FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Algorithm Selection for the Graph Coloring Problem

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Martin Schwengerer
Matrikelnummer e0625209

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Priv.-Doz. Dr. Nysret Musliu

Wien, 18.10.2012

(Unterschrift Verfasser)

(Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Algorithm Selection for the Graph Coloring Problem

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Martin Schwengerer
Registration Number e0625209

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Priv.-Doz. Dr. Nysret Musliu

Vienna, 18.10.2012

(Signature of Author)

(Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Martin Schwengerer
Schönbrunnerstraße 293/10/8, 1120 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.

(Ort, Datum)	(Unterschrift Verfasser)

Preface

This is a revised version of the master thesis *Algorithm Selection for the Graph Coloring Problem*.

In the following paragraph, we list the corrections compared to the original version. Insignificant typos and spelling errors are not marked explicitly.

Notation: p. x, t. y means page x, line y from top. Similarly p. x, b. y means page x, line y from bottom.

- **p. 23, b 8:** Changes citation source to [109]. Note that this changes the enumeration of the remaining references.
- **p. 39, first subsection:** We are using *maximal* cliques and not *maximum* cliques as graph feature.

Acknowledgements

First of all, let me note that I don't believe that many people will ever read this thesis. From my experience, I know that especially the acknowledgments are one of the first chapters that everybody skips because of time reasons or just a lack of interest. Nevertheless, I would like to thank some people which supported me in different ways.

First of all, I thank my supervisor for the chance to make my thesis about this interesting topic, inspiring talks and the great assistance.

Furthermore, I thank all researcher who provided implementations of their heuristics for the GCP. Without their support, it would not have been possible to carry out this thesis.

Special thanks go to my parents for their patience and support through the past years. In addition, I thank my brother for pointless, but funny philosophical discussions and various little help.

And last but not least, I would like to thank my girlfriend for so many things that I can not name them all here.

Abstract

The *graph coloring problem* (GCP) is one of the most-studied NP-HARD problems in computer science. Given a graph $G = (V, E)$, the task is to assign a color $c \leq k$ to all vertices $v \in V$ such that no vertices sharing an edge $e \in E$ receive the same color and that the number of used colors, k, is minimal. In the recent years, various heuristic and exact approaches for this problem have been developed. However, all of them seem to have advantages and disadvantages, which highly depend on the concrete instance on which they are applied. Consequently, designing an algorithm which finds on each graph the best coloring is hard or, by analogy to the *No Free Lunch* theorems, even impossible.

One possibility to achieve a better performance is to predict for each instance the algorithm which achieves the best performance. This task is known as *algorithm selection problem*: Given a set of algorithms and a set of intrinsic features of a particular instance, select the algorithm which is predicted to show the best performance on that instance.

This thesis investigates the application of machine learning techniques to automatic algorithm selection for the GCP. For this purpose, we first present several specific features of a graph, which can be calculated in polynomial time. Then, we evaluate the performance of 7 state-of-the-art (meta)heuristic algorithms for the GCP based on experimental results on 1265 graphs of 3 public available instance sets. The results clearly show that none of the algorithms is superior to all others. In addition, we analyze the behavior of these algorithms on classes of instances with certain attributes. The experiments show that for each of these classes, there exists at least one heuristic which performs clearly better than the rest.

In a subsequent step, we use the knowledge about the best-suited algorithm per instance in combination with intrinsic graph features to train 6 classification algorithms. These supervised learning methods are then used to predict for an unseen instance the most appropriate algorithm. For each classifier, we test multiple parameter settings. We further identify relevant subsets of features and investigate the impact of different data-preparation techniques on the performance of the classifiers. In addition, we study the effect of considering only a subset of heuristics on the overall quality of the prediction.

For a meaningful comparison with the underlying heuristics, we evaluate our proposed approach on a new generated set of instances. Our experiments show that algorithm selection based on machine learning is able to outperform all considered solvers regarding several performance criteria.

Kurzfassung

Das *Graphenfärbeproblem* (engl. Graph Coloring Problem (GCP)) ist eines der bekanntesten *NP-schweren* Probleme in der Informatik. Ziel dabei ist es, für einen gegebenen Graphen $G = (V, E)$ jedem Knoten $v \in V$ eine Farbe $c \leq k$ zuzuweisen, sodass keine zwei Knoten, welche mittels einer Kante $e \in E$ verbunden sind, die gleiche Farbe erhalten und dass die Anzahl der verwendeten Farben k minimal ist. Da das Berechnen eine exakte Lösung dieses Problems im schlimmsten Fall eine exponentielle Laufzeit benötigt, wurde im Laufe der Jahre eine Vielzahl an verschiedenen (Meta)Heuristiken für das GCP entwickelt. Viele dieser Methoden weisen gute Erfolge auf, allerdings scheint es, als würden die Ergebnisse sehr oft von der konkreten Instanz abhängen. Dementsprechend ist es schwierig, wenn (in Analogie zu den *No Free Lunch* Theoremen) nicht sogar unmöglich, einen Algorithmus zu finden, welcher auf allen Graphen optimal ist.

Ein Lösungsansatz für dieses Problem wäre, nicht nur einen Algorithmus zu verwenden, sondern, abhängig von der konkreten Instanz, immer den geeignetsten auszuwählen. Bei dieser Herangehensweise, auch bekannt als *Algorithm Selection*, wird aus einer Menge von Algorithmen anhand bestimmter Attribute einer Instanz derjenige ausgewählt, von welchem auf dieser Instanz das beste Ergebnis prognostiziert wird.

Die vorliegende Arbeit befasst sich mit der Anwendung von Techniken des *überwachten Lernens* als Algorithm Selection für das GCP. Für diesen Zweck stellen wir verschiedene relevante Attribute eines Graphen vor, welche in polynomieller Zeit berechnet werden können. Des Weiteren evaluieren wir die Performance von 7 modernen (Meta)heuristiken auf 1265 öffentlich verfügbaren Instanzen. Die Ergebnisse dieser Experimente zeigen deutlich, dass keine der Heuristiken im Allgemeinen besser als jede andere ist. Zudem beweisen die Experimente, dass auf der einzelnen Untergruppen von Instanzen jeweils ein oder mehrere Algorithmen deutlich bessere Leistung als der Rest erzielen.

Im zweiten Teil dieser Arbeit wird die Information über den besten Algorithmus je Instanz mit ihren charakteristischen Attributen kombiniert, um damit 6 verschiedene Klassifikationsalgorithmen zu trainieren. Im Zuge dieser Experimente identifizieren wir erfolgreiche Attributkombinationen und evaluieren, welchen Einfluss verschiedene Attributtransformationstechniken ausüben. Darüber hinaus untersuchen wir, wie eine verringerte Anzahl von Auswahlmöglichkeiten (d.h. das Entfernen von Algorithmen aus der Menge an Lösungsalgorithmen) die Qualität der Vorhersagen verändert.

Im letzten Teil vergleichen wir die Performance eines Systems basierend auf Algorithm Selection mit den zugrundeliegenden Heuristiken auf einer Menge eigens erstellter Instanzen.

Diese Experimente zeigen eindeutig, dass Algorithm Selection in allen betrachteten Kriterien bessere Ergebnisse als die einzelnen Algorithmen erzielen kann.

Contents

CHAPTER 1

Introduction

In computer science, there are some problems which arise more frequently and are, consequently, well-investigated. One of these is the prominent graph coloring problem (GCP), which is one of Karp's NP-COMPLETE problems [139]. This problem has its origins in the coloring the countries of maps such that no neighboring countries receive the same color. In this context, a coloring of a graph $G = (V, E)$ is an assignment of a color $c \leq k$ to all vertices $v \in V$ of the graph such that no adjacent vertices $u, v \in V : (u, v) \in E$ receive the same color and that the number of used colors, k is minimal. Although this sounds easy, finding such a coloring with only a limited number of colors can be very hard. Even more, due to its NP-completeness, it is unlikely (unless P $=$ NP) that there exist exact strategies which require less than exponential time to color an arbitrary graph. As a result, much focus has been spent on the development of (meta)heuristics approaches for the problem. These methods do not longer ensure optimal solutions, but return good colorings in a reasonable time.

For the GCP, various algorithms have been developed, starting with *greedy* algorithms [29, 157] to more sophisticated techniques from the area of *(meta)heuristics* like *local search* or *genetic algorithms*. Some of the most popular solvers in this context base on a Tabu search [122, 20], *variable neighborhood* [123, 11] or *iterated local search* [51]. Other methods built on *genetic algorithms* [90, 203] or *ant colony optimization* [17, 216, 75].

However, the different algorithms often show varying performance on different classes of instances. For the practical usage, this raises some new problems, as deciding which method is the best is not trivial and heavily depends on the concrete application. Even more, by analogy to the *No Free Lunch* theorems [244], it is highly probable that there exist no heuristic that is on all instances better than the rest. Thus, for an optimal solving of a particular instance, it would be beneficial to know in advance which method is the most appropriate one.

This problem of selecting the most suitable algorithm for a concrete instance is also known as Rice's *Algorithm Selection Problem* [214]. It is an important problem, especially for practical issues, as it allows more efficient solving and prevents worst-case scenarios. On the other hand, it is also very interesting from a theoretical point of view. Knowing the circumstances under which an algorithm performs good (or poor) may allow to develop new strategies and leads to more insights on the hardness of a problem.

In the recent years, various methods and applications for algorithm selection have been developed. One emerging technique in this context, that addresses this problem from a rather empirical point, is the usage of *machine learning*. These methods learn from given data relevant patterns to draw new conclusions (or in our case, predict the best algorithm).

In this thesis, we will consider the problem for selecting the best heuristic for the GCP using machine learning techniques. In detail, we will investigate if, *under the assumption that no heuristic is superior to all others, it is possible to extract from an instance of the GCP specific attributes which allow to predict the most appropriate heuristic*.

For this purpose, we will show that the assumption holds by an empirical investigation on the performance of 7 state-of-the-art heuristics for the GCP on instances of 3 different, public available sources. We will further identify characteristic attributes for an instance of the GCP and use this knowledge to train several classifiers for predicting on a new, unseen instance the best algorithm.

Finally, we will compare the overall performance of solvers based on algorithm selection with the underlying heuristic algorithms.

1.1 Objectives

The objectives of this thesis are:

- Identification of modern algorithms used for graph coloring and evaluation of their performance on a representative set of instances.
- Identification of important features of an GCP instance that have an impact on the performance of algorithms.
- Investigation of the application of different machine learning techniques for automated algorithm selection based on features of a specific instance.
- Comparison of the performance of overall solvers based on algorithm selection with other algorithms for the GCP.

1.2 Main Results

The main results of this thesis are:

- We investigated the performance of 7 state-of-the-art algorithms for the GCP on 1265 instances of 3 different, public available sources. Experimental results showed that none of the algorithms dominates all others and that some algorithms are more appropriate to instances with certain intrinsic features.
- We identified 78 attributes of GCP based on *clustering coefficient*, *graph size*, *greedy coloring methods*, *local search behavior*, *lower- and upper bounds*, *maximum cliques*, *node degree* and *tree decomposition*.

- We used the results of the algorithm evaluation in combination with the features to train 6 different *classifiers* for predicting the best algorithm on a new, unseen instance. In addition, we study the impact of *data-preparation* for algorithm selection and apply *feature selection* to identify important features.

- Finally, we compare our solvers based on algorithm selection with the single solver for the GCP on a new generated set of instances. The results show that our approach achieves better results than any single algorithm on all considered performance criteria.

1.3 Organization

The organization of this work is as follows: This thesis continues in Chapter 2 with relevant background information on complexity, heuristics, experimental aspects and machine learning. Chapter 3 explains the concept of algorithm selection and its relation to the *No Free Lunch* theorems. In Chapter 4, we present the GCP including solving strategies and popular heuristics. Furthermore, we identify important features of GCP instances and introduce our approach for algorithm selection for the GCP. Chapter 5 describes the design and setup of our experimental part with information about the chosen heuristics, the benchmark instances and the used techniques from the area of machine learning. The results of these experiments and a comparison of the heuristics for the GCP are presented in Chapter 6. In addition, we show in this chapter the performance of overall solvers based on algorithm selection and discuss the impact of different parameter configurations and data-preparation techniques. In Chapter 7, we conclude the results of this thesis and reflect upon further work.

CHAPTER 2

Background

2.1 NP-Problems

In computer science, a lot effort is done to solve different kinds of problems like sorting numbers, calculate complex equations or finding the shortest path for a salesman.

A *problem* in this context is usually a general question to be answered, often in terms of a set of parameters as input and certain variables whose values are unknown [94]. An *instance* of a problem is a particular case of a problem with concrete values for the parameters and *solving* the instance means to specify the unknown variables such that all given constraints are fulfilled.

The different problems in computer science can further be divided into two categories: *decision* and *optimization* ones. The former deal with the single question if something is possible or not, resulting in a YES or NO answer (and the corresponding solution) while the latter focus on finding a "good" (or the best) solution in a set of feasible solutions. Optimization problems can be substituted by decision problems, as for each optimization problem with a solution of quality m, there exists a corresponding decision problem asking if there is a solution of quality m. Consequently, any optimization problem can be replaced by calling the underlying decision problem multiple (linear) times.

Of course, not all problems are of equal difficulty and scientists are always eager to know how long some algorithm $f(n)$ for a particular problem in relation to some size measurement n will take. One method to analyze this behavior which is based on *complexity theory* is the so-called *Landau* notation [142]. This approach investigates how an algorithm performs asymptotically on an instance with respect to some input size n. In detail, this system describes the behavior of a function f with respect to n considering the *worst* ($\mathcal{O}$), the *best* (Ω), and the *average* (Θ) case as follows [142]:

$$\begin{aligned}
\mathcal{O}(g(n)) &= \{f(n) | \exists c \in \mathbb{R}, n_0 \in \mathbb{N} : \forall n \geq n_0 : |f(n)| \leq c \cdot |g(n)|\} \\
\Omega(g(n)) &= \{f(n) | \exists c \in \mathbb{R}, n_0 \in \mathbb{N} : \forall n \geq n_0 : |f(n)| \geq c \cdot |g(n)|\} \\
\Theta(g(n)) &= \{f(n) | \exists c, c' \in \mathbb{R}, n_0 \in \mathbb{N} : \forall n \geq n_0 : c \cdot |g(n)| \leq |f(n)| \leq c' \cdot |g(n)|\}
\end{aligned}$$

where n is some problem-specific variable, $g(n)$ is an arbitrary function and c, c' and n_0 are a constant factors (which are most times ignored).

For example, $f(n)$ is in $\mathcal{O}(n^2)$ means that solving any instance of the size n with $f(n)$ will at most take $c \cdot n^2$ time. Note that this is only an asymptotic upper bound for the run time and some instances may be solvable in a fraction of this value. Nevertheless, knowing this worst-case analysis is very useful for estimating the runtime. Even more, showing that a problem can never be solved in less time than some boundary allows a classification of problems into different "hardness classes".

One of the most prominent distinction in this context is the one between polynomial and non-polynomial algorithms. The former class, *polynomial time algorithms* represent functions which complexity is characterized by a maximal runtime of $\mathcal{O}(p(n))$ for some polynomial p (e.g. n^3 or $\texttt{log}\, n$). These algorithms may still take long time, but their growing rather ensures that they terminate within appropriate limits. Algorithms which runtime can not be bounded by a polynomial are denoted as *exponential time algorithms*, which, as the name implies, need an exponential amount of time to discover a solution. In contrast to the former class, it is unlikely that these algorithms terminate for larger n within reasonable time. Concerning a taxonomy for problems itself, those for which polynomial time algorithms exist are grouped in the class *P* and called *tractable* while those for which no such algorithm has been found are called *intractable*. More formally, the class *P* includes are problems that can be solved on a *deterministic Turing machine* in polynomial time, in contrast to problems of the class *NP*, which can only be solved on a *nondeterministic Turing machine* in polynomial time. An alternative definition for the class *NP* includes all decision problems their solution can be verified in polynomial time. Moreover, problems which are denoted as NP-HARD are at least as hard to solve as any problem in *NP* (but can also be harder to solve), while a problem p is NP-COMPLETE if every other problem in *NP* can be transformed into p in polynomial time.

2.2 (Meta)heuristics

As already mentioned, certain problems are so complex that, even with enormous computational power, solving them exactly may take years. Nevertheless, there exist some approaches to solve NP-HARD problems [47]:

One possibility is that, although the underlying problem is NP-HARD, a particular instance is solvable in polynomial time. Unfortunately, this effect is most times limited to a small subset of instances with certain attributes, which do not occur in practice very often.

Another possibility is to use an exhaustive search and decrease the search effort with techniques like *dynamic programming*, *branch and bound* or *backtracking*. These concepts try to reduce the search space by skipping non-promising areas or reduce re-computations. Alternatively, methods like *mathematical linear programming* or *Lagrangian techniques* can be used to efficiently solve complex subproblems. Although all of these sophisticated methods obtain large improvements compared with enumeration methods, they are still applicable only for smaller instances.

A different, but widely used alternative is stop searching for an optimal solution and instead concentrate on finding good solutions within certain time. These methods, called *(meta)heu-*

ristics, do not longer check all possible variable configurations (the *search space*) but rather perform a limited search among certain areas where good configurations are expected. In detail, these algorithms start from some (random) solution(s) and explore related variable configurations and their objective values, which guide the search process to promising areas. The underlying assumption is that good configurations tend to cluster together, as they often share some variable assignments. Thus, following an improved solution and testing modifications on it may lead to even more optimized variable decisions and better configurations. Consequently, these algorithms return, instead of a verified optimal solution, the best solution found during their survey among the solution space.

One very important aspect with some heuristic algorithms is the initial solution, as this is the starting point and effects the complete subsequent search [258]. In this context, the *basin of attraction* [198] of a solution is the area (the set of start configurations) in the search space from which a heuristic is guided to this solution. Two other often used terms are *intensification* and *diversification* in a search. The former one stands for the detailed investigation of a specific area in the search space, e.g. by optimizing only a few variables of the current state. The latter one, *diversification*, means a more expanded exploration where the search moves between different areas and different *basins of attraction*.

In the recent years, various (meta)heuristics have been developed and there exist multiple taxonomies for categorizing them (see [258] for more details). For this thesis, we use a classification introduced in [124] which separates the different approaches into *local-search based* and *population based* methods. Figure 2.1 gives an overview on the different approaches using this schema. The former group, local search (LS), contain techniques which take an initial state and try to improve it by exploring related configurations [258]. Therefore, a LS algorithm iteratively generate possible candidates (called *neighborhood*) by applying so-called *moves* on the current solution and chooses the most promising one as its as new current state. This process repeats until some termination condition, like a time limit or a lack of improvement, is fulfilled. Critical elements in a LS are the neighborhood and the acceptance criteria. Concerning the latter, the algorithm usually look for states which are better (lower in case of a minimization problem) than the current one. Two prominent paradigms therefore are *first-improvement*, which accepts the first improved solution and *best-improvement*, which selects the best solution among the whole neighborhood. However, it can be the case that no move result in a better state than the current one, so the algorithm could not choose an improving configuration and the search is stuck in a *local optima*. This is of course bad for the performance of the search, so researchers invented different techniques to escape local optima like accepting worse solutions (e.g. *simulated annealing* [140]) or adding a memory (e.g. *tabu search* [103]). Concerning the neighborhood itself, it is clear that a small neighborhood increases the speed of the procedure, as only a few candidate configurations have to be considered. The downside is that this also decreases the chance to find a good solution, and the search is more likely to get stuck in a *local optima*. On the other side, a too large neighborhood leads to a slow search that considers many unnecessary states.

The second central group are *population-based* methods. These are often inspired by natural processes like evolution (e.g. *genetic algorithms* [105]) or behavior of animals (e.g. *ant colony optimization* [72]). In contrast to LS, *population-based* methods consider several pos-

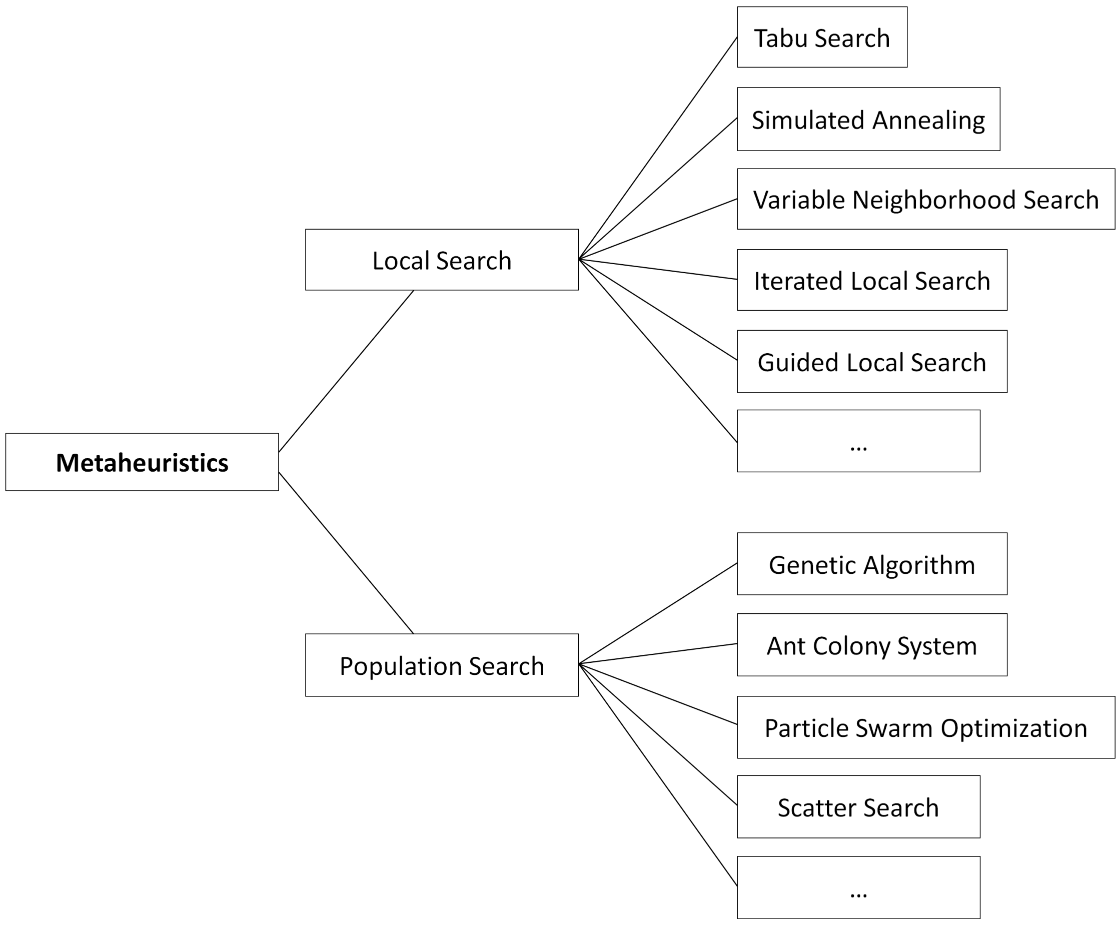

Figure 2.1: Classification of (meta)heuristics based on [258].

sible solutions at once and try to evolve them in a simultaneous manner [258]. This should help to avoid local optima and provide a more robust search. In addition, many state-of-the-art *population-based* methods are complemented with a LS to so-called *memetic algorithms* [185]. These systems try to combine the strength of both paradigms by enhancing the strong diversification abilities of maintaining multiple solutions with the intensification capabilities of the LS.

2.3 Experimental Aspects

This thesis aims to show that knowledge about the individual behavior of heuristics for some type of instance can be used to predict the best algorithm for a new, unseen instance. For this purpose, it is mandatory to examine the efficiency of the different algorithms with respect to some instance-specific attributes. Following an empirical paradigm, this is usually done in terms of a comprehensive study on the performance of the heuristics on a representative set of instances.

Unfortunately, there exist only limited appropriate data for the GCP that we could use to investigate the behavior of the algorithms. For that reasons, we were forced to carry out our own experiments. Such experiments should follow systematic criteria to allow neutral and fair conclusions about the different algorithms. For this reason, we present in the following paragraphs considerations and existing work on methodical testing and the comparison of (meta)heuristics.

2.3.1 Testing and Comparing Metaheuristics

In the field of algorithmic, huge effort has been spent on the asymptotic analysis of algorithms in the context of complexity theory. Apart of this theoretical approach, the experimental investigation of (meta)heuristics has long time been neglected [109]. However, since the 90ties, the interest on the latter subject increased, resulting in several work on methodical evaluation [13, 127, 134, 183, 14, 19]. Worth mentioning in this context is [211] where a complete tutorial and hints for the representation of results are given. Also recommendable is [218] that contains an excellent chapter on runtime comparison.

In general, we followed these guidelines for our experimental design. Nevertheless, we want to describe some aspects and considerations for our particular test setting.

2.3.2 Performance Measurement

When evaluating the performance of (meta)heuristics, usually several measurements, like the quality of the solution or the required runtime, can be observed. Furthermore, such experiments involve typically multiple instances with different size and constraints, so scientists need methods to compare the different results and draw conclusions about the overall performance. A central issue in this context is to provide instance-independent measures because otherwise, outliers or extreme values may adulterate the results [47].

For the *solution quality*, one possible measurement, which is independent of the instance and normalized, is the relative *distance* (or *error*) to the optimal solution value. More formal, the *distance* $d(c, i)$ on an instance i is defined as $d(c, i) = \frac{|c(i) - c_{\text{opt}}(i)|}{c_{\text{opt}}(i)}$ where $c(i)$ are the costs of the solution returned by the algorithm and $c_{\text{opt}}(i)$ are the optimal costs. Drawback of this approach is that it requires the knowledge of the best possible result, which is often unknown. Alternative values, like using bounds or approximations, are often only weak estimations so in practice, often the best known solution (BKS) replaces the optimum [218]. Another disadvantage of this metric is shown in [259] where some properties of error functions are defined. According to this work, the relative error is not *proper* and the same author introduce a more robust metric which requires knowledge about a reference solution. In detail, their measurement $err(k, i)$ for the cost [1] of the solution c on an instance i is defined as

$$err(k, i) = \frac{c(i) - c_{\text{opt}}(i)}{c'(i) - c_{\text{opt}}(i)}$$

[1]Note that this metric can be applied for any kind of costs. For this thesis, we will use this measurement only concerning the number of colors needed k, for which reason we will denote this metric with $err(k, i)$.

where $c_{\text{opt}}(i)$ represent the optimal and $c'(i)$ the costs of some reference point (e.g. a worse solution). Unfortunately, defining a good reference point is also often hard[2], wherefore [260] modified this approach by replacing $c'(i)$ with the cost of a generic algorithm such as a heuristic or a random solution generator [47].

An alternative to using distance metrics offer so-called *rank-based* methods [47]. Characteristic features of them are that the measurements of the algorithms (like solution quality or time) are transformed into a ranking that displays the relative performance on an instance. Then, depending on its rank, the leading algorithm achieves a descent number of points which are summarized and form the global performance indicator over all instances. Popular versions of this approach are a *classical* ranking where the earned points are distributed linearly (e.g. according to their position)[3] and the *formula one* (F1) method[4] where the received points are $10, 8, 6, 5, 4, 3, 2, 1$, respectively. One inherent benefit of these methods is that they ignore instance-specific differences in the objective function (e.g. an instance with an optimal value of 10 versus an instance with an optimal value of 10000). In addition, they are more robust compared with distance-based metrics, as one outlier does not have so much influence on the average performance. On the downside, all ranking methods comprise a loss of information, as generating rankings neglects the actual distance between algorithms, which may also be relevant.

2.3.3 Runtime Comparisons

As the title of this thesis reveals, we would like to design a system which selects the best *algorithm* for a specific problem. For this purpose, it is necessary to compare the performance (which includes in our case also the *runtime*) of the different alternatives and, as already mentioned, we follow an experimental approach. However, this leads to one problem which many scientists ignore, namely that it is almost impossible to measure the runtime of an algorithm. An algorithm is idea how to solve a problem, represented in an abstract set of commands, orders and variables. Thus, it is just some kind of instruction how to act and only the *implementation* of an algorithm is a concrete entity which can also be executed. Following this definitions, experiments can only be carried out with implementations of algorithms and not with the algorithm itself. Nevertheless, it is common usage in computer science that an algorithm is metonymic with its implementation, although this is not true.

The reason for this precisely distinction is that, though we want to compare algorithms, we have to use their implementations for the experiments. Moreover, these implementations are made in a particular *programming language* using a particular data structure and data types, which can dramatically effect the runtime. Considering these factors, the question arises how to perform fair runtime comparisons, which is one of the hardest fields in the area of experimental analysis.

[2]For the GCP, one possible reference point is the number of nodes, or the highest degree of a node plus one, as these two values are a trivial upper bound for the required number of colors.

[3]This value correlates with the average rank, wherefore we consider in this thesis only the *average rank* and the *standard deviation*.

[4]Based on a system used for Formula one between 2003 and 2009. For more info, see `https://en.wikipedia.org/wiki/Formula_one_points` (last visited 18.10.2012)

In this context, [218] describe what they call *Best Runtime Comparison Solution*, namely that all algorithms are implemented in the same language and that the source code is available so that they can be compiled on the same machine with the same compilation flags. Unfortunately, different researchers use different programming languages and have different programming skills, resulting that often approaches for one problem are encoded in unequal ways. Consequently, is very unlikely to achieve such an optimal setting.

One alternative is to omit the language requirements and use implementations written in different languages. Some people may criticize that this corrupts somehow a comparison of different algorithms as the implementation language affects the program speed. This holds especially for the field of (meta)heuristics, where often a time limit is used as stopping criteria. Indeed, we cannot deny that the choice of the programming language impacts the performance, especially when different paradigms like compiler-based (e.g. C++) versus interpreter-based approaches (e.g. Java, Python) are used. On the other side, the chosen programming language is not the only possible reason for a slow implementation - sometimes it might just be bad programming skills or improper modeling decisions which result in a slower program [67]. Case studies on this topic reveal that the ratio of execution speed of programs between the best and the worst programmer is up to $10:1$ [174].

A different situation appears when it is not possible to obtain the source code (e.g. it is lost, protected by copyright, not published etc.). In these cases, a valid alternative is to reimplement the algorithm in a preferred language. This solution would solve the problem with multiple programming languages without suffering from unequal programming skills of different programmers. Even more, this method allows using similar data structures, solution-checking methods and compilers, which eliminate some major points of critique. Even though this approach sounds promising, we would like to highlight some major downsides.

First of all, reimplementing multiple algorithms is sometimes not possible [218]. Although one of the principles of modern science is reproducibility, it can be very hard to construct exactly the same algorithm from a coarse-grained description in a publication. Often, the algorithm is described roughly to provide an overview, ignoring some small design decisions which may have an huge impact on the final program. We do not want to say that this happens on purpose! This is just an unavoidable result of needed abstraction, information filtering and lack of space.

The second argument against a manual translation to one programming language is a more practical and human problem: Neutrality and Fairness! As already mentioned before, different programmer have different skills and even if all algorithms are reimplemented by the same authors, it is not guaranteed that all algorithms are equally optimized. It is easy understandable that researchers will spend some time in optimizing their own and new algorithm where they know the underlying concepts and structures. In contrast to this, it is much harder to optimize a foreign algorithm where the programmer does not know so much about the basic idea, algorithm and dependencies. Moreover, improving an already published algorithm is less worthwhile, less motivating and sometimes a waste of rare working time. In addition, even if all implemented algorithms received the same amount of time, it is not guaranteed that the resulting programs are equal optimized. There might still be the case that the programmer is just not able to implement the heuristic in the most efficient way.

As the reader can see, both alternatives to the optimal setting involve some drawbacks. For this thesis, we stick to a recommendation of Johnson [134], who suggested to use as efficient implementation as expected for practical usage. Following this principle, we argue that most authors will optimize their own implementation as much as possible (within limited time), always in mind that their implementation has to be competitive to existing solutions. Hence, we expect that most available original implementations are improved to a certain grade of optimization for their used programming language such that the effects of different programming skills can be ignored [5]. Moreover, we argue that it does not matter in which language it is developed, as long as it is competitive with the other approaches. In other words and from a more practical (and economical) point of view: It does not matter in which language the best program is written, as long as it achieves the best solutions.

2.4 Machine Learning

In nature, one essential element of many successful species is their ability to adapt their behavior by *learning* from previous events. It is the capability to use knowledge extracted of past observations to recognize a situation and expect behavior of new entities. This a very powerful and effective ability with incredible impact on any kind of problem solving. For this reasons, many researcher spent much effort to enrich computers with such capabilities. These techniques, grouped under the term *machine learning*, are an interesting branch in the area of *artificial intelligence* (AI).

In this thesis, we want to design a system that should, like an oracle, predict for some problem the best way to solve it. However, forecasting which solution for some new, unseen task is not trivial. It requires knowledge about the different alternatives and a lot of experience. Even more, we want a robust system which is able to gain more insight into the problem than its creators. At this point, *machine learning* comes into play. These techniques allow not only to judge based on given rules, they are also capable of learning them from given observations. Hence, they can autonomously extract knowledge of training data, determine relevant patterns and use this information to predict new data.

For this thesis, we only deal with a small part of machine learning, namely *classification* and *regression*. These two techniques belong to so-called *supervised* learning methods where the training process includes data (a correct solution) given by a well-informed supervisor. In contrast to this, *unsupervised* methods do not have any knowledge about the correct solution.

2.4.1 Classification

The first topic in this context is *classification*. Given some training data T consisting on observations of a set of variables $X = \{x_1, x_2, ..., x_n\}$ for each instance[6] in T. Furthermore, each

[5]This statement does not mean that there are no runtime differences caused by the choice of the programming language. We just assume that the authors of the various programs optimized as good such that we can compare the underlying algorithms and that our measurements are not corrupted by poor programming.

[6]We want to distinguish between an *instance* for a problem (as described in Section 2.1) and an instance in the context of machine learning. The former one is a concrete case of the problem while the latter is a concrete observation of multiple attributes of an entity. Regarding this thesis, the instances of the GCP is usually a graph while its related data (attributes of the graph and algorithm information) are referred as instance in the sense of

instance belongs to a specific class $y \in Y = \{c_1, c_2, ..., c_m\}$. The process of *classification* is then determining for a new observation o the corresponding class y based on the attribute values $\{x_{1_o}, x_{2_o}, ..., x_{n_o}\}$ of o. A *classifier* in this context is a function mapping $h : X \rightarrow y$ from a set of variables X to a class value $y \in Y$ [23].

During the recent years, several different classification algorithms have been designed, following different concepts and formal background. Similar to (meta)heuristics, the *no free lunch* theorems also hold for classification algorithms, claiming that none of these is superior to the others in general [243]. Thus, finding the most suited technique manually is usually hard and requires broad domain knowledge and experience in machine learning. To face this challenge, automatic techniques, like algorithm selection, have been proposed [7, 30]. However, applying this technique here would extend the scope of this thesis so we will limit our research on a few classification algorithms.

For this thesis, we consider 6 popular machine learning methods: k-nearest neighbor (kNN), C4.5 decision trees (DT), random forests (RF), Bayesian networks (BN), support vector machines (SVM), and multilayer perceptrons (MLP). The motivation behind this selection is that these six approaches are popular techniques that have been used successfully for algorithm selection. In addition, they follow different paradigms which allows a more comprehensive comparison with respect to their usability for the GCP.

The following section presents a short overview on the chosen algorithms. For a more detailed description, we refer to the original publications and [8, 242]. For implementation details, see the *Weka* manual [23].

k-Nearest Neighbor

The *nearest neighbor* (NN) decision rule [58] is a very simple classification method based on the assumption that observations, which have closely related attribute values, are often categorized in the same class. NN-based algorithms belong to the class of *instance-based-* (IB) [4] or *lazy-* learning methods. Thus, instead of learning a model of the underlying distribution, these techniques save in the training phase all observations and corresponding classifications. As a consequence, they do not need a (time-consuming) training but with the drawback that all data is stored, which results in a higher space consumption.

Concerning a new, unclassified sample, a NN-based classifier uses the saved knowledge to find a point in the training data which minimizes a distance function of their features, called the *nearest neighbor*, and takes that point's classification as prediction for the new sample. This concept can be extended to consider k stored observations (kNN) where the majority vote of k-nearest neighbors determines the predicted class.

Essential parameters for kNN are the number of neighbors, k, the used distance function and the voting procedure [249]. Small k may increase the influence of noise in the data while a too large value of k may include many neighbors of other classes. Furthermore, kNN classifiers can use different distance functions (e.g. linear or with scaling) and, for $k > 1$, also voting methods using distance-dependent weighting.

machine learning.

Advantages of kNN classifiers are their simplicity and that they provide often good results (e.g. [151]). On the downside, computing the predictions is usually more expensive, as all training instances have to be considered [249].

Decision Trees

Decision Trees [208, 249] are a simple way to represent knowledge in a tree-like structure. The most prominent method is the C4.5 decision trees (DT) [209], a descendant of ID3 [208]. Both techniques base on a divide-and-conquer algorithm where the training set is recursively divided into subsets according to values of a single attributes. This top-down process creates on each level, starting with a set of all training observations, a decision node which splits the examples into multiple subsets depending on their values of a single feature. Then, the process continues on the subsets until either all entries of each subset have the same classification value or some other criteria is fulfilled. In that case, a leave note is created which denotes the subset's classification.

Essential component in this algorithm is the selection of the observed attribute for the current node, as this influences the depth of the search tree and the evaluation performance. Different methods have been invented. The most popular consider the *information gain* [208] or the *gain ratio* [209]. In addition, after the initial tree construction is completed, usually a *pruning phase* occurs starting from the leaves to the root to avoid overfitting [249]. During this process, the algorithm computes for each subtree (a) the estimated error of that subtree and (b) a leave node replacing it. In case that the latter does not exceed the former, the leave node replaces the subtree.

Making predictions on an existing DT is rather simple. Starting on the head node, a search algorithm compares the attribute value of the new observation with the decision rule of the current node and determines the descent node, on which the procedure repeats. When the search reaches a leave node, it stops and returns the classification of that node. Advantages of this method, which can also be implied by a hierarchical set of rules, are that the resulting classifier is intuitive, easy to understand and fast.

Random Forests

random forests (RF) [125, 126] are an extension of C4.5 decision trees (DT) by using so-called *ensemble* methods. Main idea is to connect multiple different decision tree-based classifiers to achieve higher generalization accuracies. For this purpose, t different DT are generated during the training phase by using randomly chosen subsets of the feature space.

For the prediction of an unclassified observation x, the t classifications of x are calculated. This knowledge is used to build for each possible classification c a *discriminant function* $g_c(x)$ which estimates the probability of x belonging to class c. Then, the class c with the highest probability $g_c(x)$ is used as predicted classification.

Advantages of RF are their good performance on high-dimensional data [43], a short training phase and their robustness to overfitting [126]. One disadvantage is that, compared with decision trees, RF are harder to interpret for humans as the prediction is divided among multiple trees and masked by the discriminant function.

Support Vector Machines

support vector machines (SVM) [22, 57, 234] are one of the most popular techniques in machine learning. Originally designed for binary discrimination, SVM use the training data to find the best possible decision boundary in the feature space. This decision function is based on a linear separation of the data points in a metric space using a hyperplane. Usually multiple of these hyperplanes exist, so to ensure a maximal generalization ability, SVM calculate the hyperplane which maximizes the margin between the two classes. This margin is determined by the shortest distance between the closest data points, the so-called *support vectors*, to any point on the hyperplane. In case that the data is not linearly separable, it is allowed that some points are incorrect classified by using a penalty function for finding the correct "soft" margin. Moreover, it is also possible to use non-linear classifiers which map the input vectors into a high-dimensional feature space using *kernel functions* (e.g. high order polynomials or *radial basis* functions).

For a classification of a new observation, the new data point is simply tested on the decision function to see if it belongs to the first or second class. In case of multiple classes, several (binary-splitting) SVM are combined to form a hierarchical decision procedure.

Advantages of SVM are their robustness and their sound theoretical foundation [249]. Furthermore, they require less training data than other algorithms and can handle large numbers of attributes.

Bayesian Networks

One central issue in artificial intelligence (and computer science in general) is dealing with uncertainty. Estimating, handling and reasoning with probabilistic events is often a challenging task and one of the current leading approaches to face this are Bayesian networks (BN) (also called Bayes networks, Bayesian believe networks, causal networks, or probabilistic networks) [56, 120, 23]. These networks are a graphical model to represent a set of random variables U and their (conditional) dependencies.

Formally, a BN is a directed acyclic graph where the nodes are variables with an underlying probability function and the edges represent conditional dependencies between the variables. Variables may be continuous or discrete and an arc from u to v indicates a probabilistic dependency between these nodes where v depends on u. Furthermore, the lack of an edge between two nodes implies that there is no direct dependency between them. The probability for each variable $u \in U$ is defined as $p(u|\text{pa}(u))$ where pa(u) are the *parents* of u and the probability distribution for the set of all variables U is $P(U) = \prod_{u \in U} p(u|\text{pa}(u))$.

For the purpose of classification, a BN with the feature variables X and the classes Y as nodes can be used to estimate the probability of each class, given an observation x. Thus, for all $y \in Y$, it is possible to calculate the probability $p(y|a_1(x), a_2(x), ..., a_n(x))$ and predict the class with the highest probability.

