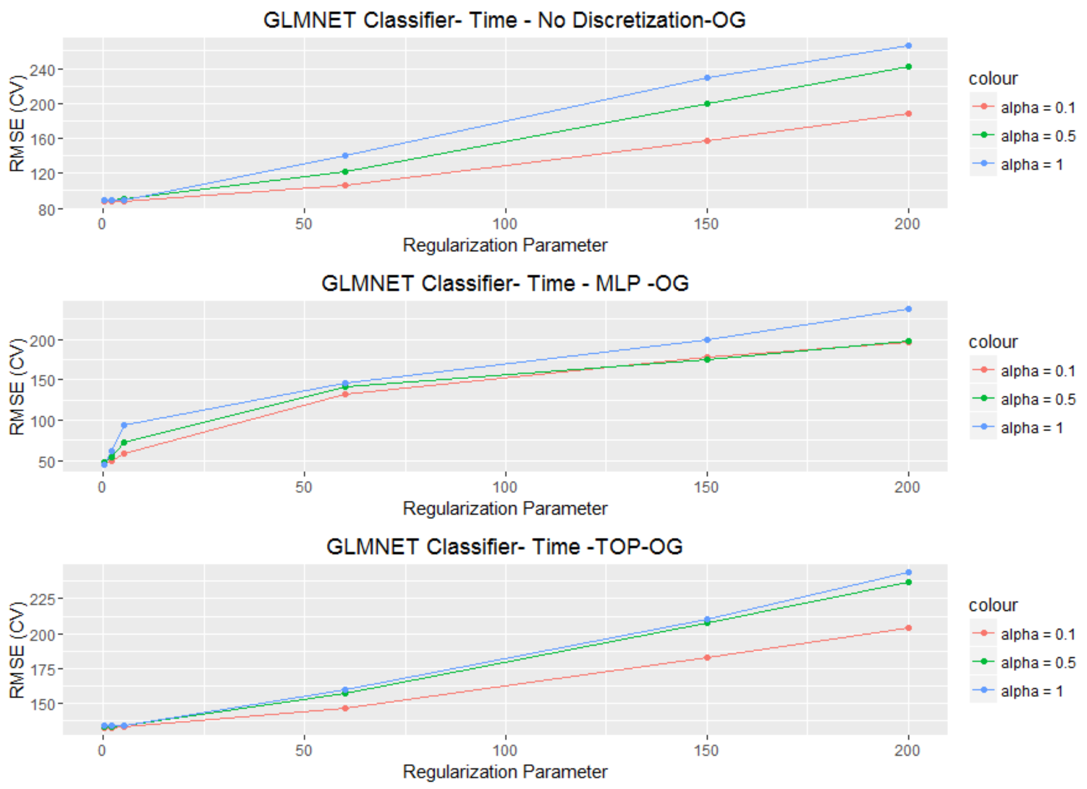


Figure 7.9: Root-mean-squared error of the Generalized Linear Model regression method on the training set using different parameter settings for the 2BPOG case. The results are shown for each algorithm using the different dicretization methods. The dot represent the mean value using cross validation for each parameter setting.

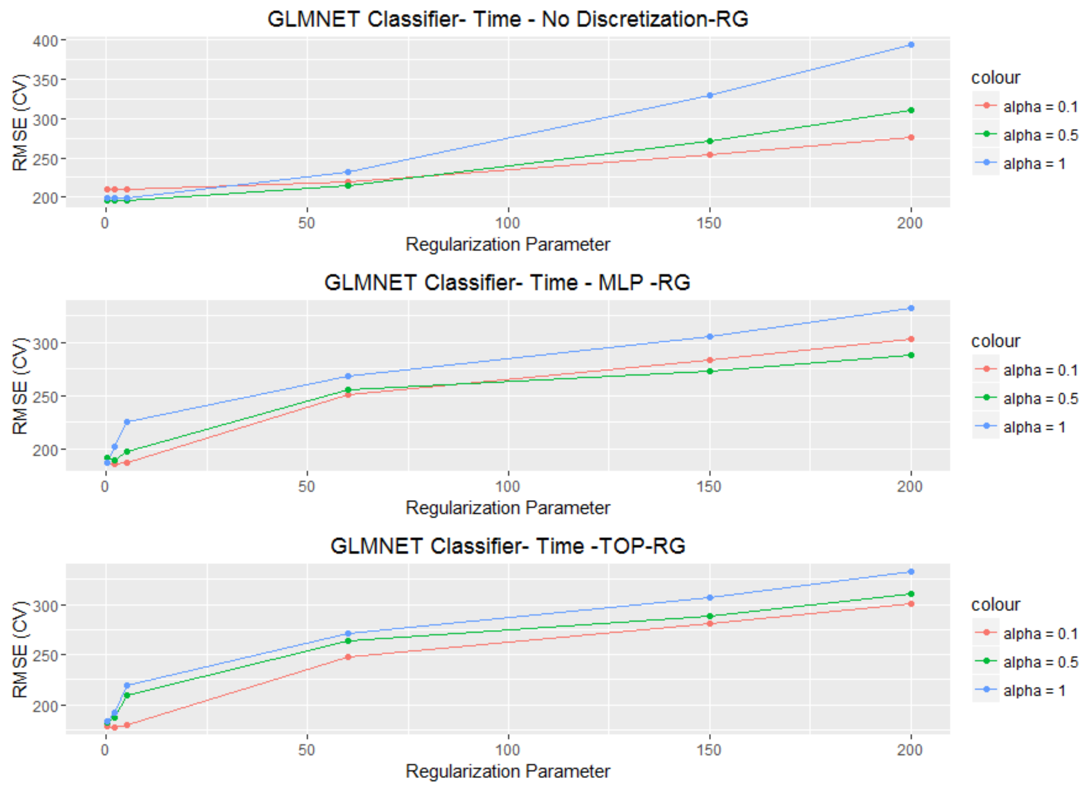


Figure 7.10: Root-mean-squared error of the Generalized Linear Model regression method on the training set using different parameter settings for the 2BPRG case. The results are shown for each algorithm using the different dicretization methods. The dot represent the mean value using cross validation for each parameter setting.

7.1.6 LOESS

The last considered regression method for the runtime prediction is the LOESS regression. We tested six different span parameter values. Figure 7.11 shows that values between 0.3 and 0.6 were able the achieve the best results. This also applies to the 2BPRG case as shown in Figure 7.12. Furthermore, the experiment shows that the LOESS regression model is not able to deal with data which were preprocessed using the *TOP* method. Another observation is that LOESS was only able to predict good values for the *CHB* algorithm of the 2BPRG cases. The method was, on the other hand, able to deliver consistently good results for the other cases.

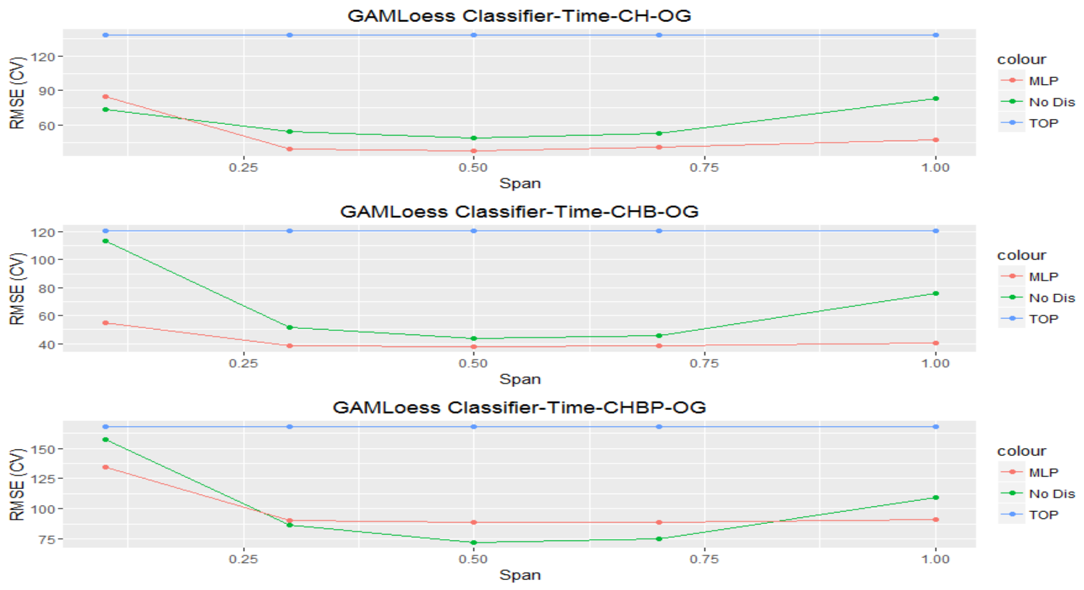


Figure 7.11: Root-mean-squared error of the LOESS regression method on the training set using different parameter settings for the 2BPOG case. The results are shown for each algorithm using the different dicretization methods. The dot represent the mean value using cross validation for each parameter setting.

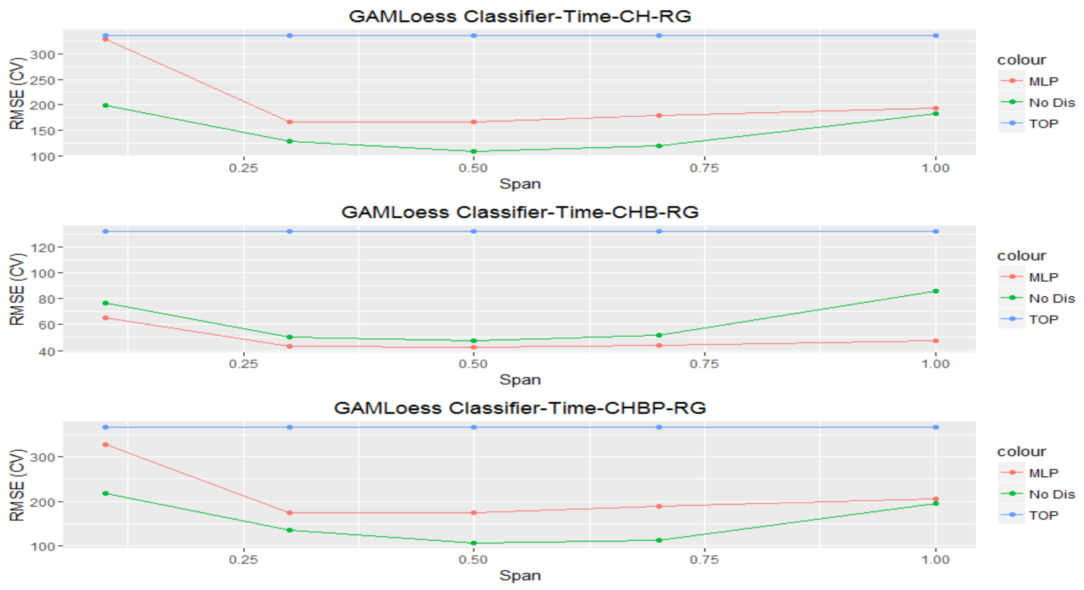


Figure 7.12: Root-mean-squared error of the LOESS regression method on the training set using different parameter settings for the 2BPRG case. The results are shown for each algorithm using the different dicretization methods. The dot represent the mean value using cross validation for each parameter setting.