Multilayer Perceptrons

A *neural network* [215] is an artificial construct trying to imitate biological reasoning in human brains. Central elements are single units (called neurons or perceptrons) which are linked within each other by a set of input and output interfaces. Input signals are weighted using a node-specific bias and if this input function exceeds the *activation function* of such a neuron, it is activates and "fires" a signal to all linked nodes, which process this signal and may be activated too. Possible activation functions are *threshold* functions, *sigmoid* functions or *radial basis* functions.

One of the most popular versions of neural networks are multilayer perceptrons (MLP) [234], which are well-suited for classification tasks (e.g. OCR [220]). As the name implies, the characteristic feature of MLP are that the perceptrons are arranged in a layered structure where each node has output links to all other nodes in the following layer. The nodes are typically connected acyclic following a *feed-forward* approach and as activation function most times a *sigmoid* one is chosen. For classification, each feature is used as separate variable of the input layer, followed by multiple *hidden* layers and an output layer representing the class prediction. The training of a MLP is usually done by *back-propagation* that uses the error of a misclassification to adapt the bias weight of the different neurons.

Advantages of MLP are their robustness and that they are suitable for many different tasks. In addition, they can model complex and non-linear dependencies. Disadvantages are their hidden inner structure, which makes them hard to monitor. Furthermore, they may require, compared with other techniques, a large training time.

2.4.2 Regression

So far, we have considered machine learning techniques that classify examples into one of a discrete set of possible categories. A slightly different problem occurs when the predicted variable is outside some predefined classes, e.g. if the target value is a continuous or real-valued number. In such cases, the task is to approximate the underlying function and estimate for a new observation the value of the chosen variable. This is done by finding a *regression function* using different techniques from the field of *regression analysis* [16]. These methods try to model the relation of one or more input variables $x_1, x_2, ..., x_n$ to one variable y. One of the most common variants in this context is *linear regression*. This techniques use a linear model of the connection between the input variables and y. Consequently, the expected value $E(y_j)$ of an new observation j can be modeled as

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

where $i = 1, ..., n$ is an index for the attributes and β_k for $k = 0, ..., n$ are weighting values. Other models inherit more complex functions like *quadratic regression* or *cubic regression*.

For the context of algorithm selection, regression is very interesting because it poses an alternative to classification-based approaches. For this purpose, a system can be trained to learn the relation between instance features and the performance criteria (e.g. the runtime). During the selection process, this program predicts via regression for all algorithms the performance on the current instance, which is then used to rank the algorithms and to determine the best selection. We want to note that this process can also be understood as some kind of classification [32] as it finally results in one discrete value. Nevertheless, it is based on a different principle with other advantages and disadvantages.

2.4.3 Discretization

In the context of machine learning, data (in form of a variable) are usually *nominal* (categorical) or *continuous* (numeric). The former class refers to data taking values of a predefined finite set of possible categories while the latter is only characterized by a linearly ordered range of values. Unfortunately, not all algorithms can handle numeric values (e.g. the *Tree Augmented Naive Bayes* classifier [86]). Some of these techniques have build-in conversion functionality (e.g. C4.5 decision trees) but if not, an explicit *discretization* can be used to transform *continuous* attributes into *nominal*-valued ones. The different discretization techniques can be classified according to their characteristics [74] like *supervised* vs. *unsupervised*, *global* vs. *local*, or *static* vs. *dynamic* methods. *Supervised* methods include the classification information of the instances for their decisions (in contrast to *unsupervised* ones), *global* vs. *local* indicates if the produced partitions are applied to the compete instance space (global) or not (local), and *static* vs. *dynamic* refers to a fixed (static) or variable (dynamic) maximum number of generated intervals.

One of the most trivial discretization is a simple uniform binning of the data, e.g. *Equal Interval Width* or *Equal Frequency Intervals* (both unsupervised, global and static method). An example for a supervised, global and dynamic method is Fayyad & Irani's minimum-descriptive length (MDL) [81] algorithm. For a more comprehensive list of different techniques, we refer to [74].

One important point, which is also worth mentioning, is that discretization is not only beneficial when the algorithm requires categorical data. Research clearly indicate that some algorithms show significant better performance when the data is transformed from numeric to nominal values [74]. For example, experimental studies [256, 100] on *naive Bayes* and kNN classifiers show that both achieve unambiguously better results on discretized data than on continuous ones.

2.4.4 Feature Selection

Basic idea of machine learning is to learn from given examples and predict new knowledge. Information itself is here considered in the form of instances described by various attributes (also called features), and someone might easy come to the conclusion the more information (features) we have of an example, the better for the learning process. Even more, it is reasonable for any kind of learning that there is no such thing as "too much information" (as long as we have the computational power to process it).

Unfortunately, practical results show that for machine learning, this assumption is often not true - more features do not ensure better performance and in some cases, they even impair accuracy of the used classifier [115, 227]. The effects of irrelevant, duplicated or correlated features differ depending on the used classifier, but are usually a longer training time [156] and lower accuracy [131, 156]. Even more, also adding relevant attributes can degrade an algorithm's performance [131].

At this point, the question arises given a set of features F, which subset $S \subseteq F$ should be used to achieve the best performance. And, more interestingly, how should we find this subset?

This process, called *feature selection*, is important subtask for each application of machine learning. Its difficulty bases on the fact that for each set of features F with $|F| = n$ attributes, there exist 2^n different subsets, so even for small n it is impossible to test all subsets. A manual selection could of course reduce the number of candidates, but this requires usually deep domain knowledge and expertise in classification algorithms. Furthermore, it is often even for humans not decidable which attributes to choose. In this context, [132] define three categories of features: *irrelevant*, *weakly* relevant and *strongly* relevant ones. Following this classification, the task is to select all strongly relevant features, no irrelevant features and a subset of weakly relevant features that yields good performance. However, other authors argue that selecting only the most relevant features leads often in a suboptimal results, especially if the variables are redundant [112]. Thus, a subset of *useful* features may contain only a few relevant features with less redundant information.

Regardless of the different notations of relevance, it is widely accepted that the choice of features is in general hard. Fortunately, there exist various automated techniques for feature selection which follow two paradigms [112]. The first one, *subset selection*, aims on finding good subsets that together have good prediction results. These techniques are often based on search algorithms like *forward selection*, *backward selection* or *genetic search* and can further be divided into *filter* and *wrapper* [132, 144] procedures. The former one, *filter* techniques, are general algorithms which require no further knowledge about the used classifiers while *wrapper*-based techniques require the intended classification method and "wrap" it for their search procedure. Both criteria evaluate complete subsets, but while filter-based algorithms decide about the "value" of attributes according to some calculated measurements (e.g. correlation), wrapper just apply the classification algorithm and use its performance as decision criterion. Advantages of the former algorithms are their versatility, as they are not tailored to only one classifier. Moreover they are usually much faster and therefore, cover a wider area of the search space. Benefit of wrapper methods are that their results are sometimes better for the targeted classifier, but with the loss of generality. Examples for filter techniques are *CFS* [115] or LVF [161].

An opposed concept to *subset selection* is to *rank* the attributes according to their individual predictive utility [112]. For this purpose, each feature is tested on some measurement (e.g. *information gain*) and a ranking of all attributes is generated. In a subsequent step, superior features (e.g. exceeding some threshold) are selected while inferior features can be discarded.

Finally, it has to be mentioned that one of the most successful preprocessing can be achieved by the combination of *feature selection* and *discretization* [100].

For more information about feature selection itself, we recommend [64] which gives an overview on the different search strategies. For an introduction of the different methods and general guidelines for feature selection in form of a checklist, we suggest reading [112].

CHAPTER 3

Algorithm Selection

3.1 What is a good algorithm?

In a previous chapter, we have presented some methodology for testing and measuring the performance of algorithms. However, there is still the question what metrics should be tested. Even more: What is a good algorithm? And how to decide if an algorithm is better than another one? Answering these simple questions seem to be very easy, but is that really the case? For this thesis, we would like to find and predict the best algorithm for a specific instance of a problem, so first of all, we have to define what is “the best” or what is “better” [1]. In many areas, it is hard to specify “better” or “the best”. For example concerning cars: Some people would suggest taking the fastest car as the best one. Others judge by the fuel consumption, while some take the price or the horsepower as relevant parameters.

> Of course there is no best possible way to sort; we must define precisely what is meant by “best”, and there is no best possible way to define “best”[Donald E. Knuth, in the context of sorting] [143][p. 181]

In algorithmics, it is a little bit easier as we do not have so much parameters, but still the definition of “better” is always in relation of the observed attribute. For the field of (meta)heuristics, [13] highlights three areas of interest: *computational effort*, *solution quality*, and *robustness*.

[1]Mathematically spoken, we are looking for a *total preorder* of the algorithms. A total preorder is a binary relation $\leq$ over a set X where the following properties hold for $\forall a, b, c \in X$:

$$a \leq a \qquad \text{(reflexivity)}$$
$$a \leq b \wedge b \leq c \rightarrow a \leq c \qquad \text{(transitivity)}$$
$$a \leq b \vee b \leq a \qquad \text{(totality)}$$

Computational effort is the most obvious candidate, and as for nearly all algorithms, interesting measurements are the runtime and the space (memory) consumption. Memory is usually bound by the implementation, the operating system and current machine, which are relative unstable (with respect to different machines), but easily expandable factors. Furthermore, as most (meta)heuristics cover only a limited area of the search space, they require much less space than complete methods and do not exceed today's memory limitations. Moreover, as most important argument, lack of memory can often be substituted by additional computational effort (which means by time). For that reasons, many researchers exclude the space requirements for their comparisons and focus on time requirement, whereby the question occurs how (and in which forms) to measure it. The most natural way is just stopping CPU execution time, a simple and easy-comparable method. Although widely used in practice, some researchers argue that CPU times are inappropriate measures, as they are hardware-dependent and hard to replicate [5]. Alternative methods, include like using the *representative operation counts* [5] base on the number of predefined program calls in the execution of the solver, resulting in a machine-independent indicator for an algorithms performance.

However, time may not be the only relevant factor. As already mentioned, the different problems in computer science can be roughly divided into two classes: *decision* and *optimization* problems. In case of a decision problem, the result of all algorithms is similar and time can be considered as most relevant metric. However, in case of an optimization problem, the new feature of *solution quality* appears, which is most likely correlated to the runtime. Even more, runtime and solution quality are often adversaries, so a better solution usually requires more runtime and more runtime may likely result in a better solution. This could end in the special case that with enough time, the algorithm could cover the whole search space, were it definitely finds the best solution. However, this would require maximal runtime, which is undesirable (and often incomputable) for (meta)heuristics. So it ends up in the question if the additional time required to find a better solution is justifiable compared with the improvements in the evaluation function. This is also closely related with the question of the termination condition (e.g. maximal runtime, number of iterations), which is discussed later. Nevertheless, the interaction between time and solution quality is a crucial issue for any comparison of algorithms dealing with optimization problems.

The third category, *robustness*, describes how well an algorithm scales with different instances of a problem. This includes the need of parameter tuning and the usability on a wide range of instances. Consequently, this measurement describes a behavior on a set of instances rather than on a particular instance. However, algorithm selection primary focuses on instance-based decisions, wherefore robustness-attributes are less relevant. As a result, we almost ignore this metric when comparing different algorithms. Nevertheless, there was one application of some robustness-related attributed in this thesis: During the creation of our training data, we discovered instances where multiple solvers performed best. For algorithm selection itself, it would be sufficient to predict only one of these solvers, but the question arises which algorithm should be used for training. One solution would be to handle this as *multi-label classification* where multiple classes can be assigned to a single instance. Unfortunately, many classifiers can not handle these ambiguous data, for what reasons transformation strategies have been developed [66]. Worth mentioning in this context is [138] where a study on different techniques for an algorithm selection of TSP solvers is presented. However, we follow a different approach by eliminating additional labels and selecting only one algorithm for training. Therefore, we break ties by preferring algorithms that are on average more successful, i.e. that showed an lower average rank, which is a robustness attribute.

Another attribute of algorithms, although often ignored, deals with one central element on many (meta)heuristics- the randomness. Nearly all state-of-the-art methods contain random elements which are used for initialization decisions, branching or just solving ties during the search. Besides the statistical relevance of the numeric values like runtime, this creates an additional feature of algorithms, its *risk* [130]. This central point poses great influence on practical usability, as even the fastest algorithm is inapplicable if it provides the correct solution only with a probability of e.g. 10%. Regardless of a decision or optimization problem, this information, which can mostly be gained via experiments, must be taken into account, either implicit (e.g. by encoding it in other attributes) or explicit as *coincidence value*.

After this short review of possible features, is it now possible to say some algorithm A is better in general than another algorithm B. Of course it is. Someone can easily argument that when algorithm A performs on all observed features $f \in F$ at least as good as B and for at least one attribute $f' \in F$ A is better than B it is always beneficial to use algorithm A and therefore, A is better than B [80][2]. Unfortunately this is seldom the case in the field of (meta)heuristics if $|F| > 1$, as most times the different features are (inverse) related. Thus, an ordering using this scheme is likely to end up with several algorithms denoted as best, each superior in one or a few features. This is called *Pareto optimality* and for further information, we refer to research in this area. For our purpose, this method is inapplicable, as we prefer a distinct (or at least dominant) solution. In this situation, researchers have only a few possibilities. Either they can try to map the different features on one value by using an artificial function, or they have to prioritize the attributes according to their importance. Both cases have the disadvantage that the weight of the features must be specified according to some criteria, which influences the complete ordering. An error or a wrong prioritization may corrupt the entire result of the research. Regardless of the chosen method, in the end there must be a non-ambiguous method, specifying for any two algorithms and given their features, if and which one is better.

3.2 No Free Lunch Theorem

The no free lunch (NFL) theorems by Wolpert and Macready [244, 245] are well-known theorems in optimization heuristics and were "the" disappointment for all researchers trying to create the best and all-dominating (meta)heuristic. Although primarily proposed for natural-inspired

[2]Mathematically spoken:

$$\begin{aligned}
\forall f \in F : f(A) \leq f(B) \quad &\wedge \qquad\qquad (A \textit{ is at least as good as } B)\\
\exists f' : f'(A) < f'(B) \quad &\rightarrow \qquad\qquad (A \textit{ is on } f' \textit{ better than } B)\\
A < B \quad & \qquad\qquad\quad (A \textit{ is better than } B)
\end{aligned}$$

heuristics like genetic search or simulated annealing, they are in general valid for all black-box [3] algorithms for optimization problems. Central statement is that for any two different algorithm A, B where A outperforms B on some cost function f, there must be a different cost function f' where B outperforms A [244]. Even more, Wolpert and Macready show that over the set of all possible problems, the average performance of any pair of algorithms is exactly the same. This is even valid for a random search, so roughly speaking, no algorithm performs in general better than just randomly exploring the search space. Note that here lies the drawback of the theorem - it is only valid over the set of all possible optimization problems. For one concrete problem, the average performance of different methods can vary and nobody will deny that heuristics specially designed for one problem often outperform general search methods [244, 76].

The NFL theorems have been addressed by many researchers [60, 210], leading to the almost no free lunch (ANFL) theorem [76] that introduces some restriction on the complexity of the considered functions. The authors proved that if a search heuristic performs well compared to the average results on some optimization problem with an evaluation function f, it must take advance on some hidden structure of f. This is also confirmed as each search strategy follows some intuition how f looks like, e.g. that inputs with large f-values are very likely located together at some *local optima*. In addition, the authors of the ANFL theorem show that for each function f, it is possible to create a function f' closely related to f which is hard to solve for this particular heuristic.

With the NFL theorem in mind, the reader might ask herself if it is even possible to choose a "good" algorithm for a particular problem and a particular instance, as all methods in general perform equal. Practice shows us clearly, that it is possible. The NFL theorem is only a general statement over all possible evaluation functions (which means all problems), not necessary valid for one particular problem or the individual instances of it. Thus, it is possible that there are classes of instances where a particular algorithm is more suitable than others. However, it is very unlikely that one algorithm is better than all others on all instances. The reason for this is that for many problems, their instances can be *reduced* to instances of other problems. In addition, also many different real world tasks can all be transformed to instances of the same problem. Consequently, although all instances of a problem follow the same semantical structure, they may imply different constraints, needs and requirements. An indicator for this is that for many problems there exist multiple methods to create a particular instance [4].

One result of the ANFL is that an algorithm A performs better if it fits to some underlying, hidden structure of the problem. It is some kind of implicit guessing and using hints in a problem's evaluation function that determines if A is beneficial for that problem (or an particular instance) or not [76]. Starting from this concept, the question arises that if the hidden structure of a problem influences the performance of algorithms, does this statement also hold for instances (or instance classes) of a particular problem. If it is possible to say that some method is better suited for functions (problems) with some features, even if these features are implicit, can we

[3]This means that the function f to be maximized (or minimized) is a black box and the algorithm has no information about the underlying problem.

[4]Regarding graph coloring problem, we refer in this context to [59], where the authors identify different classes of the problem and introduce a parametric instance generator. A different method for generating hard graphs with low chromatic number is presented in [180, 181].

distinct upon the same features to make instance-specific predictions about the performance of an algorithm?

This issue is part of the central question of this thesis (some would say the underlying null-hypothesis):

- *Are there some (hidden) attributes of an instance of a particular problem, the GCP, which indicate the performance of different algorithms?*
- *And under the assumption that no algorithm outperforms all others on all instances, can we use these attributes to predict the "best" algorithm for a new and unseen instance of the GCP?*

But how to choose these algorithms? And how to describe an instance and find relevant instance classes? These crucial questions lead us to a well-known problem, the *Algorithm Selection Problem.*

3.3 The Algorithm Selection Problem

For a particular problem, usually different researchers invent multiple different solution strategies and algorithms, each having advantages and disadvantages. Although this is highly recommendable, it entails some challenges for practical applications as selecting (or finding) the best fitting method may be hard. Even more, the "best fitting" is always related to the particular problem instance and may change as the instance changes.

3.4 Algorithm Selection: Basic Concept

The *Algorithm Selection Problem* postulated by Rice [214] faces this important question, namely given multiple algorithms, which one should be selected to solve a concrete problem instance. A schematic view on algorithm selection is given in Figure 3.1 where x is an instance of the problem space P and w is a performance criteria. The term $f(x)$ represent characteristic features of x obtained by an extrapolation method F. Given this feature, the task is to find a selection mapping S that selects the best algorithm a from the set of candidates A for x with respect to w. Then $P(a, x)$ predicts the performance p of a on x, which is finally mapped by a norm function $g(p, w)$ to the *performance* of the algorithm.

Besides the basic task of selecting the best algorithm for one instance, there exist different paradigms for algorithm selection. A central distinction can be done between *static* and *dynamic* techniques [109] where the former one select one algorithm once while the latter approaches monitor the performance of their selection and may revise their decision later. A special case of dynamic methods is *recursive algorithm selection* [152, 154, 235] where the selection procedure occurs at every recursive call.

A different topic, which often comes together with algorithm selection, are *algorithm portfolios* [106] and their design. Originally, algorithm portfolios have been intended as a collection of algorithms for a particular problem which are executed in *parallel* (concurrently) [130]. However, follow-up work also consider *sequential* (one by one) or *partly sequential* executions [253],

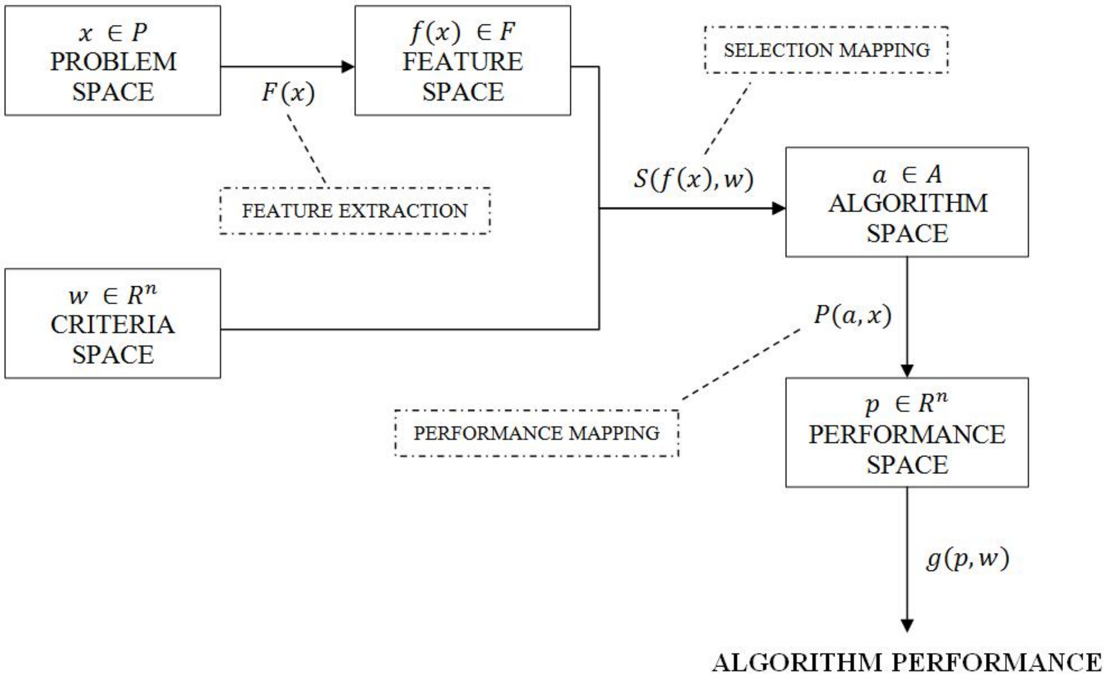

Figure 3.1: Schematic model of algorithm selection [214].

dynamic portfolios [87, 88, 89] or using a *restart* strategy [106]. In general, approaches can further be classified according to an *(a,b)-of-n* naming scheme, denoting that at least a and maximal b of a total of n algorithms are executed [253]. Thus, while the classical algorithm portfolios follow a *(n,n)-of-n* philosophy, other configurations like *3-of-n* or a *1-of-n* (the classical algorithm selection) exist. In these settings, when not all n algorithms are used, it is reasonable that only the best algorithms should be executed, which is just a different variant of the algorithm selection problem.

Moreover, it has to be distinguished between systems where the different algorithms share information between each other (e.g. in a *sequential* and *dynamic* scheduling) or not. The advantages of the former case over classical algorithm selection is that it allows to combine algorithms in a dynamic way. Thus, such a system is able to reach a better performance for an instance than any of the underlying engines alone [172]. In contrast to this, the performance of non-communicative approaches or ordinary algorithm selection is always bounded by the best solver for that particular problem.

In the following paragraph, we will explain some crucial elements for algorithm selection, namely the *feature space* and the *algorithm space*. For the *criteria space*, which considers possible metrics for an algorithm, we refer to the discussions in Section 3.1.

3.4.1 Algorithm Space

Backbone for each successful application of algorithm selection is the collection of algorithms available and in theory, no limits regarding the chosen methods exist. It is possible to mix exact and heuristic methods or even include bad or simple algorithms, as an optimal selection strategy would not use suboptimal methods anyway. The former is especially useful for hard problems where exact methods are only able to solve a subclass of instances (e.g. small ones). In practice of course, it is a little bit more complicated and, although some research has been done (especially in the field of portfolio selection, see [80]), there are no general valid concrete recipes to achieve a perfect mix of algorithms. Often, it ends up with a manual analysis on test instances to identify good candidates [253], which requires high domain and algorithm knowledge. Using this method, first a collection of aspirants is designed which is often limited by available algorithms, existing implementations and time for own developments. Then, meta-information like existing research (e.g. on benchmark instances) or complexity aspects (see Section 3.4.3) can be used for a preselection to remove suboptimal candidates. In a subsequent step, the remaining solving strategies are tested on a representative set of instances, which forms the basis for a manual selection.

3.4.2 Feature Space

Besides the algorithm collection, the choice of features which describe a concrete instance is an essential element for algorithm selection. Unfortunately, there exist no automatic way finding such features [190], as this requires usually good domain knowledge and analytic skills. Nevertheless, some approaches seem to be useful across different problems and sometimes, even features of related problems can be reused [222]. Example for such generic features usual concern metrics about the instance size or the strength (amount) of some constraints. Also using heuristics, approximations or solving subproblems may provide insightful attributes. In this context, we warmly recommend [222] which presents various features for different optimization problems.

In theory, almost anything related to the instance can be used, although not every attribute is always a good feature. [190] suggest two guidelines for good attributes: First of all, features shall be generated by any instance of the problem without any knowledge of the instance construction. This is reasonable, as additional information, like the construction method, is usually not available (as it is not part of the problem) and therefore for a general usage just worthless. The second recommendation is that the computation should be in low-order polynomial time. This is closely related to the *metareasoning-partition problem* [128] and faces the fact that the more time is needed for the meta-reasoning (which also includes feature computation), the less time is left for solving the problem. Thus a time-expensive feature extraction may undo the benefits of an excellent selection of the solving algorithm and should therefore be avoided.

Besides these two issues, there are also other reasons why features are not suitable even if they fulfill these suggestions. For example, attributes which strongly correlate to other features reveal no additional information. The same holds also for attributes taking always the same value or containing just random noise. Such features are useless and can be omitted. However, it is often not possible to determine if a feature is useful or not in advance.

A successful and widely used solution to this problem is to collect a wide range of features and use then different selection techniques to find the most appropriate one. This process is called *feature selection* (see Section 2.4.4) and is a well-studied area in machine learning.

3.4.3 Analytic Algorithm Selection

One method to compare different algorithms is based on the *complexity theory* and works by analyzing the asymptotic behavior for the best/worst/average case of an algorithm [80]. This is usually done using the *Landau* notation (see Section 2.1) which denotes the time or space complexity with respect to some size metric n. For the context of algorithm selection, knowing these characteristic values of all candidates allows to choose the one having the best best/worst/average performance for a given instance with size n. In addition, estimating the constants c, c' and n_0 for each algorithm enables a more precise selection and can prevent worst-case scenarios efficiently.

One benefit of this approach is a good abstraction of the algorithm from implementation details [80]. On the one hand, this brings advantages as different methods can be compared in general and more easily. On the other hand, algorithms may have equal asymptotic behavior, but show dissimilar performance in practice depending on the concrete instance and the implementation [129].

Due to this, analytic analysis and complexity theory is in general not suitable for an automatic algorithm selection [80]. However, complexity theory is still an essential background for many automated algorithm selection methods, which makes it a convenient candidate for a manual preselection during the portfolio design.

3.4.4 Machine Learning for Algorithm Selection

In the previous section, we describe an analytical and theoretical approach for a decision procedure. Unfortunately, it seems that these analysis is often not usable in practice, either because it is very hard to analyze the behavior of the algorithms or the results are not satisfying. An alternative method addresses this problem from a rather empirical point of view. Instead of a detailed mathematical analysis of algorithms, it is sometimes easier to evaluate their performance on different types of instances experimentally. The resulting knowledge can then be used in a subsequent step to decide which algorithm is appropriate for instances with certain characteristics. In theory, this task can be made manually. However, with increasing number of attributes and complexity, it is necessary to use more sophisticated techniques to learn under which circumstances an algorithm is to prefer. This process of learning patterns to predict the performance (or the best algorithm) is exactly the same task that various algorithms in the area of *machine learning* are designed for. Consequently, the wide portfolio of machine learning techniques are an ideal basis for algorithm selection, which is also underpinned by various successful applications [80].

Although it is widely accepted that machine learning provides the most promising techniques, there is an ongoing discussion which of the two major concepts, *classification* or *regression*, is better suited for algorithm selection. In the original work, Rice [214] encourages a structure similar to *decision-trees* and also other early implementations prefer classification-based methods (e.g. [3, 30]). However, recent approaches like [253, 32] built on regression to predict for all algorithms the performance, which is then used to choose the one with the best estimated value. According to its authors, regression is more appropriate than classification, as it allows a more adequate *error metric*. The reason for this is that one global goal of algorithm selection is to increase the average performance, so it is sometimes acceptable to use an algorithm which is nearly as good as the best if it improves the average efficiency. However, classifiers use an error metric that penalizes all misclassification equally, regardless if the predicted algorithm is almost the best or the worst. In contrast to this, the learning procedure for a regression function penalizes a large inconformity more than only a minor mismatch. Thus, a wrong prediction of a classifier results more likely in a larger performance gap to the best algorithm of that instance while using regression minimizes this risk. In addition, using regression does not require an implicit ranking nor the comparison with other algorithms. As a result, these models can be trained once and do not have to be recalculated in case of a new algorithm. Arguments against using regressions are presented in [171], where the authors point out that modeling the execution time is challenging. For example, the estimated values of `SATzilla` [253] differ by up to an order of magnitude. Furthermore, although a runtime prediction is convenient for algorithm selection, it is by no means necessary. It would be sufficient to anticipate the fastest algorithm without having knowledge how long it will take to solve the current instance. Moreover, [111] object that also classifiers can penalize misclassifications differently and that the performance difference between a mismatch of a classifier and the best solution is often small. Similar results are also reported in [178].

Because of these arguments, we believe that it is hardly decidable which approach to prefer as both have advantages and disadvantages. We can not deny that for decision problems, regression poses some benefits, especially regarding the error metric and because it is easily expandable. On the other hand, estimating the runtime is often complicated and not necessary, particularly if it would be sufficient to predict just the best algorithm. Furthermore, using regression for optimization enforces to predict the solution quality and the execution time, which (a) may be inverse correlating and (b) requires twice as much learning effort (and enables twice as much possibilities for mispredictions). Nevertheless, there are for both paradigms multiple state-of-the-art implementations. In this context, we recommend [148] where the authors compare different techniques from the area of machine learning for algorithm selection.

In the following paragraphs, we give an overview on the different approaches to algorithm selection using machine learning. One of the most famous algorithm selection systems is `SATzilla` [253] (and its successor [254]), which is a portfolio-based approach using regression to estimate the best suited SAT-solver. It has won several categories in the 2007 and 2009 SAT competition [252] and is among the state-of-the-art systems for solving this problem. A different system, which also targets SAT, is presented in [171]. In contrast to `SATzilla`, it builds on a kNN classifier to predict an algorithm and achieves, according to its authors, better performance than `SATzilla`. In addition, [137] presents *static* and *dynamic* scheduling strategies for algorithm portfolios of SAT solver. Also mentionable, in this context is [219] which utilizes a technique called *latent class models* to identify groups of similar instances. Another systems for SAT is presented in [189] that applies a *greedy* selection strategy and is, according to its authors, also competitive to `SATzilla`. Besides for SAT, there are of course other problems for which successful applications of algorithm selection exist. For example, the system CLASPFOLIO [98] is a recent approach to select the best solver for answer set programming (ASP) [99] using a *support vector regression*. Also for ASP is ME-ASP [172, 173], which builds on kNN classifier. A comparable framework for quantified Boolean formulas (QBF) is AQME [206, 207], which uses *1-nearest-neighbor*, *Decision Trees*, *Decision Rules*, and *Logistic Regression* to predict the best solver. For the well-known TSP, [223] developed a prediction using a MLP while [138] applied different methods like DT, kNN, SVM and a *naive Bayes network*. Another regression-based system is shown in [32, 31] where algorithm selection for the WDP is presented. Moreover, there exist approaches for the QAP [224], the MPE [110, 111], the NRP [178], the BEP [108, 97]), and scheduling [15]. Also worth mentioning is [182] where classification- and regression-based algorithm selection for ASP based on features of tree decomposition is presented.

Regarding learning itself, there is also various work on selecting the best learning algorithm, e.g. [3, 30, 28, 158, 7]. However, as this is not directly related to our application area, we omit further descriptions. For more information regarding this topic, we refer to [225] where further examples are given.

Besides these systems using classical techniques of *machine learning*, there are also approaches based on other *meta-learning* [242] methods. For example, [111] show in their work for the MPE a successful application of *bagging*, *boosting* and *stacking*.

Other methods built on *Markov decision processes*, e.g. for selecting branching rules in SAT algorithms [153], or for algorithm selection in *sorting* and the OSSP [152]. Still others handle the problem from a statistical point of view [83]. Their proposed solution approximates the performance of the algorithms using regression and applies simple rules for method selection. Also noteworthy is [162], where runtime prediction for *branch and bound* algorithms is used to estimate the best method.

For the field of *dynamic* algorithm selection, [10] and [42] encourage using *reinforcement learning* techniques to observe and revise the selection decision during the search. In this context, we also want to mention work on *dynamic* algorithm portfolios [87, 88, 89] and an application example about selecting QBF solver [217].

Other portfolio-based approaches are [247] for *backtracking* search and [255] for the SCOP. The latter approach is interesting, as it used *analytical hierarchy process* as external decision making tool for selecting the best portfolio. Also worth mentioning in this context is CPHYDRA [191], a very successful system for *constraint programming* which builds on *case-based reasoning* to partition CPU-time between multiple solver.

Finally, we also want to highlight HYDRA [251], a combination of *automated algorithm configuration* and *portfolio-based algorithm selection*. An application with SAT algorithms shows that this method has high potential and is competitive to portfolio-based approaches.

For further information of the state-of-the-art, including historical remarks and interdisciplinary connections, we refer to [225]. In addition, we recommend [147] (and the corresponding online bibliography [146]) where a summary on literature and a classification of the different approaches is presented.

CHAPTER 4

Algorithm Selection for the GCP

In this chapter, we present our algorithm selection approach for the graph coloring problem (GCP). We first explain the GCP and popular heuristics to solve it. Afterwards, we describe several graph features (and ways to compute them) that can be used to characterize a concrete instance of the GCP. Finally, we introduce our approach based on machine learning to automatically select the best algorithm for GCP instances.

4.1 The Graph Coloring Problem

4.1.1 Definition

Graph Coloring is a well-known and often studied problem in computer science. The decision variant, the graph coloring problem (GCP) (sometimes also *vertex coloring problem* or *k-coloring problem*) is one of Karp's NP-COMPLETE problems [139] and its origins go back to the *four color theorem* and the task of coloring real-world maps.

Given an undirected, acyclic graph $G = (V, E)$ where V is a set of vertices (nodes) and E is a set of edges $(u, v) : u, v \in V$ between these nodes. A *coloring* of G is a mapping $\phi : V \mapsto \Gamma$ which labels each vertex with one of k colors $\Gamma = \{1, ..., k\}$ such that no adjacent nodes have the same color. A coloring is denoted as *legal* (or *feasible*) if for all pairs of vertices u, v with $(u, v) \in E$, $\phi(u) \neq \phi(v)$ holds. If there are two connected nodes u, v such that $\varphi(u) = \varphi(v)$, then these nodes are *in conflict* (or *conflict nodes*) and the coloring is said to be *infeasible*.

Alternatively, a coloring can be seen as a partitioning of V into k subsets where all nodes of one subset $S_c, c \in \Gamma$ have the same color c. Then, a coloring is *legal* if for each subsets S_c, no nodes of S_c are adjacent (that is, $\forall u, v \in S_c$, $(u, v) \notin E$ holds). Such sets are also called *independent sets*.

A graph G is called *k-colorable* if there exists a legal coloring with at most k colors, and the minimum number of colors necessary for a legal coloring is denoted as *chromatic number* χ_G.

As many other NP-problems, graph coloring arises as NP-COMPLETE decision and as NP-HARD optimization problem [195]. The former case, the GCP, targets the question if there exist a legal coloring for a given graph G and number of colors k, while the optimization version, the *chromatic number problem*, aims to find the lowest possible number of colors k under which a feasible coloring of G is possible. Note that in literature, the distinction between these names are often not clear and some authors also call the optimization version as graph coloring problem.

Unlike other *NP-complete* problems (e.g. the *Hamilton path problem*), instances of the GCP are "hard on average" [237], meaning that also random instances tend to be difficult to solve. A *landscape analysis* reveals that the search space tend to contain large plateaus [179] which complicate the search process. Other results show that the landscape is mostly *rugged* [9, 149] and that various valleys, peaks and *local optima* are distributed over the whole search space [25]. In this context, several works [119, 2, 149, 12] have been done to investigate under which circumstances a graph is hard to color by investigating the *phase transition* of the GCP. Although is widely accepted that an increasing density for a fixed k tend to result in more difficult instances [61], there is not explicit correlation between graph attributes and hardness known. Consequently, there exists so far no concrete link between the *phase transition* and a specific parameter and the only measurement if an instance is hard is the computational effort to solve it [47]. Moreover, approximating the chromatic number itself is very hard [82], although many different approaches for this task exist (see [195] for more details).

The first coloring algorithms date back to the late 1960s [54, 241] and since that time, many different exact and heuristic methods have been introduced.

Graph coloring itself has many applications. Possible application areas are, besides others, scheduling [157, 101] (e.g. satellite scheduling [261], timetabling problems [241], time-tabling problems in education [35]), register allocation [44], frequency assignment [93], wireless sensor networks [167], bag rationalization [102], or circuit testing [95]. In addition, algorithms for graph coloring are also used to solve other problems like finding bounds for the *maximal clique problem* [194].

4.1.2 Exact Algorithms for the GCP

As already mentioned, the GCP is an NP-COMPLETE problem. Thus, finding the optimal solution may be very time intensive. In detail, calculating the exact chromatic number for a graph with n nodes is, to the best of our knowledge, in $\mathcal{O}((\frac{4}{3} + \frac{3^{\frac{4}{3}}}{4})^n) \approx 2.4150^n$ [78]. Nevertheless, there exist multiple methods for finding an exact solution. While early approaches focus on a depth-first search [54, 239] or *backtracking* [29], recent methods base on *column generation* [175, 107], *integer linear programming* (ILP) [33, 177, 36, 34] or *branch and bound* [38, 104, 176, 117]. Other exact methods perform linear-decomposition of the graph [166] or base on dynamic programming [78].

An alternative approach to these exact methods is to map the GCP to another problem, that is then solved separately. A good candidate in this context is the satisfiability problem (SAT), which is one of the best-studied NP-COMPLETE problems. Moreover, there exist various exact and heuristic solvers for SAT. For more information, we refer to [236] which reviews 12 different encodings of the GCP as SAT and to [233] presents techniques for symmetry breaking. Also worth mentioning is [24], which describes a *learning automata* especially designed for solving the GCP as SAT-representation.

However, all these approaches are only usable in general on small graphs up to 100 vertices [48]. For larger graphs, these algorithms become very time consuming, which inhibits a practical application on these instances. Thus, for obtaining results in reasonable runtime, the use of *heuristic* methods is in general unavoidable.

4.1.3 Heuristics for the GCP

The following section gives a short overview on the different heuristic approaches for the GCP, highlighting their principles, similarities, differences and relations within each other. To begin with, we explain different concepts and representations which highly influence the algorithms and allow a classification of the different approaches. We describe important algorithms divided into three groups: fast, but sometimes inaccurate *greedy* methods; classical *local search* approaches; and *population-based* heuristics. For each of them, we present popular algorithms whereby we focus on those used for the experimental part of this thesis. For more details including historical remarks, we refer to [91]. Other surveys can be found in [170, 118, 160]. An experimental evaluation of different algorithms is presented in [48].

Strategies

As already mentioned, the GCP is a popular and well-known problem in computer science for that many different solving methods have been developed. In the course of this research, some general strategies have been evolved that are used successfully for the GCP [90].

Some algorithms solve the *decision* problem, where the k is fixed, while other algorithms deal with the optimization problem, where the goal is to minimize k. As described before (see Chapter 2), this is a rather arbitrary choice, as the *optimization* variant can be substituted by the *decision* one in linear time. Consequently, most modern heuristics deal with the *decision* version. Besides that, another interesting aspect is if the search works internally on *complete* (*proper* [25]) or *incomplete* (*partial* [20], or *impasse* [169]) colorings. In the former case, at every state in the search all nodes have a color assigned, regardless if the color is allowed or not, while in the latter case, nodes can also be "uncolored". Another facet is if the procedure operates on *feasible* (or *legal*) or *infeasible* colorings. The first method considers only colorings where no adjacent nodes have the same color, in contrast to *infeasible* models, which allow conflict nodes. A different distinction can be made on the representation of the GCP itself - some methods follow the so-called *assignment* approach and work on finding a labeling function which assigns each node a color while others stick to a *partitioning* approach using a set formulation [90]. Although these two models are in principle similar, they offer different neighborhoods and crossover strategies.

In general, it is possible to transform the current state of a search (the current solution) to an alternative model, although this may cause drawbacks in the solution quality. For example a *feasible* and *incomplete* coloring can be converted to a *infeasible* and *complete* coloring by assigning a random color to all uncolored vertices. Such remodeling can be used to escape local optima and find a better coloring. An successful example for such strategy is the `VSS` [123] algorithm, which uses three different representations and predefined translation steps.

In the following paragraphs, we describe some popular algorithms that are also used for our evaluation.

Greedy Heuristics

Greedy (or *constructive*) heuristics are methods which build iteratively a solution based on locally optimal decisions. These algorithms usually show a short runtime, but with the drawback that each decision is final, which implies that there is no backtracking. As a consequence, bad (global) decisions often lead to non-optimal solutions. Nevertheless, many modern heuristics contain constructive algorithms, especially for obtaining initial solutions, e.g. for a following local search (LS) algorithm. Furthermore, these heuristics can easily be modified to generate colorings with a fixed number of colors k by eventually returning illegal solutions, which is also widely used for other (meta)heuristics [48]. In the following paragraphs, we present the most popular greedy algorithms. For a more detailed list, we refer to [150].

Random Order Sequential: The random order sequential (ROS) is one of the simplest algorithm for graph coloring. Given a graph , the algorithm creates a (random) sequence of vertices $S = \{v_1, v_2, \ldots, v_n\}$ and assigns to v_1 the color 1. Then, in the following iterative procedure, all other nodes v_i for $i = \{2, \ldots n\}$ receive the lowest possible color such that for each node v_i no adjacent node v_j ($j < i$) has the same color. It is clear that the quality of this solution highly depends on the order of the nodes and although there exists for any graph an ordering which would produce an optimal coloring[1] , practical results are in general below other constructive methods [48, 52].

DSATUR: As already mentioned, the quality of ROS highly depends on finding a good ordering, which seems to be hard too. Brélaz [29] introduced therefore a new way to rank the nodes based on their *saturation degree*, which is defined as the number of different colors of adjacent nodes. His proposed algorithm, DSATUR (DSAT), uses a similar coloring method as ROS but instead of a fixed ordering of the nodes, it chooses the one with highest saturation degree for coloring. In case of a tie (and for the first node), the node with maximal degree to still uncolored nodes is preferred; and if the tie remain, it is broken randomly. With this dynamic ordering, the algorithm greedily colors vertices with the lowest possible color, resulting in a time complexity of $O(n^3)$ and a space complexity of $O(n^2)$ [52].

Recursive Largest First: The recursive largest first (RLF) [157] algorithm is a greedy method which builds on the extraction of large independent sets. Given a graph, it assigns the color $c = 1$ to a node v with largest degree to the nodes of the set of uncolored nodes U_1. Next, it moves all nodes adjacent to v in a set U_2, containing nodes which cannot be colored with color c. Then, it adds, as long as U_1 is not empty, the vertex $u \in U_1$ with largest degree to nodes in U_2 whereby ties are broken up by preferring minimal node degree. After a node u is chosen (and colored

[1]The proof of this proposition is trivial: Given an optimal coloring for a graph G where for all color classes C_i and C_j with $1 \leq i, j \leq k-1, i < j$, $|C_i| \geq |C_j|$ holds. Then, an ordering of the vertices according to their colors result in a sequence of vertices for which the described coloring method returns an optimal coloring.

with c), its adjacent nodes are moved from U_1 to U_2. When no further nodes can be added, the construction of the color class c is complete and the entire process is repeated using the next available color on the subgraph of G, containing only uncolored nodes U_2. The time complexity of RLF is $O(n^3)$ while its space requirements is $O(n^2)$. Compared with DSAT and ROS, RLF provides on many classes of instances significant better results, but with the drawback of a higher runtime [48]. Improved versions of RLF can be found in [50].

Local Search Heuristics

One downside of *greedy* algorithms are their irreversible decisions, which often lead to colorings with far more colors than the *chromatic* number. One alternative are heuristics based on a local search (LS). Instead of building a valid solution step-by-step, these methods systematically explore the search space to improve their current solution.

Since the formulation of the GCP, various techniques following this paradigm have been introduced. In the following paragraph, we present the most popular one, which are also considered for our experimental part.

TABUCOL: One of the most popular heuristics for the GCP is the `TABUCOL` algorithm by Hertz and Werra [122]. Introduced in the late 80ties, it became de facto the reference algorithm for any new developed method and is often used a LS in memetic algorithms. As the name implies, `TABUCOL` is a classical tabu search that saves applied moves and forbids their revision for a period of iterations. The algorithm works on *infeasible* colorings using a *partitioning approach* where the neighborhood of a current state is obtained by moving a single node u of one class V_i to another class V_j. After one move, the color class i is marked as forbidden for u, except this would result in a configuration better than the best solution found so far (aspiration criteria). Originally, the algorithm works with a fixed tabu tenure of 7, but today's implementation are based on an improved version [90], which uses a dynamic tabu tenure $t = \texttt{random}(A) + \alpha \cdot n_c$ where $\texttt{random}(A)$ returns a random number between 0 and $(A - 1)$, n_c is the number of conflict nodes and α is a parameter. Proposed values for the parameters are $A = 10$ and $\alpha = 0.6$. Furthermore, the improved method restricts the neighborhood move by considering only conflict nodes. The evaluation function is the number of conflict nodes and at the end, the solution with the lowest number of these nodes is returned.

PartialCol: `PartialCol` [20] is a prototype for two heuristics using a tabu list. In contrast to `TABUCOL`, these algorithms work with *feasible*, but *incomplete* colorings. A current solution is represented by the k color classes and a set of uncolored nodes V_{k+1}. The search aims to reduce V_{k+1} to $\{\}$, using $|V_{k+1}|$ as objective function. The neighborhood is defined as moving one node u from V_{k+1} to a color class V_i. In case that this configuration is infeasible, nodes adjacent to u are shifted to V_{k+1} until the solution is legal again. At each iteration, `PartialCol` selects the node u and the color class V_i which results in the smallest $|V_{k+1}|$. In addition, it sets V_i tabu for all vertices removed from V_i for a certain period of time. The authors of `PartialCol` experimented with different calculations of this *tabu tenure*, which result in the two versions, `Dyn-PartialCol` and `Foo-PartialCol`. The former one uses a tenure similar to TABUCOL by just replacing the number of conflict nodes with the size of V_{k+1}, resulting in $t = \texttt{random}(10) + 0.6 \cdot |V_{k+1}|$. The latter approach operates with a *reactive tabu tenure* based on the *fluctuation* of the objective function. If the value of this function is stable over some period of time, the tabu tenure is increased to escape the current search region. To prevent a too high tenure, the variable is slightly evaporated along the search process, resulting that the search is alternating between diversification (with high tenure) and intensification (when the tenure is low).

Iterated Local Search: Another search strategy, which is also used for the GCP, is the iterated local search (ILS), a concept which combines an external LS with an explicit diversification mechanism. Starting from an initial solution, the method first calculates some local optima s^* using the LS. Then, it iteratively uses a perturbation procedure to gain a new state s' from s^*, applies the LS on s' and replaces the current incumbent s^* by the new local optima if some acceptance criteria are fulfilled. This process is then repeated from the (maybe new) configuration s^* until the termination conditions are satisfied.

For the context of the GCP, different implementations of the ILS have been developed [193, 163]. We consider here the version of Chiarandini and Stützle [51, 47]. In this approach, the initial solution is created with the DSAT heuristic followed by a color reduction until the solution is infeasible. As LS, an one-opt neighborhood enhanced with a tabu list, comparable to TABUCOL, is used and the perturbation is applied always to the best solution found so far. For the perturbation itself, the authors suggest a *recoloring*, where a certain number of colors $k_r = \gamma \cdot k$ (where γ is a parameter) are removed and the corresponding nodes are recolored using the DSAT [51] or ROS [47] method. To prevent a stagnation on the current state, it is ensured that the new color of each node is different from the one before the recoloring.

Guided Local Search: guided local search (GLS) [238] is an adaptive heuristic which tries to escape local optima by a dynamic objective function. Instead of using the classical cost function of a problem, it augments the cost function with penalties that are updated every time when the underlying search finds a local optima. Key element is that the penalties are updated on some attributes of the current solution so that the method avoids reentering this state again and is guided to new, promising search regions.

Chiarandini adapted this scheme for the GCP [47]. This algorithm uses weights associated to edges that cause a conflict as penalties which modify the original cost function. Moreover, it uses an one-opt exchange neighborhood for the LS and whenever a local optima is found, it allows $sw = 20$ non-worsening moves (sidewalks) before the weights (and the evaluation function) are updated.

Population Bases Heuristics

Central element of the local search paradigm is that these methods maintain one current solution during the search process, which is improve by exploring related configurations (its neighborhood). A different, but also widely used alternative is to omit this limitation and consider a whole set (or a population) of solutions. These techniques, also denoted as *population-based* heuristics, can also be used to solve the GCP.

Similar to the LS-based methods, there exist various *population-based* algorithms to solve GCP. In the following paragraphs, we introduce several popular state-of-the-art solvers that are also considered for the evaluation.

HEA: One of the first successful evolutionary approaches for the GCP is hybrid evolutionary algorithm (HEA) [90]. This method is a *genetic algorithm* using a LS instead of a *mutation* operator and a specialized crossover operator, called *Greedy Partition Crossover* (GPX). HEA represents solutions using a *partitioning approach* working on complete, but illegal colorings. Starting from an initial population obtained by a modified DSAT algorithm, it iteratively applies the LS followed by a crossover operation and a population update. The LS itself is a *tabu search* similar to TABUCOL, which improves the individual solutions. One factor of the success of HEA is the *GPX* crossover. This operator creates of two parents one offspring by building iteratively new subsets based on the parents subsets with maximum cardinality. Consequently, the offspring inherits large color classes of both ancestor, which form the basis for the following LS.

MACOL: In the article introducing HEA [90], its authors identify that one important element for the success of population-based methods for the GCP is the diversity of the population. This factor balances the search between *diversification* and *intensification* and prevents it from converging to early to a local optima. This phenomena leads to the development of MACOL [164] which uses a similar concept of a *memetic algorithm* as HEA. Instead of DSAT, MACOL, applies a modified version of the DANGER [104] algorithm for the initial population in combination with elements for diversity control: before a new individual is accepted, a distance metric to the already generated colorings is calculated and, in case that the solution is too close to existing ones, the new individual is not accepted. Note that according to the authors of MACOL, using a pure random initial method does not result in worse solutions but may prolong the search slightly. The following LS is again TABUCOL with a *critical one-move* neighborhood. However, instead of the *GPX*, MACOL uses an *Adaptive Multi-Parent Crossover* (AMPaX). This operator differs in two major points to the *GPX*: First, *AMPaX* uses two or more parents per offspring (between 2 and 6, chosen randomly) and second, in each step of the crossover operation, *AMPaX* chooses a parent and a color class adaptively (instead of a successive, alternating way). In addition, after selecting an independent set from one parent, this parent is ignored for a few number of steps to avoid focusing on a single parent. Another difference to HEA is the *Pool Updating* strategy, which balances intensification and diversification of the search on the basis of the *distance* between the quality of existing solutions and the quality of the new one.

MMT: MMT [169] is a two-phase hybrid heuristic based on an evolutionary algorithm and a *set covering* formulation. In contrast to other population-based approaches, the first phase deals with *partial feasible* solutions and is using a *tabu* list with an *impasse class neighborhood* as LS. Crossover is done with a modified *GPX* and to enforce diversity, MMT applies a distance-sensitive pool updating considering for each individual its score (sum of the degree of uncolored nodes) and the number of uncolored nodes. The second phase, called *Column Optimization*, utilizes a *set covering* (or *set partitioning*) formulation of the problem. This is an integer linear programming (ILP) model that is also used for other GCP algorithms [175]. Considering this formulation, MMT applies the *Lagrangian heuristic algorithm* CFT to find further improved solutions.

Multi-agent Fusion Search: The multi-agent fusion search (MAFS) algorithm [250] is an application of a *multi-agent optimization framework* for the GCP. Basic elements are independent *agents* who share information within each other using a communication protocol. Furthermore, the system inherits two forms of knowledge representation: declarative knowledge, which is represented in a symbol structure called *chunks*, and procedural information processes (like an algorithm), which are implemented as *rules*. The fusion search itself is a collaboration of a *recombination search* and a LS, where the former one is responsible for "navigating" through the search landscape while the latter one performs a more intensive low-level search. The agents are initialized based on a dummy solution obtained by DSAT, which is for each individual modified and optimized using the LS. This search itself is again TABUCOL, although also other variants have been tested. The most important parts is the *recombination search*, which is based on a grouping approach similar to HEA or MACOL. To create a new configuration, this method first extracts of the parents' color classes *maximal independent sets*. These sets are then recombined in a *alternate-greedy* way whereby to prevent that the offspring is not to closely related one of its parents, the independent sets are chosen alternatively of the individual ancestors. Finally, redundant vertices are removed by keeping the node with the smallest k value.

Other Approaches

As already mentioned, the GCP is one of the most studied problems in computer science and as a result, various different techniques for solving it have been developed. In the following paragraph, we list different methods in the literature and although we found more than 40 approaches, we have to admit that this list might be incomplete. For a more comprehensive view, we group the algorithms in the following classes.

Methods based on Local Search: Concerning the algorithms based on LS, it is interesting that a large number of heuristics build upon a *tabu* list. Most of these techniques try to enhance the classical tabu search, e.g. by a guiding the search to some regions [202, 204] or adding further reactive elements [45]. Other algorithms focus on an adaptive neighborhood exploration [69] or combine a tabu search with *simulated annealing* [192]. Another interesting hybrid approach in this context is VSS [123], which is build on a tripartite search space using TABUCOL, PartialCol and a third technique alternately for an effective search. A different idea follow HCD [37] and its successor CHECKCOL [41], which assign weights to vertices to escape local optima. There are further approaches based on multi-phased local search (TPA) [40] and IGrAll [39] and applications of a GRASP [155] whereas latter is primary designed for sparse graphs. Moreover, there are implementations using *simulated annealing* [135], *variable neighborhoods* (VNS) [11] or a *very large neighborhood* (VLNS) [231, 49], although the latter is according to its authors less effective. Other heuristics, which we do not describe further, are RCC [71], an *iterated greedy* method [62], a *Novelty* algorithm [47], an approach based on the *minimum-conflict* principle used in constraint satisfaction problem (CSP) [47], and lmXRLF/lsII [141].

Population-based Approaches: Besides the different variants using a LS, there exist a wide range of population-based methods for the GCP. One of the first approach with these techniques built on a classical *genetic algorithm* [65]. Unfortunately, this method does not perform well [91], which leaded to the invention of more sophisticated techniques using for example an *uniform independent set* crossover [73] or no crossover at all [184, 77]. However, the first really successful GAs apply instead of the *mutation* operation, a *local search* to improve the current solution. We have already introduced one of the first successful methods, HEA [90]. More recent algorithms are AMACOL [92], EVOCOL [201], and EVODIV [203], which also include elements for controlling the diversity of the population. Another solver, EXTRACOL [248], reduces the graph by extracting large independent sets before the basic solving procedure. There are further approaches based on *ant colony optimization* [17, 216, 75] and an *ant local search* [200]. Other naturally inspired methods are the *honey bee algorithm* BeesCol [18], *hybrid immune algorithm* [63], *quantum annealing* [228], and using a *gravitational swarm* [212, 213]. Moreover, there are applications of a *scatter search* [116], a combination of a GA with simulated annealing [84], and approaches working with multiple, concurrent applied searches like MEA [45] or [205]. Also worth mentionable are two parallel genetic algorithms [1, 221] which are well suited for multi-core application.

Unconventional Approaches: In the end, we would like to present some unconventional approaches which we could not classify to one of the previous groups. These methods are usually not popular techniques for (meta)heuristics. One group in this context are a couple of solvers based on the combination of *cellular automata* and *learning automata* [229, 230, 6, 79]. In these algorithms, each cell is associated to a vertex and has certain actions, which are trained to find the best action (color) for the graph.

Other approaches use a *neural network* (e.g. [197, 96, 70]) or a *DNA* computation [257]. One hardware-based solution is described in [246] where the authors solve the GCP with the help of *coupled oscillators*.

4.2 Features

One central element for algorithm selection is a representative set of attributes (also called features) that shows individual characteristics of an instance. These features can be used to estimate the *hardness* of an instance. Features must be easy (and fast) to compute and should cover different constraints, aspects and representation of the problem.

For the GCP, we choose the following features based on graph invariants, greedy heuristics and local search elements. For simplicity, we introduce a naming scheme where each feature x belongs to a certain class C and is denoted as C_x.

Feature elements targeting single nodes (e.g. *node degree* or *betweenness centrality*) leading in a statistical population, a frequency distribution (or just a set of values) enforce using *aggre-*

gate functions for a meaningful outcome. For the proposed approach, Table 4.1 displays the used aggregate functions (and corresponding shortcuts). All features are, when suitable, normalized using $n = |E|$, $m = |V|$ or (in case of maximum or minimum of a set of values) by their mean value.

Name	Shortcut	Description
minimum	min	minimum value of the population
maximum	max	maximum value of the population
mean	mean	arithmetic mean of the population
variation coefficient	vc	a normalized measure of dispersion
median	med	median (or second quartile) of the population
first quartile	q25	the value that splits lowest 25% of the data
third quartile	q75	the value that splits lowest 75% (or highest 25%) of the data
entropy	e	entropy in the population

Table 4.1: Aggregate functions for a set of values.

Note that we are aware that some of the following features may seem redundant or useless. Nevertheless, we think that the elimination of features should occur in the process of feature selection based on scientific evidences rather than eliminating based on an educated guess. Therefore, we decided to generate a wide range of features and evaluate their usability in a subsequent step, the *feature selection* (see Section 2.4.4 for general information and Section 5.6 for implementation details).

Note that some of the following features have also been used for other applications of algorithm selection. For example, information about the *degree* of the nodes is used in [253] while [182] also considers attributes of a *tree decomposition* for their selection procedure. Furthermore, [222] describes size-related attributes and recommends the usage of statistical properties in case of a set of values.

Graph Size

This class of features (denotes as S_x) targets the size of the graph represented by the number of nodes $S_n = |V|$ and the number of edges $S_e = |E|$. In addition, we use the ratio between these values $S_{\text{ne}} = \frac{m}{n}$, its multiplicative inverse $S_{\text{en}} = \frac{n}{m}$ and the *density*, which is defined as $S_d = \frac{2 \cdot m}{n \cdot n - 1}$.

Node Degree

The *degree* of a node u is the number of adjacent nodes (the number of edges of u) and therefore a measurement how constrained u is. For the corresponding feature class D_x we take the set of node degrees and calculate the features $D_{\text{min}}, D_{\text{max}}, D_{\text{mean}}, D_{\text{med}}, D_{\text{q25}}, D_{\text{q75}}, D_{\text{vc}}, D_{\text{e}}$ normalized by the number of nodes. Note that D_{mean} is similar to S_{en} and can therefor be omitted. We included it in this listing only for the sake of completeness.

Maximal Clique

A *clique* in a graph $G = (V, E)$ is a subset of vertices $C \subseteq V$ such that for each two vertices $u, v \in C$ there exists an edge $(u, v) \in E$. A *maximal clique* is a clique which cannot be enlarged by adding an additional adjacent vertex without loosing its clique state. This also implies that a maximal clique is not included in any other clique of G.

For our purpose of finding good graph features, the cliques of a graph reveal interesting details about its colorability. First of all, the size (also called cardinality) of each clique forms a lower bound of colors needed, as each node in a clique must receive a different color. In addition, the size and number of the cliques where a particular node u is included may also indicates how strong the adjacent nodes of u are connected with each other. To find large cliques, we designed a greedy construction heuristic that works as follows: Starting from a clique C containing a single node u, it iteratively adds one node v which (a) is adjacent to all nodes of C and (b) shares the highest number of neighbors with all nodes $w \in C$, until no further nodes can be added. Roughly speaking, the algorithm enlarges the clique by selecting the node with the most legal expansion possibilities.

With this algorithm, we can find for each node $u \in V$ as start node a maximal clique $\mathrm{mc}(n)$ (not necessarily the largest one) and its cardinality $|\mathrm{mc}(n)|$ whereby we only keep the latter for further usage. From this set of clique sizes, we calculate the maximal clique features CS_{min}, CS_{max}, CS_{mean}, CS_{med}, CS_{q25}, CS_{q75}, CS_{vc}, and CS_{e} normalized by the number of nodes n. In addition, we add the computation time CS_{time} and the size of the greatest found clique CS_{m} (without normalization) to our feature set. The reason for the latter is that CS_{m} forms a lower bound and therefore may be useful.

Betweenness Centrality

The *betweenness centrality* [85] of a node is a measurement how central a node is within the graph. Given a node u, it is defined as the number of shortest paths from all vertices to all other nodes that pass through u. More formally, the betweenness centrality $g(u)$ of u is

$$g(u) = \sum \frac{\sigma_{s,t}(u)}{\sigma_{s,t}} | s, t \in V, s \neq u \neq t$$

where $\sigma_{s,t}$ is the number of shortest paths from s to t and $\sigma_{s,t}(u)$ is the number of shortest paths that include u.

For calculating the betweenness centrality for all nodes of a graph, we use an algorithm by Brandes [26] with runtime $O(n \cdot m)$, resulting in a set of betweenness centrality values of the nodes. Again, we apply various aggregate functions and normalize these outcome using n,m and the mean value, resulting in the features BC_{e}, BC_{vc}, $\text{BC}^{m}_{\text{min}}$, $\text{BC}^{m}_{\text{max}}$, $\text{BC}^{m}_{\text{mean}}$, $\text{BC}^{m}_{\text{med}}$, $\text{BC}^{m}_{\text{q25}}$, $\text{BC}^{m}_{\text{q75}}$, $\text{BC}^{n}_{\text{min}}$, $\text{BC}^{n}_{\text{max}}$, $\text{BC}^{n}_{\text{mean}}$, $\text{BC}^{n}_{\text{med}}$, $\text{BC}^{n}_{\text{q25}}$, $\text{BC}^{n}_{\text{q75}}$, $\text{BC}^{\text{BC}_{\text{mean}}}_{\text{min}}$, $\text{BC}^{\text{BC}_{\text{mean}}}_{\text{max}}$, $\text{BC}^{\text{BC}_{\text{mean}}}_{\text{q25}}$, $\text{BC}^{\text{BC}_{\text{mean}}}_{\text{q75}}$, and BC_{time}.

As already mentioned, calculating the betweenness centrality requires finding all shortest paths between any two nodes. These paths can further be used to determine the distance between two nodes (called *eccentricity*) which might also be an useful graph invariant (which is described later). As these information is already computed with the algorithm by Brandes, we modified the method slightly to save the corresponding value and gain the eccentricity features for free.

Clustering Coefficient

The *clustering coefficient* is a degree how strongly a graph is clustered together. There are two different kinds of clustering coefficient - a *global clustering coefficient* and a *local clustering coefficient*. The *global clustering coefficient* [165] is the ratio between the number of closed triples (which equals the number of closed triangles multiplied with 3) and the total number of triples where a (closed) *triple* consists of three nodes which are connected with two (three) undirected edges. Formally, it is defined as

$$\mathrm{CC_g} = \frac{3 \cdot \text{no. triangles}}{\text{no. connected triples}} = \frac{\text{no. closed triples}}{\text{no. connected triples}}.$$

The *local clustering coefficient* [240] of a node u indicates how strong the adjacent nodes of u are connected and how close they are to forming a clique. It is defined as the proportion of the number of edges between its neighbors and the maximal possible amount of edges between them. Suppose that u has d_u neighbors and the number of edges between them is

$$e_u = |\{(v, v') : (u, v) \in E, (u, v') \in E, (v, v') \in E\}|,$$

then the *local clustering coefficient* is $C_u = \frac{e_u}{d_u \cdot (d_u - 1)}$. A value $C_u = 1$ means that all adjacent nodes of u are connected within each other while a coefficient of 0 denotes that there exist no edges between the neighbors of u.

For the aspect of finding graph features based on the clustering coefficient CC_x, we use the set of clustering coefficients and calculate the aggregate functions described in the Table 4.1 to receive $\mathrm{CC_{min}}$, $\mathrm{CC_{max}}$, $\mathrm{CC_{mean}}$, $\mathrm{CC_{med}}$, $\mathrm{CC_{q25}}$, $\mathrm{CC_{q75}}$, $\mathrm{CC_{vc}}$, and $\mathrm{CC_e}$. In addition, we also save the time needed for calculating the clustering coefficient in the variable $\mathrm{CC_{time}}$

Although the clustering coefficient is a normalized value and therefore a good statistical measurement, it has one major drawback - it hides the information about the size of the neighborhood of a node. To counter this lack of information, we compute for each node u a *weighted clustering coefficient* by multiplying the clustering C_u with the degree of u. Then, the features $\mathrm{WCC_{min}}$, $\mathrm{WCC_{max}}$, $\mathrm{WCC_{mean}}$, $\mathrm{WCC_{med}}$, $\mathrm{WCC_{q25}}$, $\mathrm{WCC_{q75}}$, $\mathrm{WCC_{vc}}$, and $\mathrm{WCC_e}$, are calculated based on these values using the mentioned aggregate functions.

Eccentricity

This class of features considers the *distance* between the nodes of a graph. Consider two vertices $u, v \in V$ of a graph. The *distance* between u and v is defined as the number of edges on the shortest path between them [113] (or infinite, if no path exists). Then, the *eccentricity* of a node u is the greatest distance between any node $v \in V : v \neq u$, indicating how *far* u is from the most distant node in the graph. The *minimum eccentricity* of a graph G is called *radius*, while the *maximum eccentricity* is the *diameter* of G.

For our purpose as graph metric, we consider the radius ($\mathrm{EC_{min}}$), diameter ($\mathrm{EC_{max}}$) as well as $\mathrm{EC_{mean}}$, $\mathrm{EC_{med}}$, $\mathrm{EC_{q25}}$, $\mathrm{EC_{q75}}$, $\mathrm{EC_{vc}}$, and $\mathrm{EC_e}$. In addition, we normalize the radius, the diameter and the first- and third quartile with the mean eccentricity EC_{mean}, resulting in the variables $\text{EC}_{\text{min}}^{\text{EC}_{\text{mean}}}$, $\text{EC}_{\text{max}}^{\text{EC}_{\text{mean}}}$, $\text{EC}_{\text{q25}}^{\text{EC}_{\text{mean}}}$, $\text{EC}_{\text{q75}}^{\text{EC}_{\text{mean}}}$.

Local Search Features

local search (LS) is a widely-used technique in (meta)heuristics and all of here discussed algorithms for the GCP include some local search component. As a logical consequence, it may be useful to gather performance information of a LS itself. This idea is closely related to the concept of *landmarking*, which is a successful applied approach for algorithm selection (e.g. [196, 159]). Moreover, also `SATzilla` [253] and `CPHYDRA` [191] use features extracted from short runs of simple solvers. Of course an entire and highly sophisticated LS would be too time-consuming, for which reason we only search for the first local optima in combination with a fixed number of iterations and a time limit. For our LS, we use a neighborhood changing the color of one conflict node applied in a best-improvement manner by minimizing the number of conflict edges. One drawback is that each LS needs a (not necessary legal) initial solution, usually created randomly or in a greedy way. For this purpose, we decided to use a greedy procedure with random components that returns for a given number k a coloring with at most k colours. As we want to observe the progress of the LS, we ensure that the found coloring is not legal (for a non-trivial graph) by selecting k_{init} as follows:

$$k_{\text{init}} = \max\{\text{GC}_{\text{best}} \cdot 0.9, k_{\text{LB}}\}$$

whereby GC_{best} is an upper bound obtained by the minimal colors needed of DSAT and RLF and k_{LB} is the lower bound, CS_{m}. As we are using random elements and for more stable results, we execute the search 10 times with different values as random seed. Every time a local optima is found, we store the iteration number (nto), the number of conflict edges (ce) and number of conflict nodes (cn) as graph features. Furthermore, also if no local optima is found, we save at the end of one run the number of conflict edges (cee), number of conflict nodes (cne), total improvement (i)[2] and the improvement per iteration (ii). Based on these values, we calculate the average over the 10 runs, resulting in the attributes LS_{ce}, LS_{cn}, LS_{cee}, LS_{cne}, LS_{nto}, LS_{ii}, and LS_{i}. In addition, we also record the number of local optima found LS_{nlo} and the runtime LS_{t}.

Greedy Coloring Methods

As already mentioned (see Section 4.1.3), there exist two widely-used greedy coloring methods for the GCP. These algorithms, DSAT [29] and RLF [157], are often used as initial procedures for various other heuristic algorithms. Furthermore, they are fast and result in a legal coloring that forms an upper bound, which makes them perfectly suitable for our needs.

For our feature class GC_{x}, we calculate for each of these two algorithms the number of colors needed GC_{DSAT} (GC_{RLF}), and their runtime $\text{GC}_{T-\text{DSAT}}$ ($\text{GC}_{T-\text{RLF}}$). Furthermore, we take the minimum number of colors $\text{GC}_{\text{best}} = \min\{\text{GC}_{\text{DSAT}}, \text{GC}_{\text{RLF}}\}$ and the ratio between them, $\text{GC}_{\text{R/D}} = \frac{\text{GC}_{\text{RLF}}}{\text{GC}_{\text{DSAT}}}$ and $\text{GC}_{\text{D/R}} = \frac{\text{GC}_{\text{DSAT}}}{\text{GC}_{\text{RLF}}}$, as additional attributes.

[2]the number of conflict edges at the beginning minus the number of conflict edges at the end

Beside these very useful attributes, a complete coloring obtained by such a greedy method provides additional benefits - it also includes the construction of independent sets. The reason for this is that in a coloring for a graph, all nodes with the same color assigned form an independent set and as these information comes for free with each colouring, we use it for additional information about the graph (more precisely, the found colouring). For this purpose, we evaluate the size of the different color classes and calculate the features $h_{\text{min}}, h_{\text{max}}, h_{\text{mean}}, h_{\text{med}}, h_{\text{q25}}, h_{\text{q75}}, h_{\text{vc}}, h_{\text{e}}$ ($h \in \{\text{ID}, \text{IR}\}$), which are normalized using the number of nodes n.

Tree Decomposition

Tree decomposition is a mapping of a graph into a tree structure. Formally, a *tree decomposition* of a graph $G = (V, E)$ consists of a tree T such that for each node $t \in T$, there exists a subset of vertices $V_t \subseteq V$. Furthermore, for each of these subsets $V_t, t \in T$ the following statements hold [114]:

- for each vertex $v \in V$ there is some V_t such that $v \in V_t$ (Node coverage)
- for all edges $(u, v) \in E$ there exists some V_t including u and v (Edge coverage)
- for any two sets V_{t_2}, V_{t_2} of the nodes $t_1, t_2 \in T$ containing both a node $v \in G$ ($v \in V_{t_1} \wedge v \in V_{t_2}$) and any third node $t_3 \in T$ lying on the path from t_1 to t_2, the node v is also included in V_{t_3}.

In this context, the *width* of a tree decomposition is the size of its largest set V_t minus one and the *treewidth* of a graph G is the minimum width over all possible tree decompositions of G.

Tree decomposition itself is applied to different problems in mathematic and computer science. Tree decomposition characterizes the difficulty of solving a particular problem, and usually, problems that have small tree width can be solved more efficiently. Unfortunately, finding a tree decomposition with lowest width for a graph is NP-HARD for which reason this is usually computed using heuristic approaches. For our purpose, we use a *minimum-degree* heuristic implemented in the *hypertree library* [68]. Note that the resulting width still can be improved with (meta)heuristic techniques like genetic algorithms [187] or iterated local search [186]. However, these techniques consume usually more time. Therefore, we decided to use *minimum-degree* heuristic to find a decomposition because it is a simple, but very efficient method. Given a tree decomposition for a graph, we use its width normalized by the number of nodes as feature TD_{width}. Moreover, we also store the time needed for the graph decomposition TD_{time}.

Lower- and Upper Bound

This class of features, denoted as B_x, targets the question how much space for improvement is given, starting from a trivial solution as upper bound k_{UB} to a theoretical lower bound k_{LB}. Central idea is that the range between these values may indicate how complex (or easy) it is to find colorings with less than k_{UB} colors. In addition, an upper bound can also be used as initial number of colors for the optimization variant of the GCP as well as starting number for our evaluation of the different algorithms. Note that also some heuristics itself use lower and upper bounds for their calculations (e.g. `MMT` [169]).

For our purpose, we can use other, already introduced features as bounds. For the upper bound, we can either use the maximal node degree $D_{\text{max}} + 1$, or the number of colors needed when using the greedy algorithms DSAT or RLF. As both, DSAT and RLF, always require less or equal $D_{\text{max}} + 1$ colors, we take the value $k_{\text{UB}} = \min\{\text{GC}_{\text{DSAT}}, \text{GC}_{\text{RLF}}\}\text{GC}_{\text{best}}, k_{rlf}\}$ as upper bound.

For the lower bound, one obvious limit for the number of colors is the cardinality of a *maximum clique* (also denoted as clique number $\omega(G)$ of a graph G). Recall that a *maximum clique* is a clique of largest possible size in G, which forms a valid lower bound as each node in these clique must be colored with a different color. Unfortunately, finding the maximum clique (the maximum clique problem) is also NP-HARD [94] and therefore not suitable as feature. Nevertheless, also the cardinality of any other clique represents a lower bound and as we already search for a *maximal clique* $\text{mc}(n)$ for each node n, this data can be used for the lower bound

$$k_{\text{LB}} = \max\{|\text{mc}(n)| : n \in V\}$$

of the chromatic number.

Note that other lower bounds, which are maybe more tight, can be found in [121], although most of them require higher computational effort and are therefore less applicable [168].

Regarding graph features, we use the ratio between the lower and upper bound $B_{\text{lu}} = \frac{k_{\text{LB}}}{k_{\text{UB}}}$, its inverse $B_{\text{ul}} = \frac{k_{\text{UB}}}{k_{\text{LB}}}$, the distance between the bounds normalized by the lower bound $B_{\text{dist}}^{k_{\text{LB}}} = \frac{k_{\text{UB}} - k_{\text{LB}}}{k_{\text{LB}}}$ and upper bound $B_{\text{dist}}^{k_{\text{UB}}} = \frac{k_{\text{UB}} - k_{\text{LB}}}{k_{\text{UB}}}$ as instance attributes.

4.2.1 Remarks

During some preliminary experiments, we discovered that calculating the *betweenness centrality* and the *eccentricity* might require very long time. Although the used algorithms have a worst-case complexity which is polynomial with respect to the graph size, they need up to one hour for their computation (e.g. on very large graphs with 4000 nodes). For this reasons, we removed these features from our collection. Nevertheless, we mention them in this work for the sake of completeness. Moreover, they might contain important information, so in case that faster algorithms (or heuristics) for these features are invented, it is definitely worth making further investigations on these graph features.

4.3 Proposed Approach

According to Rice's original work, algorithm selection consists of the main components:

- the problem space *P*,
- the feature space *F*,
- the algorithm space *A*, and
- the performance space *Y*.

Following this notation, our proposed system targets instances for the GCP (P) (see Section 4.1.1) of which we extract up to 78 different attributes (F) (see Section 4.2). Furthermore, we consider state-of-the-art heuristics for the GCP (A) (see Section 4.1.3) and use the quality of the obtained coloring and the runtime as performance criteria (Y) (see Section 3.1).

For the decision procedure itself, we follow a *empirical* approach using classification algorithms from the area of *machine learning*. These *classifiers* are trained with a representative amount of *training data* consisting of the *features* and the *best algorithm* according to Y for each instance. For predicting the most appropriate heuristic for a new graph, such a system calculates the feature of the graph and uses this information with the trained model to determine the corresponding algorithm.

CHAPTER 5

Experimental Setup and Environment

In this chapter, we describe the environment and setup for our experiments. In the first part, we explain the settings for evaluation of different (meta)heuristic approaches for GCP. In detail, we list the used heuristics, their parameter configuration and the benchmark instances. Afterwards, we describe the setup for experiments with machine learning techniques used for algorithm selection problem. We list the chosen classification algorithms and describe the techniques that were applied for *feature selection* and *discretization*.

5.1 Algorithms for the Graph Coloring Problem

For the experiments, we needed a wide-range of algorithms for the GCP. Therefore, we analyzed different approaches and selected several state-of-the-art algorithms as candidates for our comparison. We contacted the authors of these algorithms to get their original implementations and used public available programs. Due to pleasant support of various researcher, we ware able to collect 12 different (meta)heuristic algorithms for the GCP, namely `EVODIV` [201, 203], `GLS` [47], `HEA` [90], `ILS` [51], `MACOL` [164], `MAFS` [250], `MEA` [45], `MMT` [169] (only the component containing the genetic algorithm), `PRTS` [45], `FOO-PARTIALCOL` [20] (further abbreviated to `FPC`), `SA/TS` [192], and `TABUCOL` [122] (further denoted as `TABU`). Unfortunately, we had to exclude `PRTS`, `MEA` and `SA/TS` because they were incompatible with our test environment. These solvers were compiled for a Windows operating system and enforced the usage of a graphical user interface, which was no compatible with our test setup. In addition, we also removed `EVODIV` from out algorithm set because it showed poor performance compared to other algorithms. In the end, we had 8 different (meta)heuristic algorithms in 3 different programming languages (C,C++ and Java). In the following paragraph, we explain why we have chosen these algorithms.

The main reason for selecting the `TABU` solver, because it is one of the most-studied heuristics and is often used as LS in various population-based algorithms for the GCP. In addition, according to a comparison by Chiarandini [47], `TABU` is besides `HEA` and `ILS` the most effective algorithm for random graphs. The same study also identified `GLS` as best algorithm for geometric graphs and it is, as ILS, also well-suited for *Leighton* graphs. HEA is chosen because it shows good performance on *flat* graphs and it is used as basis for many other evolutionary heuristics that are applied for GCP. This also motivated us to use MACOL, which is a direct successor of HEA. Moreover, it achieves the best known solution (BKS) on some *hard* graphs of the *Dimacs* challenge. We selected FPC and MMT because we also wanted to use algorithms working with *partial* colorings and these two candidates are the correspondent versions of TABU and HEA. The last competitor, MAFS, is included because it shows good performance on large graphs. Table 5.1 displays the chosen algorithms and their parameters. Note that concerning the parameter settings, we used the values proposed in the original publications and that were suggested by its developers. We did not apply any parameter tuning nor did we test the effect of instance-specific settings. The only parameter which we vary is the stopping criteria (the time limit or the number of iterations).

Name	Parameters
GLS [47]	$\lambda = 1$, no. sidewalks $= 20$
FPC [20]	$\alpha = 0.5$
HEA [90]	$\alpha = 0.5$, no. tabu Iterations $= 10000$
ILS [51]	$\alpha = 0.5$, $\texttt{ilsprob} = 100$, $\texttt{pert} = 0.0(ROS)$, $\gamma = 0.35$
MACOL [164]	no. tabu iterations $\alpha = 75000$, $\lambda = 0.8$, population size $p = 20$, $\texttt{RPN} = 20$
MAFS [250]	no. agents $= 25$, no. iterations $= 70 \cdot$ time limit
MMT [169]	no. tabu Iterations $= 20000$, $\texttt{tabu tenure} = 45$, (population size) $p = 20$
TABU [122]	$\alpha = 0.5$

Table 5.1: Algorithms for the GCP and corresponding parameter configurations used for the evaluation.

We also made some experiments with other approaches like one that transforms the GCP to SAT(ColorSat [233]), constraint satisfaction problem (CSP) (GeCol [107]), and integer linear programming (ILP) using a supernodal formulation [34]. Additionally, we also implement a program for answer set programming (ASP) [99]. Unfortunately, the results of these solvers obtained by preliminary tests were always far behind those achieved by the above algorithms. Therefore, we did not include these approaches in our final comparison.

5.1.1 Variations of the Algorithm Space

After our preliminary experiments, we selected 8 different algorithms for solving the GCP which should, in combination with a classification algorithm, form our system for algorithm selection for the GCP. However, the performance of the different classification algorithms depends, besides the used attributes and the data itself, also on the number of possible classes. Thus, it is possible that by removing heuristics from the set of possible algorithms the overall performance is improved.

To evaluate this aspect, we also investigated the usage of only a subset of algorithms. In detail, we evaluated if and how removing some heuristic changes the performance of the classification algorithm for algorithm selection. Therefore, we used several subsets of algorithms denoted as hx with $x \in \{8, 7, 6, 5, 4, 3\}$ where x stands for the best x heuristics. As selection criteria, we took the number of first places according to our ranking scheme. The best algorithm is the one which achieved on the most instances the best coloring in the shortest time. Note that also other configurations are possible, e.g. selecting an algorithm that dominates on a particular subset of instances with some specific attributes. However, this is beyond the scope of this thesis and is therefore left for the future work.

5.2 Benchmark Graphs

5.2.1 Training Data

For the evaluation of different algorithms, we took as training data three different public available sets of instances. The first set, further denoted as `dimacs`, consists of difficult graphs from the *Graph Coloring and its Generalizations*-series (COLOR02/03/04) [1] which builds up on the well-established Dimacs Challenge [133]. This set includes benchmark instances from the *coloring* and the *clique* part of the Dimacs Challenge and are graphs obtained by various construction methods. The second and third set of instances, denoted as `chi500` and `chi1000`, are provided by Marco Chiarandini and Thomas Stützle [53] and contain 520 instances with 500 nodes and 740 instances with 1000 nodes respectively. These instances are created using Culberson's [62] random instance generator by controlling various parameters like the *edge density* or the *edge distribution*. The former variable, denoted as p, determines the ratio of edges between the vertices. Considered values for these two sets are $p = \{0.1, 0.5, 0.9\}$. Regarding the edge distribution, the graphs can be classified into three groups: *uniform graphs* (G), *geometric graphs* (U) and *weight biased graphs* (W). In an *uniform graph* the edges are assigned for each pair of nodes with a fixed probability p which is equal to the desired density. *Geometric* graphs are created by uniformly locating the nodes in a two-dimensional square and assigning an edge between two nodes if their euclidean distance is less or equal to some parameter r. For the last category, *weight biased graphs*, the nodes are first assigned to independent sets. Then, a weight is assigned to each pair of nodes and the edges are iteratively created with a probability proportional to the weight of the pair of nodes. Each time a new edge is generated, these weights are decreased such that large cliques becomes unlikely. to prevent the creation of edges between nodes in one independent set, the weight of these pairs is set to 0. Considering the second and third set of instances, Table 5.2 gives an overview on the number of graphs with different attributes. As the reader may have noticed, the number of instances of the different subgroup is not distributed homogeneously. Especially sparse graphs with a density of 0.1 are less represented. In addition, these sparse graphs tend to be less hard to color. Consequently, the number of *hard* instances in these graphs is smaller than on the graphs with higher density. Also worth mentioning is that the number of uniform graphs is larger compared to the other types of graphs.