7.2 Comparison of Regression Models

After the first evaluation of the individual machine learning regression models, we summarize the results in Figure 7.13 for the 2BPOG and in Figure 7.14 for the 2BPRG case to compare the individual performances with each other.

Figure 7.13 compares the results for the 2BPOG case. The graph shows clearly different performances for each of the three algorithms regardless of the preparation of the data set. Especially for the *TOP* preprocessing method, there seems to be a significant difference between the performances of the machine learning procedures. Good results can be accomplished especially by the *kNN*, *RF* and the *RT*. However, these results also very clearly for the individual algorithms. The *GLM* and *SVM* procedures seem to not be suitable for the algorithm runtime prediction.

Concerning the 2BPRG, the Figure 7.14 reveals that the results of the *RF* for each of the three BPP algorithms belong to the best ones. Similar good results could otherwise only be achieved by the *kNN* method. Another interesting finding is that the *LOESS* regression model shows some fluctuating results for the different preprocessing methods.

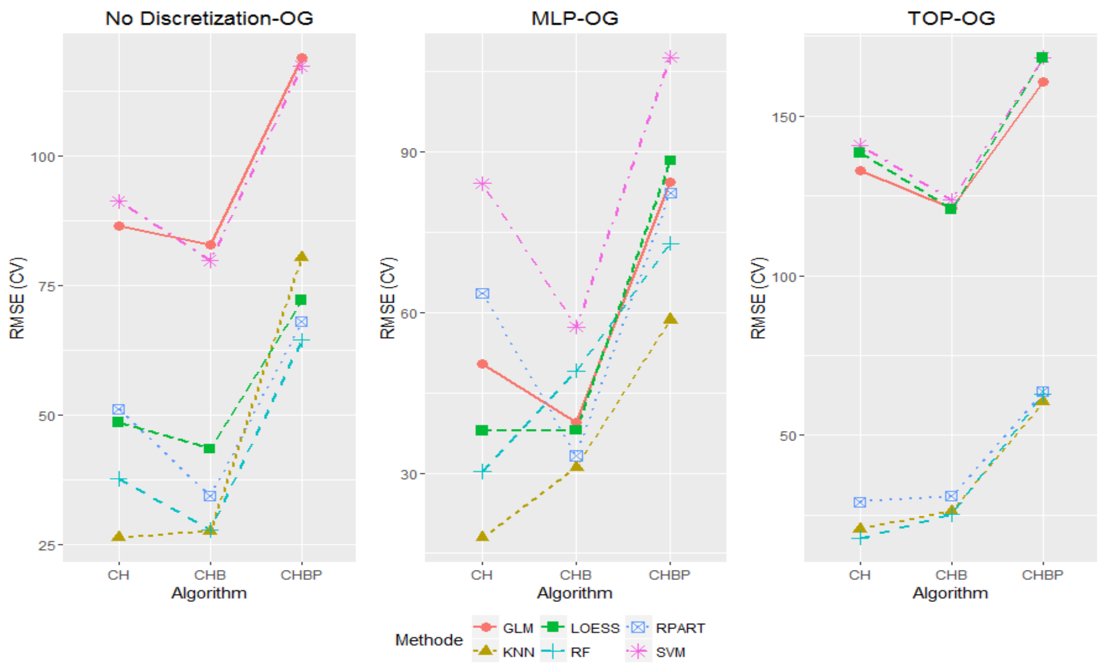


Figure 7.13: RMSE of the tested machine learning regresion prediction models using the three available algorithms for the 2BPOG case. The dots represent the best (lowest) value achieved with the most promising parameter configurations for each machine learning method.

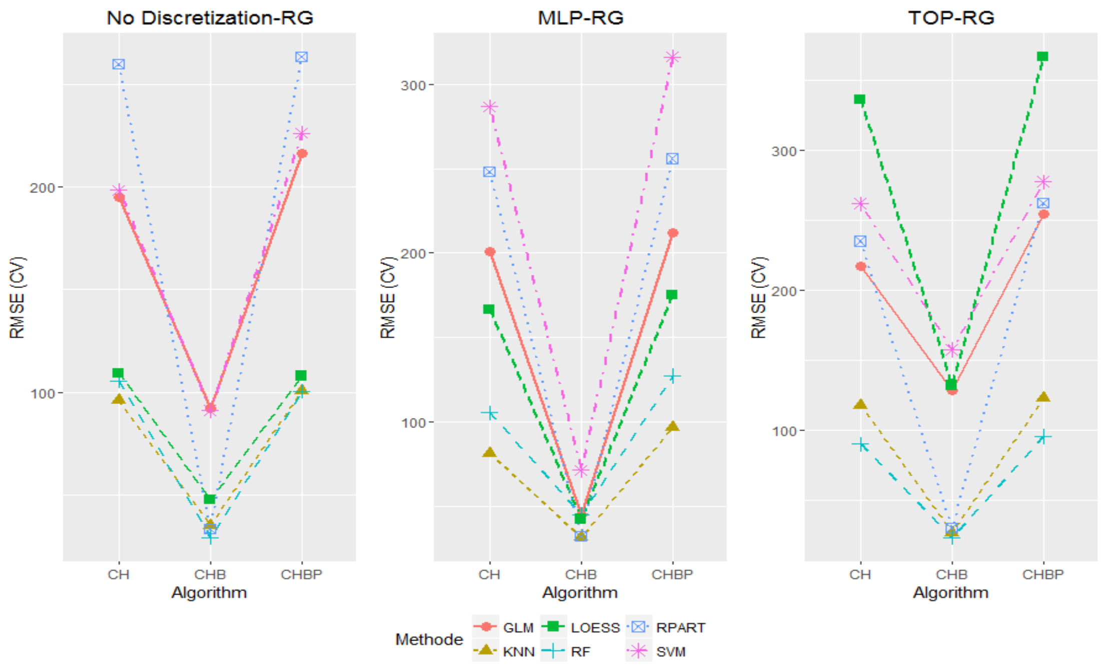


Figure 7.14: RMSE of the tested machine learning regression prediction models using the three available algorithms for the 2BPRG case. The dots represent the best (lowest) value achieved with the most promising parameter configurations for each machine learning method.

7.2.1 Effects of Discretization

So far, we have evaluated the different parameter configurations and compared the results to each other. In the next paragraph, we want to analyze the effects of the discretization methods for the runtime prediction approach.

Starting with the 2BPOG case, Figure 7.13 indicates that the best performance for the CH and CHB are achieved with the non-discretized data. For the $CHBP$ algorithm the kNN machine learning method was able to obtain the best performance using the MLP preprocessing method. Table 7.1 gives a complete overview of the results compared to each other. Keep in mind that the evaluation criteria is the $RMSE$ for this experiment. This means a negative percentage range in the analysis indicates a smaller $RMSE$ and therefore an improvement of the prediction model. Some noteworthy results from Table 7.1 are that the kNN method performs significantly better using the MLP and the TOP preprocessed data sets. Further, the GLM, $LOESS$ and SVM shows a better performance only with the MLP preprocessed data. The RF , on the other hand, shows a positive development when using the data from the TOP method.

Method	MLP		TOP	
	avg (%)	best (%)	avg (%)	best (%)
GLM	- 39.54	- 39.31	+ 43.91	+ 43.91
KNN	- 69.08	- 19.89	- 67.60	- 20.35
RT	+ 24.72	+ 17.18	+ 154.63	- 18.79
RF	+ 15.95	+ 16.75	- 16.95	- 19.51
SVM	- 4.13	- 13.61	+ 50.18	+ 50.06
LOESS	- 16.64	+ 0.004	+ 86.99	+ 160.16

Table 7.1: Impact on the RMSE (in percent) for the 2BPOG case when using discretized data.

For the 2BPRG case, Table 7.2 and Figure 7.14 give an overview of the effect of the discretization methods. Interestingly, the *RF* method is able to perform better with both discretized data sets compared to the 2BPOG case. The other procedures follow most of the time the same patterns as for the 2BPOG case. The exact results can be taken from Table 7.2.

Method	MLP		TOP	
	avg (%)	best (%)	avg (%)	best (%)
GLM	- 8.93	- 8.93	+ 19.26	+ 19.26
KNN	- 52.24	- 9.21	- 48.81	+ 15.59
RT	- 18.74	- 3.52	- 19.73	- 5.10
RF	+ 30.00	+ 18.35	+ 146.90	- 10.88
SVM	+ 38.61	+ 30.86	+ 35.16	+ 35.16
LOESS	+ 26.30	+ 45.35	+ 129.38	+ 215.17

Table 7.2: Impact on the RMSE (in percent) for the 2BPRG case when using discretized data.

7.3 Analysis of the Best Configuration per Regression Model

The following section gives an overview of the results for the runtime prediction approach. Once again, all results are computed using a repeated 10-fold-cross-validation on the training data set. For the three algorithms, the best configurations combined with the most promising preprocessing method are summarized and compared to each other. The results are displayed for each of the three algorithms separately in the following tables.

The following tables represent the best configurations for the 2BPOG case of the BPP. Table 7.3 contains the results for the CH algorithm. It is easy to see that the RF is able to perform best with a RMSE of 17.76413 closely followed by the kNN prediction model. All other methods are not able to compete with these two.

Algorithm	**Preprocessing Method**	**RMSE**	**Configuration**
GLM	MLP	50.40345	none
kNN	MLP	17.86678	k = 2
RT	TOP	29.01001	cp = 0.01
RF	**TOP**	**17.76413**	**mtry = 19**
SVM	MLP	84.11483	C = 15
LOESS	MLP	37.97607	span = 0.5 & degree = 1

Table 7.3: Summary of the best performing parameter settings for the *CH* algorithm of 2BPOG case with respect to the RMSE.