[1]available at `http://mat.gsia.cmu.edu/COLOR04/`

	$n = 500$			$n = 1000$		
Type	$p = 0.1$	$p = 0.5$	$p = 0.9$	$p = 0.1$	$p = 0.5$	$p = 0.9$
G	30	60	90	40	140	70
U	30	110	60	50	200	70
W	30	60	50	30	90	50

Table 5.2: Number of instances of the set `chi500` (left) and `chi1000` (right) separated by density and graph type.

5.2.2 Test Data

For the final evaluation of our algorithm selection approach with the underlying heuristics for the GCP, we used as a `test set` a complete new and unseen set of instances. For this purpose, we used Culberson's instance generator to construct instances of different size, density and type. We adopted the main parameters of Chiarandini's graphs and focused on *uniform* (G), *geometric* (U) and *weight biased* (W) graphs. We used 4 different sizes $n = \{500, 750, 1000, 1250\}$ with density values $p = \{0.1, 0.5, 0.9\}$. For each parameter setting we created 5 graphs, leading to a total of 180 instances. Note that for the *weight biased graphs* the density is also affected by the initial weighting W. To achieve the desired values $p = \{0.1, 0.5, 0.9\}$, we set W according to [59] to $\{2, 115, 404\}$, $\{3, 173, 605\}$, $\{5, 232, 804\}$, $\{6, 290, 1003\}$ for graphs of size 500, 750, 1000 and 1250, respectively.

5.2.3 Time Limit

As already mentioned, the GCP is an NP-hard problem, so deciding whether a graph is k-colorable or not may take very long time and usually it is not possible to estimate the time needed. To prevent programs to run for a very long time, we specified for each instance some time limit after which the execution is stopped. Different approaches for such threshold have been used in practice like using fixed values (ranging from 5 to 120 hours [164, 203, 248]) or comparing with some reference algorithm, e.g. `TABU` with a fixed number of iterations [48]. We decided to use a maximal runtime based on instance attributes itself. In detail, we set the time limit t_{max} as

$$t_{max} = \text{min}(3600, \sqrt{|E|} \cdot x)$$

where $|E|$ is the number of edges and x is 15, 5 and 3 for the sets `dimacs`, `chi500` and `chi1000`, respectively. For the `test set` which contains graphs of different size, we stick to the values used for `chi1000` ($x = 3$). These values for x were obtained experimentally.

Our general assumption is that the chosen time limit does not influence the ranking of the algorithms, as long as it is not customized to one heuristic and if it is between some "rational" boundaries. This means that on the one hand, the time limit should be long enough so that the data input and the initialization processes does not effect the timing. On the other hand, the time limit should be within practical limits, because an unrealistic long value is not useful for real-world applications.

In this context, according to our results, these limits do not restrict the search. For example, on the instances of `chi1000`, the average time needed for the best solution on the *hard* instances is only 21.58% of the allowed value t_{max}. Even more, 90% of the best solutions are found within 62.66% of t_{max}.

5.2.4 Trivial-, Easy- and Hard- Instances

The graph coloring instances used in this work are of different difficulty. For algorithm selection, easy instances are usually uninteresting - calculating features and performing predictions for the best algorithm may often take longer than even the slowest search procedure that solves the problem optimally. Therefore, we are interesting in problems that are hard to solve.

So we came up with the questions (a) how to define *easy* and *hard* (or *difficult*) instances, and (b) how to assign an instance to a particular category before applying a solver.

The first question is hardly discussed in literature and usually this depends on the specific problem. Typically, in the literature an instance is classified as hard or easy based on attributes like [46]:

- the number of possible solutions,
- the size of basins of attraction, or
- the frequency of nearly-optimal local optima.

Beside these theoretical aspect, there are some common practices described in literature which are often based on implicit mutual agreement or an established historical classification (e.g., used in [118]). Typical criteria for classifying an instance as easy are for example a short runtime of multiple solvers to achieve a best known solution (e.g., used in [164]). Also the comparison to low-level algorithms that obtain the optimal solution indicates that an instance is of low complexity (e.g., used in [250, 47, 51]). On the other hand, hard instances are usually characterized by a lack of an exact solution, relatively long runtime and a strong variations in the performance of different heuristics.

Also, to denote in which category an instance belongs to is not trivial. Of course, this can be performed by the application of one or multiple algorithms, but from the practical point of view, this is often inefficient, unusable or just not possible. Much more interesting would be to classify in advance and before applying some solvers. This question is closely related to finding the *phase transition* [46], an idea following the observation that easy instances are either underconstrained or overconstrained and that the hard problems usually occur on the boundary of these two regions. This area, where the difficulty of the problem changes abruptly, is called *phase transition* and can be described for each problem by a critical values of some *order parameters*. Unfortunately, finding these transition may also be complex and requires explicit empirical research.

One possible practical alternative might also be machine learning - under the precondition that extracting the relevant features is efficient (with respect to time), machine learning classifiers seem as the perfect decision tool for classifying instances as *easy* or *hard*.

Another different approach is implemented in *SATzilla* [253], which runs as first step a general algorithm for a short predefined period of time and checks whether this method finds a solution or not. Only in the latter case, when the instance seems to be hard, the program applies its algorithm selection techniques.

For this thesis, we will ignore these questions and just focus on hard instances. Therefore, we manually separated our instances into four categories: *trivial*, *trivial2*, *easy* and *hard*.

The first class, *trivial*, contains instances where we already get the best known solution during the computation of features. In detail, we use greedy coloring techniques and an algorithm for finding maximal cliques to obtain lower and upper bounds for the chromatic number. If these two values equal, it is impossible to find a better coloring. Therefore, the instance is trivial. The second category, denoted as *trivial2*, follows a related principle: It includes all instances where, starting with the color number obtained by the greedy algorithms, no further improvements can been found by any search algorithm. For these instances, it seems that the greedy approaches return the best known solutions. We further denote instances as *easy* if the heuristics find better solutions than the greedy methods, but these results are found by at least 50% of the solvers and in at most 5 seconds. And finally, all other instances which do not belong to one of the introduced categories are categorized as *hard*.

We focus on hard instances because these instances require the most effort to solve them. Therefore, the algorithm selection can be useful for these instances.

5.3 Test Methodology & Experimental Environment

5.3.1 Evaluation System

All tests have been performed on the same system, a Transtec CALLEO 652 Server containing 4 nodes, each with 2 AMD Opteron Magny-Cours 6176 SE CPUs ($2 \cdot 12 = 24$ cores with 2.3GHz) and 128 GB memory. For resource control and job scheduling, we used the Condor workload management system [2] (for more information about the Condor system, we refer to [226]). Each algorithm is executed for each instance on a separate core with a memory limit of 5 GB. The programs itself are compiled, in cases where we got the source code, on the same machine using the *g++* compiler with optimization level *-O3*. For the Java program, we use the JDK version 1.6.0_23.

5.3.2 Algorithm Evaluation

As already mentioned, we gathered both *decision*- and *optimization* methods for GCP. Luckily for us, all optimization-based approaches also print the time when they found a new (better) coloring during their search. This allowed us to treat both variants in an uniform way using similar input (the instance, start number of colors k and time limit t_{limit}). Moreover, we ensured by an external time limit that no heuristic takes longer then the provided time limit and also limited the memory to 5 GB. As output of each execution, we collected for each number of colors k the time needed to find that coloring.

On each particular instance, we took the lowest number of colors obtained by the greedy coloring methods and reduced it by one (formally, $k = k_s = min\{\text{GC}_{\text{DSAT}}, \text{GC}_{\text{RLF}}\} - 1$). So, we assumed that all solvers are able to find a coloring with $k_s + 1$ colors. Then, we called each algorithm $n = 10$ times ($n = 20$ for the `dimacs` instances) using different random seeds. After

[2] `http://research.cs.wisc.edu/condor/`

all executions had been finished, we checked whether a coloring with k colors has been found. If so, we recorded the runtime, and, after checking if we reached a lower bound or not, repeated the process with $k-1$ colors. For solvers that solve the optimization version of GCP, we recorded the computation time for all numbers of colors $k-1, ..., k_x$ which have been found by all n program calls separately. Then, we continued by searching for a coloring using $k_x - 1$ colors. For both types of solvers, the process stopped if none of the n executions found a coloring using k colors, or if a lower bound (e.g. obtained by the clique size) was reached.

After applying the algorithms to all instances, we calculated for each algorithm, each instance and each number of colors k for which a coloring has been found these two parameters:

(a) the median time needed and

(b) how often a coloring has been found (the *risk*).

Having this information for each instance and all algorithms, we measure the performance of a method as follows: Let a be an algorithm, i be an instance and $m_a^i(k)$ be the median time needed for a to solve i with k colors. Furthermore, let k_a^i be the lowest number of colors of a on i where the risk of a is above 50%. Then, we define the *performance measurement* $f(a,i) = \langle k_a^i, m_a^i(k_a^i) \rangle$.

Using this measurement, we are able to rank different methods according to their reached number of colors k and the computation time for a particular instance. More formally: Let i be an instance and a and b be two algorithms. Then a is better or equal b on instance i ($a \leq_i b$) with $f(a,i) = \langle k_a^i, m_a^i(k_a^i) \rangle$, $f(b,i) = \langle k_b^i, m_b^i(k_b^i) \rangle$ if and only if

- $k_a^i < k_b^i$, or
- $k_a^i = k_b^i$ and $m_a^i(k_a^i) \leq m_b^i(k_b^i)$.

It is easy to see that $\leq_i$ is a *total preorder* over the set of all algorithms A, as it fulfills *reflexivity*, *transitivity* and *totality*.

Using this binary relation, we can define the "best" algorithms [3] B for an instance i as

$$B^i = \{a \in A : \forall b \in A (a \leq_i b)\}$$

5.3.3 Classifier Evaluation

Besides measurements for the used GCP algorithms, we also need metrics indicating the performance of the algorithm selection itself. One of the most interesting metrics for this is of course how often the "best" algorithm is predicted. As we were using classifiers, this value is related to the *accuracy* of a classifier, which is the percentage of correct classifications among all instances. However, in our experiments, we also investigated the effect of using only a subset of the tested heuristics. Unfortunately, using only the accuracy for this kind of comparison is less informative and does not contain information about the usability of algorithm selection.

[3]Note that according to our definition, there can be multiple algorithms that give the best result. Although this is rare in practice (especially on hard instances), it is unpractical for machine learning. Therefore, in case of a tie, we use an ordering based on the average rank of the algorithms and select the method that is ranked on the first place.

Therefore, we introduce the term *success rate*, which indicates how often the algorithm selection returns the best algorithm among a set of tested heuristics. In detail, the *success rate* $s(c, I, A)$ of a classifier c on a set of instances I and the algorithm space A is defined as

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

where $c(i)$ is the predicted algorithm for the instance i. Note that for the rest of this thesis, we will only use the success rate considering all tested heuristics $A = \texttt{h7}$.

One advantage of this metric is that it also considers algorithm predictions that are not correct with respect to the training data, but which are also ranked on the first place (because there were multiple best algorithms). Thus, it takes all cases into account where the predicted algorithm obtains the best solution, regardless if it is the expected class in the training data or not.

Besides this measurement for the number of best-selected algorithms, another interesting information is how "close" the prediction of a classifier is to the optimal value and how the algorithm selection performs in comparison with single heuristics for the GCP. For this purpose, we treat the classifiers like a normal heuristic and use for each instance the performance of the predicted algorithm as the classifiers result. This allows us to apply different evaluation criteria used for other comparisons.

In detail, we consider three different measurements: $err(k, i)$, a *classical ranking* (in terms of the *average rank* and the *standard derivation*) and a *formula one* ranking.

5.4 Discretization

As already mentioned, various classifiers show significant better performance when they are trained with nominal data instead of continuous values. To evaluate the results of such a discretization, we used two different discretization methods in our experiment which transformed our numeric features into nominal ones. The first one is the classical minimum-descriptive length (MDL) algorithm by Fayyad and Irani [81], while the second method is a derivation of MDL by Kononenko [145] using a different criteria (further denoted as Kononenko's criteria (KON)). Concerning the nomenclature of our data sets, we denote a set which has been discretized using the first method with the extension `mdl`, while sets obtained by Kononenko's criteria with `kon`. In addition, we also use the non-discretized data set for the following classification, which is marked with `none`.

5.5 Chosen Classification Algorithms

One key assumption for this thesis is that different algorithms show diverse performance on the same data. This includes of course also classification techniques from machine learning. Therefore, before each successful application of classification, first appropriate algorithms must be selected. This task of choosing the best machine learning algorithm is itself an algorithm selection problem and is subject to various research projects (e.g. [3, 30, 27, 28, 158, 7]), where also automated approaches have been developed.

However, applying such selection methods is outside of the scope of this thesis, wherefore we test exemplary the performance of several well-known machine learning-algorithms. For the classification part of our experiments we used six different machine learning techniques, namely: Bayesian networks (BN), C4.5 decision trees (DT), k-nearest neighbor (kNN), multi-layer perceptrons (MLP), random forests (RF), and support vector machines (SVM).

For all these techniques, we used the implementation included in the *Weka* software collection, version *3.6.6*. Many of these methods offer a wide range of parameters, which may effect the performance and the success of the learning task. Therefore, we manually identified important variables and used for each learning algorithm multiple settings. The following paragraphs describe the most important decisions. Final parameter settings included in the original *Weka*-calls can be found in Appendix A.1.

For the *Bayesian Network* (BN), one of the most important parameters is the maximal number of parent nodes P that each node can obtain. The more parent nodes, the more complex relations can be represented, but also the more possible structures exist, which results in a larger search space. For our application, we tested 5 settings denoted as `BN`*p* with $p = 1, ..., 5$. Another parameter worth mentioning is the method for finding the *most probable believe-network structure*, which is itself a NP-HARD problem [55]. For this purpose, we used the K2 heuristic [55] which is a hill-climbing method combined with a strict variable ordering.

Concerning the *C4.5 Decision Tree* (C4.5), we experimented with four settings regarding different *confidence factor* C (used for pruning) and minimal number of objects per leave, M. Table 5.3 gives a more detailed view on the settings and the concrete variable values. Note that, to avoid confusions, we denote the configurations `DT`*x* with $x = 1, ...4$.

Setting	C	M
`DT1`	0.250	2
`DT2`	0.125	2
`DT3`	0.250	3
`DT4`	0.250	4

Table 5.3: Parameter settings for the DT classifier.

Setting	c	Kernel	e
`SMO1`	1.0	Poly	1.0
`SMO2`	1.0	Poly	1.2
`SMO3`	1.0	Poly	1.4
`SMO4`	1.5	Poly	1.0
`SMO5`	1.5	Poly	1.4
`SMO6`	2.0	Poly	2.0
`SMO7`	3.0	Poly	2.0
`SMO8`	2.0	PUK	-

Table 5.4: Parameter settings for the SVM classifier .

Concerning the *Random Forest* (RF), we experimented with the depth, the number of chosen attributes and the number of trees. Based on our experiments, have not observed any advantage in cutting the depth or limit the number of chosen attributes. Therefore, we considered only two settings, specifying the number of trees used to 10 (denoted as `RF1`) and 15 (`RF2`).

Regarding the *Support-Vector Machine* (SVM), we used the sequential minimal optimization (SMO) algorithm [199], which is a fast and easy to train SVM. Relevant parameters are the complexity parameter c where we tested values for $c \in \{1.0, 1.5, 2.0, 3.0\}$, and the underlying kernel function. As kernel function, we used a polynomial kernel with $K(x, y) = \langle x, y\rangle^p$ or $K(x, y) = (\langle x, y\rangle + 1)^p$ and a *Pearson VII function-based universal* kernel (PUK) [232]. For the former one, we tested different values for the exponent $e \in \{1.0, 1.2, 1.4, 2\}$ while for the latter one, we used the default parameters. Table 5.4 shows the different settings for SVM. Note that we named the configurations with `SMO`*x* with $x = 1, ...8$ according to the SMO algorithm.

Concerning the *Multilayer Perceptron*, we almost entirely used the default parameters from *Weka*. The number of hidden layers was set to $\frac{\text{(no. attr.+no. classes)}}{2}$ and the number of training iterations was 500. We experimented with different learning rates and selected two settings with a value of 0.3 (denoted as `MLP1`) and 0.4 (`MLP2`) respectively.

For the *k-Nearest Neighbor*, an obvious variable is the number of neighbors k. We experimented with $k = \{1, 3, 5, 7, 9\}$ and the corresponding settings are denoted as `IB`*k* (according to the name of the classifier, `IBk` in *Weka*). For other parameters like the search algorithm or distance weighting, we used the default values of *Weka*.

5.6 Feature Selection

For feature selection, we used two search methods: a *genetic* search and a *best-first* selection strategy with backtracking abilities. The former one is a simple implementation of the classical genetic algorithm by Goldberg [105], using a $\langle 0, 1\rangle$-vector for the chosen features as chromosome. Important parameters for this method are the crossover probability C, the number of generations G, the population size R and the mutation probability M. We used a parameter setting of $C = 0.6$, $G = 100000$, $R = 50$ and $M = 0.033$. These values are, except the number of generations, the default setting of the *Weka* system. We also want to note that for the expanded features, we reduced the maximal generations to $G = 10000$ because of runtime reasons.

The second method was a *greedy hill climbing* algorithm which is enhanced with (limited) backtracking. The search starts with an empty or predefined set of attributes and adds iteratively the feature with the highest improvement concerning the chosen measurement. Backtracking is limited by the number of consecutive non-improving decisions N, which was set to $N = 10000$ for our application based on preliminary experiments. Moreover, we used the empty set as start set.

For the evaluation, we used in both methods the *CfsSubsetEval* (CFS) criteria [115]. This method evaluates a subset by measuring for each feature its prognostic value concerning the classification, in combination with the degree of redundancy between the attributes. Using this two measurements, CFS prefers feature subsets which have a low correlation within each other, but are highly correlated with the classification value.

Concerning our experiments, we first performed a feature selection on our base features, which are denoted as `b`. The resulting data sets of this process are denoted with `bff` for the best-first forward selection, and `gen` for the genetic search. After the feature selection, we combined for each subset of algorithms `h`*x*, $x \in \{8, 7, 6, 5, 4, 3\}$ and discretization method $disc \in \{\texttt{none}, \texttt{mdl}, \texttt{kon}\}$ the two resulting feature subsets and applied on these sets a process called *basis function expansion*. In this process, the features of an instance i are multiplied pairwise and the product is added as new feature $x_{i,j} \cdot x_{i,k}$ for $j = 1...m$ and $k = j + 1...m$. In addition, we also add the quotient $x_{i,j}/x_{i,k}$ of each pair of features. As result, we obtained for all subsets of algorithms three sets of expanded features, denoted as:

- `e1` from the feature subsets using no discretization,
- `e2_mdl` from the subsets based on *mdl* as discretization method, and
- `e3_kon` from the data using the *kon* criteria.

Note that for the last two, we performed the expansion on the non-discretized features and applied the discretization procedure afterwards. The feature selection itself occurred on the discretized attributes, but as these features are nominal, it is no longer possible to multiply or divide them. After this expansion, we performed another stage of feature selection to determine for each expanded set two reduced subsets.

All feature selection variants were performed using a 10-fold cross-validation. Using this method, *Weka* returns as result for each attribute the occurrence in the 10 attribute sets. For the basic features, we took all attributes which were selected at least once within the 10 folds. For the expanded attributes, this procedure resulted in very large sets of features, especially when we were using the *genetic* search. Therefore, we reduced the chosen features by considering only attributes which were selected in at least x folds with $x \in \{3, 6, 8, 9\}$ such that the total number of attributes is around 100. This is necessary as the execution time for testing some classifiers (especially the MLP) raises dramatically with increasing size of the attribute set, which makes systematic testing impracticable.

CHAPTER 6

Experimental Results and Evaluation

In this chapter we present an evaluation of 8 state-of-the-art solvers for the GCP on the instances of three public available instance sets. Main issue of interest is to find for each instance the algorithm that achieves the best results and to analyze the performance of the different heuristics on subsets of instances. For this purpose, we calculate several performance measurements of the algorithms and investigate the impact of graph features on the behavior of the algorithms.

In the second part of this chapter, we investigate automatic algorithm selection for the GCP using machine learning techniques. In detail, we train several popular classification algorithms and study their performance using several parameter configurations and data-*discretization* techniques. Furthermore, we compare the results of the different classifiers and investigate the effect of *feature selection* and *reducing the algorithm portfolio*. Finally, we compare our solvers based on automated algorithm selection with the underlying heuristic algorithms on a set of new generated instances.

6.1 Heuristics Evaluation

To begin with, we take a look on the results on the different sets of instances and analyze the behavior of solvers on different subsets of instances.

6.1.1 Instance Set chi500

Of the 520 graphs with 500 nodes from the instance set `chi500`, we detected 65 *trivial* instances which can be solved optimal by greedy algorithms. After our experiments with the chosen heuristics, we discovered 8 instances where none of the heuristics found a better coloring than the greedy methods (marked as *trivial2*). We further classified 170 instances as *easy* (according to our definitions in Section 5.2.4) and categorized the remaining to 277 instances as *hard*. To compare different algorithms, we calculated multiple performance metrics of the heuristics.

The first and maybe most interesting issue is the comparison of the performance between the different algorithms. For this purpose, we rank all methods according to our metric described in 5.3.2 and count how often a heuristic gives best results. Figure 6.1 shows for each algorithm the number of instances on which it can obtain the best performance. Note that according to our

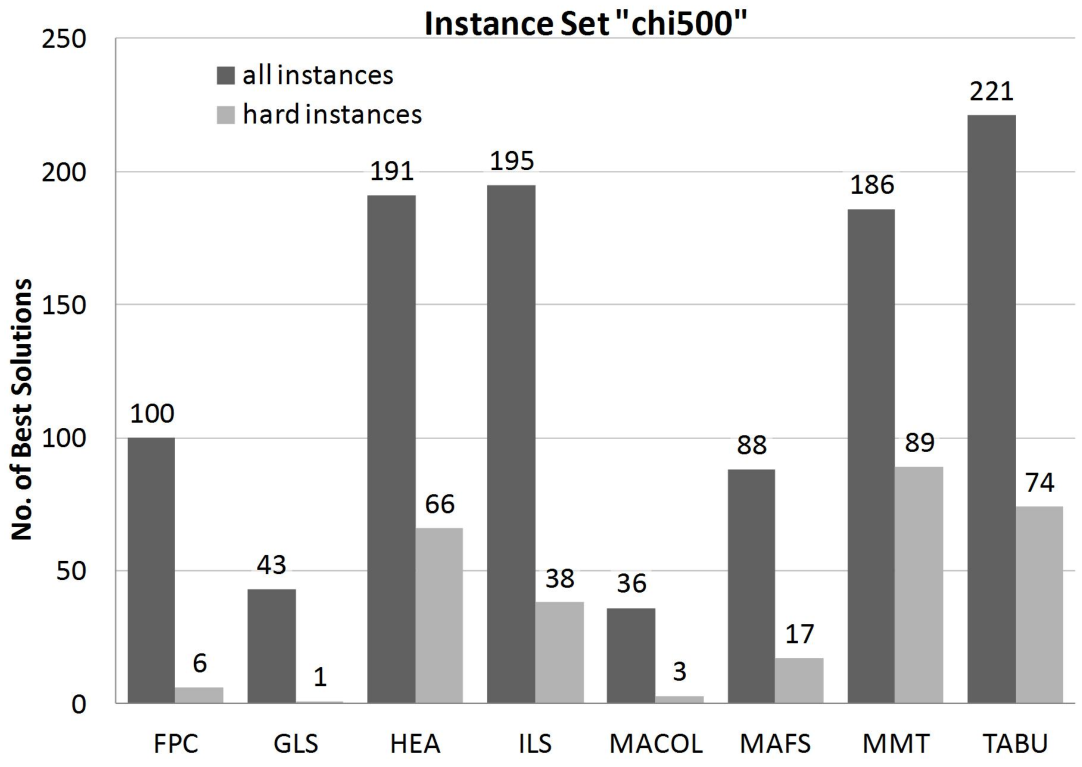

Figure 6.1: Number of instances of the test set `chi500` on which the algorithms obtain the best solution.

definition, there can be multiple programs performing equal on one instance, so it might be that for an instance there are several heuristics achieving the best result. In detail, on 154 (90.1%) of the easy instances at least two methods found the best solution in equal time while on the hard instances, only 17 (6.1%) times no unique best algorithm was detected. In addition, the data clearly indicate that counting the number of first places on all instances is not representative for the success on hard instances. For example, `HEA` and `ILS` have nearly similar number of best solutions on all instances (191 to 195), but considering only hard graphs shows that the former obtained on 66 instances best results while the latter performed only in 38 cases best. Also `FPC`, who achieved on 100 of all 447 tested instances the best results, showed on hard graphs with 6 best-solved instances a worse performance than `MAFS`, which achieved on 88 graphs, including 17 hard ones, the best solution.

More interesting are the results on the different subsets of instances. Therefore, we separated the instance set into 9 subgroups depending on their class (G, U and W, see Section 5.2) and their density (0.1, 0.5, and 0.9). Figure 6.2 shows the result of the hard instances according to these subsets. As expected, the different algorithms show different performances concerning the

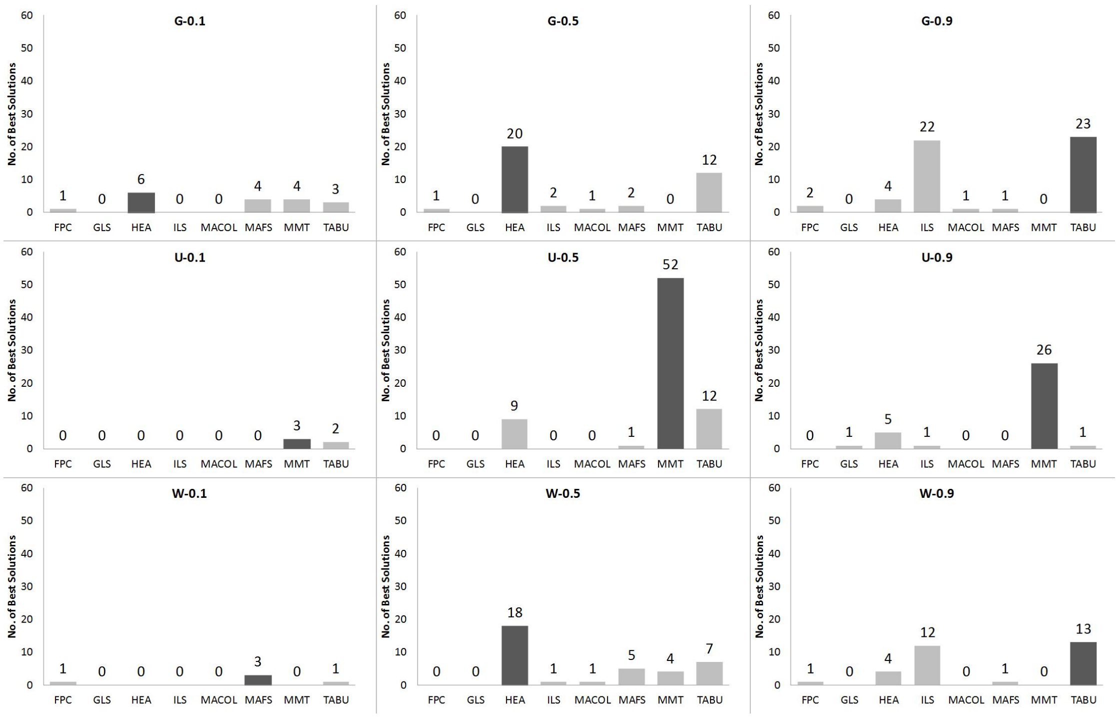

Figure 6.2: Number of hard instances from `chi500` on which the algorithms obtain the best solution. The graphs are separated according to their class and the density.

graph attributes. Even more, the results clearly show that no algorithm outperforms all others on all subsets. The figures also illustrates that only few graphs with a low density of 0.1 are classified as *hard* and the results on these three subsets do not clearly identify any dominant algorithm. Concerning a density of 0.5, the results on the instances of `G-0.5` and `W-0.5` reveal a strong performance of `HEA`, followed by `TABU`. This is interesting as `HEA` uses `TABU` as embedded LS, which seems a successful approach on these graphs. A complete different result can be observed on the subgroup `U-0.5`, where `MMT` outperformed other heuristics on most instances. Also on the other geometric graphs, grouped in `U-0.1` and `U-0.9`, `MMT` achieved the highest number of best solutions, which indicate that `MMT` is in general well-suited for these type of graphs. On the remaining two subsets with high density (`G-0.9` and `W-0.9`) `ILS` and `TABU` showed the best performance.

Concerning the distance $err(k, i)$ to the best colorings found so far, Figure 6.3 gives a more detailed view (the results are again separated by the density and the class of the graphs). Remarkable is especially `MMT`, which has a very low value on the subsets `U-0.5` and `U-0.9`. This indicates that this method finds in our experiments most of the time the best coloring on these instances. In addition, the results suggest that the success of this algorithm is due to finding colorings using fewer colors and not necessarily because of a quick search. A different, but also interesting observation is that `GLS` achieves in almost all subsets a worse performance than any other algorithm. Only on the instances of `U-0.9` and `U-0.5` it shows competitive results, but

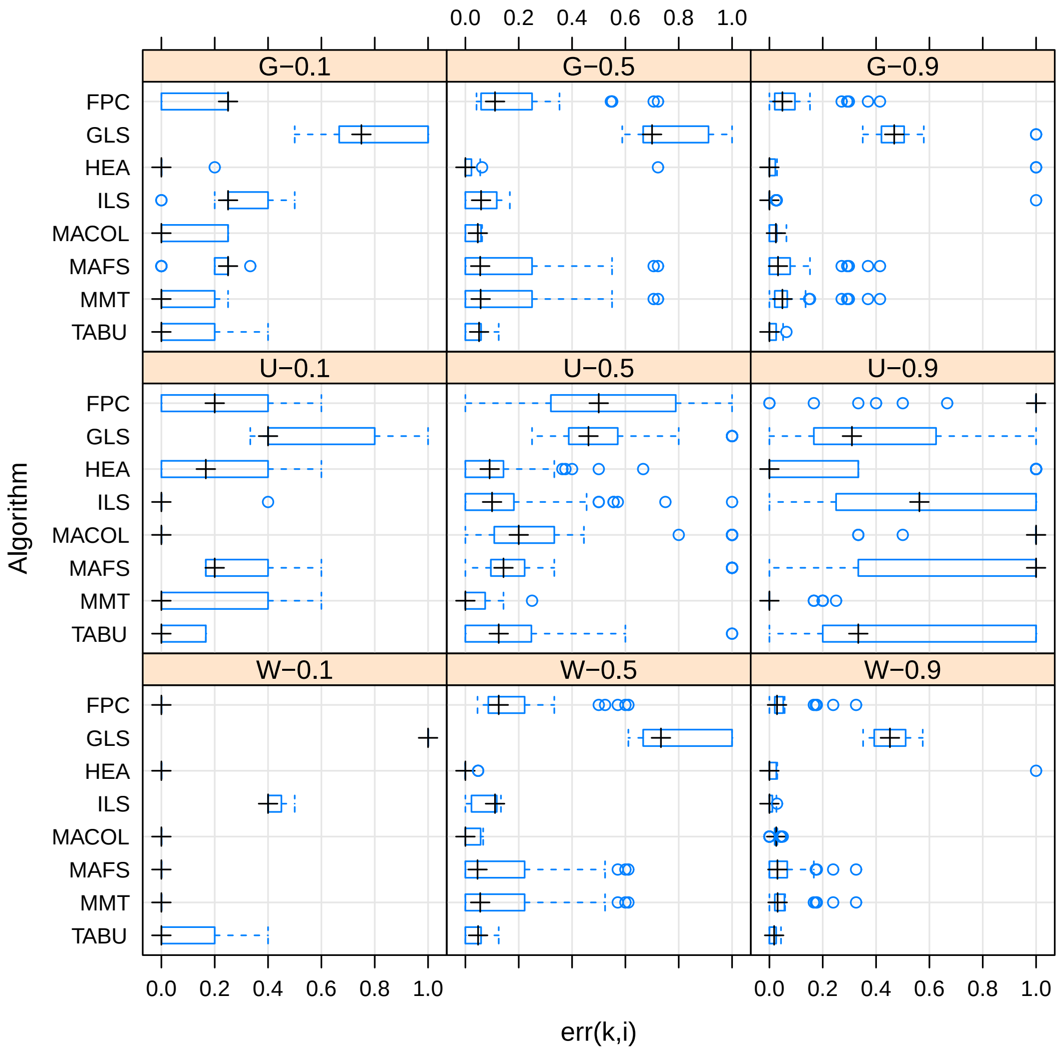

Figure 6.3: Distance metric $err(k, i)$ of the algorithms on the hard instances of `chi500`. The results are grouped by the graph class and the density.

it is never among the leading approaches. Worth mentioning is also that the values of $err(k, i)$ on the sets G-0.9 and W-0.9 are in general low. One explanation could be that the greedy methods achieve worse results compared with other heuristics. Therefore, upper bounds used are relative high and there is much space for progress by finding better colorings. This is also justified because on average, the best coloring found by the heuristics has 22.44% less colors than the solution obtained by the greedy algorithms.

A similar observation is reported in [47] using the same instances where the improvement of heuristics over the RLF function is studied.

The figures also highlight that the median of the different algorithms are often located at similar areas among one subset, which shows a similar performance of these solvers. Only on the subsets containing the geometric graphs (U-$\star$) there is no such clustering.

Another interesting measurement for the performance are the results concerning our ranking method. Table 6.1 contains the results using a *classical ranking* and a *formula one* ranking. The

Algorithm	Classical		F1	No. Ranks							
	avg	std		1	2	3	4	5	6	7	8
FPC	5.26	1.72	1063	6	14	24	37	75	49	43	29
GLS	6.74	1.75	640	1	12	16	10	9	23	77	129
HEA	**2.47**	1.25	**2036**	66	95	63	33	14	5	1	0
ILS	3.57	1.86	1638	38	59	46	54	31	18	31	0
MACOL	4.74	1.42	1201	3	15	34	72	57	69	25	2
MAFS	4.70	1.74	1239	17	14	45	38	55	62	45	1
MMT	3.17	2.06	1845	89	51	26	17	39	40	15	0
TABU	2.75	1.51	1939	74	61	58	42	30	10	1	1

Table 6.1: Ranking results on the hard instances of chi500. The best results among this ranking are marked bold.

second and third column show the average rank and the standard derivation using a classical ranking. The lower the value, the better is the algorithm compared with the other competitors. It is easy to see that GLS achieves on average the worst rank while the most successful algorithms in this category is HEA followed by TABU and MMT. Surprisingly, MACOL, which does not obtain on many graphs the best solution, is clearly better than FPC and competitive to MAFS. Concerning the ranks using the *formula one* (F1) method, which is given in the third column, the results are quite similar and only the gaps between the values differ. The reason for this is that the method rewards top rankings more than good average results, which sometimes effects the ranking of closely related competitors. The remaining columns in this table give detailed information how often an algorithm achieves a particular rank. As the reader may notice, GLS shows on 129 of the 277 (46.57%) graphs the worst performance while more robust methods like HEA or TABU are almost never ranked worse than on the sixth place.

For a more detailed view, Figure 6.4 displays the ranks on the hard instances grouped by the different subsets. Areas of interest are especially subgroups where many algorithms achieve similar distance metric $err(k, i)$ (see Figure 6.3), like for example G-0.9 or W-0.9. On these sets, we can clearly see that, although many heuristics find on average solutions with closely related number of colors, there are clear patterns concerning the rankings. In both cases ILS

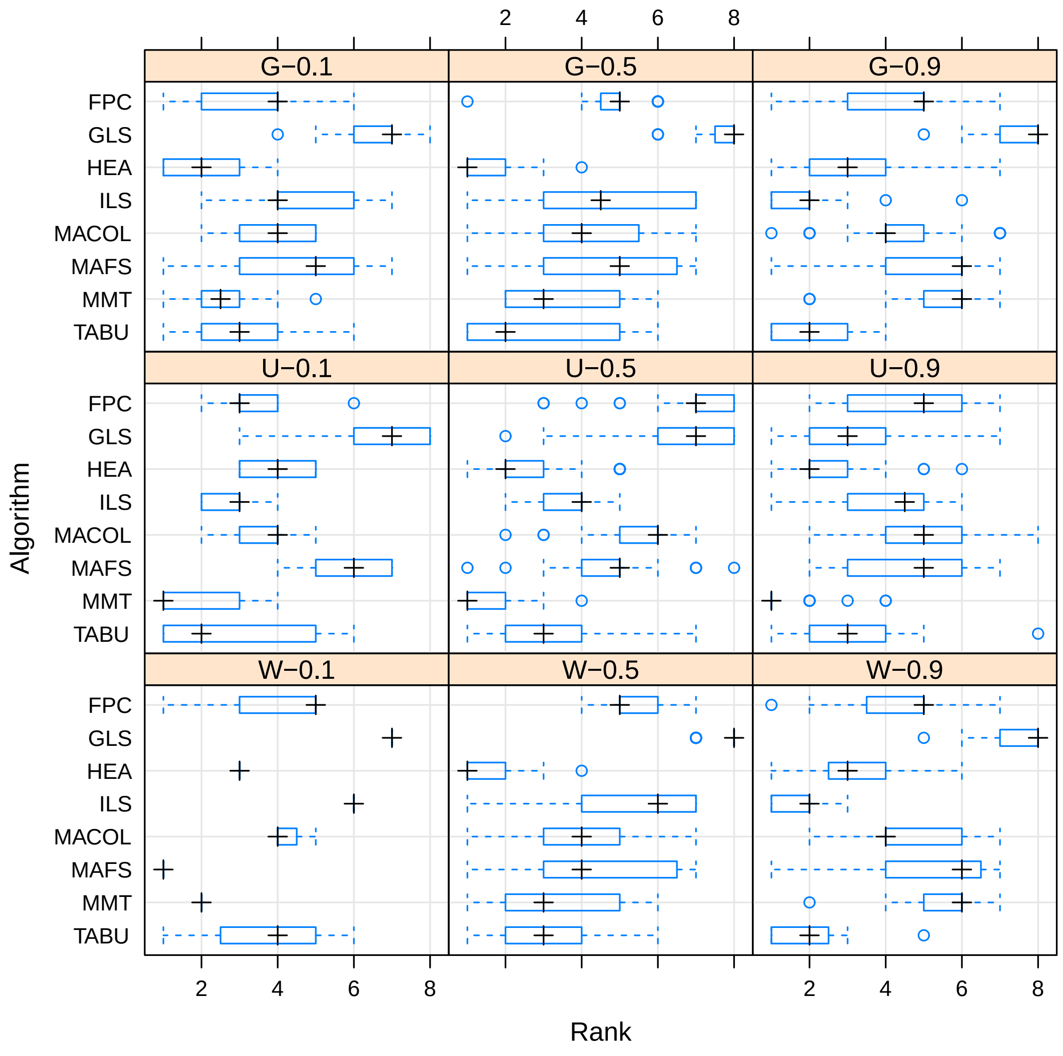

Figure 6.4: Boxplot diagram showing the ranking of the algorithms on the hard instances of `chi500`. The results are grouped according to graph class and the density.

and TABU are the leading approaches followed by HEA while MMT is always located at the lower ranks. Another observation is that GLS is on most subclasses the worst performing algorithm. Only on U-0.9 it achieves a median rank of 3, but as on these instances MMT is clearly the dominant algorithm, it is not a sufficient criteria to keep GLS in our algorithm set.

As consequence of these results, we excluded GLS for further tests. Main reason is that it shows poor performance with respect to $err(k, i)$ and is only a few times the best algorithm. The latter holds also for MACOL, but due its better results concerning the distance metric we decided to apply further tests on this heuristic. The results of GLS surprised us, as this method is one of the best on *geometric random graphs* in the experiments by Chiarandini [47]. One explanation is that our experiments involved much larger computation time, which may allow other algorithms to find better solutions while GLS reaches quickly colorings for higher k, but is unable to make further improvements (i.e. find better colorings).

6.1.2 Instance Set chi1000

The second instance set, chi1000, contains 740 instances, from which we separated 71 as *trivial* after the feature computation. We further classified 5 instances as *trivial2* after the experiments with our algorithms where none of our heuristics found a better coloring on that graphs than the greedy methods. From the remaining 664 graphs, we identified 136 as *easy* and the remaining 528 as *hard*.