Next, we analyze the outcome for the *CHB* method. The Table 7.4 shows again the *RF* performs best on the training set using the *TOP* discretization technique. More interesting is the fact that almost all approaches deliver acceptable results compared to each other. Only the performance of the *SVM* might be inferior in relation to its competitors.

Algorithm	**Preprocessing Method**	**RMSE**	**Configuration**
GLM	MLP	39.57406	none
kNN	TOP	26.04224	k = 2
RT	TOP	30.93993	cp >0.01
RF	**TOP**	**25.13520**	**mtry = 14**
SVM	MLP	57.34458	C = 15
LOESS	MLP	37.99581	span = 0.5 & degree = 1

Table 7.4: Summary of the best performing parameter settings for the *CHB* algorithm of 2BPOG case with respect to the RMSE.

Distinctly different results are shown for the *CHBP* algorithm in Table 7.5 . Compared to the other two heuristics, the prediction models provide significantly poorer results. The best approaches uses the *kNN* method and provides us with a *RMSE* vale of 60.40159. This value is twice as bad as the best value for the *CHB* heuristic and more than three times worse than the comparable value for the *CH* algorithm. A possible explanation would be the complexity of the algorithm since it is the most complex compared with the other two.

Algorithm	**Preprocessing Method**	**RMSE**	**Configuration**
GLM	MLP	84.3053	none
kNN	**none**	**60.40159**	**k = 7**
RT	TOP	63.64074	cp >0.01
RF	TOP	62.71340	mtry = 16
SVM	MLP	107.7158	C = 15
LOESS	none	72.18040	span = 0.5 & degree = 1

Table 7.5: Summary of the best performing parameter settings for the *CHBP* algorithm of 2BPOG case with respect to the RMSE.

So far, we have evaluated the results for the 2BPOG case. The next step is to analyze the performance of the three algorithms for the 2BPRG case. For this purpose, we summarized the results in the Table 7.6 , 7.7 and 7.8. In general, it is noticeable that the results of the 2BPOG case are not as good as those for the 2BPOG case. Starting with the *CH* algorithm, Table 7.6 shows the detailed analysis. The performance is not as good as for the 2BPOG case as already mentioned. Especially the *GLM*, the *SVM* and also the *RT* method are not able to produce usable results. The best model is using the *kNN* method with the *MLP* discretized data and a RMSE of 81.60648.

Algorithm	**Preprocessing Method**	**RMSE**	**Configuration**
GLM	none	194.9283	none
kNN	**MLP**	**81.60648**	**k = 2**
RT	TOP	235.0692	cp = 0.0005
RF	TOP	90.12391	mtry = 14
SVM	none	198.1537	C = 1
LOESS	none	109.2510	span = 0.5 & degree = 1

Table 7.6: Summary of the best performing parameter settings for the *CH* algorithm of 2BPRG case with respect to the RMSE.

Surprisingly, for the *CHB* algorithm the approaches are able to produce a good outcome. The individual results are summarized in Table 7.7. With a RMSE of 23.63004, the prediction model is able to produce a smaller error than for the 2BPOG case. The impression we got from Figure 7.14 are confirmed by the numbers in the table. All other procedures, except the *SVM* method, can also provide very good results for the *CHB* algorithm. An explanation for this behavior cannot be made on the basis of the available data.

Algorithm	**Preprocessing Method**	**RMSE**	**Configuration**
GLM	MLP	45.18718	none
kNN	TOP	27.06684	k = 2
RT	TOP	30.38953	cp = 0.0002
RF	**TOP**	**23.63004**	**mtry = 14**
SVM	MLP	71.54584	C = 15
LOESS	MLP	42.81039	span = 0.5 & degree = 1

Table 7.7: Summary of the best performing parameter settings for the *CHB* algorithm of 2BPRG case with respect to the RMSE.

The performance for the *CHBP* is similarly poor as for the 2BPOG case. Table 7.8 displays the results concerning the RMSE. As expected, the best performance was achieved by the *RF* method with a RMSE of 95.84077 followed by the *kNN* method with 96.88509. Both models are able to achieve their best performance using the *TOP* discretized data set. Compared to the other methods the *LOESS* approach is able to reach the third place. The other processes produce a RMSE of well over 200 and are therefore no longer suitable for a practical use.

Algorithm	**Preprocessing Method**	**RMSE**	**Configuration**
GLM	MLP	212.2038	none
kNN	TOP	96.88509	k = 2
RT	TOP	255.8048	cp = 0.0001
RF	**TOP**	**95.84077**	**mtry = 14**
SVM	none	225.9977	C = 1
LOESS	none	107.9664	span = 0.5 & degree = 1

Table 7.8: Summary of the best performing parameter settings for the *CHBP* algorithm of 2BPRG case with respect to the RMSE.

The results for the training set indicates that the *RF* and the *kNN* prediction models are best suited for the algorithm runtime prediction approach. Furthermore, the results of the training set show that the *TOP* preprocessing method has a positive influence on the RMSE. In 4 of 6 cases, the *RF* method achieved the best results with the *TOP* discretized data and the *kNN* used once the *MLP* preprocessing method in the best setup. In the next paragraph, these results are to be evaluated on the basis of the test sets in order to give a final judgment on the usability for the algorithm runtime approach for the three tested algorithms.

Algorithm	**Prediction Model**	**RMSE**	**R^2**	**Data Span (Sec)**
2BPOG				
CH	RF_TOP_CH_OG	12.4867063	0.9995902	1 - 2990
CHB	RF_TOP_CHB_OG	34.2816482	0.9909219	1 - 1947
CHBP	kNN_CHBP_OG	39.745708	0.992555	1 - 2995
2BPRG				
CH	kNN_MLP_CH_RG	197.033200	0.979931	1 - 6722
CHB	RF_CHB_RG	31.035428	0.994268	1 - 2183
CHBP	RF_TOP_CHBP_RG	59.2276826	0.9968807	1 - 6923

Table 7.9: Performance metrics for the runtime prediction and the underlying heuristics on the test set.

7.4 Comparison of Regression Models for the Runtime Prediction on the Test Set

In the previous chapter, we compared the results of the different configurations for the runtime prediction model with the various machine learning techniques. After we tested the created models with the designated training set we have to evaluate them using some previously unseen problem instances. The test set consist of 111 randomly selected problem instances from the two publicly available datasets and the dataset we created for this thesis.

The already presented RMSE is used as an evaluating parameter. The RMSE is calculated from the square root of the average forecast error. The larger the RMSE, the worse is the adaptation of the model. It is necessary to obtain the smallest possible RMSE by controlling the influencing factors in order to improve the quality of a model. In order to determine the best models from the training set, it was enough to compare the different RMSE values among each other and to select the model with the smallest value. For the evaluation of the models in a more realistic environment, it is important to consider the RMSE in connection with the data itself. This means a RMSE of 30 can be good if the initial data set has a span of $0 - 3000$. If, however, our data shows a span of 10 to 50, then a model with a RMSE of 30 is not usable in practice.

As a further reference point for the goodness of the model, we also refer to the R^2 for each prediction model. The R^2 is an indication of how well the independent variables are able to explain the variance of the dependent variables. The values for R^2 are always between 0% (useless model) and 100 % (perfect model adjustment). Note that the R^2 is a measure of quality for describing a linear relationship. It can be easily interpreted as the fraction of the variance of the dependent variable (in our case the runtime), which can be explained by the independent variable (the calculated features). For the evaluation of the best models, we used the test set with the parameter configurations that showed the best performance during the preceding experiments.

The RMSE for both cases of the BPP is summarized in Table 7.9. Both cases show a very good performance on the test set and even surpass sometimes the results of the training set. Only the *CH* algorithm for the 2BPRG case remains behind our expectations. This particular case is interesting since it is the only one where the MLP preprocessing method was able to achieve the best results during the test phase. The plots of the observed values against the predicted values give a visual representation of the results and can help us to understand how well the model fits. For the 2BPOG case Figure 7.15 shows the performance of the CH algorithm. The distribution of the results, amongst other things, from the use of the well-known benchmark instances. These instances include a large share of problems with only a few items to pack. The runtime of the large instances are only covered by the problem instance which we introduced. Figure 7.16 and 7.17 show especially for the smaller values some higher residuals which leads to the slightly increased RMSE compared to the *CH* algorithm.

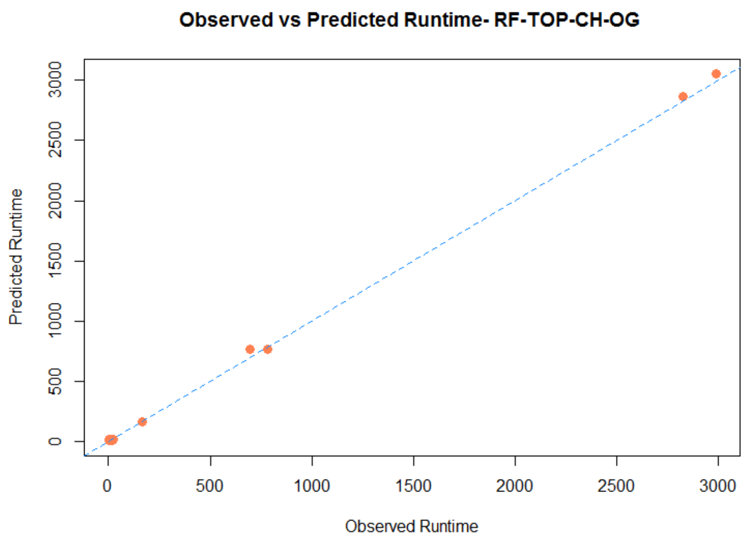


Figure 7.15: Visual comparison of models for runtime predictions of previously unseen test instances. In each scatter plot, the x-axis shows true runtime and the y-axis runtime as predicted by the respective model. Each dot represents the runtime of an unseen instance for the 2BPOG case - CH heuristic