Figure 6.5 gives an overview on the number of best solutions each algorithm reaches. As on the set chi500, MMT shows an very good performance and achieves the highest number of best solutions. Also TABU, HEA and ILS reach many times the best solution. Mentionable is also the performance of MACOL, which is on no instance among the best heuristics. Considering only the hard instances, the ranking stays the same although some algorithms seem to achieve many of their first places on easy instances (e.g. TABU and ILS, which are both ranked on 92 easy graphs at the first place).

Figure 6.6 reveals a closer look on the results on the different subsets. We can see that again almost all classes have their most appropriate algorithm. On five of the nine groups (G-0.1, G-0.5, U-0.1, U-0.5, U-0.9), MMT is the best method, which supports the observations from chi500 that this algorithm is well-suited for geometric graphs. On G-0.5, MMT it is closely followed by HEA, which also shows very good results on W-0.5. These results also correspond to the results on chi500 where HEA is also very successful on instances with density 0.5. On the subsets of graphs with high density G-0.9 and W-0.9, TABU and ILS seems to be adequate. These algorithm show also on some graphs with medium density the best performance. One difference to the instances of chi500 is observed on the subset W-0.1, where FPC clearly achieves the highest number of best solutions. One explanation is that the graphs of the equivalent subgroup of chi500 are mostly classified as *easy* so that the group W-0.1 of chi500 consists only of 5 instances, which do not provide much insight into the performance of the heuristics on that graphs. The algorithm MAFS is again on no subset the dominating algorithm but shows on instances with medium density, especially on G-0.5, good performance.

Note that the number of instances are not uniform distributed among the different instance subset (neither in chi500 nor in chi1000), which affects the results concerning the total number of instances on which an algorithm showed best performance. Especially U-0.5 consists

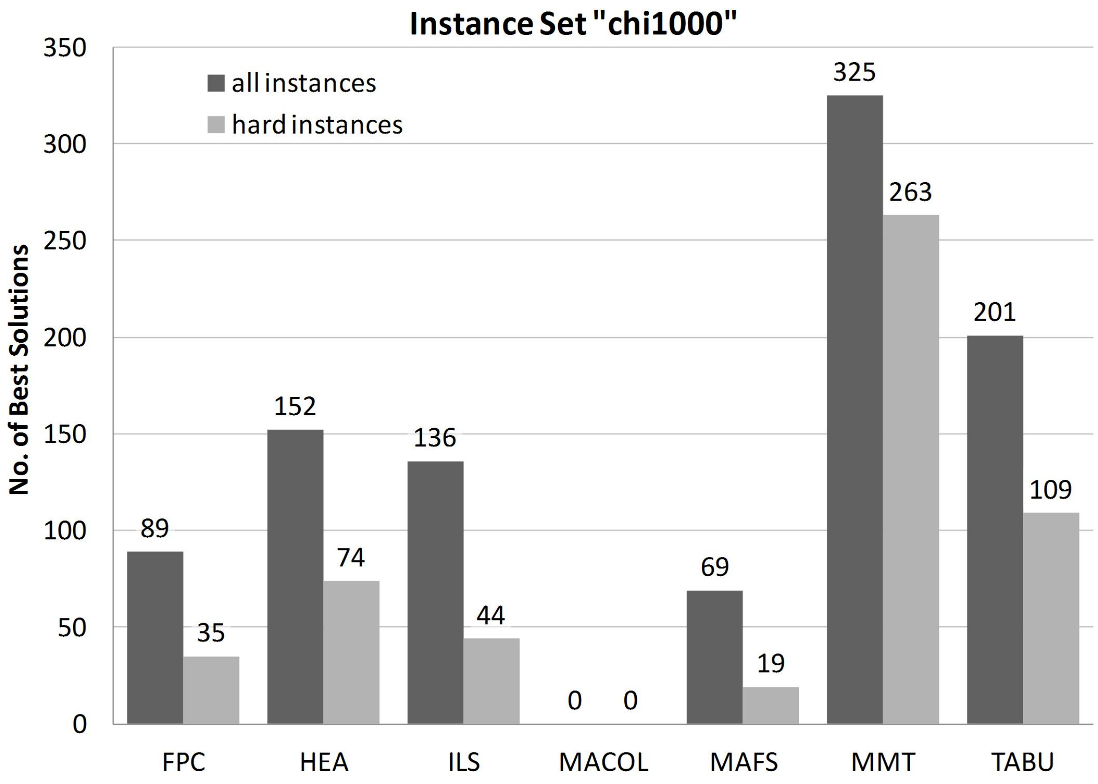

Figure 6.5: Number of instances from the set `chi1000` on which the algorithms show best performance.

of more hard graphs than any other subgroup which also has an effect on the gap between `MMT` and other heuristics (see Figure 6.5 and Figure 6.6).

Concerning the distance metric $err(k, i)$, Figure 6.7 reveals some interesting explanations for the ranking of the algorithms. First of all, it seems that on some instance sets (`G-0.9`, `W-0.5`, `W-0.9`), the distance between the different algorithms is very small meaning that the number of colors found differ only slightly. In contrast to this, on the geometric graphs, and especially on `U-0.5` and `U-0.9`, the values diverge and on these subsets `MMT` achieves the lowest values. A different, but also interesting behavior are the results of `MAFS` on the instances of `G-0.1` and `G-0.5`. On the former subset, this algorithm shows the worst results while on the related instances of `G-0.5` it is one of the best heuristics. Unfortunately, we could not find some explanation for this behavior. Also mentionable is that `HEA` and `ILS` fail to improve their initial colorings on some instances of `G-0.9`. This is remarkable as `HEA` was handled as robust method [160] and `ILS` is among the best in this subgroup.

Table 6.2 shows the results using the *classical* and the *formula-one* ranking. In addition, it contains the detailed information how often a particular rank is obtained. The data on this table clearly reveal that algorithms `MMT` and `HEA` are on average the most effective ones followed by `TABU`. Then, with some gap, lies `ILS` before `MAFS` while `FPC` and `MACOL` are located at

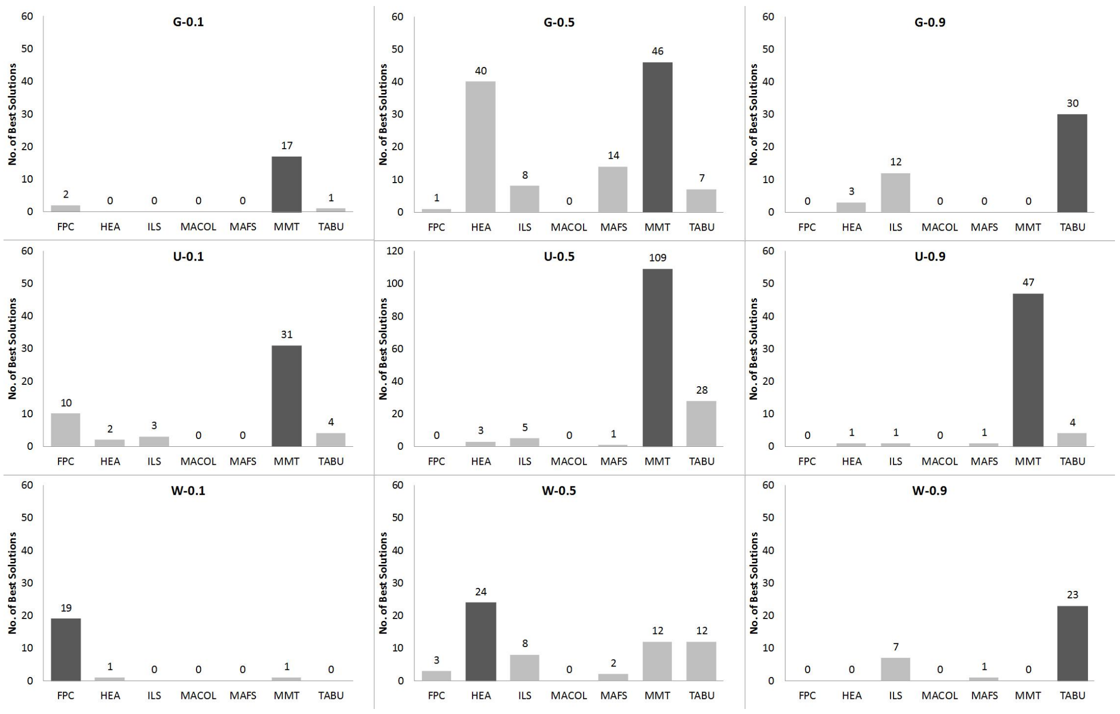

Figure 6.6: Number of hard instances from the set `chi1000` on which the algorithms show best performance. The results are grouped according to the class of the graphs and the density. Note that the y-axis on the diagram for `U-0.5` has a different scale.

Algorithm	Classic		F1	No. Ranks						
	avg	std		1	2	3	4	5	6	7
FPC	4.64	1.91	2424	35	50	78	81	80	73	131
HEA	2.79	1.39	3631	74	206	104	81	35	18	10
ILS	4.14	1.87	2732	44	78	69	135	67	41	94
MACOL	5.59	1.15	1818	0	16	24	29	115	253	91
MAFS	4.22	1.58	2623	19	61	110	91	139	58	50
MMT	**2.30**	1.67	**4141**	263	80	68	40	40	29	8
TABU	2.97	1.50	3513	109	109	120	105	63	10	12

Table 6.2: Ranking results on the hard instances of `chi1000`.

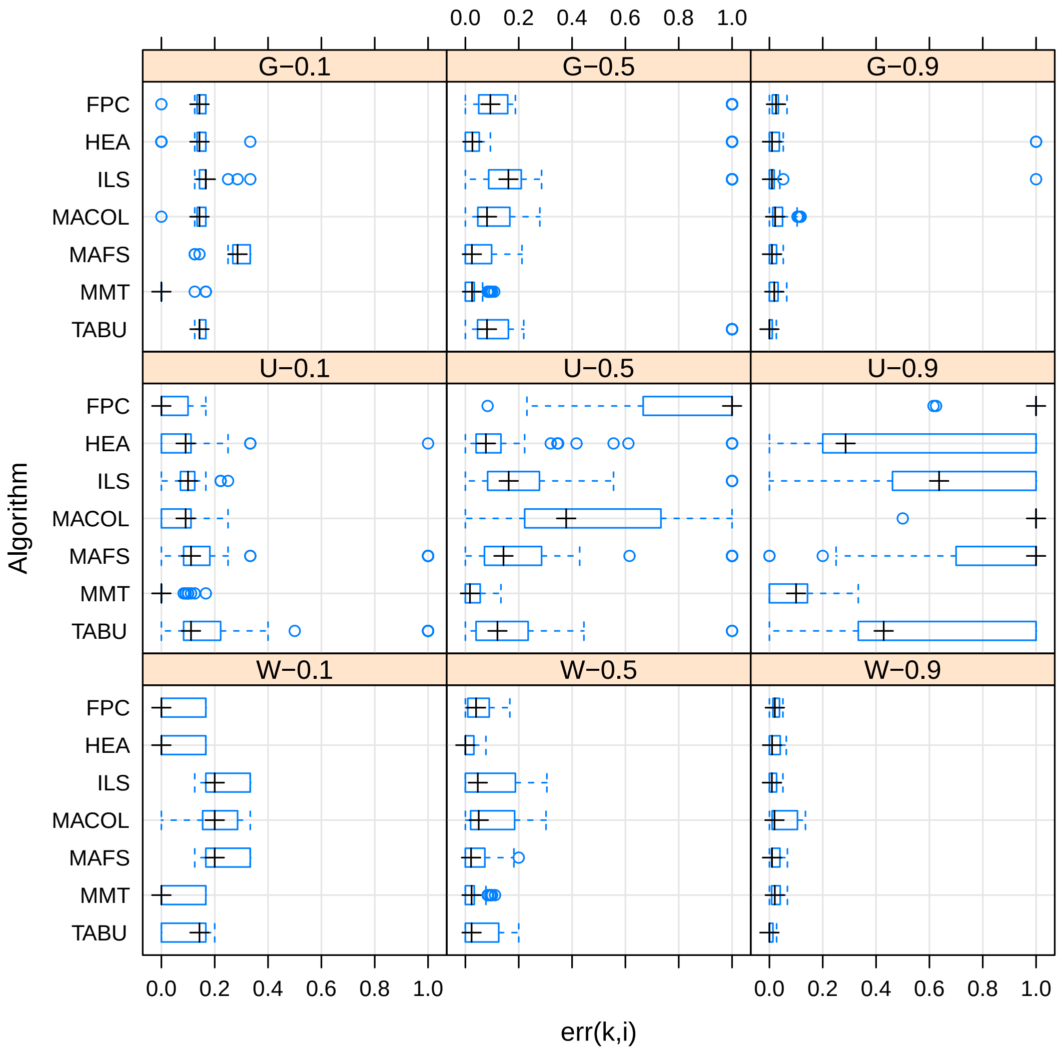

Figure 6.7: Distance metric $err(k, i)$ of the algorithms on the hard instances of `chi1000`. The results are grouped by the graph class and the density.

the end of the ranking. The formula-one method enlarges the distance between `MMT` and `HEA`, mostly because it is not normalized and depends on the number of instances where the algorithm achieves the first place. However, using this criteria does not change the order of the heuristics compared with using average rank, although the gaps between some competitors change slightly.

For a more insightful illustration, Figure 6.8 shows the ranking results on the hard instances grouped by the different subsets. As expected from the $err(k, i)$ measurement, `MMT` achieves

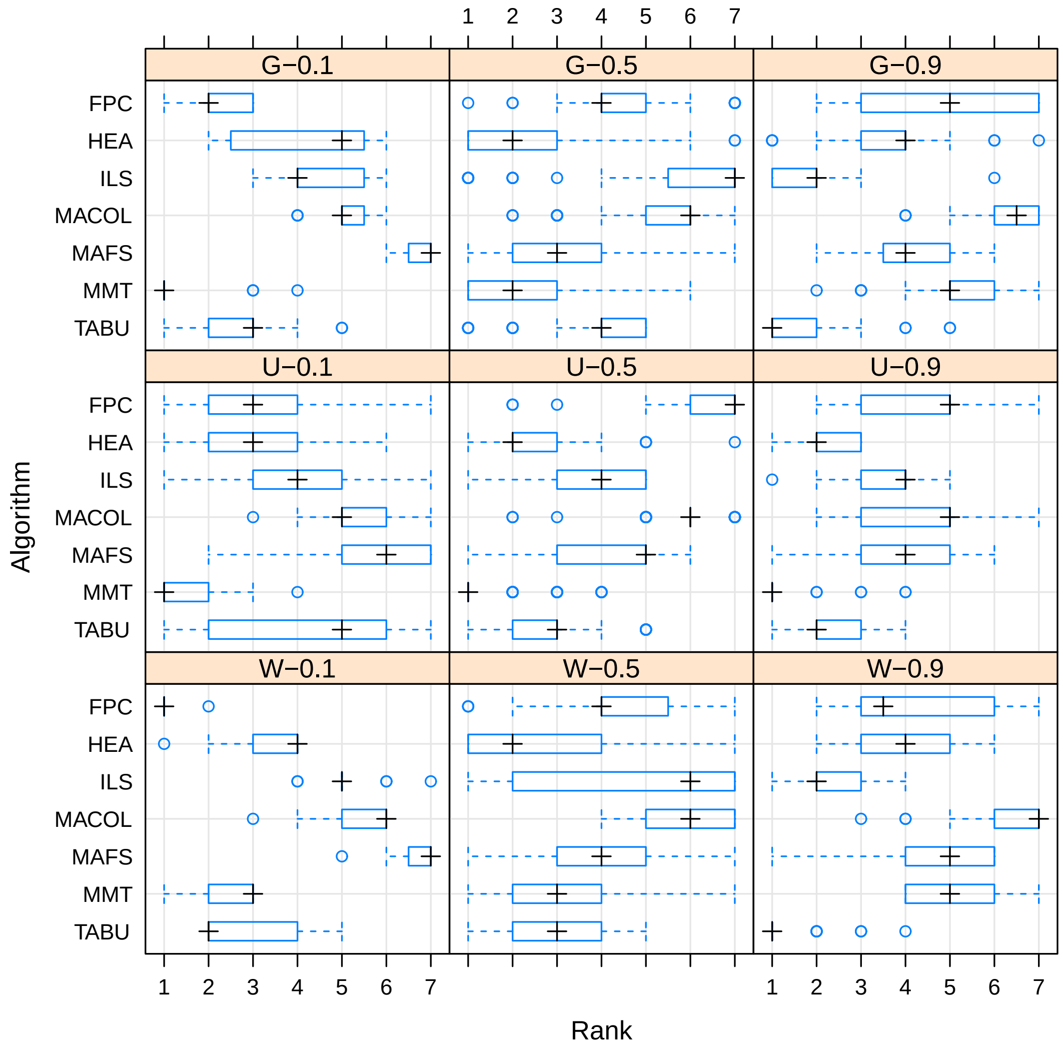

Figure 6.8: Boxplot diagram showing the ranking of the algorithms on the hard instances of `chi1000`. The results are grouped according to graph class and the density.

ordinary good results on the *geometric* graphs `U-*` while on the subsets `G-0.9` and `W-0.9`, `TABU` is the best-suiting algorithm, followed by `ILS`. One observation worth mentioning is that on some subsets, where MMT obtained very low values of $err(k, i)$, MMT is not the one with the lowest average rank. Especially on the *uniform* and *weighted* graphs, this method is many times ranked on mid-level positions although the required number of colors seems to be low. This confirmed our assumption that the success of MMT is most times not caused by a fast search, but rather by the quality of the found solution. A good example for this is W-0.1 where FPC is very effective. On these instances, MMT, HEA and FPC perform equally with respect to the distance $err(k, i)$, but FPC is apparently faster and therefore to prefer. Another example is W-0.9 where the median rank of MMT is only 5 although it offers competitive results concerning $err(k, i)$.

6.1.3 Dimacs Instances

The third set containing the instances of the *DIMACS* challenge consists of 174 graphs of different size where we identified 33 of them as *trivial*. We further separated 35 instances where no better coloring has been found during our experiments (marked as *trivial2*) and from the remaining 106 graphs, we classified 52 as *easy* and 54 as *hard*.

Figure 6.9 gives an overview on the number of best solutions each algorithm achieves. In

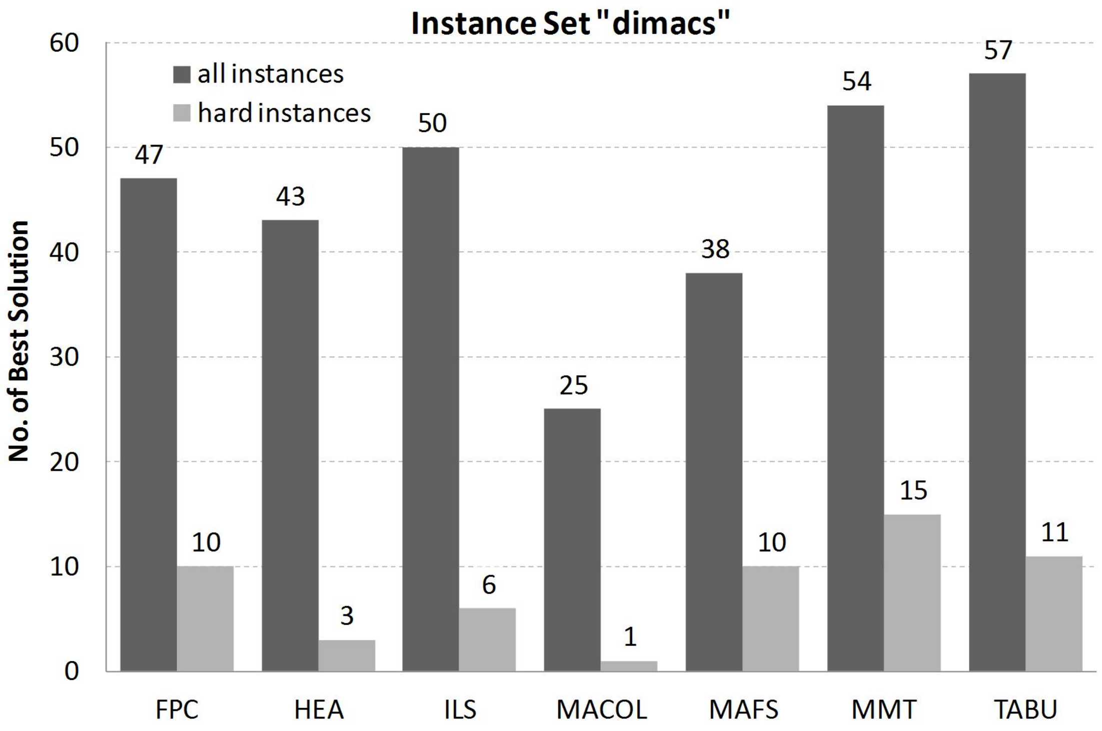

Figure 6.9: Number of instances from the set dimacs on which the algorithms show best performance.

contrast to the previous two instance sets, all heuristics showed on at least one hard instance the best solution. Moreover, the gaps between the different algorithms are rather small, which may also depend on the small number of instances. The method which has at the most instances the best performance is again `MMT`, but only with a small margin of 4 instances to its successor `TABU`.

Figure 6.10 gives a detailed view on the distance $err(k, i)$ on some selected subsets of instances. Surprisingly, `MAFS`, which does not shine out so far, shows the best performance

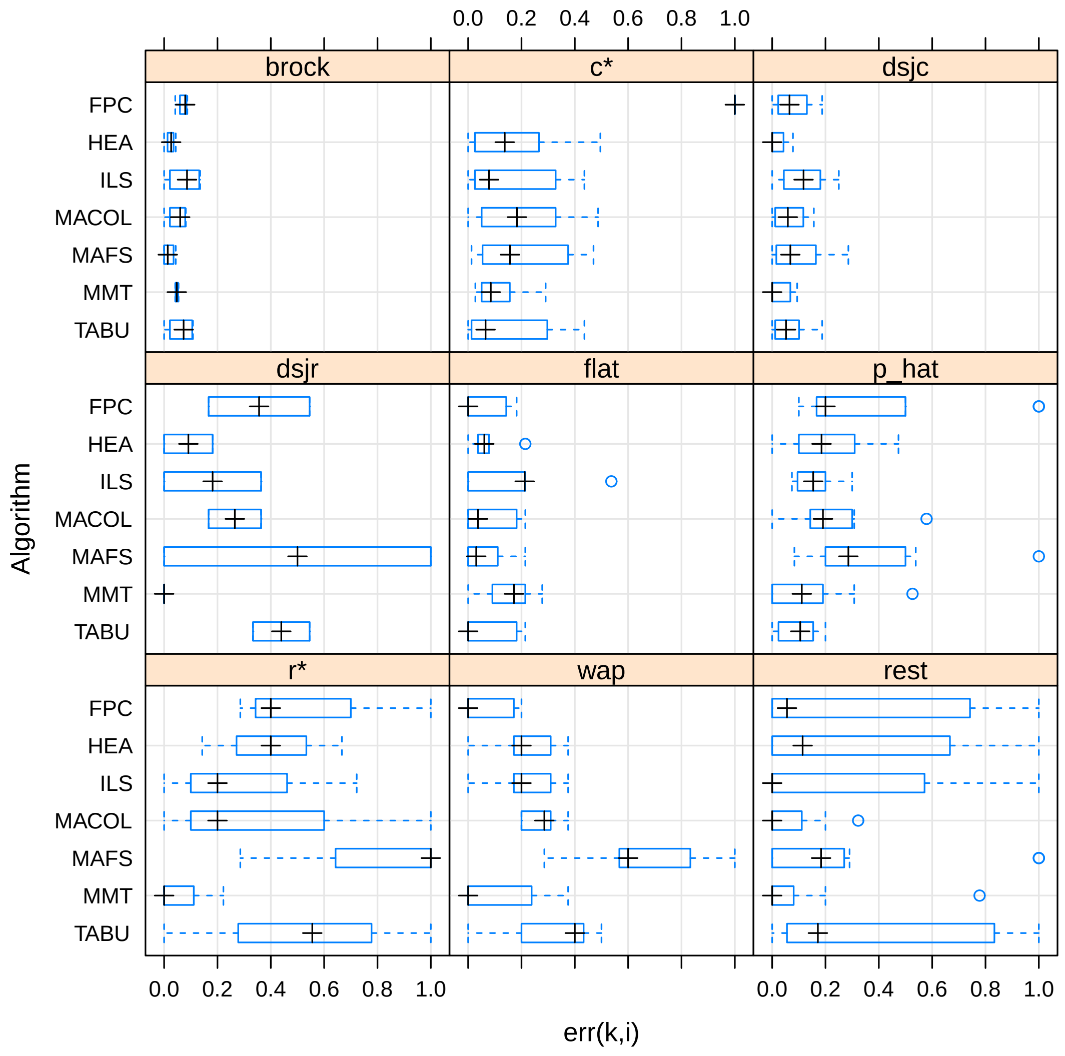

Figure 6.10: Distance metric $err(k, i)$ of the algorithms on the hard instances of `dimacs`. The results are grouped by the graph class and the density.

on the graphs of the set `brock`, closely followed by `HEA`. On the graphs grouped in `c*`, `TABU`, `ILS` and `MMT` are well-suited while `FPC` fails completely to find any coloring lower than the *greedy* methods. On the instances of `dsjc`, `MMT` and `HEA` are the leading approaches and on `dsjr`, `MMT` is able to find on all instances the best known solution. In contrast to this, this algorithm performs badly on flat graphs where HEA, MACOL, MAFS, FPC, and TABU form the most appropriate algorithms. The results on subset p_hat reveal a rather inhomogeneous distribution among the different heuristics where TABU is the most robust and successful one. On the graphs of r*, MMT is again able to find in most cases the best coloring while MAFS is, as for the instances of dsjr and wap, inappropriate. Concerning the latter subset, FPC and MMT are well-suited with slightly advantages for FPC. We further conclude the results of the remaining graphs in the category rest, where besides MMT, MACOL and MAFS achieve good performance.

As already mentioned, one central disadvantage of the *distance* $err(k, i)$ is that it only considers the number of colors and ignores the runtime. To include also the runtime of the heuristics, we also analyzed the rank within the tested algorithms, which combines the solution quality and the required time. Figure 6.11 shows the ranking of the algorithms separated by the different subsets. One interesting observation is that, although HEA and MAFS show similar values for $err(k, i)$ on the set brock, the latter achieves a lower median rank which indicates that is requires less time than HEA. A similar conclusion can be drawn between HEA and MMT on the instances of dsjc where HEA's median rank is below the value of MMT. Also on the graphs of dsjr, where MMT finds on all instances the best solution, is MMT's mean rank equal to the one of HEA and above the one of the ILS. Concerning the graphs of c*, the algorithm with the lowest ranks is TABU followed by ILS. On the subset dsjc the best heuristic is HEA while on dsjr, ILS shows the strongest performance. On flat graphs is FPC the leading heuristic followed by MAFS while on the subset p_hat these two algorithms are the worst and TABU is the best. On the remaining three subsets r*, wap and rest MMT reaches the lowest mean rank although on the last group the results are diffuse.

6.1.4 Conclusion

Our experiments clearly show that none of the algorithm is superior to the others on all classes of graphs. Although this observation is not new and it coincides to previous work [160, 47], it is one essential requirement for algorithm selection. On most of our considered subclasses of instances, there are one or two heuristics which perform better than the rest and which qualify themselves to be included in a potential set of candidate solvers. Furthermore, many algorithms showing good results on some subset of instances perform bad on other classes, which conforms to our assumption that heuristics follow some hidden structure which highly influences the progress of the search.

The second observation is that there must be some intrinsic features of a graph that influence the performance of the different coloring algorithms and that can be ascribed to construction parameters like the graph class. Thus, the different methods to build graphs result in instances with different inner structure that also influences the solving of the GCP

These are two crucial preconditions for algorithm selection and the goal for the following experiments is to evaluate if these inherent attributes can be extracted using simple metrics and if they allow a prediction of the best-suited algorithm for a particular instance.

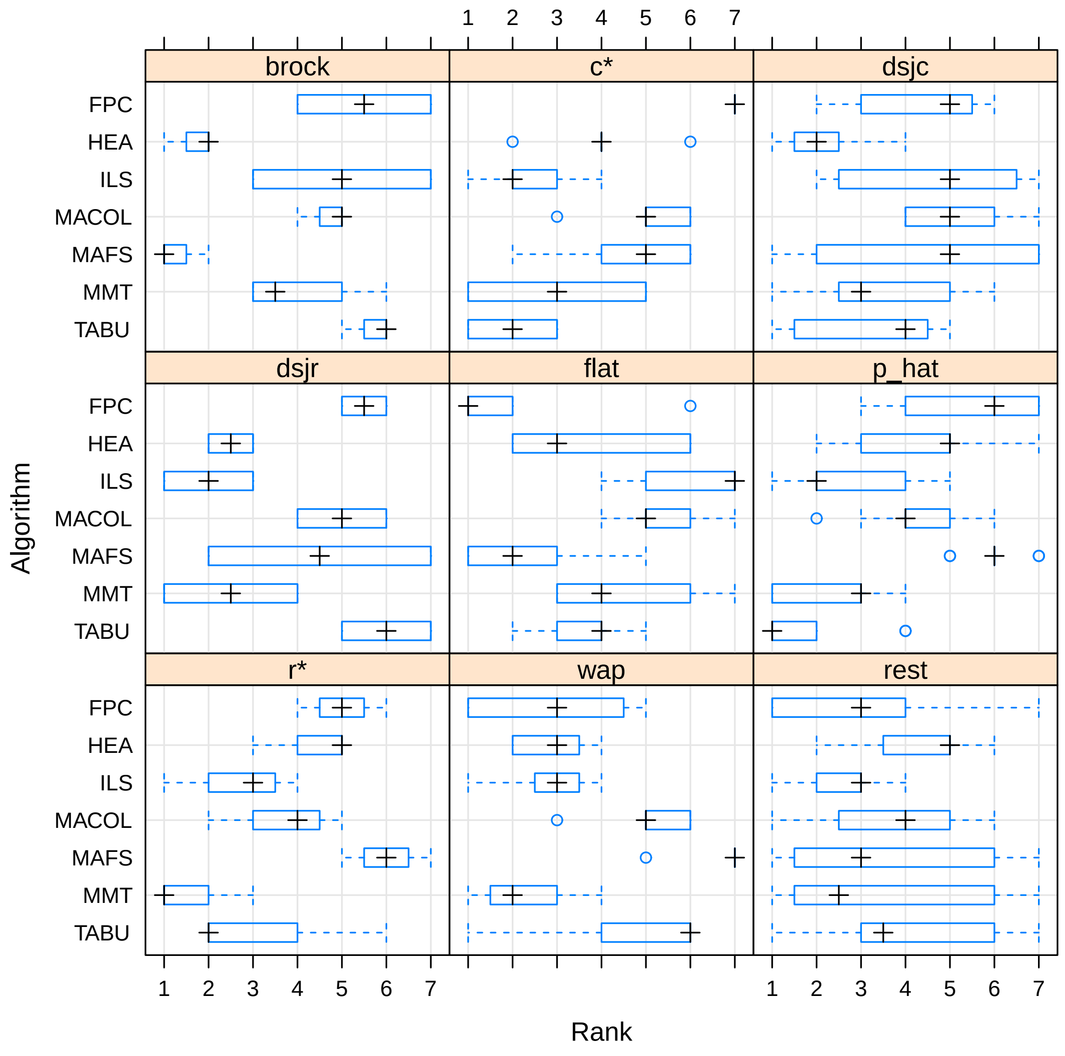

Figure 6.11: Boxplot diagram showing the ranking of the algorithms on the hard instances of `dimacs`. The results are grouped according to graph class and the density.

6.2 Solvers based on Algorithm Selection

6.2.1 Terminology

Before we take a look on the results of our experiments regarding algorithm selection based on machine learning, let us shortly describe some terminology and nomenclature used in the following section.

As mentioned before, we tested graphs from three different sets, called `chi500`, `chi1000` and `dimacs`. Unfortunately, the last set contains only 54 *hard* instances (according to our definition), but usually more instances are needed for a meaningful application of machine learning. Therefore, we combined these instances with those of `chi500` and `chi1000` to one big set denoted as `mixed`. This set comprises all 859 *hard* instances and represents, as it is much more heterogeneous than `chi500` and `chi1000`, a more realistic and harder setting for an algorithm selection. For that reason, we will focus our evaluation on the `mixed` set of instances.

In our experiments, we tested different instance sets, preparation steps and feature subsets. To provide a better overview on these modifications, we introduce here a naming scheme for the data sets. In detail, the different sets are labeled as *set*_h*x*_*base*_*disc*_*fs* where $set \in \{\text{chi500}, \text{chi1000}, \text{mixed}\}$ represent the set of instances, hx with $x \in \{3, ...8\}$ the number of used heuristics, $base \in \{\text{u}, \text{e1}, \text{e2}, \text{e3}\}$ [1] the attributes on which the feature selection is applied, $disc \in \{\text{none}, \text{mdl}, \text{kon}\}$ the applied discretization method and $fs \in \{\text{none}, \text{bff}, \text{gen}\}$ the method used for feature selection. For example, the data set `chi500_h7_b_mdl_bff` represent the data for the instances of `chi500` using the best 7 algorithms. Moreover, the features contains of non-expanded attributes (`b`), which are discretized using the MDL criteria and selected via a best-first forward selection search.

Concerning the used heuristics, we test 5 sets containing the best $x \in \{3, ...8\}$ heuristics with respect to the number of first places in our ranking. Therefore we start with all algorithms `h7` (see Section 5.1.1) and exclude successively the algorithms with the lowest success. A detailed description about the different sets of heuristics and the sorting criteria is shown in Table 6.3.

6.2.2 Feature Selection

Before we take a look on the performance of the various classification algorithms, let us shortly discuss the results of our feature selection. Main purpose of this process is to eliminate redundant or useless attributes so that the classifier can focus on the relevant characteristics of the instances. To analyze the importance of the different classes of attributes, we count for each attribute how often it is selected, sum this value up according to their class and normalize the sum by the number of attribute selections. Figure 6.12 shows the average number of attributes per class on the selected subsets of the unexpanded data sets. From the figure it is apparent that data sets which have been discretized contained significant less attributes compared to the non-discretized ones. One reason for this might be that in the course of the discretization, the values of some features are all mapped to one single nominal value. Thus, all instances have

[1] We applied a *basis function expansion* on the results of the feature selection using the numerical values and those of the discretized ones. The result of the non-discretized cases is marked as `e1` while `e2` (`e3`) is based on outcome of the feature selection on data discretized with the *MDL* (KON) criteria.

Rank	Algorithm	No. Best Solution	Algorithm Set				
1	MMT	367					
2	TABU	194				h4	h3
3	HEA	143			h5		
4	ILS	88	h7	h6			
5	FPC	51					
6	MAFS	46					
7	MACOL	4					
8	GLS	1					

Table 6.3: Ranking according to the total number of best solutions on the instances of `mixed`. Note that `GLS` has only been tested on the instances of `chi500` and is listed for the sake of completeness.

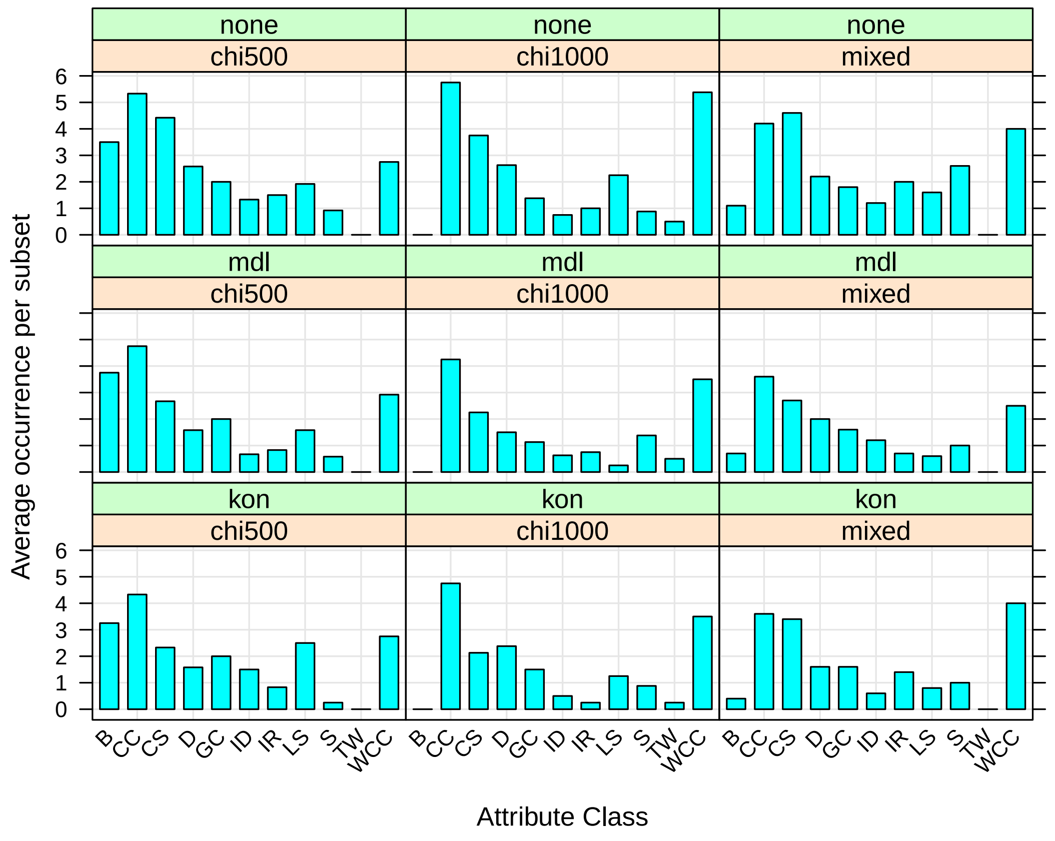

Figure 6.12: Average number of attributes selected per attribute class in the unexpanded data sets.

on that features the same value, which makes it meaningless and therefore, it is removed by the feature selection. Another explanation is that the discretized data are more significant than the original one and therefore, less attributes are needed for a similar performance with respect to the selection criteria. Nevertheless, these are just some possible explanations and more detailed investigations in this topic are left for future work.

Concerning the chosen attributes, it is easy to see that features of some classes are more frequently used than others. Surprisingly, the frequency of being selected varied over the three sets of instances, especially between `chi500` and `chi1000`. For example, attributes concerning the lower and upper bound are frequently used in `chi500` while never on the data of `chi1000`, in contrast to the attributes based on the tree width, which are only selected for the data of `chi1000`.

The most selected features are in almost any settings those based on the *clustering coefficient*, the *weighted clustering coefficient* and the *clique size*. In detail, in each selected subset there are on average between 2 and 6 attributes of that three classes, which indicates high importance to these features. Note that these figures highly depends on the number of attributes per type, which is for some classes higher than others (e.g. the class CC (*clustering coefficient*) has 10 different attributes while the class S (*size features*) 5 and the class TW *tree width* only 2 attributes). Nevertheless, we believe that it reveals some insight on the selected attributes and the importance of the classes.

Concerning the expanded attributes, Figure 6.13 shows the results using all selected subsets (expanded and unexpanded). Note that, as the expanded attributes are based on attributes of two classes, we count them twice - once for each class of the underlying attributes.

Again, attributes based on the *clustering coefficient* or the *weighted clustering coefficient* are highly represented while the features concerning the *clique size* occurred, in relation to the results on the unexpanded attributes, less frequently. On the feature sets of `chi500`, attributes depending on *bounds*, the *node degree* and *local search* are often selected while for the data of `chi1000`, besides *clustering coefficient* and the *weighted clustering coefficient*, also the class of features using the *clique size* is important. On the data of `mixed`, again the classes *clustering coefficient* and *weighted clustering coefficient* are those with the most selected members.

Another interesting result is that for all three sets of instances, there were some features which are included in almost any selected subset. For example, on the data sets for `chi500` the attribute $\mathrm{GC}_{\mathrm{D/R}}$ is selected in 95.89% of the subsets, closely followed by $\mathrm{GC}_{\mathrm{R/D}}$ which is included in 89.04% of the subsets. On the data sets for `chi1000`, the most-selected attribute is CC_{g} (83.33%) followed by $\mathrm{CC}_{\mathrm{mean}}$ (75.00%). These attributes occurred also in the results from the feature selection on data sets containing all graphs very often ($\mathrm{GC}_{\mathrm{D/R}}$ (81.01%), $\mathrm{GC}_{\mathrm{R/D}}$($50.63\%$), CC_{g} (64.56%), $\mathrm{CC}_{\mathrm{mean}}$ (82.28%)). This indicates that these attributes have a higher correlation to the classification variable. However, the attribute with the highest number of occurrences in the data of *mixed* is $\mathrm{CC}_{\mathrm{max}}$ (86.08%). For more information about the single attributes, we refer to the Appendix A.4 where a detailed listing of the most-selected attributes can be found.