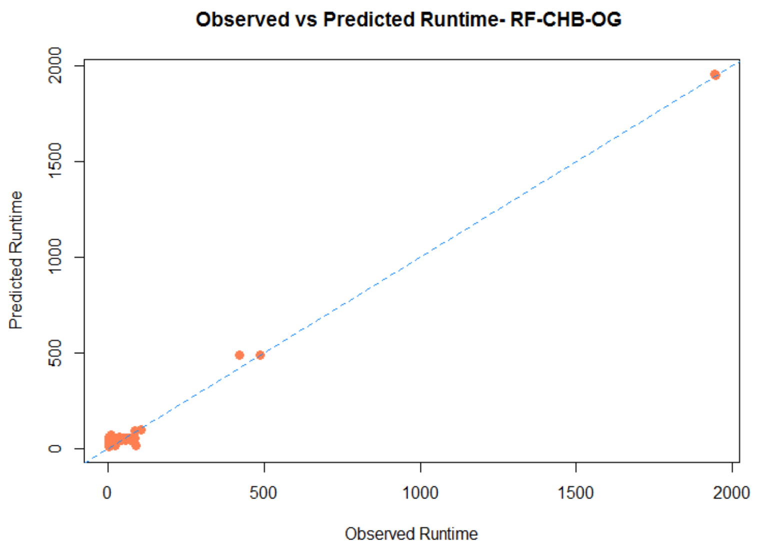


Figure 7.16: Visual comparison of models for runtime predictions of previously unseen test instances. In each scatter plot, the x-axis shows true runtime and the y-axis runtime as predicted by the respective model. Each dot represents the runtime of an unseen instance for the 2BPOG case - CHB heuristic

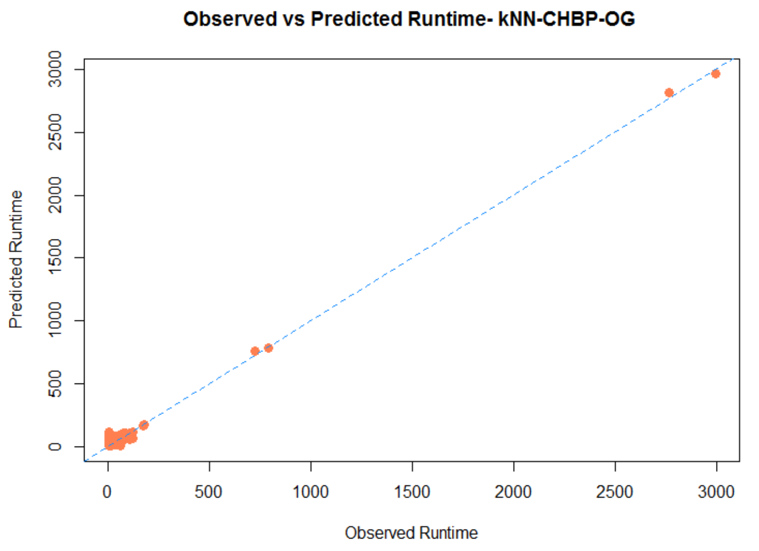


Figure 7.17: Visual comparison of models for runtime predictions of previously unseen test instances. In each scatter plot, the x-axis shows true runtime and the y-axis runtime as predicted by the respective model. Each dot represents the runtime of an unseen instance for the 2BPOG case - CHBP heuristic

Figures 7.18, 7.19, and 7.20 show some qualitative differences in predictions for the 2BPRG case. It can be well recognized that the increased RMSE for the CH algorithms is due to the outliers in the 5000 sec range as shown in Figure 7.18. The other figures show again a good fitting with some noise for the lower part of the prediction.

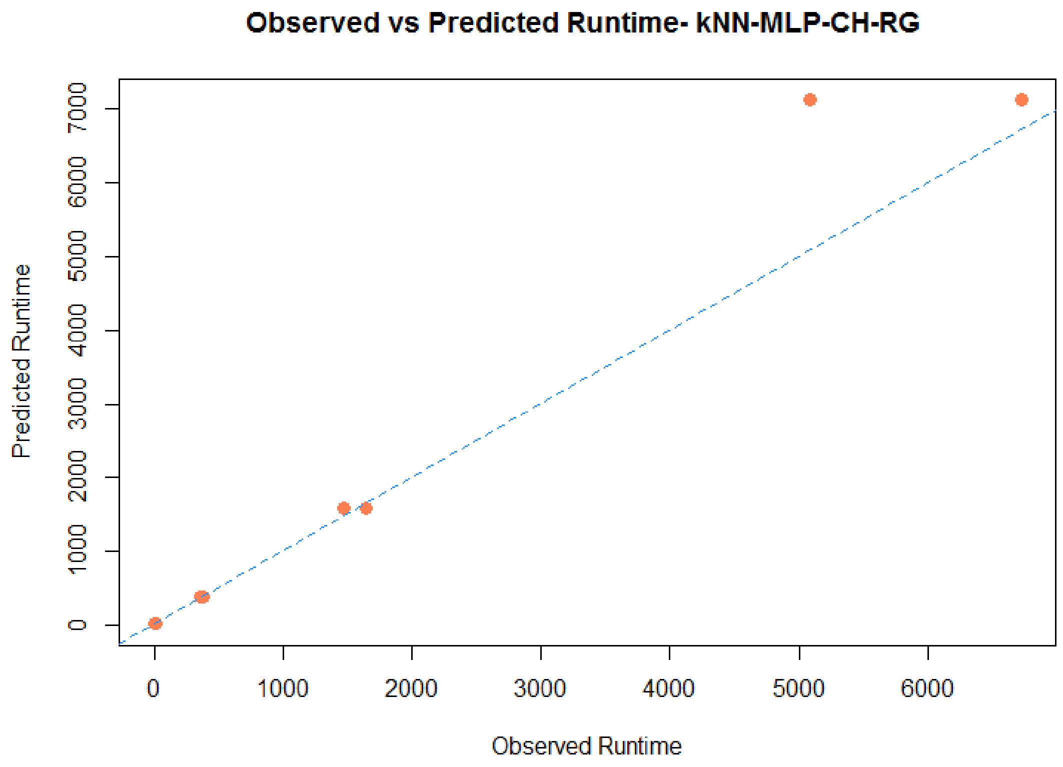


Figure 7.18: Visual comparison of models for runtime predictions of previously unseen test instances. In each scatter plot, the x-axis shows true runtime and the y-axis runtime as predicted by the respective model. Each dot represents the runtime of an unseen instance for the 2BPRG case - CH heuristic

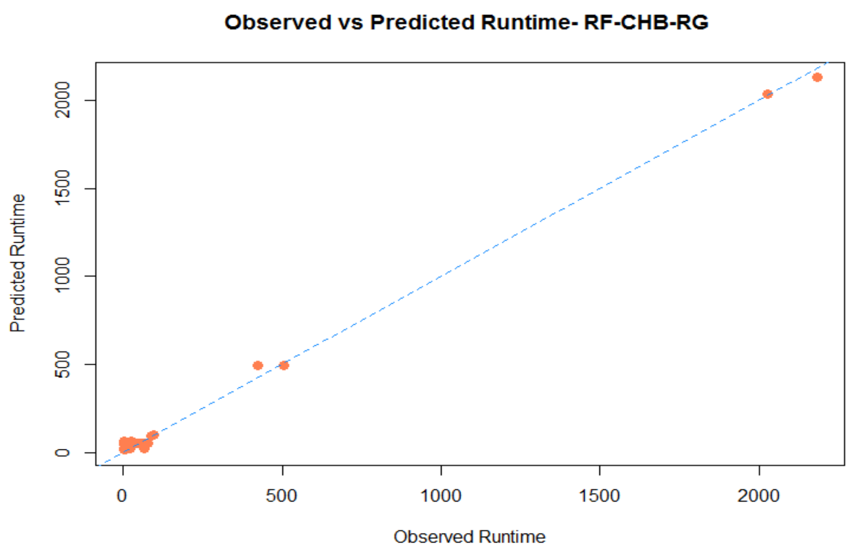


Figure 7.19: Visual comparison of models for runtime predictions of previously unseen test instances. In each scatter plot, the x-axis shows true runtime and the y-axis runtime as predicted by the respective model. Each dot represents the runtime of an unseen instance for the 2BPRG case - CHB heuristic

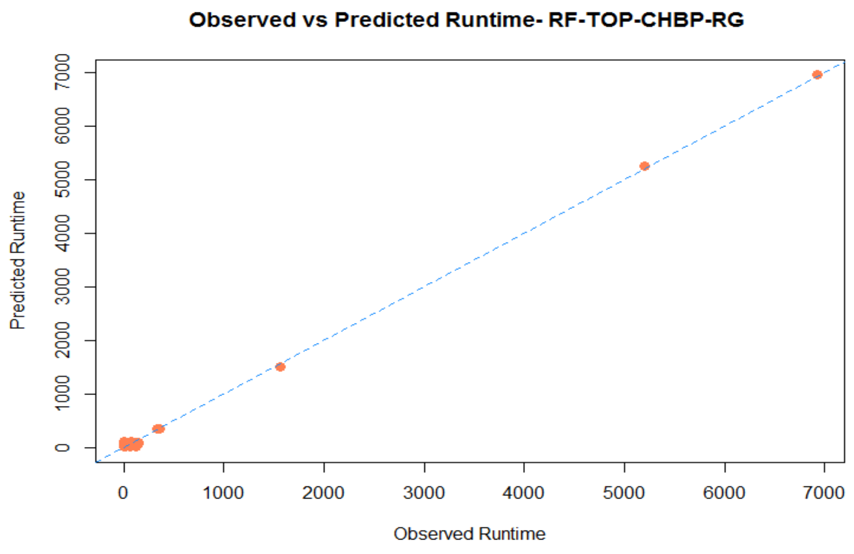


Figure 7.20: Visual comparison of models for runtime predictions of previously unseen test instances. In each scatter plot, the x-axis shows true runtime and the y-axis runtime as predicted by the respective model. Each dot represents the runtime of an unseen instance for the 2BPRG case - CHBP heuristic

7.4.1 Variable Importance for the Runtime Prediction

The last section deals with the analysis of the importance of the features for the runtime prediction models.

For each of the three algorithms, we performed a variable importance evaluation and visualized the results. The function used a method which automatically scales the importance scores for each feature between $\{-1, \dots, 0, \dots, 1\}$. A negative variable importance could appear if pairs of features offer almost the same kind of information which could lead in extreme cases to different results. However, most of the time this features only add unnecessary noise in the prediction model because they provide no additional information for the model.

In our experiment, we see these behavior in Figure7.21, 7.22 and 7.25. In all these cases, the TOP discretized data was used to build the best prediction model for the algorithm. This could be a sign that the data is too simplified or categorized by using the *TOP* preprocessing method.

Another important outcome is that some models use only a fraction of the available features to produce the predictions (mostly the *RF* method) as we can see in the Figures 7.21, 7.22, 7.25, and 7.26 . In these models, the $NumberOfItems$ is the most important feature for the prediction of the running time.

Concerning the *kNN* prediction models, Figure 7.23 and 7.24 show that almost all features are used for this method.

The most important variable is the $TotalItemSize$ and for the other model the *OddItems* as presented in Chapter 5.3.1. Only the *OneBinItems* as well as *BigItems* and *SmallItems* seem to have no influence on the predictions of this model.

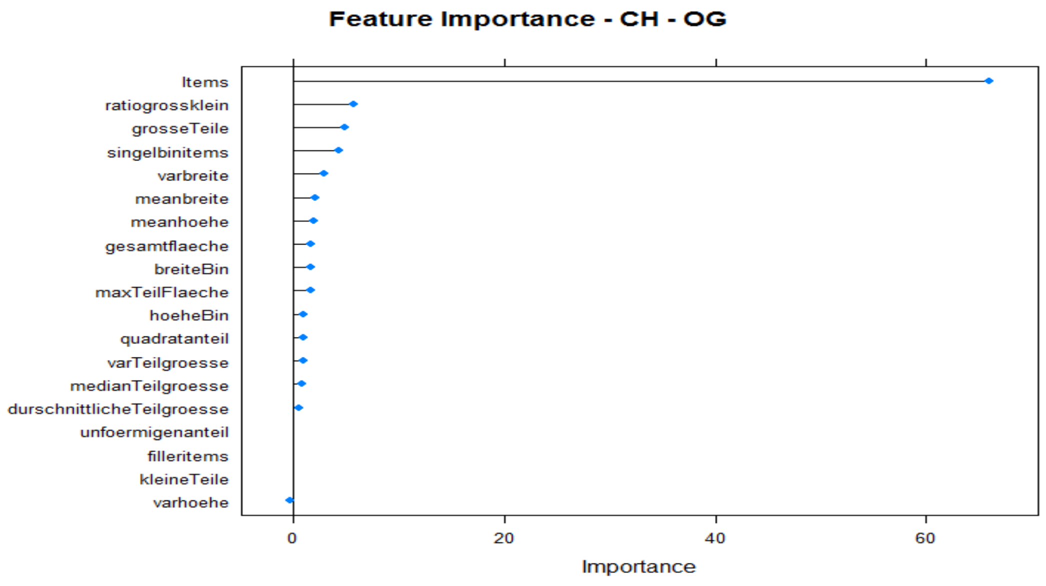


Figure 7.21: Visual overview of the feature Importance for the runtime predictions models. Each plot represents the most important features for the selected machine learning method of the 2BPOG case - CH heuristic.

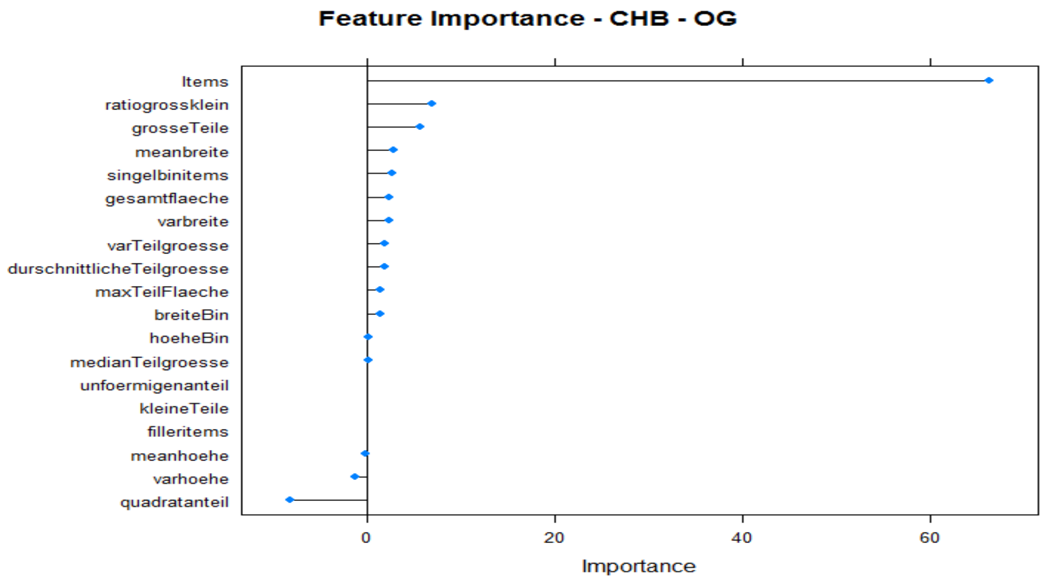


Figure 7.22: Visual overview of the feature Importance for the runtime predictions models. Each plot represents the most important features for the selected machine learning method of the 2BPOG case - CHB heuristic.

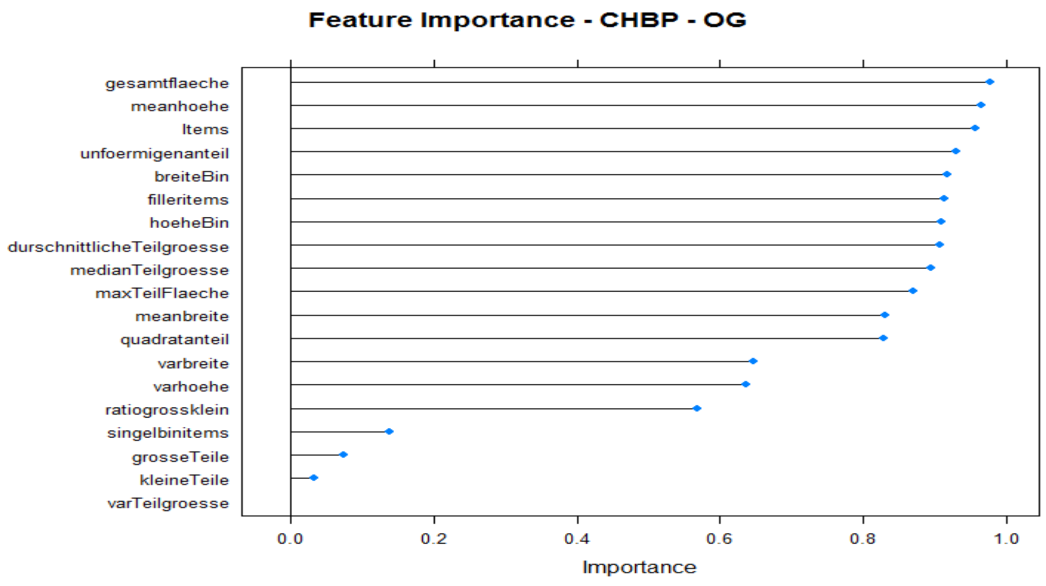


Figure 7.23: Visual overview of the feature Importance for the runtime predictions models. Each plot represents the most important features for the selected machine learning method of the 2BPOG case - CHBP heuristic.

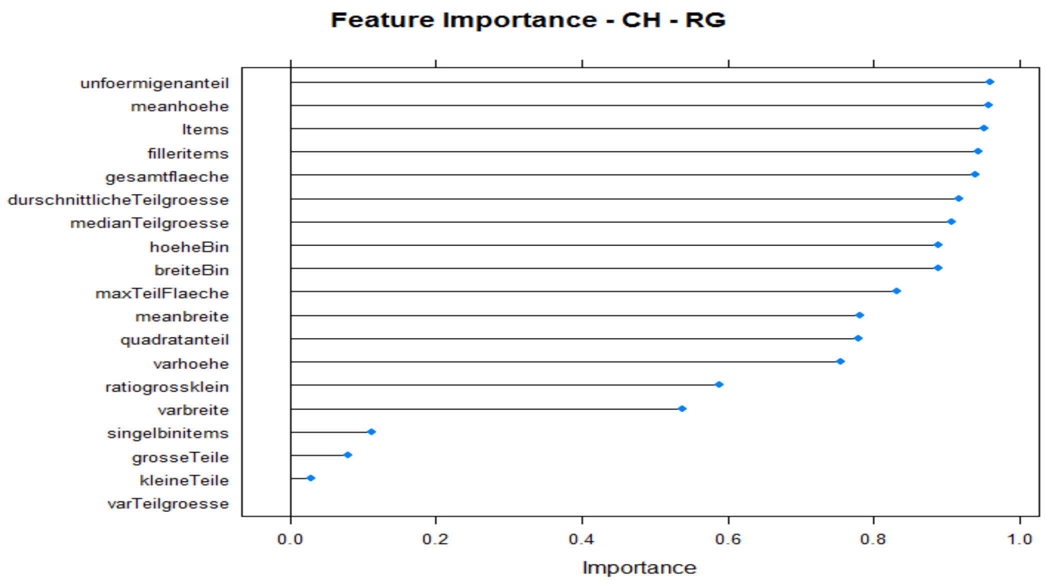


Figure 7.24: Visual overview of the feature Importance for the runtime predictions models. Each plot represents the most important features for the selected machine learning method of the 2BPRG case - CH heuristic.

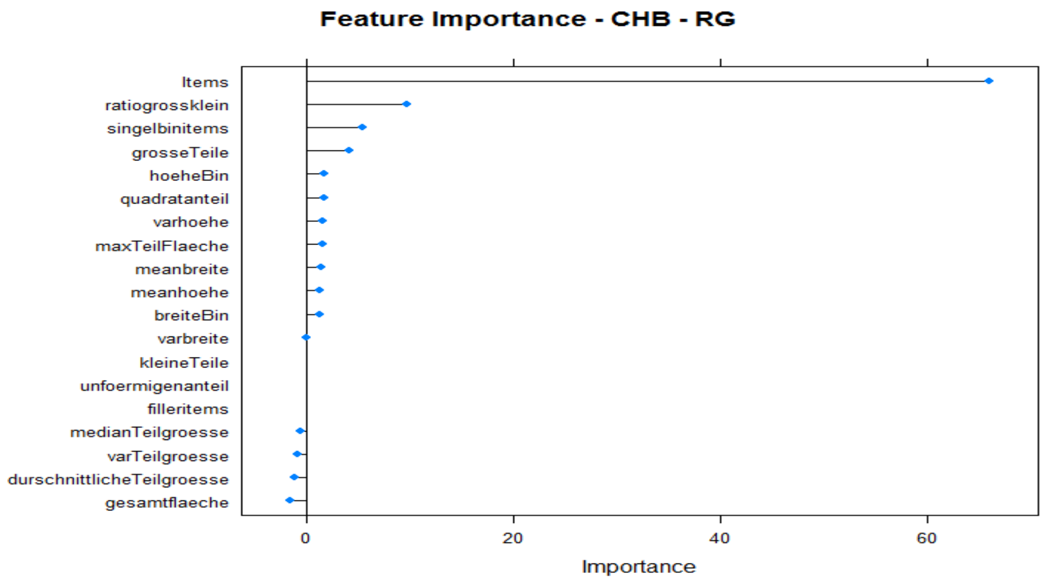


Figure 7.25: Visual overview of the feature Importance for the runtime predictions models. Each plot represents the most important features for the selected machine learning method of the 2BPRG case - CHB heuristic.