A different aspect of this evaluation is also to detect attributes which do not correlate with the best algorithm and are therefore never selected. As also seen in Figure 6.12, one class of attributes which is almost never chosen are the features concerning the *tree decomposition*. The

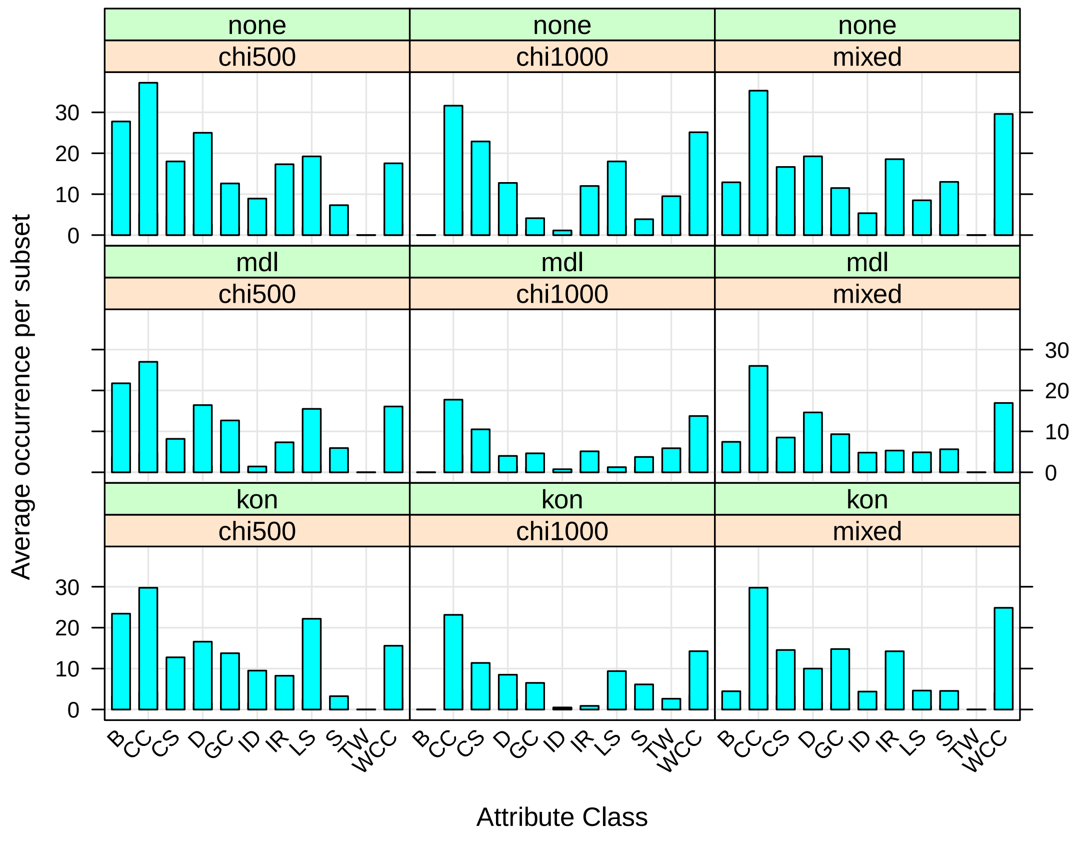

Figure 6.13: Average number of attributes selected per attribute class in all data sets.

attribute TD_{time} is never selected and the minimal tree width TD_{width} is only used on the data sets of `chi1000`. This let us conclude that these metrics are in general not very important for algorithm selection for the GCP.

Another features which are (almost) never selected are the results of the two greedy methods, GC_{RLF} and GC_{DSAT} and the time needed for their calculation, $\text{GC}_{T-\text{DSAT}}$ and $\text{GC}_{T-\text{RLF}}$. Only GC_{DSAT} is selected in two subsets while the others are never chosen. The reason for this is that these variables are not normalized and contain therefore none general meaning. Moreover, multiple related attributes like $\text{GC}_{\text{D/R}}$ or $\text{GC}_{\text{R/D}}$ are among the most-used ones, which leads us to the conclusion that the greedy algorithms `DSAT` and `RLF` are important, but only in combination with other attributes.

There are of course other variables which are never selected (e.g. LS_{ii} or D_{max}), but most surprising for us was that also the number of nodes S_n is one of the unused attributes. This was expected for the data sets of `chi500` and `chi1000` where all instances have the same size, but also on the set of all instances, `mixed`, this attribute is never selected. One explanation is that the number of nodes does not vary so much to have a high dependency on the performance of the algorithms.

6.2.3 Classifier Parameter Evaluation

In the following section, we analyze the performance of the different parameter settings for the classifiers. For this purpose, we take a look on the accuracy on the classifications grouped by the used discretization method and the algorithm space.

Bayesian Network

As mentioned before, we tested 5 different settings for the BN classifier. These settings differ on the number of parent nodes each node can have. Figure 6.14 shows the accuracy values of the different configurations on the observations from the instance set `mixed`. Considering the different settings, the plot clearly shows that using only 1 parent node always leads to suboptimal results and that by incrementing the value of this parameter, also the accuracy increases. At value of 3, this effect stagnates and additional parent nodes do not further improve the results. In addition, the diagram reveals that the less algorithms for the GCP are used, the higher the average accuracy gets, regardless of the used parameter setting. This is reasonable, as the less heuristics are used, the less classes are available and the higher are the number of classifications for the remaining algorithms, which makes it easier for the classification algorithms. Another interesting observation is that on all tested data sets, the classifier seems to perform on the non-discretized data sets worse than the using those from the groups *MDL* or *KON*.

k-Nearest Neighbor

Figure 6.15 shows the results of the different parameter configurations for the kNN classifier on the instances of the data set *mixed*. For this algorithm, we experimented with the number of nearest neighbors *k* (denoted as IB*k*) whereby we tested values of $k \in \{1, 3, 5, 7, 9\}$. The experiments clearly illustrate that on the non-discretized features a larger neighborhood is more successful than considering only a small number of neighbors. This effect is especially observable on the dataset of `chi500`, where the difference between using 1 and 9 nearest neighbors is up to 10%. On the discretized data of `mixed`, this effect on the average accuracy almost disappears. Only between using 1, 3 and 5 neighbors, marginal improvements can be observed. Although there is on average no advance in using higher values, the best results were almost always obtained when using a large neighborhood. In addition, increasing the number of neighbors leads on average almost never to decreasing accuracy. Therefore, although larger k requires more computational effort, it may result in a better performance and can therefore be recommended.

Decision Tree

For the C4.5 decision trees (DT) classifier, we tested 4 parameter configurations with different *confidence factor* and number of objects per leave node (see Table 5.3).Figure 6.16 gives an overview on the accuracy of these settings on the instances of *mixed*. Unfortunately, none of the configurations seems to be clearly better than the others. On the non-discretized data, the

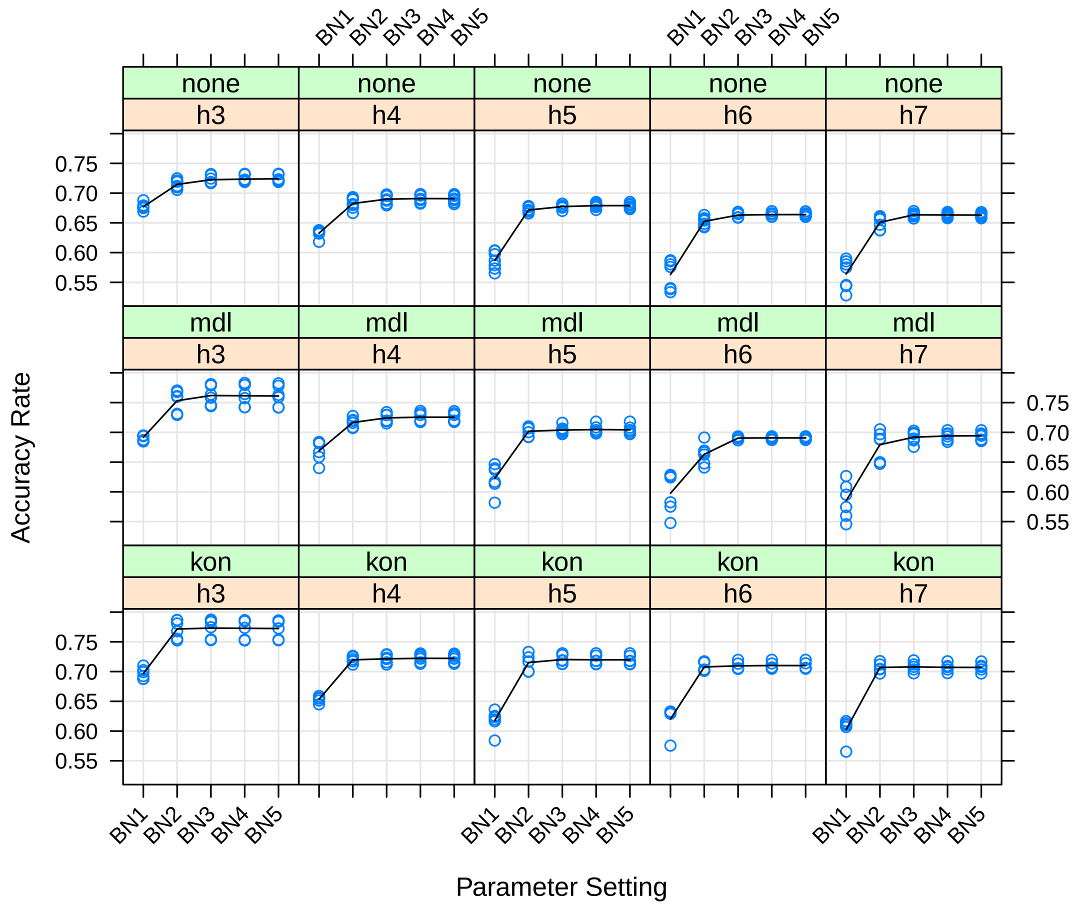

Figure 6.14: Accuracy of the BN classifier on the instances of *mixed* using the different parameter settings. The results are grouped by discretization method and algorithm set. The dots represent a result on a particular data set and the black line indicates the mean value.

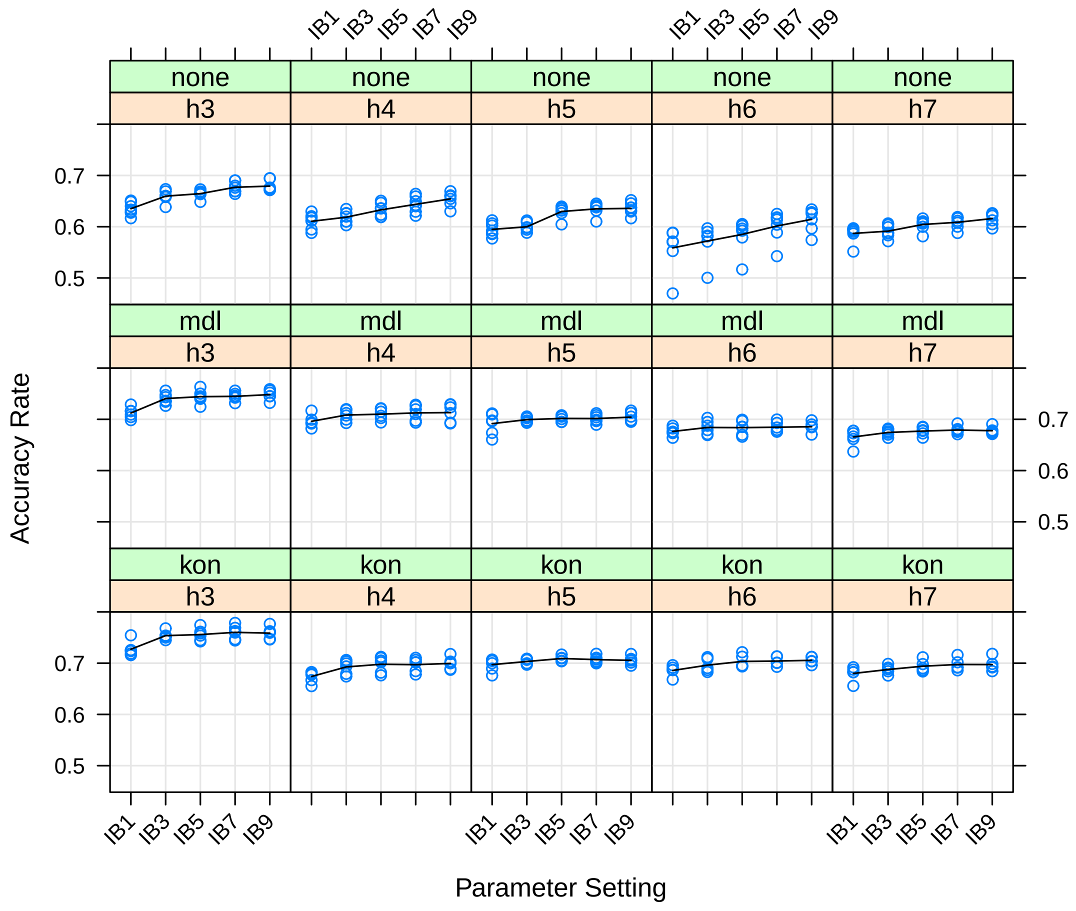

Figure 6.15: Accuracy of the kNN classifier on the instances of `mixed` using different parameter settings. The results are grouped by the discretization method and the algorithm set. The dots represent a result on a particular data set and the black line indicates the mean value.

settings `DT2` and `DT4` perform slightly better than the rest while when using discretized features, these configurations perform worse than `DT1` and `DT3`. In the latter cases using nominal values, the default parameters of *Weka* (`DT1`) work on average marginally better than the others. Remarkable is the gab between using the best 4 and 3 algorithms, where the accuracy increases on average up to 5% with the decreasing amount of used heuristics.

Random Forest

For the random forests (RF) classifier, we tested 2 parameter settings that differ only in the number of generated trees (`RF1` using 10 *decision trees*, which is the default value of *Weka*, and `DRF2`, which comprises 15 trees). Figure 6.17 gives an overview on the results on the data

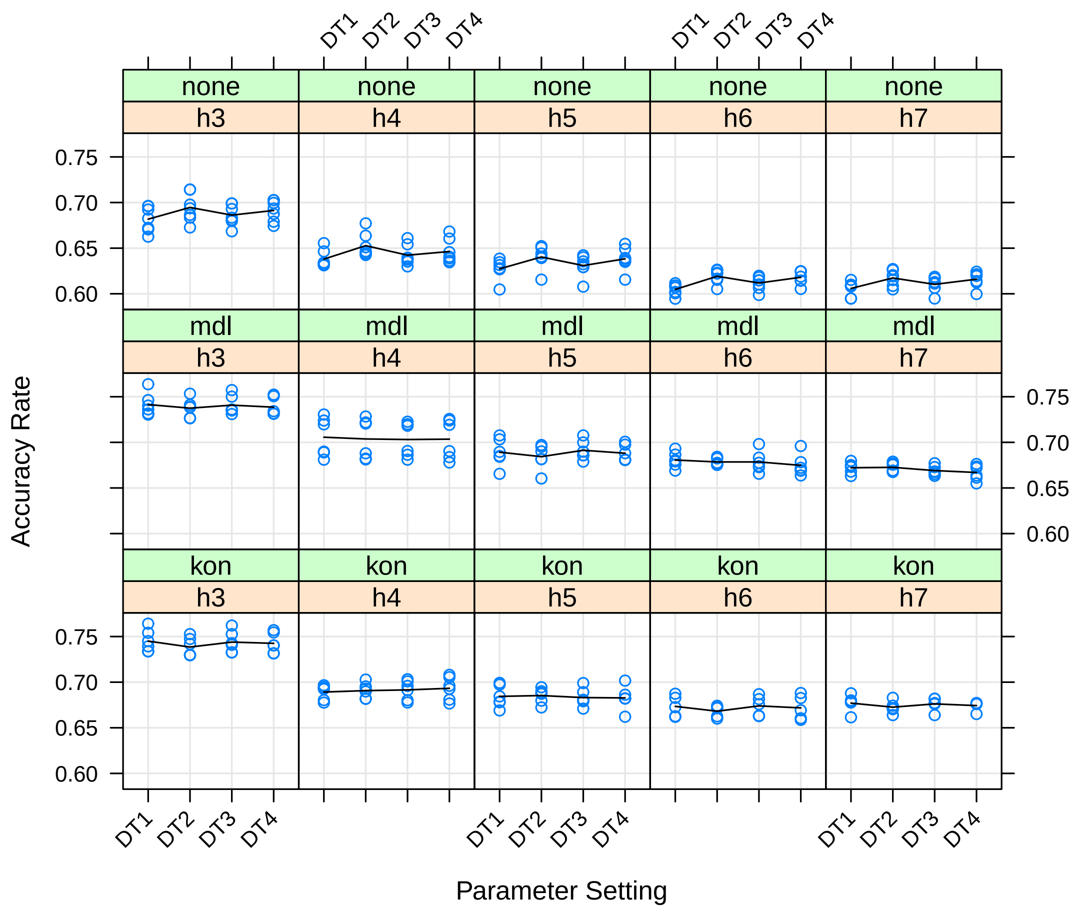

Figure 6.16: Accuracy of the C4.5 decision trees (DT) classifier on the instances of `mixed` using different parameter settings. The results are grouped by the discretization method and the algorithm set. The dots represent a result on a particular data set and the black line indicates the mean value.

sets of `mixed`. The data show that both settings are performing nearly equal with some minor advantages for the second configuration, especially on the non-discretized data.

Multilayer Perceptron

Concerning the multilayer perceptrons (MLP), we tested 2 configurations which only differ on their learning rate (`MLP1` with a value of 0.3 and `MLP2` with a value of 0.4). The results for the data sets of `mixed` are displayed in Figure 6.18. On the non-discretized data, the accuracy of both settings is on average nearly similar and varies only in correlation with a decreasing amount of heuristics used. The graphic further reveals that this classifier has major problems on the

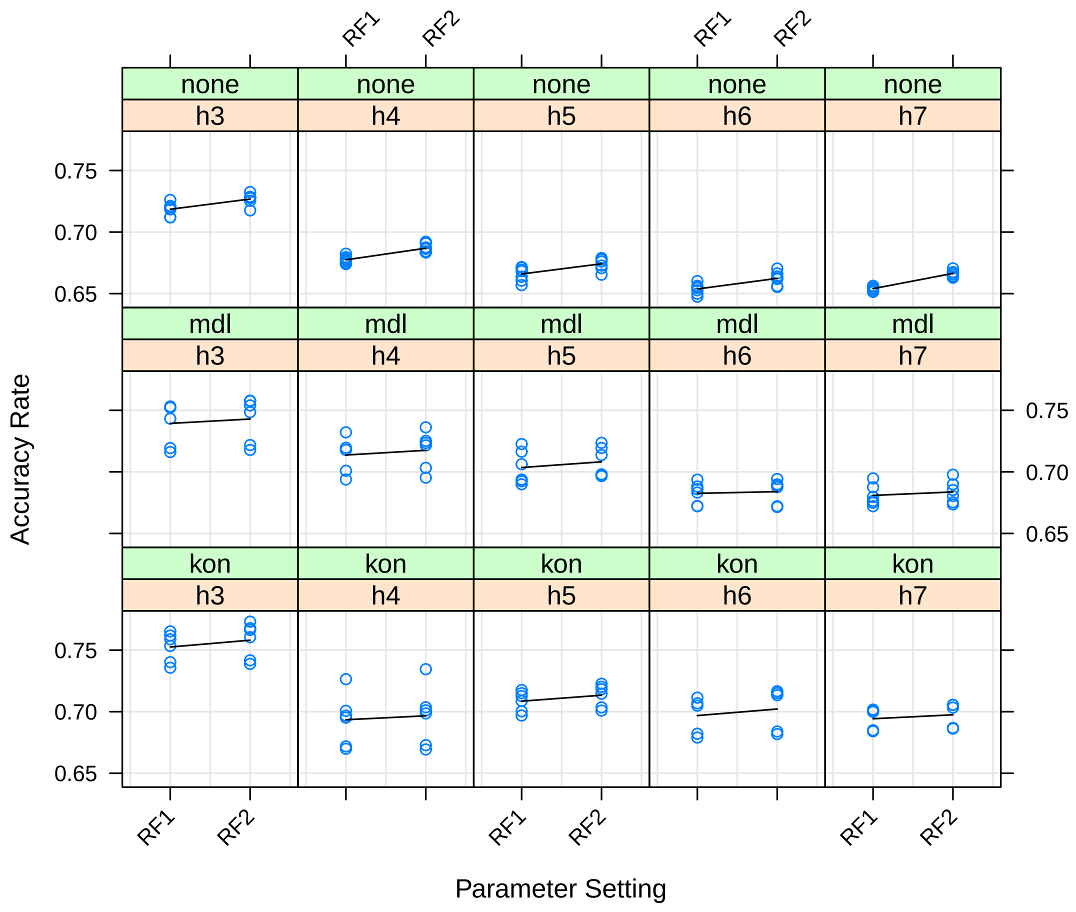

Figure 6.17: Accuracy of the random forests classifier on the instances of `mixed` using the two tested parameter settings. The results are grouped by the discretization method and the algorithm set. The dots represent a result on a particular data set and the black line indicates the mean value.

discretized data and in particular when using the best 6 or 7 algorithms for the GCP. Especially with larger amounts of features, a MLP approach required much more time compared with the other methods. In detail, testing one classifier took on 10 of the 186 data sets of `mixed` more than 12 hours and the average accuracy on these cases was 50% for the configuration `MLP1` and 33% for `MLP2`. Surprisingly, using a similar amount of continuous parameters does not result to such a behavior, which leads us to the conclusion that this classifier is not suited for discretized data. Concerning the 2 parameter settings, our results indicate that when they are applied on discretized data, the default setting of *Weka* is with 65% slightly better than using an increased learning rate, which results in an average accuracy of 60%. On the numeric data, both settings seem to be equal successful (an accuracy of 64% versus 65%) with also competitive

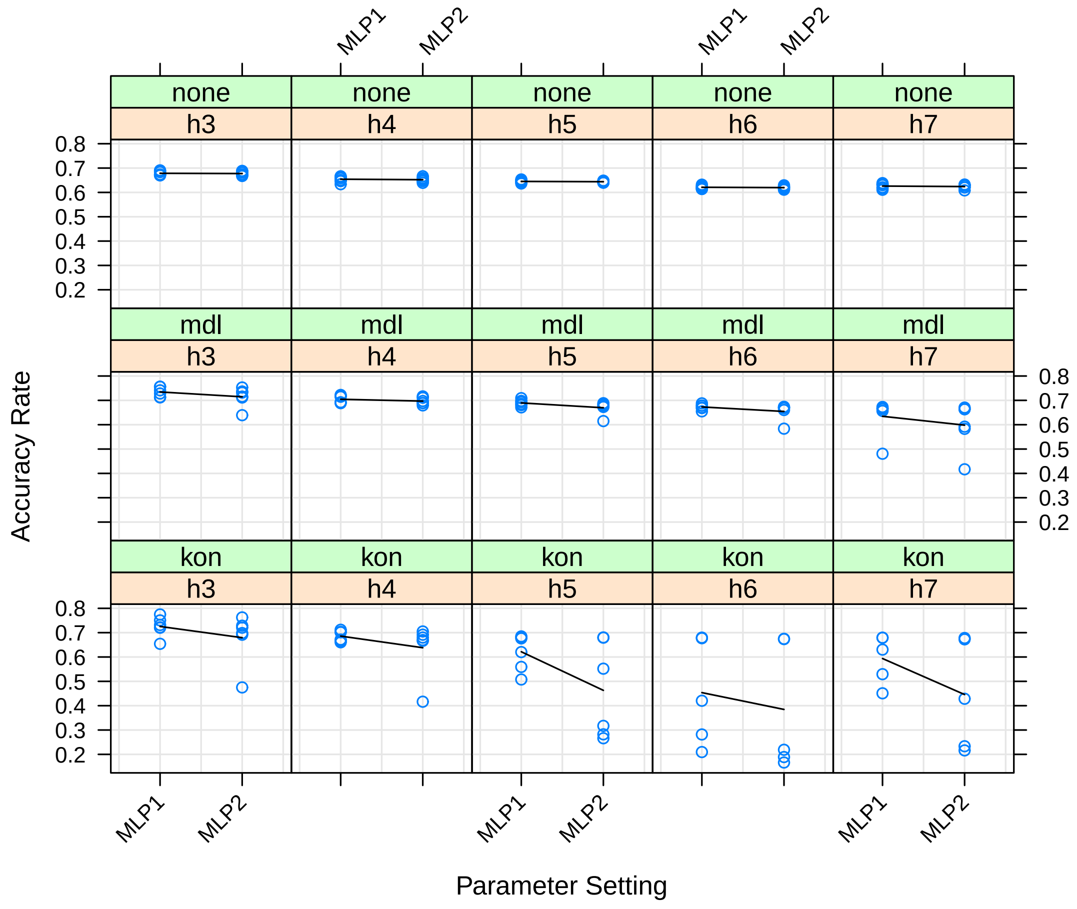

Figure 6.18: Accuracy of the MLP classifier on the instances of `mixed` using the two tested parameter settings. The results are grouped by the discretization method and the algorithm set. The dots represent a result on a particular data set and the black line indicates the mean value.

results concerning the runtime.

Support Vector Machines

The last considered classifier uses support vector machines (SVM) and we tested 8 parameter configurations with different *kernel functions*, *exponentials* and *complexity parameters* (see Table 5.4). Figure 6.19 shows the results of the parameter settings on the data sets of `mixed` separated by the discretization method and the considered heuristics for the GCP. Concerning the non-discretized data, it is easy to see that using a PUK kernel (`SMO8`) leads to the best results. The highest average accuracy using the polynomial kernel can be achieved with the settings `SMO6` and `SMO7`. On the discretized cases, the most suitable configurations are `SMO1` and

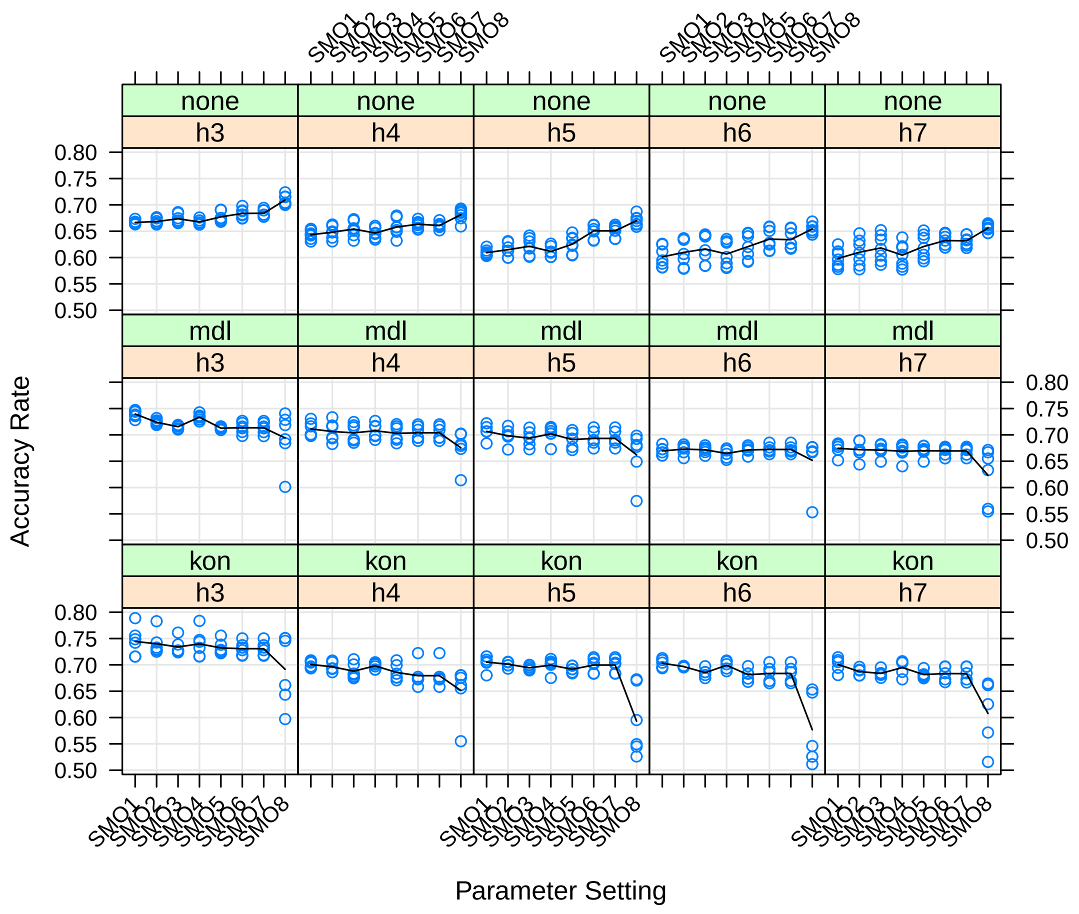

Figure 6.19: Accuracy of the SVM classifier on the instances of `mixed` using the tested parameter settings. The results are grouped by the discretization method and the algorithm set. The dots represent a result on a particular data set and the black line indicates the mean value.

`SMO4` while the approach using a PUK kernel (`SMO8`) obtained by far the worst values. Another observation is that on nearly all data sets, there was no difference between the configurations `SMO6` and `SMO7` which indicates that increasing the complexity factor further does not enhance the accuracy. In general, the accuracy of the discretized data is, depending on the configuration, up to 10% better than using only numerical values (e.g. see the best values of the configuration `SMO1` using the algorithm set `h3`).

6.2.4 Variation in the Algorithm Space

One question that arose during our experiments was if and how the choice of the algorithm space influences the performance of the classifier. Assuming that, the less choices a classifier has, the more likely it will predict the correct class, we wanted to see if dropping some algorithms result in a higher total performance. For that reason, we applied all our tests using only the best $x \in \{3, ..., 8\}$ heuristics. Figure 6.20 displays the results concerning the *success rate* of the different classifiers separated by the discretization method. Recall that the *success rate* s(c,I,A) of a solver is the ratio between the number of instances an which the solver achieves the best solution and the total number of instances. Thus, this metric gives information about the performance of a solver in relation to a set of solver (in our case, the set of tested algorithms `h7`). It is easy to see that using only the best 3 heuristics always leads to worse performance

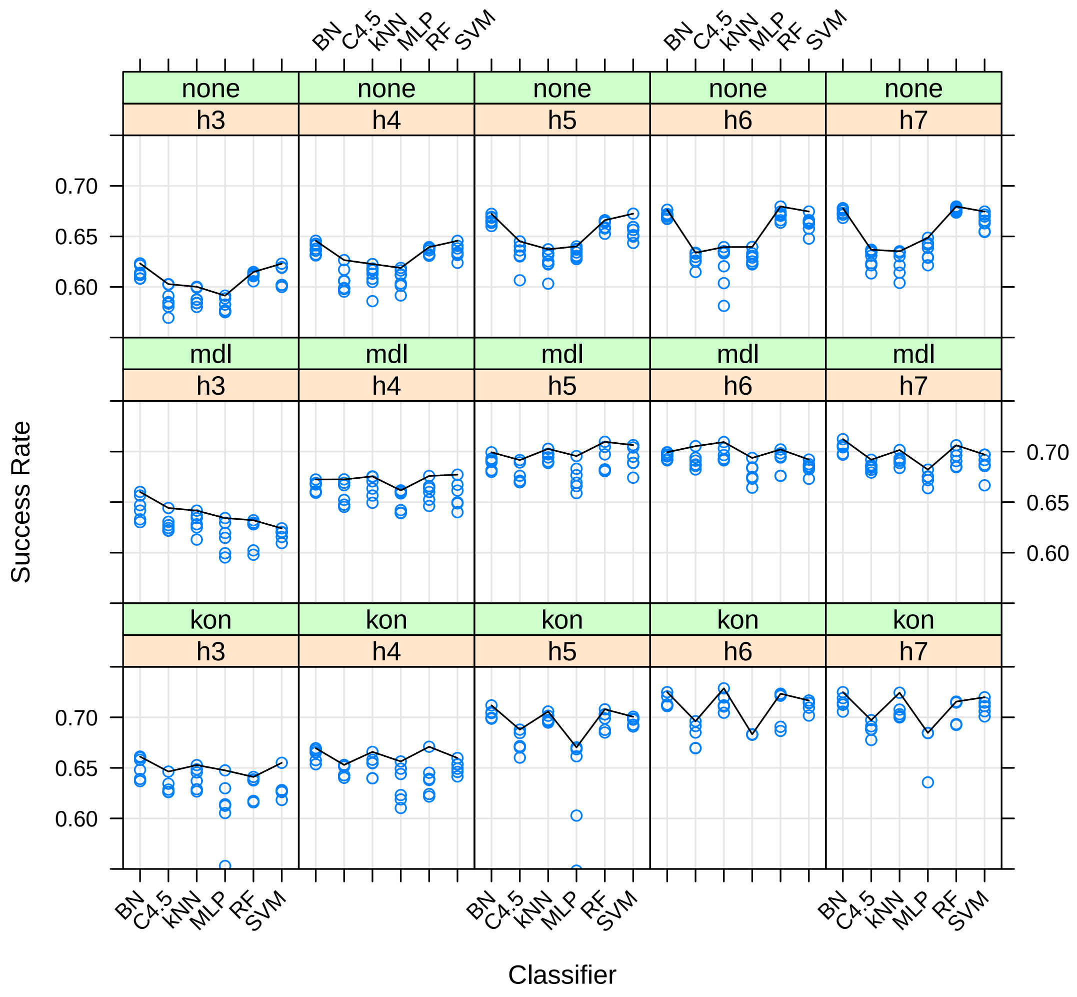

Figure 6.20: Success rate s(c,I,A) of the tested classifiers using the best $x \in \{3, 4, 5, 6, 7\}$ heuristics (denoted as hx) on the instances of `mixed`. The dots represent a result on a particular attribute subset and parameter configuration while the line displays the best achieved result.

compared to using 4 or more algorithms. Especially between the algorithm sets *h3*, *h4* and *h5* all classifiers show larger gaps on their achieved success rate. For example, the classifier using a Bayesian network achieved on the non-discretized data sets of *h3* a best result of 62%, on *h4* a value of 65% and on *h5* a success rate of 67%. Considering the algorithm sets with more heuristics, our data show that the benefit of including additional algorithm decreases in relation to the performance of the new algorithm. Thus, changing the algorithm set from *h5* to *h6* (include `MAFS`) or adding `MACOL` to the set *h6* results only in minor advantages on the success rate.

6.2.5 Comparison of Classifiers

Besides the question of selecting a good algorithm portfolio, one important issue is the performance of the different classification algorithms in our experiments. For this purpose, we compared the success rate s(c,I,A) of the classifiers using their best parameter configuration. Figure 6.21 shows the highest achieved success rate per classifier separated by the discretization method and the used algorithm set. Again, it is easy to see that for all classifiers and starting

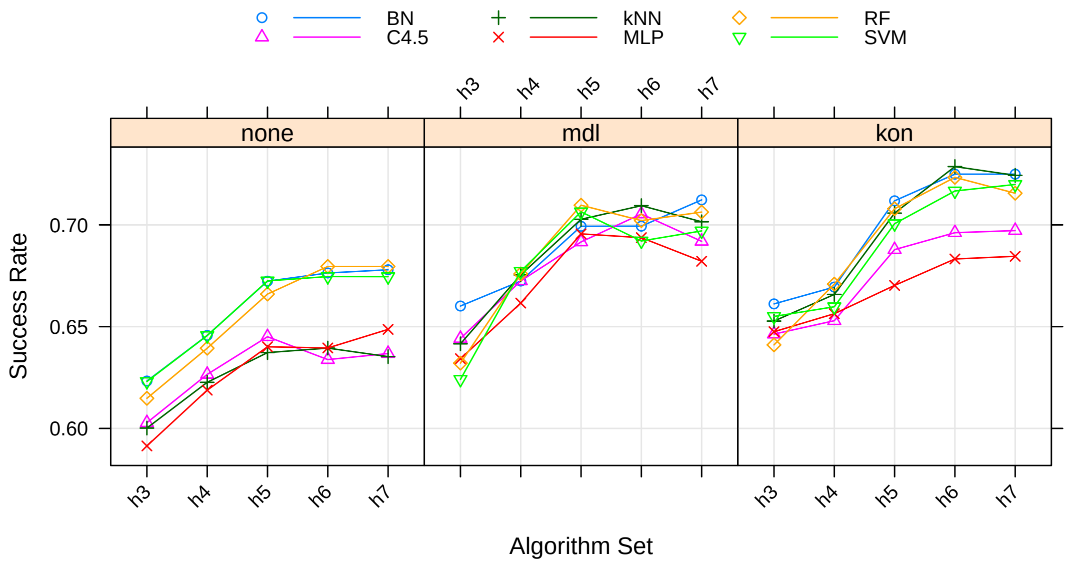

Figure 6.21: Success rate of the tested classifiers on the instances of `mixed`. The values are the best (highest) values using different attribute subsets and parameter configurations.

with the algorithm set `h3`, adding further heuristics increases the overall performance. Then, at a certain point the improvement stagnates and only minor progress, or in some cases, even a loss of the performance, occur. In the case of the numeric attributes, this effect occurs at the set `h5`, from where the classifier SMO, BN, kNN and MLP show only small changes compared to `h6`. Even more, the DT classifier has on `h6` a worse performance than on `h5` and only RF is able to use the additional heuristics for a better algorithm selection.

Method	BN		C4.5		kNN	
	avg (%)	best(%)	avg(%)	best(%)	avg(%)	best(%)
MDL	+ 2.54	+ 3.43	+ 5.38	+ 6.03	+ 7.42	+ 7.00
KON	+ 3.50	+ 4.70	+ 4.97	+ 5.22	+ 7.84	+ 8.92
Method	MLP		RF		SVM	
MDL	+ 2.64	+ 4.69	+ 2.32	+ 3.01	+ 2.85	+ 3.18
KON	- 7.47	+ 3.59	+ 2.58	+ 4.38	+ 3.07	+ 4.52

Table 6.4: Improvements of the success rate s(c,I,A) (in percent) when using discretized data in relation to the results achieved with non-discretized data on the instances of `mixed`.

Concerning the data sets prepared with the MDL criteria, the graphic shows that also some classifiers reach a peak using `h5`. For example, the RF and SMO show their best performance using `h5` and loose both on the success rate when they are applied with `h6`. In contrast to this, kNN reaches its best results on `h6` while BN seems to be the only classifier which makes advances of the bigger choice of algorithms of `h7`.

On the data where we applied Kononenko's criteria (KON) for discretization, it is observable that except BN and DT, all classifier reach a higher success rate when tested with the algorithm set `h3` than the corresponding settings on the non-discretized data or using the standard MDL criteria. However, applied with `h4` many classifiers achieve worse results compared to the classical MDL criteria. Especially MLP and DT show only a flat increase on their success rate. The other four classifiers reveal in general a slightly better performance than using the MDL criteria with a peak on `h6` for BN, kNN and RF while SMO reached its highest values on `h7`.

6.2.6 Effects of Discretization

Besides a comparison of the different considered heuristics, Figure 6.20 and Figure 6.21 reveal also some insights for the effects of the different methods of discretization. As mentioned before, transforming numerical values into nominal ones can have a significant effect on the performance of the classifier. Hence, we tested, apart from using the original data, two discretization methods denoted as MDL and KON. Concerning the impact of this data preparation, Table 6.4 clearly show that all classifiers can achieve higher values than using nominal values.

In detail, the column *avg* refers to the average success rate of the best parameter configuration for each data set while the column *best* represents the difference between the best value obtained on all data sets using no numerical values and the best value of the discretized data sets. As the reader can see, both discretization variants improve the best reached success rate whereby using the classical MDL method raises the value on average by 4.36% while using Kononenko's criteria increases s(c,I,A) by 3.80%. However, the best parameter and portfolio configurations, the benefits of discretized values for some classifier are up to $+7.00\%$ with MDL and even $+8.92\%$ using KON. Also worth mentioning are the bad results of the MLP using KON, where on average the success rate decreases by 7.47%.

Setting	Data Set	Accuracy (%)	$s(c, I, A)$ (%)	$err(k, i)$ (%)	Rank avg	Rank stdev	Rank F1	t (min)
BN3	h7_e3_kon_bff	71.89	72.49	4.06	1.50	1.01	7810	1
IB5	h6_e3_kon_gen	72.14	72.86	4.20	1.49	0.99	7827	1
DT3	h6_e2_mdl_bff	69.81	70.54	4.86	1.57	1.09	7724	1
MLP1	h5_e2_mdl_gen	70.91	69.56	4.45	1.57	1.10	7715	52
RF2	h6_e3_kon_gen	71.66	72.33	4.19	1.50	1.00	7816	1
SMO1	h7_e3_kon_gen	71.44	71.98	4.55	1.53	1.04	7773	1

Table 6.5: Summary of the best-performing parameter settings with respect to the *success rate* s(c,I,A) of the different classifiers on the instance set `mixed`.

6.2.7 Analysis of the Best Configuration per Classifier

To conclude these various parameter configurations and used algorithm set, we manually selected for each classifier the setting with the highest *success rate*. Table 6.5 gives an overview on these results considering the different performance measurements.