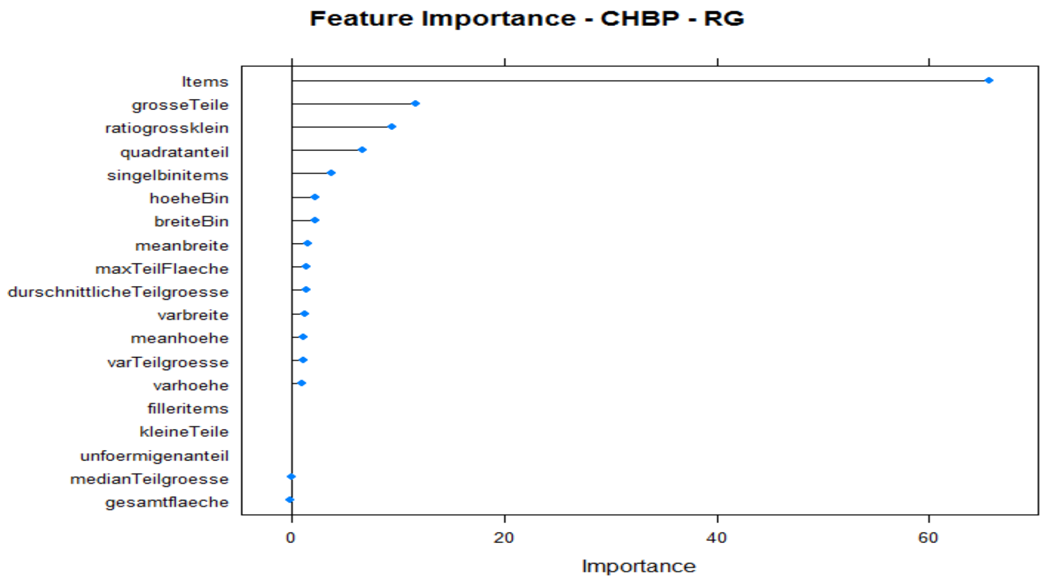


Figure 7.26: Visual overview of the feature Importance for the runtime predictions models. Each plot represents the most important features for the selected machine learning method of the 2BPRG case - CHBP heuristic.

CHAPTER 8

Conclusion and Future Works

This thesis featured the algorithm selection and algorithm runtime prediction approach for the 2-dimensional bin packing problem(BPP). We focused on the 2BPRG case (where items may be rotated by 90 degrees and guillotine cuts are required) and on the 2BPOG case (where Items are oriented and guillotine cuts are required) of the bin packing problem. For the used machine learning techniques we computed 23 features. Furthermore, we collected experimental results of 10 state of the art bin packing heuristics (**FFIHOGJ04**, **BFIHOGJ04**, **CFIHOGJ04**, **CH**, **CHB**, **CHBP**, **GBL**,**A-B**,**A-B New** and **SVC2BPRG**) on 500 instances of 2 different public available sets of problem instances.

Using the gathered information we trained six classification and regression models for the BPP. We used a series of well-known machine learning techniques like Naive Bayes, Decision Trees, k-Nearest Neighbor, Random Forests, Support Vector Machines, Neural Networks, Generalized Linear Models and the LOESS regression. We expanded the experiments with different parameter settings for the prediction models and investigated the effects of two discretization techniques. Furthermore, we examined the effects of different algorithm sets on the quality of the predictions.

The first lessons we learned was that the algorithm selection approach for the BPP is not beneficial if only the number of bins is used as the main benchmark criteria. The first experiments clearly show that most of the 500 instances are solved best by the at least three of the presented algorithms. These three algorithms are superior to the others. Therefore, one of the essential requirements for the algorithm selection approach is not given. These observations lead to the conclusion that the algorithm selection approach is not helpful for the 2D BPP using only the bins as the performance measure.

Our second finding shows that the results of the individual algorithms clearly provide heterogeneous solutions for the benchmark set using the bins and time as the evaluation criteria for the 2BPRG and the 2BPOG case. Some algorithms performed good but they were not able to dominate the others. Furthermore, many algorithms show a better performance on some subgroups than on the others, which could indicate some hidden structure that influences the performance of the algorithms.

For the algorithm selection approach our experiments showed that the k-Nearest Neighbor, Random Forests, and the Support Vector Machines are the most appropriate machine learning procedures for the algorithm selection based on classification for the 2BPOG case. All models had been tuned using 10- fold cross-validation to find the best setup for each prediction model Additionally, we were able to find a positive effect of using the data processed with discretization methods, especially the Minimum Description Length Principle (MDL) for the 2BPOG case. For the 2BPRG case, the decision trees and the other machine learning methods mentioned above were also able to develop good strategies to solve the problem instances.

The results of our algorithm runtime prediction experiment showed that good results can be accomplished especially by the k-Nearest Neighbor, Random Forests, and the Decision Trees machine learning procedures for both cases. Another interesting observation shows that the Generalized Linear Models and Support Vector Machines procedures produced weaker results than the other methods.

In the final step of this thesis, we evaluated both approaches using the test set. Using the best heuristics sets, machine learning techniques and configuration for the algorithm selection we were able to show that our approach is able to achieve better results than any single heuristic alone. For the algorithm runtime prediction, we were able to produce rather good predictions for the run times of previously unseen problem instances.

Future works for the algorithm selection approach could focus on including further algorithms like exact solvers and newly introduced methods. Another interesting topic would be to experiment with other cases of the bin packing problem to examine if the algorithm selection approach can also be usefully applied.
Another consideration is the use of combined portfolio implementations, which execute more the one heuristic for each problem instance. This approaches could achieve better performances especially for problem instances with many items. For the algorithm runtime prediction, an extension of the benchmark instances would be desirable to better evaluate and test the presented approach. Additional, a broader field of heuristics could help to build a better prediction model. The results of the algorithm selection approach could also be used in a next iteration for the algorithm selection problem to make the process even more optimal.

List of Figures

List of Tables

List of Algorithms

Index

Bibliography

[Aar05] Scott Aaronson. Guest column: Np-complete problems and physical reality. *ACM Sigact News*, 36(1):30–52, 2005.

[Aha92] David W Aha. Generalizing from case studies: A case study. In *Proc. of the 9th International Conference on Machine Learning*, pages 1–10, 1992.

[AKA91] David W Aha, Dennis Kibler, and Marc K Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991.

[Alp14] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2014.

[AO96] Ravindra K Ahuja and James B Orlin. Use of representative operation counts in computational testing of algorithms. *INFORMS Journal on Computing*, 8(3):318–330, 1996.

[AS06] Shawkat Ali and Kate A Smith. On learning algorithm selection for classification. *Applied Soft Computing*, 6(2):119–138, 2006.

[BBM04] Thomas Bartz-Beielstein and Sandor Markon. Tuning search algorithms for real-world applications: A regression tree based approach. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, pages 1111–1118. IEEE, 2004.

[BFOS84] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. Classification and regression trees. wadsworth & brooks. *Monterey, CA*, 1984.

[BGK$^+$95] Richard S Barr, Bruce L Golden, James P Kelly, Mauricio GC Resende, and William R Stewart. Designing and reporting on computational experiments with heuristic methods. *Journal of heuristics*, 1(1):9–32, 1995.

[BGV92] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[BH03] Michael Berthold and David J Hand. *Intelligent data analysis: an introduction*. Springer Science & Business Media, 2003.

[BHK06] Edmund K Burke, Matthew R Hyde, and Graham Kendall. Evolving bin packing heuristics with genetic programming. In *Parallel Problem Solving from Nature-PPSN IX*, pages 860–869. Springer, 2006.

[Bis06] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

[BKK$^+$16] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, et al. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016.

[BLS05] Nikhil Bansal, Andrea Lodi, and Maxim Sviridenko. A tale of two dimensional bin packing. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 657–666. IEEE, 2005.

[BMB13] Mohammad Reza Bonyadi, Zbigniew Michalewicz, and Luigi Barone. The travelling thief problem: the first step in the transition from theoretical problems to realistic problems. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 1037–1044. IEEE, 2013.

[BPC63] Thomas Bayes, Richard Price, and John Canton. *An essay towards solving a problem in the doctrine of chances.* C. Davis, Printer to the Royal Society of London, 1763.

[Bre94] Eric Allen Brewer. *Portable high-performance supercomputing: high-level platform-dependent optimization.* PhD thesis, Citeseer, 1994.

[Bre95] Eric A Brewer. High-level optimization via automated statistical modeling. In *ACM SIGPLAN Notices*, volume 30, pages 80–91. ACM, 1995.

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

[Bro93] Carla E Brodley. Addressing the selective superiority problem: Automatic algorithm/model class selection. In *Proceedings of the tenth international conference on machine learning*, pages 17–24, 1993.

[BRY98] Andrew Barron, Jorma Rissanen, and Bin Yu. The minimum description length principle in coding and modeling. *IEEE Transactions on Information Theory*, 44(6):2743–2760, 1998.

[BS00] Pavel B Brazdil and Carlos Soares. A comparison of ranking methods for classification algorithm selection. In *European Conference on Machine Learning*, pages 63–75. Springer, 2000.

[BS13] Christian Blum and Verena Schmid. Solving the 2d bin packing problem by means of a hybrid evolutionary algorithm. *Procedia Computer Science*, 18:899–908, 2013.

[BSDC03] Pavel B Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003.

[BW87] Judith O Berkey and Pearl Y Wang. Two-dimensional finite bin-packing algorithms. *Journal of the operational research society*, 38(5):423–429, 1987.

[CASDC11] Tak Ming Chan, Filipe Alvelos, Elsa Silva, and JM Valério De Carvalho. Heuristics with stochastic neighborhood structures for two-dimensional bin packing and cutting stock problems. *Asia-Pacific Journal of Operational Research*, 28(02):255–278, 2011.

[CCT15] Yi-Ping Cui, Yaodong Cui, and Tianbing Tang. Sequential heuristic for the two-dimensional bin-packing problem. *European Journal of Operational Research*, 240(1):43–53, 2015.

[CDG88] William S Cleveland, Susan J Devlin, and Eric Grosse. Regression by local fitting: methods, properties, and computational algorithms. *Journal of econometrics*, 37(1):87–114, 1988.

[CF11] Christoforos Charalambous and Krzysztof Fleszar. A constructive bin-oriented heuristic for the two-dimensional bin packing problem with guillotine cuts. *Computers & Operations Research*, 38(10):1443–1451, 2011.

[CG91] William S Cleveland and Eric Grosse. Computational methods for local regression. *Statistics and Computing*, 1(1):47–62, 1991.

[CGS92] William S Cleveland, Eric Grosse, and William M Shyu. Local regression models. *Statistical models in S*, 2:309–376, 1992.

[CH67] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[Chi05] Marco Chiarandini. *Stochastic local search methods for highly constrained combinatorial optimisation problems.* PhD thesis, Technische Universität, 2005.

[CJEH07] François Clautiaux, Antoine Jouglet, and Joseph El Hayek. A new lower bound for the non-oriented two-dimensional bin-packing problem. *Operations Research Letters*, 35(3):365–373, 2007.

[CKY08] Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 96–103. ACM, 2008.

[Coo00] Stephen Cook. The p versus np problem. *The millennium prize problems*, pages 87–104, 2000.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[DF00] Glenn De'ath and Katharina E Fabricius. Classification and regression trees: a powerful yet simple technique for ecological data analysis. *Ecology*, 81(11):3178–3192, 2000.

[DIM16] Maxence Delorme, Manuel Iori, and Silvano Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1–20, 2016.

[DJH$^+$08] Piet De Jong, Gillian Z Heller, et al. *Generalized linear models for insurance data*, volume 10. Cambridge University Press Cambridge, 2008.

[DJW02] Stefan Droste, Thomas Jansen, and Ingo Wegener. Optimization with randomized search heuristics—the (a) nfl theorem, realistic scenarios, and difficult functions. *Theoretical Computer Science*, 287(1):131–144, 2002.

[DKS$^+$95] James Dougherty, Ron Kohavi, Mehran Sahami, et al. Supervised and unsupervised discretization of continuous features. In *Machine learning: proceedings of the twelfth international conference*, volume 12, pages 194–202, 1995.

[DL97] Manoranjan Dash and Huan Liu. Feature selection for classification. *Intelligent data analysis*, 1(1-4):131–156, 1997.

[DMV02] Mauro Dell'Amico, Silvano Martello, and Daniele Vigo. A lower bound for the non-oriented two-dimensional bin packing problem. *Discrete Applied Mathematics*, 118(1):13–24, 2002.

[DP97] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2-3):103–130, 1997.

[Dyc90] Harald Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159, 1990.

[DZ02] Michael Doumpos and Constantin Zopounidis. *Multicriteria decision aid classification methods*, volume 73. Springer Science & Business Media, 2002.

[ELH08] Jane Elith, John R Leathwick, and Trevor Hastie. A working guide to boosted regression trees. *Journal of Animal Ecology*, 77(4):802–813, 2008.

[Ewa12] Roland Ewald. *Automatic Algorithm Selection for Complex Simulation Problems.* Springer, 2012.

[FD92] Emanuel Falkenauer and Alain Delchambre. A genetic algorithm for bin packing and line balancing. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 1186–1192. IEEE, 1992.

[FGG97] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.

[FI93] Usama Fayyad and Keki Irani. Multi-interval discretization of continuous-valued attributes for classification learning. 1993.

[Fin98] Eugene Fink. How to solve it automatically: Selection among problem solving methods. In *AIPS*, pages 128–136, 1998.

[Fle13] Krzysztof Fleszar. Three insertion heuristics and a justification improvement heuristic for two-dimensional bin packing with guillotine cuts. *Computers & Operations Research*, 40(1):463–474, 2013.

[FSVdV07] Sándor P Fekete, Jörg Schepers, and Jan C Van der Veen. An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, 55(3):569–587, 2007.

[Fun89] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989.

[GACVO09] L Gonzalez-Abril, Francisco Javier Cuberos, Francisco Velasco, and Juan Antonio Ortega. Ameva: An autonomous discretization algorithm. *Expert Systems with Applications*, 36(3):5327–5332, 2009.

[GE03] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[GH88] David E Goldberg and John H Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[GKK$^+$11] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Torsten Schaub, Marius Thomas Schneider, and Stefan Ziller. A portfolio solver for answer set programming: Preliminary report. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 352–357. Springer, 2011.

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New generation computing*, 9(3-4):365–385, 1991.

[GSCK00] Carla P Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of automated reasoning*, 24(1-2):67–100, 2000.

[GTM$^+$05] Leonardo Garrido, Hugo Terashima-Marín, et al. Building hyper-heuristics through ant colony optimization for the 2d bin packing problem. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 654–660. Springer, 2005.

[Guo03] Haipeng Guo. *Algorithm selection for sorting and probabilistic inference: a machine learning-based approach.* PhD thesis, Citeseer, 2003.

[Hal99] Mark A Hall. *Correlation-based feature selection for machine learning.* PhD thesis, The University of Waikato, 1999.

[HB90] Eric J Horvitz and John S Breese. *Ideal partition of resources for metareasoning.* Knowledge Systems Laboratory, Medical Computer Science, Stanford University, 1990.

[HBZS13] Wei Han, Julia A Bennell, Xiaozhou Zhao, and Xiang Song. Construction heuristics for two-dimensional irregular shape bin packing with guillotine constraints. *European journal of operational research*, 230(3):495–504, 2013.

[HCR$^+$00] Elias N Houstis, Ann C Catlin, John R Rice, Vassilios S Verykios, Naren Ramakrishnan, and Catherine E Houstis. Pythia-ii: a knowledge/database system for managing performance data and recommending scientific software. *ACM Transactions on Mathematical Software (TOMS)*, 26(2):227–253, 2000.

[HDH$^+$99] Adele E Howe, Eric Dahlman, Christopher Hansen, Michael Scheetz, and Anneliese Von Mayrhauser. Exploiting competitive planner performance. In *European Conference on Planning*, pages 62–72. Springer, 1999.

[HHHLB06] Frank Hutter, Youssef Hamadi, Holger H Hoos, and Kevin Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In *International Conference on Principles and Practice of Constraint Programming*, pages 213–228. Springer, 2006.

[HJR$^+$00] Joseph L Hellerstein, TS Jayram, Irina Rish, et al. *Recognizing end-user transactions in performance management.* IBM Thomas J. Watson Research Division, 2000.

[HJY$^+$10] Ling Huang, Jinzhu Jia, Bin Yu, Byung-Gon Chun, Petros Maniatis, and Mayur Naik. Predicting execution time of computer programs using sparse polynomial regression. In *Advances in neural information processing systems*, pages 883–891, 2010.

[HLH97] Bernardo A Huberman, Rajan M Lukose, and Tad Hogg. An economics approach to hard computational problems. *Science*, 275(5296):51–54, 1997.

[HMP10] Pierre Hansen, Nenad Mladenović, and José A Moreno Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

[HN97] James V Hansen and Ray D Nelson. Neural networks and traditional time series methods: a synergistic combination in state economic forecasts. *IEEE transactions on Neural Networks*, 8(4):863–873, 1997.

[Ho95] Tin Kam Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.

[Ho98] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

[HT01] EBCH Hopper and Brian CH Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. *European Journal of Operational Research*, 128(1):34–57, 2001.

[HXHLB14] Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[HY01] David J Hand and Keming Yu. Idiot's bayes—not so stupid after all? *International statistical review*, 69(3):385–398, 2001.

[JKP$^+$94] George H John, Ron Kohavi, Karl Pfleger, et al. Irrelevant features and the subset selection problem. In *Machine learning: proceedings of the eleventh international conference*, pages 121–129, 1994.

[Joh97] George H John. *Enhancements to the data mining process*. PhD thesis, stanford university, 1997.

[Jol02] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[K$^+$95] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Stanford, CA, 1995.

[Kar72] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[KCHS11] Jorge Kanda, Andre Carvalho, Eduardo Hruschka, and Carlos Soares. Selection of algorithms to solve traveling salesman problems using meta-learning1. *International Journal of Hybrid Intelligent Systems*, 8(3):117–128, 2011.

[KGM12] Lars Kotthoff, Ian P Gent, and Ian Miguel. An evaluation of machine learning in algorithm selection for search problems. *AI Communications*, 25(3):257–270, 2012.

[KMS$^+$11] Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In *International Conference on Principles and Practice of Constraint Programming*, pages 454–469. Springer, 2011.

[Knu76] Donald E Knuth. Big omicron and big omega and big theta. *ACM Sigact News*, 8(2):18–24, 1976.

[Kot12] Lars Kotthoff. *On algorithm selection, with an application to combinatorial search problems*. PhD thesis, University of St Andrews, 2012.

[Kot14] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60, 2014.

[KPP04] Hans Kellerer, Ulrich Pferschy, and David Pisinger. Introduction to np-completeness of knapsack problems. In *Knapsack problems*, pages 483–493. Springer, 2004.

[Kri51] Daniel G Krige. A statistical approach to some basic mine valuation problems on the witwatersrand. *Journal of the Southern African Institute of Mining and Metallurgy*, 52(6):119–139, 1951.

[LBNA$^+$03] Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden, and Yoav Shoham. Boosting as a metaphor for algorithm design. In *International Conference on Principles and Practice of Constraint Programming*, pages 899–903. Springer, 2003.

[LBNS02] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *International Conference on Principles and Practice of Constraint Programming*, pages 556–572. Springer, 2002.

[LBNS09] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Empirical hardness models: Methodology and a case study on combinatorial auctions. *Journal of the ACM (JACM)*, 56(4):22, 2009.

[LBV12] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. Selecting classification algorithms with active testing. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 117–131. Springer, 2012.

[LCT03] TW Leung, Chi Kin Chan, and Marvin D Troutt. Application of a mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 145(3):530–542, 2003.

[LD04] John Levine and Frederick Ducatelle. Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, 55(7):705–716, 2004.

[LH13] Chung-Shou Liao and Chia-Hong Hsu. New lower bounds for the three-dimensional orthogonal bin packing problem. *European Journal of Operational Research*, 225(2):244–252, 2013.