From these figures, it is apparent that *discretization* is one of the key factors for a successful algorithm selection on the GCP. All results of the Table 6.5 are reached on nominal attributes with only marginal differences between using MDL or KON. As the latter seems to provide slightly better outcome (with respect to the best configurations per classifier), we focus for the remaining comparisons on data prepared with this method.

Furthermore, we decided to fix our set of used heuristics to `h6`. The reason for this is that all classifiers show high performance on that set and neither adding additional algorithms nor removing some lead to significant improvements concerning the *success rate*.

For a more detailed statistical analysis, we executed all classifiers (except the MLP) using their best configurations with a 10-fold cross-validation on the data set `h6_e3_kon_gen`. Then we applied a *corrected resampled T-test* [188] on these results. These experiments, applied with a level of significance of $\alpha = 0.05$, reveal that `BN3`, `IB5` and `RF2` are significant better than `DT3` while all other pairwise comparisons do not show significant differences.

A detailed view on the performance with respect to the different classes (heuristics) can be seen with the help of a *confusion matrix*. Table 6.6 shows the aggregated and normalized values over 20 runs for the BN classifier. The figures reveal that the classifier focus especially on classifying `MMT`, `HEA` and `TABU`, which is reasonable, as these are the most-successful algorithms. Remarkable is also the low true positive (TP) rate on `ILS`, where most cases are classified as `TABU` instead of the correct heuristic.

A slightly different behavior can be seen in Table 6.7 where the *confusion matrix* of `IB5` is given. In addition, this This classifier shows slightly lower success predicting instances for `MMT` compared with BN, but a higher TP rate regarding `ILS` and `TABUB`.

The *confusion matrix* for the setting `RF2` is presented in Table 6.8. The data here show that `RF2` and `IB5` have similar performance concerning prediction of the different classes.

For a better overview on the prediction per class on the different heuristics, we summarize

FPC	HEA	ILS	MAFS	MMT	TABU	← classified as
3.6	0.2	0.0	0.1	1.1	0.7	FPC
0.1	10.1	0.5	0.3	4.2	1.5	HEA
0.1	0.8	2.7	0.0	0.6	3.3	ILS
0.0	0.7	0.2	2.1	1.5	0.3	MAFS
0.5	2.1	0.2	0.2	39.1	0.7	MMT
0.0	1.3	0.7	0.0	6.7	13.8	TABU

Table 6.6: Confusion matrix of the setting `BN3` of 20 runs using the data set `mixed_h6_e3_kon_gen`. Note that the values are normalized by the number of instances.

FPC	HEA	ILS	MAFS	MMT	TABU	← classified as
3.0	0.4	0.1	0.1	1.4	0.6	FPC
0.1	10.4	0.9	0.5	3.3	1.5	HEA
0.2	0.3	4.4	0.1	0.3	2.2	ILS
0.0	0.6	0.4	2.7	0.7	0.4	MAFS
0.5	3.0	0.2	0.7	36.4	2.1	MMT
0.0	1.0	2.2	0.1	4.0	15.2	TABU

Table 6.7: Confusion matrix of the setting `IB5` of 20 runs using the data set `h6_e3_kon_gen`. Note that the values are normalized by the number of instances.

FPC	HEA	ILS	MAFS	MMT	TABU	← classified as
3.7	0.4	0.1	0.2	0.8	0.5	FPC
0.2	10.1	0.5	0.6	3.8	1.5	HEA
0.2	0.4	4.3	0.2	0.5	2.0	ILS
0.1	0.7	0.3	2.5	1.0	0.3	MAFS
0.6	3.0	0.2	0.7	36.2	2.2	MMT
0.2	1.0	1.7	0.3	4.4	14.9	TABU

Table 6.8: Confusion matrix of the setting `RF2` of 20 runs using the data set `h6_e3_kon_gen`. Note that the values are normalized by the number of instances.

the individual results in Table 6.9. This statistic clearly shows that all classifiers offer good results concerning `MMT`, which is the appropriate choice in the majority of instances. In detail, all classifiers show a TP rate of up to 91.35% (average value 84.62%) for this class, which is recommended on 42.84% of the 859 observations. In contrast to this, from the examples of the second biggest group, `TABU`, are on average only 64.06% correct classified. Although this is the desired behavior, it poses some risk because the success of `MMT` is intensified by the inhomogeneous training set. On a different (balanced) training data, the performance of `MMT` might be inferior in relation to its competitors. On such data, the learning algorithms would have to increase their prediction rate on the other classes to achieve similar performance.

Class	No. Best		BN	C4.5	kNN	RF	SVM
	abs.	(%)					
FPC	48	5.59	63.65	65.42	54.06	65.52	68.75
HEA	143	16.65	60.38	56.05	62.48	60.49	63.81
ILS	65	7.57	35.46	47.23	58.62	56.46	57.54
MAFS	41	4.77	43.41	46.95	55.98	51.83	47.32
MMT	368	42.84	91.35	83.68	84.93	84.59	78.55
TABU	194	22.58	61.29	59.72	67.42	66.19	65.70
TOTAL	859		71.34	68.14	72.14	71.66	69.56

Table 6.9: Percentage of correct predictions per class on the data set `mixed_h6_e3_kon_gen`. The second column represents the number of instances where the heuristic achieves the best solution.

6.2.8 Comparison of Algorithms for the GCP using Cross-Validation

So far, we have evaluated different parameter configurations, classifier, discretization methods and used heuristics. In the following paragraph, we compare the best found classifier with the underlying heuristics to show the benefits of algorithm selection. For this purpose, we selected for each classifier the parameter configurations which achieved high success rate and inspect their behavior on the different subsets of instances.

In detail, we tested each classifier 20 times using a 10-fold cross validation on the data set `mixed_h6_e3_kon_gen`. Figure 6.22 shows the average number of correct predictions per classifier separated by the three instance sets. For a good comparison with the tested solvers for the GCP, the graphic contains also for each algorithm the number of instances on which they show the best performance. The diagram clearly illustrate that 5 of 6 tested classifiers achieve nearly similar results. Only the MLP fails completely on this data set. This method requires more than 24 hours for one cross-validation and its results are even below those of the underlying heuristics. Nevertheless, the other approaches show an overwhelming performance by obtaining on up to 625.9 (72.86%) instances the best solution. Compared with the best single heuristic for the GCP, `MMT`, this is an improvement by 259 (30%). Even more, this increase can be observed on all three tested sets of instances, `chi500`, `chi1000` and `dimacs`.

A more detailed comparison is given in Table 6.10 where we evaluate the classifiers using different metrics. Due to the poor performance of MLP on the data discretized with KON, we also add the results of the data set `mixed_h5_e2_mdl_gen` for this classifier. According to this data, we can say that our systems for algorithm selection are clearly better on all evaluated measurements. All tested classifier, except MLP, show a lower *distance* $err(k,i)$ than the best heuristic, `MMT` (up to -1.35%) and are way ahead of the other algorithms (between 8% and 32%). Concerning the ranking, the classifiers reach on average a rank between 1.49 and 1.57, which is much better than the best single heuristic (`MMT` with 2.63). In addition, the algorithm selection shows a lower standard derivation of the ranking (average values around 1.04 versus 1.64) and win also using the formula-one criteria (7827 versus 6348) by an increase of $+23.30\%$.

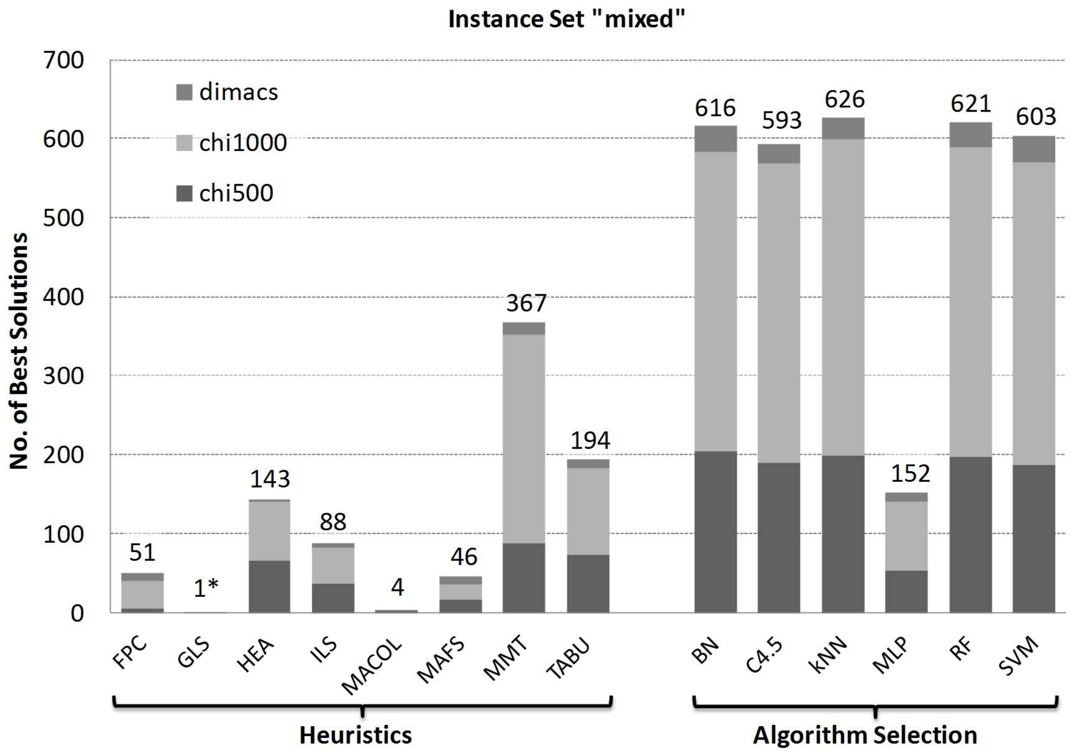

Figure 6.22: Number of predicted best algorithms of the different classifiers on the data set mixed_h6_e3_kon_gen in comparison with the underlying (meta)heuristics. The classifiers are tested 20 times using 10-fold cross-validation. The asterisk denotes that this heuristic is only tested on chi500. Please note that the MLP classifier is only tested once because of time reasons.

Next, we analyze the behavior on the different subsets of the instance sets chi500 and chi1000. For this purpose, we compare the results of the RF classifier using the setting RF2 with the heuristics for the GCP. Note that according to our experiments, the data of the other classifiers are in the majority of cases similar to those presented here, for what reason we omit these additional results.

Figure 6.23 illustrates the results of RF2 on the instances of chi500. It can be seen that the classifier chooses for all subsets appropriate algorithms and is in 6 of the 9 subgroups better than any single heuristic. On some subsets, especially U-0.5 and U-0.9, the proposed approach is not better than MMT which indicate that the used features might be unable to characterize these cases in a perfect way. Another noteworthy observation is that on the subset G-0.5 it seems that the classifier is able to distinguish well between the two dominating algorithms HEA and TABU and achieves relative good success rate.

The results for the instances of the larger set chi1000 are given in Figure 6.24. As on the previous figure, the diagram clearly shows that the classifier is able to identify on all subsets appropriate heuristics and the number of correct classifications is always at least near to those

Solver	$s(c, I, A)$ (%)	$err(k, i)$ (%)	Rank avg	Rank stdev	Rank F1
Algorithm Selection					
BN	71.68	**3.78**	1.53	1.07	7772
C4.5	68.99	4.62	1.58	1.06	7693
kNN	**72.86**	4.20	**1.49**	0.99	**7827**
RF	72.33	4.19	1.50	1.00	7816
MLP	17.69	26.32	4.02	2.05	4665
MLP’	69.56	4.45	1.57	1.10	7715
SVM	70.16	5.02	1.57	1.11	7718
Heuristics					
FPC	5.94	36.57	4.74	1.83	3829
HEA	16.65	12.23	2.72	1.37	6001
ILS	10.24	19.26	3.89	1.88	4720
MACOL	0.47	25.29	5.22	1.32	3294
MAFS	5.36	21.72	4.36	1.68	4159
MMT	42.84	5.41	2.63	1.86	6348
TABU	22.58	16.00	2.92	1.54	5791
Best (heuristic)	42.84	5.41	2.63	1.86	6348
Best (AS)	**72.86**	**3.78**	**1.49**	**0.99**	**7827**

Table 6.10: Comparison of several performance metrics between the an cross validation of the classifiers on the data `mixed_h6_e3_kon_gen` and the underlying heuristics on the instances of `mixed`. Note that `MLP'` denotes a configuration on the set `mixed_h5_e2_mdl_gen`.

of the best algorithm. In detail, on 6 of the subsets is our algorithm selection better than the best underlying method and the gap on the remaining 3 groups is always marginal. The data further indicate that on the instances of `W-0.5`, where multiple heuristics perform best, the classifier is able to discriminate between the different graph types as its performance is better than any single solver for the GCP. On the other side, the prediction rate on `G-0.5` is in relation to `MMT` and `HEA` not very well, which is a strong clue that the classifier is not always able to distinguish when to apply one of these two algorithms. This is interesting, as on the subset `G-0.5` of `chi500`, RF shows good performance. However, on the instances of `chi500`, the most successful solvers are `HEA` and `TABU` while on the instances of `chi1000`, `MMT` achieves the highest number of best solution. This diverse information in the data could explain the suboptimal performance of the classifier.

6.2.9 Comparison with Algorithms for the GCP on the Test Set

In the previous section, we compare the predictions of the different classifiers with the single heuristics for the GCP on the data set *mixed*, which is also used for the learning process itself.

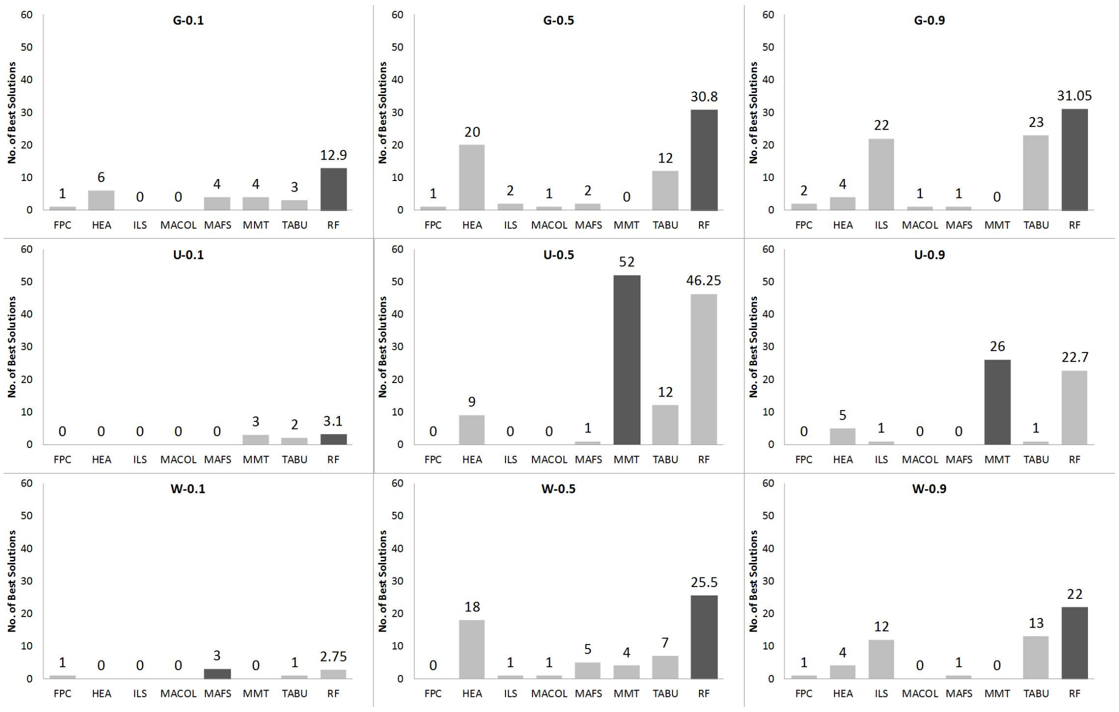

Figure 6.23: Number of predicted best algorithms on the instances of `chi500` using the setting `RF2` in comparison with the underlying (meta)heuristics. The classifier is tested 20 times on the data set `mixed_h6_e3_kon_gen` using 10-fold cross-validation.

However, a more realistic scenario is that an already trained classifier has to choose an algorithm on new, unseen instances. To simulate this application, we trained each considered learning algorithm using the instance set `mixed_h6_e3_kon_gen` and evaluated their performance on the `test set`. This also allows a fair comparison between the underlying algorithms and our approach based on algorithm selection, as the performance of the former on these instances is not visible for the latter and can not affect their predictions.

Concerning the `test set` itself, we considered again only *hard* instances. In detail, from the 180 instances we classified 16 as *trivial* after the feature computation. Based on the data of the heuristic evaluation, we further separated 9 as *trivial2* and 3 as *easy* leading to 152 remaining *hard* instances.

As for the training set, we transformed all numeric attributes into discrete ones using KON. However, this is a supervised method that uses the information about the correct class for finding good decision boundaries and as this information is prohibited for the test set, we discretized based on the nominal values of `mixed_h6_e3_kon_gen`.

Then we trained each classifier with the data of `mixed_h6_e3_kon_gen` as training set and used `test set` as test set. We further used the parameter configuration which showed the best performance during previous experiments (see Table 6.5).

The most interesting evaluation criteria is of course the number of instances on which the

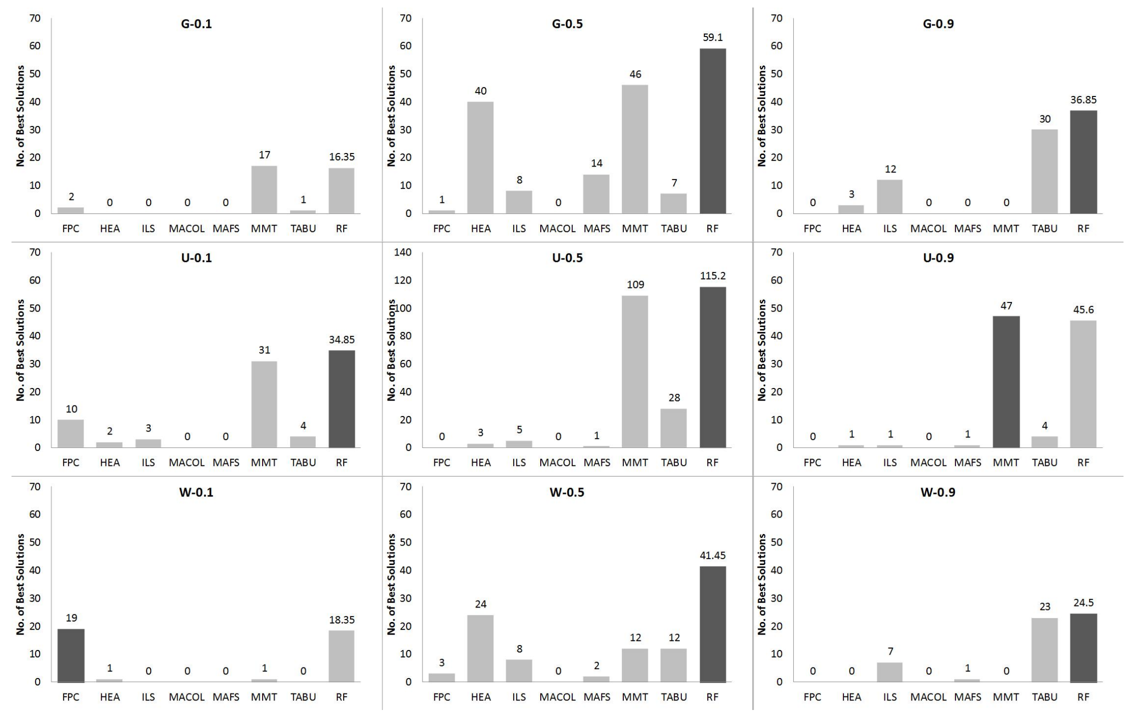

Figure 6.24: Number of predicted best algorithms on the instances of `chi1000` using the setting `RF2` in comparison with the underlying (meta)heuristics. The classifier is tested 20 times on the data set `mixed_h6_e3_kon_gen` using 10-fold cross-validation.

solvers show the best performance. Figure 6.25 gives on overview on this measurement. From this diagram, it is easy to see that all learning strategies except MLP accomplish a higher number of first ranks than any single solver for the GCP. The most successful classifiers are RF, BN and kNN which forecast on up to 68.59% of the 156 graphs the most appropriate algorithm.

A more detailed view on the results using different metrics is given in Table 6.11. The figures point out that again `MMT` is the best single heuristic with respect to the number of first places in our ranking. Moreover, it accomplishes the lowest average distance $err(k, i)$ with a larger gap to the other approaches. Surprisingly, when we look at the average rank, `MMT` is not ranked first because `TABU` and `HEA` show both a lower value. However, concerning the formula-one score, `MMT` lies again in front of `HEA`, but is still behind `TABU`.

Compared with our solvers using all algorithms and an automatic algorithm selection mechanism, it is easy to see that for all considered metrics except $err(k, i)$ at least one system shows a stronger performance than the best single heuristic. The best selection mechanism provides clearly RF, which is on all criteria except the $err(k, i)$ better than the other classifier. In detail, this system achieves a *success rate* of 70.39% ($+33.55\%$ compared with `MMT`) and an average rank of 1.51 (-1.07 compared with `TABU`). Using the formula-one ranking, it reaches 1386 points ($+292$ or $+26.7\%$ to the results of `TABU`). Only on the metric $err(k, i)$ `MMT` shows with 4.63% a lower value than RF, which predictions have an average distance of 6.44%. Surpris-

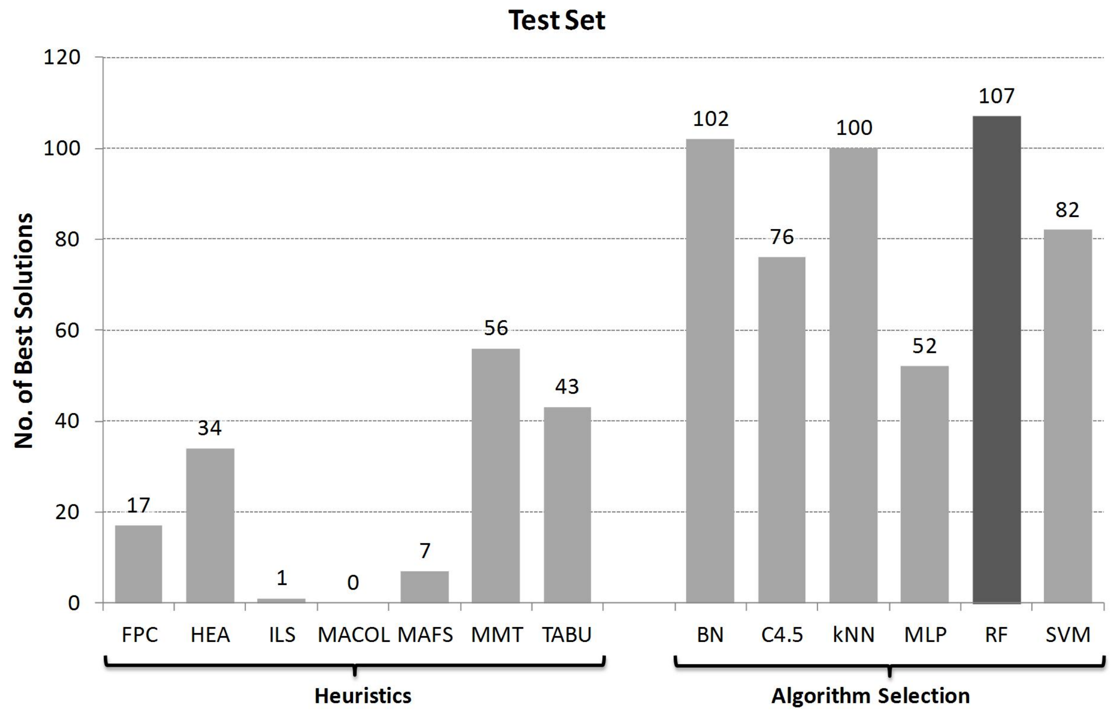

Figure 6.25: Number of instances from the `test set` on which a solver show best performance.

ingly, the approach based on a DT, which performs suboptimal concerning s(c,I,A) and the ranking criteria, has with 4.90% one of the lowest value of $err(k, i)$ from all solvers based on algorithm selection. Only kNN achieves with 4.88 a slightly lower value. The worst performance of the classifiers shows clearly MLP, which results concerning the number of instances where it performs best are even below those of `MMT`. In combination with the long runtime, these data confirm that this machine learning technique is in combination with KON not suited for the GCP.

It is now clear that it is possible to implement algorithm selection for the GCP using machine learning techniques. Let us now analyze the behavior of the most successful classifier, RF, on the different types of graphs. For this purpose, we separated the results according to the graph class and density and evaluate the performance of the heuristics and the RF with respect to that subsets.

Figure 6.26 shows the amount of graphs on which the different methods show the best performance. This diagram reveals that our solver based on algorithm selection is on 5 of the 9 subsets better or equal the best heuristic. Unfortunately, on the groups `G-0.5` and `W-0.5` our approach is not able achieve better or equal performance than any single solver. This is surprising as the best heuristic on that instances is `HEA`, which showed also on the corresponding training data good results. Consequently, it seems that the classifier was not able to learn this pattern correctly. On the groups `U-0.9` and `W-0.1` the algorithm selection fails by predicting

Solver	No. Best Solution	$s(c, I, A)$ (%)	$err(k, i)$ (%)	Rank avg	Rank stdev	Rank F1
Heuristics						
FPC	17	11.18	25.43	3.39	1.53	919
HEA	34	22.37	15.25	2.74	1.43	1065
ILS	1	0.66	21.97	3.99	1.56	784
MACOL	0	0.00	28.13	5.17	1.23	588
MAFS	7	4.61	31.71	5.34	1.94	585
MMT	56	36.84	**4.63**	2.88	1.99	1077
TABU	43	28.29	19.47	2.57	1.25	1094
Algorithm Selection						
BN	102	67.11	5.85	1.58	1.02	1360
C4.5	76	50.00	4.90	2.26	1.62	1204
IBK	100	65.79	4.88	1.61	1.17	1357
MLP	52	34.21	22.92	2.64	1.54	1091
RF	**107**	**70.39**	6.44	**1.50**	1.07	**1386**
SVM	82	53.95	9.37	2.10	1.58	1240
Best (heuristic)	56	36.84	**4.63**	2.57	1.25	1094
Best (AS)	**107**	**70.39**	4.88	**1.50**	1.07	**1386**

Table 6.11: Performance metrics of the algorithm selection and the underlying heuristics on the *test set*.

on only 3 of 10 and 6 of 20 graphs the correct algorithm. The reason for this bad results on the former subset might be in the performance of algorithms: In contrast to the training data, where `MMT` is the dominant method, on the test data also `MAFS` obtains in 4 cases the best solution. Thus, the trained patterns might not fit and this leads the classifier to so many mispredictions. We could not clearly identify why the results of the heuristics differ compared to the training set. One possible explanation for the increased success of `MAFS` is that in contrast to the training set, the new created instances do not hide any fixed coloring.

However, the suboptimal prediction rate on the latter subset can not be explained, as `FPC` is also in related subset `W-0.1` of `chi1000` the best algorithm. Thus, is seems that the classifier is just not able to learn this pattern correctly.

Concerning the distance measurement $err(k, i)$, Figure 6.27 shows a box-whisker diagram with the results on the different subsets. The figures clearly show that algorithm selection achieves in almost all subsets a very low distance which often correlates with the best heuristic. Only on the subsets `U-0.9` there is a larger gap between the RF-based approach and the values of `MMT`. Also noteworthy is that the distance of the heuristics on the instances of `U-0.5` fluctuates and varies much more compared with the other subsets and that `MMT` achieves by far the best solutions. A similar behavior can also be discovered at the training data. However, on the related instances of `U-0.9` this does not hold for all techniques: only `MMT` and `MAFS` achieve on

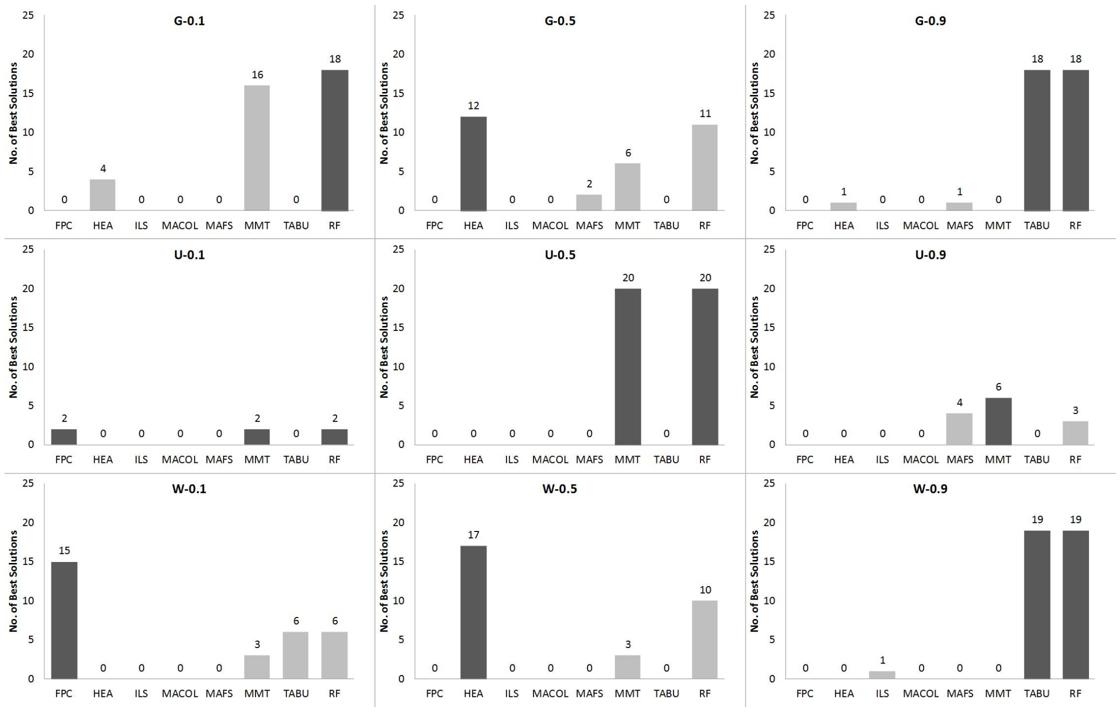

Figure 6.26: Number of graphs where a method shows the best performance on the instances of the `test set` grouped by the graph type and the density.

average improvements over the *greedy* algorithms while the other heuristics are often not able to find a better coloring. Regarding our classification approach, it seems that the RF is able to learn the pattern for the class `U-0.5` such that it predicts always the correct algorithm (`MMT`). On the instances of `U-0.9`, its performance is worse than `MMT`. However, it is still above those of the other heuristics.

A more detailed view on the ranking of the different approaches is given in Figure 6.28. As in the training data, the diagram reveals that no heuristic performs best on all types of graphs. Moreover, the ranks of the algorithms between related subsets may vary strong. A good example for this are the sets `G-0.5` and `G-0.9` where for each algorithm, its median rank changes between the former and the latter sets. Also worth mentioning are the results on `U-0.9` where most algorithms achieve a median rank of 2.5 except `MMT` and `MAFS`. This also correlates to the solution distance (see Figure 6.27) and is mainly caused by the fact that in many cases, only `MAFS` and `MMT` are able to improve the initial number of colors. As a result, these two algorithms achieve better ranks and are therefore to prefer.

Concerning the algorithm selection approach, the figure show that the proposed method is on every subgroup successful. In detail, on 6 subsets (`G-*`, `U-0.1`, `U-0.5` and `W-0.9`) the median rank of our method is 1 and also on the remaining 3 types of graphs, this value is never higher than 2. It is clear to see that in most of the cases, the box and whisker correspond with the dominating heuristic. Only on the subsets `U-0.9`, `W-0.1` and `W-0.5`, our method based on a

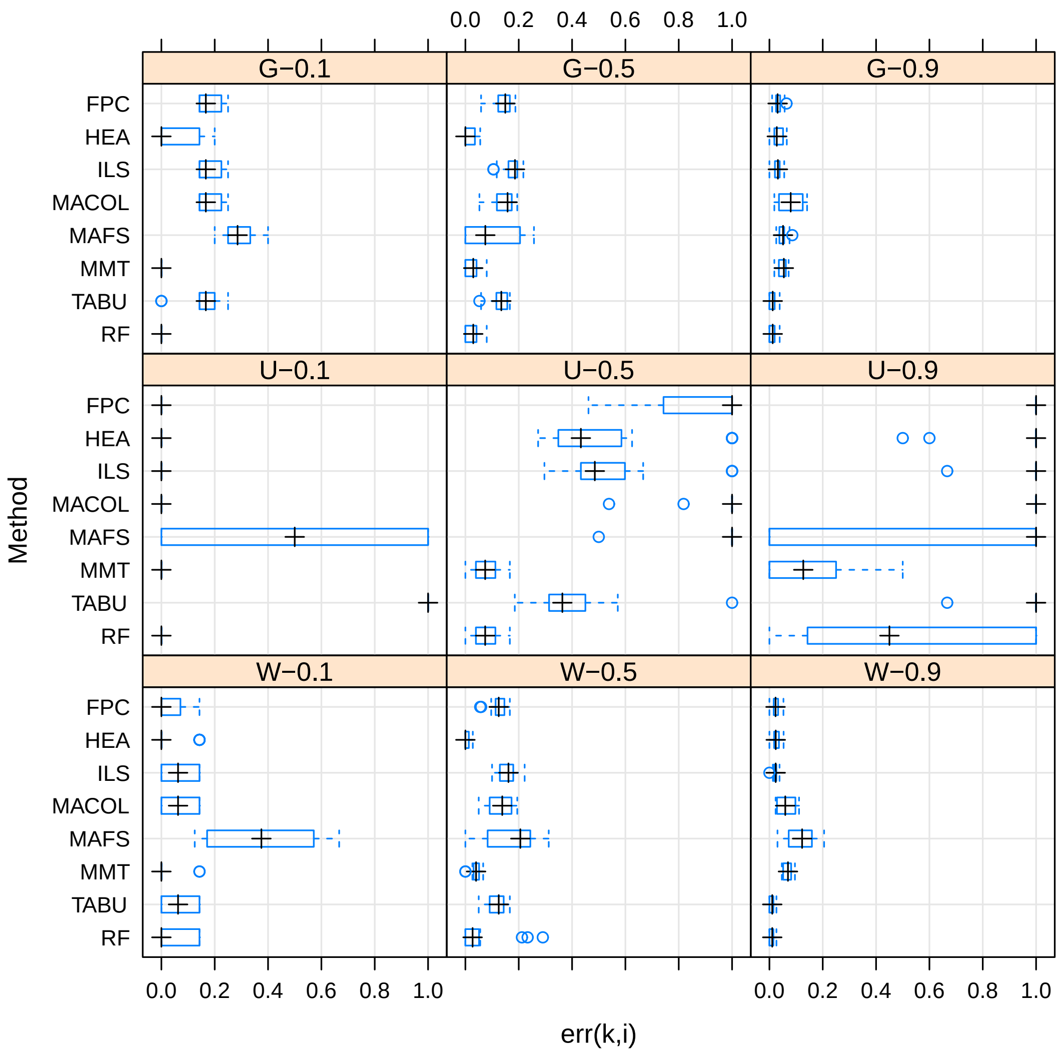

Figure 6.27: Distance metric $err(k, i)$ of the heuristics and the algorithm selection on the hard instances of the `test set` grouped by the graph class and the density.

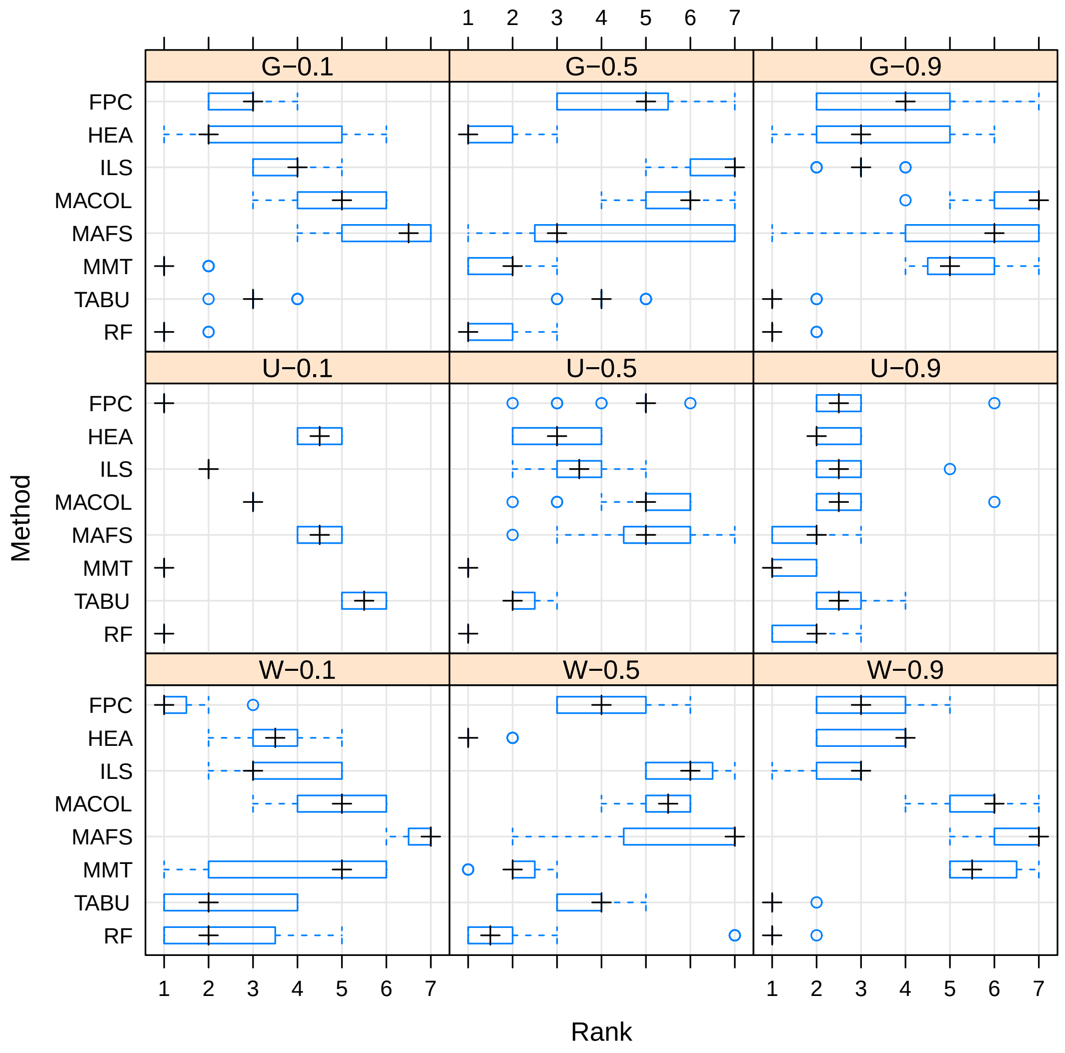

Figure 6.28: Ranking results of the heuristics and the algorithm selection on the hard instances of the `test set` grouped by the graph class and the density.

RF differ to the single algorithms for the GCP. However, compared with any single heuristic, the classifier-based approach achieves a much lower average rank (1.50 versus 2.57) because it exploits the strengths of the heuristics on the different subsets.

CHAPTER 7

Conclusion and Future Work

In this thesis, we have presented our approach to algorithm selection for the graph coloring problem (GCP) based on machine learning. For this purpose, we identified 78 characteristic attributes of a graph. We further gathered empirical data from experimental analysis of 7 heuristics (`HEA`, `ILS`, `MACOL`, `MAFS`, `MMT`, `TABUCOL`, and `Foo-PartialCol`) on 1265 instances of 3 different, public available sets of instances.

These information was used to train 6 classifiers (Bayesian networks, C4.5 decision trees, k-nearest neighbor, multilayer perceptrons, random forests, and support vector machines). We further experimented with different parameter settings for the classifiers and tested the effect of two discretization techniques. In addition, we investigated how a reduced algorithm portfolio effects the accuracy and the overall quality of the prediction.

The main findings of this thesis are:

- In our experiments, no heuristic was on all instances better than the rest.
- Some algorithm tend to be more successful on instances with certain attributes.
- Algorithm selection using classifiers outperforms the underlying heuristics.
- Supervised discretization techniques increase the performance of almost all classifiers.
- Removing worse algorithms from the portfolio may increase the overall performance of the algorithm selection.

In addition, our experiments showed that Bayesian networks, k-nearest neighbor and random forests are the most successful machine learning strategies to solve the algorithm selection problem for the GCP.

Finally, we evaluated the performance of solvers that include all heuristics and an algorithm selection mechanism and compared it with the heuristics for the GCP on a set of new generated instances. The experiments on these graphs, which have not been used for training the classifiers, clearly showed that our proposed approach is able to achieve on more instances the best performance than any heuristic alone.