[Lin00] James K Lindsey. *Applying generalized linear models.* Springer Science & Business Media, 2000.

[LL+98] Lionel Lobjois, Michel Lemaître, et al. Branch and bound algorithm selection by performance prediction. In *AAAI/IAAI*, pages 353–358, 1998.

[LL00] Michail G Lagoudakis and Michael L Littman. Algorithm selection using reinforcement learning. In *ICML*, pages 511–518. Citeseer, 2000.

[LL01] Michail G Lagoudakis and Michael L Littman. Learning to select branching rules in the dpll procedure for satisfiability. *Electronic Notes in Discrete Mathematics*, 9:344–359, 2001.

[LLP01] Michail G Lagoudakis, Michael L Littman, and Ronald Parr. Selecting the right algorithm. In *Proceedings of the 2001 AAAI Fall Symposium Series: Using Uncertainty within Computation, Cape Cod, MA*, 2001.

[LMM02] Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems: A survey. *European journal of operational research*, 141(2):241–252, 2002.

[LMV99] Andrea Lodi, Silvano Martello, and Daniele Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11(4):345–357, 1999.

[LMV02] Andrea Lodi, Silvano Martello, and Daniele Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123(1):379–396, 2002.

[LS97] Patrick Langley and Stephane Sage. Scaling to domains with irrelevant features. *Computational learning theory and natural learning systems*, 4:51–63, 1997.

[M+97] Tom M Mitchell et al. Machine learning. wcb, 1997.

[McC84] Peter McCullagh. Generalized linear models. *European Journal of Operational Research*, 16(3):285–292, 1984.

[MDC11] Tommy Messelis and Patrick De Causmaecker. An algorithm selection approach for nurse rostering. In *Proceedings of the 23rd Benelux Conference on Artificial Intelligence*, pages 160–166. Nevelland, 2011.

[MN13] Charles E McCulloch and John M Neuhaus. *Generalized linear mixed models.* Wiley Online Library, 2013.

[MPR15] Marco Maratea, Luca Pulina, and Francesco Ricca. Multi-level algorithm selection for asp. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 439–445. Springer, 2015.

[MPW97] Subramani Mani, Michael Pazzani, and John West. Knowledge discovery from a breast cancer database. *Artificial Intelligence in Medicine*, pages 130–133, 1997.

[MS13] Nysret Musliu and Martin Schwengerer. Algorithm selection for the graph coloring problem. In *International Conference on Learning and Intelligent Optimization*, pages 389–403. Springer, 2013.

[MSKH15] Mario A Muñoz, Yuan Sun, Michael Kirley, and Saman K Halgamuge. Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. *Information Sciences*, 317:224–245, 2015.

[MSSS11] Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Non-model-based algorithm portfolios for sat. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 369–370. Springer, 2011.

[MT77] Frederick Mosteller and John Wilder Tukey. Data analysis and regression: a second course in statistics. *Addison-Wesley Series in Behavioral Science: Quantitative Methods*, 1977.

[Mur06] Kevin P Murphy. Naive bayes classifiers. *University of British Columbia*, 2006.

[MV98] Silvano Martello and Daniele Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management science*, 44(3):388–399, 1998.

[Nab02] Ian Nabney. *NETLAB: algorithms for pattern recognition.* Springer Science & Business Media, 2002.

[NB72] John Ashworth Nelder and R Jacob Baker. *Generalized linear models.* Wiley Online Library, 1972.

[NLBH$^+$04] Eugene Nudelman, Kevin Leyton-Brown, Holger H Hoos, Alex Devkar, and Yoav Shoham. Understanding random sat: Beyond the clauses-to-variables ratio. In *International Conference on Principles and Practice of Constraint Programming*, pages 438–452. Springer, 2004.

[NMJ13] Mladen Nikolić, Filip Marić, and Predrag Janičić. Simple algorithm portfolio for sat. *Artificial Intelligence Review*, 40(4):457–465, 2013.

[Nud05] Eugene Nudelman. *Empirical approach to the complexity of hard problems.* PhD thesis, Stanford University, 2005.

[PAVOT10] Francisco Parreño, Ramón Alvarez-Valdés, JF Oliveira, and José Manuel Tamarit. A hybrid grasp/vnd algorithm for two-and three-dimensional bin packing. *Annals of Operations Research*, 179(1):203–220, 2010.

[PM09] Sergey Polyakovsky and Rym M'Hallah. An agent-based approach to the two-dimensional guillotine bin packing problem. *European Journal of Operational Research*, 192(3):767–781, 2009.

[PM14] Josef Pihera and Nysret Musliu. Application of machine learning to algorithm selection for tsp. In *Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on*, pages 47–54. IEEE, 2014.

[PT07] Luca Pulina and Armando Tacchella. A multi-engine solver for quantified boolean formulas. In *International Conference on Principles and Practice of Constraint Programming*, pages 574–589. Springer, 2007.

[Qui86] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[Qui93] JR Quinlan. C4. 5: Programs for empirical learning morgan kaufmann. *San Francisco, CA*, 1993.

[Ras06] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[RFB07] D Ruan, Jeffrey A Fessler, and JM Balter. Real-time prediction of respiratory motion based on local regression methods. *Physics in medicine and biology*, 52(23):7137, 2007.

[Ric76] John R Rice. The algorithm selection problem. *Advances in computers*, 15:65–118, 1976.

[Ris01] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM New York, 2001.

[RKF83] Estelle Russek, Richard A Kronmal, and Lloyd D Fisher. The effect of assuming independence in applying bayes' theorem to risk estimation and classification in diagnosis. *Computers and Biomedical Research*, 16(6):537–552, 1983.

[RNI95] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[RSMBH02] Peter Ross, Sonia Schulenburg, Javier G Marín-Bläzquez, and Emma Hart. Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 942–948. Morgan Kaufmann Publishers Inc., 2002.

[Sch12a] Youssef Hamadi Marc Schoenauer. Learning and intelligent optimization. 2012.

[Sch12b] Martin Schwengerer. *Algorithm selection for the graph coloring problem.* na, 2012.

[SM09] Kate A Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):6, 2009.

[SML12] Kate Smith-Miles and Leo Lopes. Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research*, 39(5):875–889, 2012.

[SMP11] Dharm Singh, Neha Mehta, and Pallavi Purohit. Text based image recognition using multilayer perceptron. *Special issues on IP Multimedia Communications,(1)*, pages 143–146, 2011.

[SMvH11] Kate Smith-Miles and Jano I van Hemert. Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence*, 61(2):87, 2011.

[TBB+91] Sebastian B Thrun, Jerzy Bala, Eric Bloedorn, Ivan Bratko, Bojan Cestnik, John Cheng, Kenneth De Jong, Saso Dzeroski, Scott E Fahlman, D Fisher, et al. The monk's problems a performance comparison of different learning algorithms. 1991.

[TMM+81] DM Titterington, GD Murray, LS Murray, DJ Spiegelhalter, AM Skene, JDF Habbema, and GJ Gelpke. Comparison of discrimination techniques applied to a complex data set of head injured patients. *Journal of the Royal Statistical Society. Series A (General)*, pages 145–175, 1981.

[TS94] BS a Todd and R Stamper. The relative accuracy of a variety of medical diagnostic programs. *Methods of information in medicine*, 33(4):402–416, 1994.

[Vap13] Vladimir Vapnik. *The nature of statistical learning theory.* Springer science & business media, 2013.

[VG08] Allen Van Gelder. Another look at graph coloring via propositional satisfiability. *Discrete Applied Mathematics*, 156(2):230–243, 2008.

[VMOR12] Stefan Voß, Silvano Martello, Ibrahim H Osman, and Catherine Roucairol. *Meta-heuristics: Advances and trends in local search paradigms for optimization.* Springer Science & Business Media, 2012.

[VR13] William N Venables and Brian D Ripley. *Modern applied statistics with S-PLUS.* Springer Science & Business Media, 2013.

[WFHP16] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2016.

[WKQ$^+$08] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.

[WLM$^+$17] Markus Wagner, Marius Lindauer, Mustafa Mısır, Samadhi Nallaperuma, and Frank Hutter. A case study of algorithm selection for the traveling thief problem. *Journal of Heuristics*, pages 1–26, 2017.

[WM$^+$95] David H Wolpert, William G Macready, et al. No free lunch theorems for search. Technical report, Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[WM97] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[Wol96] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

[WOZL13] Lijun Wei, Wee-Chong Oon, Wenbin Zhu, and Andrew Lim. A goal-driven approach to the 2d bin packing and variable-sized bin packing problems. *European Journal of Operational Research*, 224(1):110–121, 2013.

[WvB07] Huayue Wu and Peter van Beek. On portfolios for backtracking search in the presence of deadlines. In *Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on*, volume 1, pages 231–238. IEEE, 2007.

[WW05] Darrell Whitley and Jean Paul Watson. *Complexity Theory and the No Free Lunch Theorem*, pages 317–339. Springer US, Boston, MA, 2005.

[XHHLB07] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla-07: the design and analysis of an algorithm portfolio for sat. In *International Conference on Principles and Practice of Constraint Programming*, pages 712–727. Springer, 2007.

[XHHLB08] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, 32:565–606, 2008.

[XHHLB12] Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Evaluating component solver contributions to portfolio-based algorithm selectors. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 228–241. Springer, 2012.

[XHLB07] Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. Hierarchical hardness models for sat. In *International Conference on Principles and Practice of Constraint Programming*, pages 696–711. Springer, 2007.

[XHLB10] Lin Xu, Holger Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *AAAI*, volume 10, pages 210–216, 2010.

[XHS+12] Lin Xu, Frank Hutter, Jonathan Shen, Holger H Hoos, and Kevin Leyton-Brown. Satzilla2012: Improved algorithm selection based on cost-sensitive classification models. *Proceedings of SAT Challenge*, pages 57–58, 2012.

[YMTS09] Salik R Yadav, Raja Ram MR Muddada, MK Tiwari, and Ravi Shankar. An algorithm portfolio based solution methodology to solve a supply chain optimization problem. *Expert Systems with Applications*, 36(4):8407–8420, 2009.

[YW03] Ying Yang and Geoffrey I Webb. On why discretization works for naive-bayes classifiers. In *Australasian Joint Conference on Artificial Intelligence*, pages 440–452. Springer, 2003.

[ZGZ+11] Zhaoyi Zhang, Songshan Guo, Wenbin Zhu, Wee-Chong Oon, and Andrew Lim. Space defragmentation heuristic for 2d and 3d bin packing problems. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 699, 2011.