For future work, we will consider a concrete implementation of the presented approach, which is so far only an experimental patchwork consisting of multiple programs and scripts. In addition, including further algorithms as well as experiments on other types of graphs would be interesting. Such an extension could also include *exact* solvers and smaller instances. The algorithm selection's duty would be to decide whether an instance can be colored exactly or not (like in [109]).

Another issue to investigate is a *regression*-based approach using runtime and solution quality predictions. This technique, which is successfully used for other systems, is an alternative to our classification-based approach and in this context, a comparison on the suitability for the GCP is definitely interesting.

Finally, it is also worth considering a *portfolio* system which execute more than one heuristics. Especially *dynamic* approaches which can combine different heuristics offer the chance to achieve a higher performance than any single algorithm on alone.

APPENDIX A

Appendix

A.1 Weka Commands

The following tables contain the command line parameters for our experiments using the *Weka* framework. Please note that <*dataset.arff*> denotes the used data set, i.e. the training observations consisting on the instance features and the best suited algorithm. For our evaluation, we tested all classifiers 20 times using a random seed $S = \{1, ..., 20\}$. All following commands are for performing a 10-fold cross-validation. For using a separate test set, see the *Weka* manual.

Name	Command Line
DT1	`java weka.classifiers.trees.J48 -C 0.25 -M 2 -t <dataset.arff>`
DT2	`java weka.classifiers.trees.J48 -C 0.125 -M 2 -t <dataset.arff>`
DT3	`java weka.classifiers.trees.J48 -C 0.25 -M 3 -t <dataset.arff>`
DT4	`java weka.classifiers.trees.J48 -C 0.125 -M 4 -t <dataset.arff>`

Table A.1: Command line calls for the C4.5 decision trees (DT) classifier.

Name	Command Line
RF1	`java weka.classifiers.trees.RandomForest -I 10 -K 0 -S 1 -t <dataset.arff>`
RF2	`java weka.classifiers.trees.RandomForest -I 15 -K 0 -S 1 -t <dataset.arff>`

Table A.2: Command line calls for the random forests (RF) classifier.

Name	Command Line
MLP1	`java weka.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a -t` *<dataset.arff>*
MLP2	`java weka.classifiers.functions.MultilayerPerceptron -L 0.4 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a -t` *<dataset.arff>*

Table A.3: Command line calls for the multilayer perceptrons (MLP) classifier.

Name	Command Line
BN1	`java weka.classifiers.bayes.BayesNet -D -Q weka.classifiers.bayes.net.search.local.K2 - -P 1 -S BAYES -E weka.classifiers.bayes.net.estimate.SimpleEstimator - -A 0.5 -t` *<dataset.arff>*
BN2	`java weka.classifiers.bayes.BayesNet -D -Q weka.classifiers.bayes.net.search.local.K2 - -P 2 -S BAYES -E weka.classifiers.bayes.net.estimate.SimpleEstimator - -A 0.5 -t` *<dataset.arff>*
BN3	`java weka.classifiers.bayes.BayesNet -D -Q weka.classifiers.bayes.net.search.local.K2 - -P 3 -S BAYES -E weka.classifiers.bayes.net.estimate.SimpleEstimator - -A 0.5 -t` *<dataset.arff>*
BN4	`java weka.classifiers.bayes.BayesNet -D -Q weka.classifiers.bayes.net.search.local.K2 - -P 4 -S BAYES -E weka.classifiers.bayes.net.estimate.SimpleEstimator - -A 0.5 -t` *<dataset.arff>*
BN5	`java weka.classifiers.bayes.BayesNet -D -Q weka.classifiers.bayes.net.search.local.K2 - -P 5 -S BAYES -E weka.classifiers.bayes.net.estimate.SimpleEstimator - -A 0.5 -t` *<dataset.arff>*

Table A.4: Command line calls for the Bayesian networks (BN) classifier.

Name	Command Line

IB1	`java weka.classifiers.lazy.IBk -K 1 -W 0 -A`
	`"weka.core.neighboursearch.LinearNNSearch -A`
	`ẅeka.core.EuclideanDistance -R first-last" -t` *<dataset.arff>*

IB2	`java weka.classifiers.lazy.IBk -K 3 -W 0 -A`
	`"weka.core.neighboursearch.LinearNNSearch -A`
	`ẅeka.core.EuclideanDistance -R first-last" -t` *<dataset.arff>*

IB3	`java weka.classifiers.lazy.IBk -K 5 -W 0 -A`
	`"weka.core.neighboursearch.LinearNNSearch -A`
	`ẅeka.core.EuclideanDistance -R first-last" -t` *<dataset.arff>*

IB4	`java weka.classifiers.lazy.IBk -K 7 -W 0 -A`
	`"weka.core.neighboursearch.LinearNNSearch -A`
	`ẅeka.core.EuclideanDistance -R first-last" -t` *<dataset.arff>*

IB5	`java weka.classifiers.lazy.IBk -K 9 -W 0 -A`
	`"weka.core.neighboursearch.LinearNNSearch -A`
	`ẅeka.core.EuclideanDistance -R first-last" -t` *<dataset.arff>*

Table A.5: Command line calls for the k-nearest neighbor (kNN) classifier.

Name	Command Line
SMO1	java weka.classifiers.functions.SMO -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W *SEED* -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0" -t *<dataset.arff>*
SMO2	java weka.classifiers.functions.SMO -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W *SEED* -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.2" -t *<dataset.arff>*
SMO3	java weka.classifiers.functions.SMO -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W *SEED* -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.4" -t *<dataset.arff>*
SMO4	java wweka.classifiers.functions.SMO -C 1.5 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W *SEED* -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0" -t *<dataset.arff>*
SMO5	java weka.classifiers.functions.SMO -C 1.5 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W *SEED* -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.4" -t *<dataset.arff>*
SMO6	java weka.classifiers.functions.SMO -C 2.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W *SEED* -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 2.0" -t *<dataset.arff>*
SMO7	java weka.classifiers.functions.SMO -C 3.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W *SEED* -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 2.0" -t *<dataset.arff>*
SMO8	java -W weka.classifiers.functions.SMO -C 2.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W *SEED* -K "weka.classifiers.functions.supportVector.Puk -C 250007 -O 1.0 -S 1.0" -t *<dataset.arff>*

Table A.6: Command line calls for the sequential minimal optimization (SMO) classifier.

A.2 Detailed Results

The following section contains detailed results using discretized (denoted according to the used method mdl or kon and *non-discretized* (marked with none) data. For each of the three classes, we present the best parameter configurations for the 6 tested classifiers and the results on the data set mixed.

The set of instances mixed is the combination of the hard instances of chi500, chi1000 and dimacs and consists of 859 instances which we classified as *hard* according to our criteria.

In the following tables, we show the *accuracy* of the classification algorithm, the s(c,I,A), the $err(k,i)$ and two rank-based metric. In addition, the column *BKS* denotes the number of instances on that our solvers based on algorithm selection finds a coloring requiring as less colors as the best known solution (BKS).

Setting	Data Set	Accuracy (%)	$s(c,I,A)$ (%)	$err(k,i)$ (%)	Rank avg	Rank stdev	Rank F1	BKS
BN3	h7_e1_none_bff	66.98	67.79	4.69	1.61	1.13	7657	771
IB9	h6_b_none_gen	63.38	63.95	4.89	1.68	1.15	7537	754
DT4	h5_b_none_gen	65.47	64.51	5.27	1.69	1.18	7542	745
MLP1	h7_b_none_bff	63.81	64.87	4.86	1.68	1.18	7557	761
RF2	h6_b_none	67.04	67.96	4.86	1.56	1.00	7708	765
SMO8	h6_b_none	66.84	67.46	4.16	1.60	1.10	7666	775

Table A.7: Summary of the best-performing settings with respect to the *success rate* for the different classifiers on the instance set mixed using non-discretized features.

Setting	Data Set	Accuracy (%)	$s(c,I,A)$ (%)	$err(k,i)$ (%)	Rank avg	Rank stdev	Rank F1	BKS
BN4	h7_e2_mdl_gen_50	70.36	71.23	4.38	1.55	1.11	7754	781
IB3	h6_e2_mdl_gen_50	70.28	70.94	4.63	1.57	1.12	7722	754
DT3	h6_e2_mdl_bff	69.81	70.54	4.86	1.57	1.09	7724	755
MLP1	h5_e2_mdl_gen	70.91	69.56	4.45	1.57	1.10	7715	763
RF2	h5_e2_mdl_gen_50	72.36	70.97	4.23	1.53	1.02	7770	769
SMO1	h5_e2_mdl_bff	72.20	70.64	4.64	1.59	1.16	7706	761

Table A.8: Summary of the best-performing settings with respect to the *success rate* for the different classifiers on the instance set mixed using the MLD criteria.

Setting	Data Set	Accuracy (%)	$s(c, I, A)$ (%)	$err(k, i)$ (%)	Rank avg	Rank stdev	F1	BKS
BN3	`h7_e3_kon_bff`	71.89	72.49	4.06	1.50	1.01	7810	783
IB5	`h6_e3_kon_gen`	72.14	72.86	4.20	1.49	0.99	7827	778
DT1	`h7_e3_kon_bff`	68.78	69.72	4.74	1.59	1.11	7689	756
MLP2	`h7_b_kon_bff`	67.80	68.46	5.05	1.60	1.11	7666	752
RF2	`h6_e3_kon_gen`	71.66	72.33	4.19	1.50	1.00	7816	778
SMO1	`h7_e3_kon_gen`	71.44	71.98	4.55	1.53	1.04	7773	764

Table A.9: Summary of the best-performing settings with respect to the *success rate* for the different classifiers on the instance set `mixed` using Kononenko's MDL criteria.

A.3 Feature Subsets

The following section contains selected feature subsets that were obtained by the feature selection on the data of the instances of `mixed`.

Setname	Features
h5_b_none_gen	$B^{k_\mathrm{UB}}_\mathrm{dist}$, CC_g, CC_max, $\mathrm{CC}_\mathrm{mean}$, CC_vc, CS_max, CS_m, CS_mean, CS_min, CS_q25, $\mathrm{GC}_\mathrm{D/R}$, $\mathrm{GC}_\mathrm{R/D}$, ID_e, IR_min, LS_ce, LS_nlo, D_min, D_vc, S_e, S_en, S_ne, WCC_e, $\mathrm{WCC}_\mathrm{mean}$, $\mathrm{WCC}_\mathrm{min}$, WCC_vc
h6_b_none	$B^{k_\mathrm{LB}}_\mathrm{dist}$, $B^{k_\mathrm{UB}}_\mathrm{dist}$, B_lu, B_ul, CC_e, CC_g, CC_max, $\mathrm{CC}_\mathrm{mean}$, CC_med, CC_min, CC_q25, CC_q75, $\mathrm{CC}_\mathrm{time}$, CC_vc, CS_e, CS_max, CS_m, CS_mean, CS_min, CS_q25, CS_q75, CS_time, CS_vc, $\mathrm{GC}_\mathrm{best}$, $\mathrm{GC}_\mathrm{DSAT}$, $\mathrm{GC}_{T-\mathrm{DSAT}}$, GC_RLF, $\mathrm{GC}_{T-\mathrm{RLF}}$, $\mathrm{GC}_\mathrm{D/R}$, $\mathrm{GC}_\mathrm{R/D}$, ID_e, ID_max, $\mathrm{ID}_\mathrm{mean}$, ID_med, ID_min, ID_q25, ID_q75, ID_vc, IR_e, IR_max, $\mathrm{IR}_\mathrm{mean}$, IR_med, IR_min, IR_q25, IR_q75, IR_vc, LS_i, LS_ii, LS_ce, LS_cee, LS_cn, LS_cne, LS_nlo, LS_nto, LS_t, D_e, D_max, D_mean, D_med, D_min, D_q25, D_q75, D_vc, S_d, S_e, S_en, S_n, S_ne, TD_width, TD_time, WCC_e, $\mathrm{WCC}_\mathrm{max}$, $\mathrm{WCC}_\mathrm{mean}$, $\mathrm{WCC}_\mathrm{med}$, $\mathrm{WCC}_\mathrm{min}$, $\mathrm{WCC}_\mathrm{q25}$, $\mathrm{WCC}_\mathrm{q75}$, WCC_vc
h6_b_none_gen	CC_g, CC_max, $\mathrm{CC}_\mathrm{mean}$, CC_vc, CS_max, CS_mean, CS_min, CS_q25, $\mathrm{GC}_\mathrm{D/R}$, $\mathrm{GC}_\mathrm{R/D}$, $\mathrm{ID}_\mathrm{mean}$, IR_max, $\mathrm{IR}_\mathrm{mean}$, IR_min, LS_i, LS_nlo, D_min, D_vc, S_e, S_en, S_ne, WCC_e, $\mathrm{WCC}_\mathrm{mean}$, $\mathrm{WCC}_\mathrm{min}$, WCC_vc
h7_b_none_bff	B_lu, CC_g, CC_max, $\mathrm{CC}_\mathrm{mean}$, CC_vc, CS_max, CS_mean, CS_min, CS_q25, $\mathrm{GC}_\mathrm{D/R}$, ID_e, $\mathrm{ID}_\mathrm{mean}$, IR_max, IR_min, LS_cne, D_e, D_min, D_vc, S_en, S_ne, WCC_e, $\mathrm{WCC}_\mathrm{mean}$, $\mathrm{WCC}_\mathrm{min}$, WCC_vc
h7_e1_none_bff	$B^{k_\mathrm{LB}}_\mathrm{dist}/\mathrm{CC}_\mathrm{g}$, $B^{k_\mathrm{LB}}_\mathrm{dist}/\mathrm{CC}_\mathrm{mean}$, $B^{k_\mathrm{LB}}_\mathrm{dist}\cdot\mathrm{CC}_\mathrm{g}$, $B^{k_\mathrm{LB}}_\mathrm{dist}\cdot\mathrm{CC}_\mathrm{mean}$, $B^{k_\mathrm{LB}}_\mathrm{dist}\cdot CS_\mathrm{max}$, $B^{k_\mathrm{LB}}_\mathrm{dist}\cdot CS_\mathrm{mean}$, $B^{k_\mathrm{LB}}_\mathrm{dist}\cdot CS_\mathrm{min}$, $B^{k_\mathrm{LB}}_\mathrm{dist}\cdot CS_\mathrm{q25}$, $B^{k_\mathrm{LB}}_\mathrm{dist}\cdot D_\mathrm{min}$, $B^{k_\mathrm{LB}}_\mathrm{dist}\cdot S_\mathrm{en}$, $B^{k_\mathrm{LB}}_\mathrm{dist}/\mathrm{ID}_\mathrm{mean}$, $B^{k_\mathrm{LB}}_\mathrm{dist}/S_\mathrm{ne}$, $B^{k_\mathrm{UB}}_\mathrm{dist}/\mathrm{CC}_\mathrm{max}$, $B^{k_\mathrm{UB}}_\mathrm{dist}/\mathrm{CC}_\mathrm{mean}$, $B_\mathrm{lu}\cdot\mathrm{CC}_\mathrm{max}$, $B_\mathrm{lu}\cdot\mathrm{CC}_\mathrm{vc}$, $B_\mathrm{lu}/\mathrm{IR}_\mathrm{max}$, $B_\mathrm{lu}/D_\mathrm{vc}$, $B_\mathrm{lu}/\mathrm{WCC}_\mathrm{min}$, CC_g, $\mathrm{CC}_\mathrm{g}/B^{k_\mathrm{LB}}_\mathrm{dist}$, $\mathrm{CC}_\mathrm{g}\cdot\mathrm{CC}_\mathrm{mean}$, $\mathrm{CC}_\mathrm{g}\cdot\mathrm{IR}_\mathrm{max}$, $\mathrm{CC}_\mathrm{g}\cdot D_\mathrm{min}$, $\mathrm{CC}_\mathrm{g}\cdot S_\mathrm{en}$, $\mathrm{CC}_\mathrm{g}/S_\mathrm{ne}$, $\mathrm{CC}_\mathrm{g}/\mathrm{WCC}_\mathrm{vc}$, $\mathrm{CC}_\mathrm{max}/B^{k_\mathrm{UB}}_\mathrm{dist}$, $\mathrm{CC}_\mathrm{max}\cdot CS_\mathrm{max}$, $\mathrm{CC}_\mathrm{max}\cdot CS_\mathrm{mean}$, $\mathrm{CC}_\mathrm{max}\cdot S_\mathrm{en}$, $\mathrm{CC}_\mathrm{max}/D_\mathrm{vc}$, $\mathrm{CC}_\mathrm{max}/S_\mathrm{ne}$, $\mathrm{CC}_\mathrm{mean}$, $\mathrm{CC}_\mathrm{mean}\cdot\mathrm{IR}_\mathrm{max}$, $\mathrm{CC}_\mathrm{mean}\cdot\mathrm{IR}_\mathrm{min}$, $\mathrm{CC}_\mathrm{mean}\cdot\mathrm{WCC}_\mathrm{min}$, $\mathrm{CC}_\mathrm{mean}/CS_\mathrm{max}$, $\mathrm{CC}_\mathrm{mean}/\mathrm{IR}_\mathrm{min}$, $\mathrm{CC}_\mathrm{mean}/\mathrm{WCC}_\mathrm{vc}$, $\mathrm{CC}_\mathrm{vc}/\mathrm{WCC}_\mathrm{min}$, $CS_\mathrm{max}/\mathrm{CC}_\mathrm{mean}$, $CS_\mathrm{max}\cdot D_\mathrm{e}$, $CS_\mathrm{max}/\mathrm{GC}_\mathrm{D/R}$, $CS_\mathrm{mean}\cdot D_\mathrm{e}$, $CS_\mathrm{mean}/CS_\mathrm{q25}$, $CS_\mathrm{min}\cdot D_\mathrm{e}$, $\mathrm{GC}_\mathrm{DSAT}/D_\mathrm{min}$, $\mathrm{GC}_\mathrm{D/R}/\mathrm{WCC}_\mathrm{min}$, $\mathrm{GC}_\mathrm{R/D}/S_\mathrm{en}$, $\mathrm{ID}_\mathrm{e}\cdot\mathrm{IR}_\mathrm{min}$, $\mathrm{ID}_\mathrm{e}/\mathrm{IR}_\mathrm{max}$, $\mathrm{ID}_\mathrm{mean}\cdot\mathrm{GC}_\mathrm{DSAT}$, $\mathrm{ID}_\mathrm{mean}/D_\mathrm{e}$, $\mathrm{ID}_\mathrm{mean}/\mathrm{WCC}_\mathrm{e}$, $\mathrm{IR}_\mathrm{max}\cdot S_\mathrm{en}$, $\mathrm{IR}_\mathrm{max}/\mathrm{IR}_\mathrm{min}$, $\mathrm{IR}_\mathrm{max}/D_\mathrm{vc}$, $\mathrm{IR}_\mathrm{max}/S_\mathrm{ne}$, $\mathrm{IR}_\mathrm{max}/\mathrm{WCC}_\mathrm{vc}$, $\mathrm{LS}_\mathrm{cne}\cdot\mathrm{IR}_\mathrm{min}$, $\mathrm{LS}_\mathrm{cne}/D_\mathrm{e}$, $\mathrm{LS}_\mathrm{cne}/\mathrm{WCC}_\mathrm{e}$, $D_\mathrm{e}\cdot\mathrm{IR}_\mathrm{min}$, $D_\mathrm{e}/\mathrm{ID}_\mathrm{mean}$, $D_\mathrm{e}/\mathrm{LS}_\mathrm{cne}$, $D_\mathrm{min}/\mathrm{CC}_\mathrm{vc}$, $D_\mathrm{min}\cdot\mathrm{ID}_\mathrm{mean}$, $D_\mathrm{min}\cdot\mathrm{IR}_\mathrm{max}$, $D_\mathrm{min}\cdot\mathrm{IR}_\mathrm{min}$, $D_\mathrm{min}\cdot S_\mathrm{ne}$, $D_\mathrm{min}/\mathrm{GC}_\mathrm{DSAT}$, $D_\mathrm{min}/\mathrm{GC}_\mathrm{D/R}$, $D_\mathrm{min}/S_\mathrm{en}$, $D_\mathrm{vc}/\mathrm{CC}_\mathrm{max}$, $D_\mathrm{vc}/\mathrm{IR}_\mathrm{max}$, $D_\mathrm{vc}/\mathrm{WCC}_\mathrm{mean}$, S_en, $S_\mathrm{en}\cdot S_\mathrm{ne}$, $S_\mathrm{en}/D_\mathrm{min}$, $S_\mathrm{ne}/B^{k_\mathrm{LB}}_\mathrm{dist}$, $S_\mathrm{ne}/\mathrm{WCC}_\mathrm{min}$, $\mathrm{WCC}_\mathrm{e}\cdot CS_\mathrm{max}$, $\mathrm{WCC}_\mathrm{e}\cdot\mathrm{IR}_\mathrm{min}$, $\mathrm{WCC}_\mathrm{e}\cdot D_\mathrm{e}$, $\mathrm{WCC}_\mathrm{e}\cdot D_\mathrm{vc}$, $\mathrm{WCC}_\mathrm{e}/\mathrm{IR}_\mathrm{min}$, $\mathrm{WCC}_\mathrm{e}/\mathrm{LS}_\mathrm{cne}$, $\mathrm{WCC}_\mathrm{mean}\cdot\mathrm{IR}_\mathrm{max}$, $\mathrm{WCC}_\mathrm{mean}\cdot\mathrm{LS}_\mathrm{cne}$, $\mathrm{WCC}_\mathrm{mean}/\mathrm{GC}_\mathrm{R/D}$, $\mathrm{WCC}_\mathrm{mean}/\mathrm{IR}_\mathrm{min}$, $\mathrm{WCC}_\mathrm{mean}/D_\mathrm{vc}$, $\mathrm{WCC}_\mathrm{min}/\mathrm{CC}_\mathrm{vc}$, $\mathrm{WCC}_\mathrm{min}\cdot\mathrm{IR}_\mathrm{max}$, $\mathrm{WCC}_\mathrm{min}\cdot\mathrm{LS}_\mathrm{cne}$, $\mathrm{WCC}_\mathrm{min}/\mathrm{GC}_\mathrm{D/R}$, $\mathrm{WCC}_\mathrm{min}/S_\mathrm{ne}$, $\mathrm{WCC}_\mathrm{vc}/\mathrm{CC}_\mathrm{g}$, $\mathrm{WCC}_\mathrm{vc}/\mathrm{CC}_\mathrm{mean}$, $\mathrm{WCC}_\mathrm{vc}/\mathrm{IR}_\mathrm{max}$, $\mathrm{WCC}_\mathrm{vc}/\mathrm{WCC}_\mathrm{mean}$

Table A.10: Selected subsets of non-discretized features based on the feature selection on the data set `mixed`.

Setname	Features
h5_e2_mdl_bff	$\mathrm{CC_g}$, $\mathrm{CC_g}/\mathrm{CC_{max}}$, $\mathrm{CC_g}\cdot\mathrm{CC_{mean}}$, $\mathrm{CC_g}/CS_{\mathrm{max}}$, $\mathrm{CC_g}/D_{\mathrm{vc}}$, $\mathrm{CC_{max}}$, $\mathrm{CC_{max}}\cdot CS_{\mathrm{max}}$, $\mathrm{CC_{max}}\cdot D_{\mathrm{min}}$, $\mathrm{CC_{max}}\cdot S_e$, $\mathrm{CC_{max}}/\mathrm{GC_{D/R}}$, $\mathrm{CC_{max}}/\mathrm{WCC_{mean}}$, $\mathrm{CC_{mean}}$, $\mathrm{CC_{mean}}\cdot D_{\mathrm{min}}$, $\mathrm{CC_{mean}}/\mathrm{ID_e}$, $CS_{\mathrm{max}}\cdot CS_{\mathrm{q25}}$, $CS_{\mathrm{max}}/D_{\mathrm{min}}$, $CS_{\mathrm{mean}}\cdot D_{\mathrm{vc}}$, $CS_{\mathrm{mean}}/CS_{\mathrm{q25}}$, $CS_{\mathrm{q25}}/D_{\mathrm{min}}$, $\mathrm{GC_{D/R}}$, $\mathrm{GC_{D/R}}/\mathrm{WCC_e}$, $\mathrm{GC_{R/D}}/\mathrm{WCC_e}$, $\mathrm{ID_e}/\mathrm{WCC_e}$, $D_{\mathrm{vc}}/\mathrm{CC_{max}}$, $D_{\mathrm{vc}}\cdot\mathrm{ID_e}$, $D_{\mathrm{vc}}/\mathrm{WCC_{mean}}$, $S_e/\mathrm{CC_{max}}$, S_e/D_{min}, S_e/D_{vc}, $S_e/\mathrm{WCC_e}$, $\mathrm{WCC_e}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{max}}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{mean}}$, $\mathrm{WCC_{mean}}$, $\mathrm{WCC_{mean}}\cdot D_{\mathrm{min}}$, $\mathrm{WCC_{mean}}/CS_{\mathrm{max}}$, $\mathrm{WCC_{mean}}/D_{\mathrm{vc}}$
h5_e2_mdl_gen	$\mathrm{CC_g}$, $\mathrm{CC_g}\cdot\mathrm{CC_{mean}}$, $\mathrm{CC_g}/D_{\mathrm{vc}}$, $\mathrm{CC_{max}}$, $\mathrm{CC_{max}}\cdot CS_{\mathrm{max}}$, $\mathrm{CC_{max}}\cdot D_{\mathrm{min}}$, $\mathrm{CC_{max}}\cdot S_e$, $\mathrm{CC_{max}}/\mathrm{GC_{D/R}}$, $\mathrm{CC_{mean}}$, $CS_{\mathrm{max}}\cdot CS_{\mathrm{q25}}$, $CS_{\mathrm{max}}/D_{\mathrm{min}}$, $\mathrm{GC_{D/R}}/\mathrm{WCC_e}$, $D_{\mathrm{min}}/CS_{\mathrm{max}}$, $D_{\mathrm{vc}}/\mathrm{WCC_{mean}}$, $S_e/\mathrm{CC_{max}}$, S_e/D_{min}, $\mathrm{WCC_e}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{max}}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{mean}}$, $\mathrm{WCC_{mean}}$, $\mathrm{WCC_{mean}}/\mathrm{CC_{max}}$, $\mathrm{WCC_{mean}}\cdot D_{\mathrm{min}}$, $\mathrm{WCC_{mean}}/D_{\mathrm{vc}}$
h5_e2_mdl_gen_50	$\mathrm{CC_g}$, $\mathrm{CC_g}\cdot\mathrm{CC_{mean}}$, $\mathrm{CC_g}/\mathrm{ID_e}$, $\mathrm{CC_g}/D_{\mathrm{vc}}$, $\mathrm{CC_{max}}$, $\mathrm{CC_{max}}\cdot\mathrm{CC_{mean}}$, $\mathrm{CC_{max}}\cdot CS_{\mathrm{max}}$, $\mathrm{CC_{max}}\cdot D_{\mathrm{min}}$, $\mathrm{CC_{max}}\cdot S_e$, $\mathrm{CC_{max}}/\mathrm{GC_{D/R}}$, $\mathrm{CC_{mean}}$, $CS_{\mathrm{max}}\cdot CS_{\mathrm{mean}}$, $CS_{\mathrm{max}}\cdot CS_{\mathrm{q25}}$, $CS_{\mathrm{max}}/D_{\mathrm{min}}$, $\mathrm{GC_{D/R}}/\mathrm{CC_{max}}$, $\mathrm{GC_{D/R}}/\mathrm{WCC_e}$, $D_{\mathrm{min}}/CS_{\mathrm{max}}$, $D_{\mathrm{vc}}\cdot\mathrm{ID_e}$, $D_{\mathrm{vc}}/\mathrm{WCC_{mean}}$, S_e, $S_e/\mathrm{CC_{max}}$, S_e/D_{min}, $\mathrm{WCC_e}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{max}}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{mean}}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{q25}}$, $\mathrm{WCC_{mean}}$, $\mathrm{WCC_{mean}}/\mathrm{CC_{max}}$, $\mathrm{WCC_{mean}}\cdot D_{\mathrm{min}}$, $\mathrm{WCC_{mean}}/D_{\mathrm{vc}}$
h6_e2_mdl_bff	$\mathrm{CC_g}$, $\mathrm{CC_g}/\mathrm{CC_{max}}$, $\mathrm{CC_g}\cdot\mathrm{CC_{mean}}$, $\mathrm{CC_{max}}$, $\mathrm{CC_{max}}\cdot CS_{\mathrm{max}}$, $\mathrm{CC_{max}}\cdot D_{\mathrm{min}}$, $\mathrm{CC_{max}}\cdot\mathrm{WCC_e}$, $\mathrm{CC_{max}}/D_{\mathrm{vc}}$, $\mathrm{CC_{max}}/S_{\mathrm{ne}}$, $\mathrm{CC_{max}}/\mathrm{WCC_{mean}}$, $\mathrm{CC_{mean}}$, $\mathrm{CC_{mean}}/CS_{\mathrm{min}}$, $\mathrm{CC_{mean}}/\mathrm{WCC_{mean}}$, $CS_{\mathrm{max}}\cdot CS_{\mathrm{min}}$, $CS_{\mathrm{max}}/D_{\mathrm{min}}$, $CS_{\mathrm{min}}/\mathrm{CC_{mean}}$, $\mathrm{GC_{D/R}}$, D_{min}, $D_{\mathrm{min}}/D_{\mathrm{vc}}$, $D_{\mathrm{min}}/S_{\mathrm{en}}$, $D_{\mathrm{vc}}/\mathrm{CC_g}$, $D_{\mathrm{vc}}/\mathrm{CC_{max}}$, $D_{\mathrm{vc}}/\mathrm{WCC_{mean}}$, S_{en}, $S_{\mathrm{en}}\cdot S_{\mathrm{ne}}$, $S_{\mathrm{en}}/D_{\mathrm{min}}$, $\mathrm{WCC_e}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{max}}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{q25}}$, $\mathrm{WCC_{mean}}$, $\mathrm{WCC_{mean}}/D_{\mathrm{vc}}$
h6_e2_mdl_gen_50	$\mathrm{CC_g}$, $\mathrm{CC_g}\cdot\mathrm{CC_{mean}}$, $\mathrm{CC_{max}}$, $\mathrm{CC_{max}}\cdot CS_{\mathrm{max}}$, $\mathrm{CC_{max}}\cdot D_{\mathrm{min}}$, $\mathrm{CC_{max}}\cdot S_{\mathrm{en}}$, $\mathrm{CC_{max}}/D_{\mathrm{vc}}$, $\mathrm{CC_{mean}}$, $\mathrm{CC_{mean}}/CS_{\mathrm{min}}$, $CS_{\mathrm{max}}/D_{\mathrm{min}}$, $CS_{\mathrm{min}}/\mathrm{CC_{mean}}$, $\mathrm{GC_{D/R}}/\mathrm{GC_{R/D}}$, $D_{\mathrm{min}}\cdot S_{\mathrm{ne}}$, $D_{\mathrm{min}}/CS_{\mathrm{max}}$, $D_{\mathrm{min}}/S_{\mathrm{en}}$, $D_{\mathrm{vc}}/\mathrm{CC_{max}}$, $D_{\mathrm{vc}}/\mathrm{WCC_{mean}}$, S_{en}, $S_{\mathrm{en}}\cdot S_{\mathrm{ne}}$, $S_{\mathrm{en}}/D_{\mathrm{min}}$, $S_{\mathrm{ne}}/\mathrm{CC_{max}}$, $\mathrm{WCC_e}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{max}}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{q25}}$, $\mathrm{WCC_{mean}}$, $\mathrm{WCC_{mean}}/D_{\mathrm{vc}}$
h7_e2_mdl_gen_50	$B^{k_{\mathrm{UB}}}_{\mathrm{dist}}/\mathrm{CC_g}$, $B^{k_{\mathrm{UB}}}_{\mathrm{dist}}/\mathrm{CC_{max}}$, $B^{k_{\mathrm{UB}}}_{\mathrm{dist}}/\mathrm{CC_{mean}}$, $B^{k_{\mathrm{UB}}}_{\mathrm{dist}}\cdot CS_{\mathrm{max}}$, $B^{k_{\mathrm{UB}}}_{\mathrm{dist}}/D_{\mathrm{min}}$, $B^{k_{\mathrm{UB}}}_{\mathrm{dist}}/\mathrm{WCC_{mean}}$, $B_{\mathrm{lu}}\cdot\mathrm{CC_g}$, $B_{\mathrm{lu}}\cdot\mathrm{CC_{max}}$, $B_{\mathrm{lu}}\cdot\mathrm{CC_{mean}}$, $B_{\mathrm{lu}}/\mathrm{ID_{mean}}$, $B_{\mathrm{lu}}/\mathrm{WCC_{min}}$, $\mathrm{CC_g}$, $\mathrm{CC_g}/B^{k_{\mathrm{UB}}}_{\mathrm{dist}}$, $\mathrm{CC_g}/B_{\mathrm{lu}}$, $\mathrm{CC_g}\cdot\mathrm{CC_{max}}$, $\mathrm{CC_g}\cdot\mathrm{CC_{mean}}$, $\mathrm{CC_g}\cdot S_{\mathrm{en}}$, $\mathrm{CC_g}/S_{\mathrm{ne}}$, $\mathrm{CC_{max}}$, $\mathrm{CC_{max}}/B^{k_{\mathrm{UB}}}_{\mathrm{dist}}$, $\mathrm{CC_{max}}/\mathrm{CC_g}$, $\mathrm{CC_{max}}\cdot CS_{\mathrm{max}}$, $\mathrm{CC_{max}}\cdot CS_{\mathrm{mean}}$, $\mathrm{CC_{max}}\cdot D_{\mathrm{e}}$, $\mathrm{CC_{max}}\cdot D_{\mathrm{min}}$, $\mathrm{CC_{max}}\cdot S_{\mathrm{en}}$, $\mathrm{CC_{max}}\cdot\mathrm{WCC_{min}}$, $\mathrm{CC_{max}}/D_{\mathrm{min}}$, $\mathrm{CC_{max}}/S_{\mathrm{en}}$, $\mathrm{CC_{max}}/S_{\mathrm{ne}}$, $\mathrm{CC_{max}}/\mathrm{WCC_{mean}}$, $\mathrm{CC_{max}}/\mathrm{WCC_{min}}$, $\mathrm{CC_{mean}}$, $\mathrm{CC_{mean}}/B^{k_{\mathrm{UB}}}_{\mathrm{dist}}$, $\mathrm{CC_{mean}}\cdot\mathrm{WCC_{min}}$, $\mathrm{CC_{mean}}/\mathrm{WCC_{mean}}$, $CS_{\mathrm{max}}\cdot\mathrm{GC_{R/D}}$, $CS_{\mathrm{max}}\cdot D_{\mathrm{e}}$, $CS_{\mathrm{max}}/\mathrm{GC_{D/R}}$, $CS_{\mathrm{max}}/D_{\mathrm{min}}$, $CS_{\mathrm{mean}}\cdot D_{\mathrm{e}}$, $CS_{\mathrm{mean}}/CS_{\mathrm{q25}}$, $CS_{\mathrm{q25}}\cdot D_{\mathrm{e}}$, $\mathrm{GC_{DSAT}}/D_{\mathrm{min}}$, $\mathrm{GC_{D/R}}$, $\mathrm{GC_{D/R}}/CS_{\mathrm{max}}$, $\mathrm{GC_{D/R}}/\mathrm{GC_{R/D}}$, $\mathrm{GC_{D/R}}/\mathrm{WCC_e}$, $\mathrm{GC_{D/R}}/\mathrm{WCC_{min}}$, $\mathrm{GC_{R/D}}$, $\mathrm{GC_{R/D}}/\mathrm{WCC_e}$, $\mathrm{GC_{R/D}}/\mathrm{WCC_{min}}$, $\mathrm{ID_e}$, $\mathrm{ID_e}/\mathrm{CC_{mean}}$, $\mathrm{ID_{mean}}\cdot\mathrm{GC_{DSAT}}$, $\mathrm{ID_{mean}}/D_{\mathrm{e}}$, $\mathrm{ID_{mean}}/\mathrm{WCC_e}$, $\mathrm{IR_{mean}}/D_{\mathrm{e}}$, $\mathrm{IR_{mean}}/\mathrm{WCC_e}$, $D_{\mathrm{e}}/\mathrm{CC_g}$, $D_{\mathrm{e}}\cdot\mathrm{GC_{DSAT}}$, $D_{\mathrm{e}}/\mathrm{ID_{mean}}$, $D_{\mathrm{e}}/D_{\mathrm{min}}$, $D_{\mathrm{e}}/\mathrm{WCC_{min}}$, D_{min}, $D_{\mathrm{min}}/B^{k_{\mathrm{UB}}}_{\mathrm{dist}}$, $D_{\mathrm{min}}/\mathrm{CC_{max}}$, $D_{\mathrm{min}}\cdot\mathrm{ID_{mean}}$, $D_{\mathrm{min}}\cdot\mathrm{IR_{mean}}$, $D_{\mathrm{min}}\cdot S_{\mathrm{ne}}$, $D_{\mathrm{min}}/CS_{\mathrm{max}}$, $D_{\mathrm{min}}/\mathrm{GC_{DSAT}}$, $D_{\mathrm{min}}/D_{\mathrm{e}}$, $D_{\mathrm{min}}/S_{\mathrm{en}}$, $D_{\mathrm{min}}/S_{\mathrm{ne}}$, S_{en}, $S_{\mathrm{en}}\cdot S_{\mathrm{ne}}$, $S_{\mathrm{en}}/D_{\mathrm{min}}$, $S_{\mathrm{ne}}/\mathrm{CC_{max}}$, $S_{\mathrm{ne}}/\mathrm{WCC_{min}}$, $\mathrm{WCC_e}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{max}}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{mean}}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{q25}}$, $\mathrm{WCC_e}\cdot\mathrm{GC_{DSAT}}$, $\mathrm{WCC_e}\cdot\mathrm{GC_{D/R}}$, $\mathrm{WCC_e}\cdot\mathrm{GC_{R/D}}$, $\mathrm{WCC_e}\cdot\mathrm{ID_e}$, $\mathrm{WCC_e}\cdot D_{\mathrm{e}}$, $\mathrm{WCC_e}/\mathrm{GC_{D/R}}$, $\mathrm{WCC_e}/\mathrm{GC_{R/D}}$, $\mathrm{WCC_{mean}}$, $\mathrm{WCC_{mean}}/B^{k_{\mathrm{UB}}}_{\mathrm{dist}}$, $\mathrm{WCC_{mean}}/\mathrm{CC_{max}}$, $\mathrm{WCC_{mean}}/\mathrm{CC_{mean}}$, $\mathrm{WCC_{mean}}\cdot S_{\mathrm{en}}$, $\mathrm{WCC_{min}}$, $\mathrm{WCC_{min}}/B_{\mathrm{lu}}$, $\mathrm{WCC_{min}}\cdot\mathrm{GC_{D/R}}$, $\mathrm{WCC_{min}}\cdot\mathrm{GC_{R/D}}$, $\mathrm{WCC_{min}}\cdot D_{\mathrm{min}}$, $\mathrm{WCC_{min}}\cdot S_{\mathrm{en}}$, $\mathrm{WCC_{min}}/\mathrm{GC_{D/R}}$, $\mathrm{WCC_{min}}/\mathrm{GC_{R/D}}$, $\mathrm{WCC_{min}}/D_{\mathrm{e}}$, $\mathrm{WCC_{min}}/S_{\mathrm{ne}}$

Table A.11: Selected subsets of features discretized with the *MDL* criteria, based on the feature selection on the data set `mixed`.

Setname	Features
h6_e3_kon_gen	$\mathrm{CC_g}$, $\mathrm{CC_g}\cdot\mathrm{CC_{max}}$, $\mathrm{CC_g}\cdot\mathrm{CC_{mean}}$, $\mathrm{CC_g}\cdot\mathrm{GC_{D/R}}$, $\mathrm{CC_g}/CS_{\mathrm{max}}$, $\mathrm{CC_g}/CS_{\mathrm{min}}$, $\mathrm{CC_g}/\mathrm{GC_{R/D}}$, $\mathrm{CC_g}/D_{\mathrm{vc}}$, $\mathrm{CC_g}/\mathrm{WCC_{vc}}$, $\mathrm{CC_{max}}$, $\mathrm{CC_{max}}\cdot CS_{\mathrm{max}}$, $\mathrm{CC_{max}}\cdot CS_{\mathrm{mean}}$, $\mathrm{CC_{max}}\cdot CS_{\mathrm{min}}$, $\mathrm{CC_{max}}\cdot\mathrm{GC_{R/D}}$, $\mathrm{CC_{max}}\cdot D_{\mathrm{min}}$, $\mathrm{CC_{max}}\cdot S_e$, $\mathrm{CC_{max}}\cdot S_{\mathrm{en}}$, $\mathrm{CC_{max}}\cdot\mathrm{WCC_e}$, $\mathrm{CC_{max}}\cdot\mathrm{WCC_{min}}$, $\mathrm{CC_{max}}/\mathrm{GC_{D/R}}$, $\mathrm{CC_{max}}/D_{\mathrm{vc}}$, $\mathrm{CC_{max}}/S_{\mathrm{ne}}$, $\mathrm{CC_{max}}/\mathrm{WCC_{mean}}$, $\mathrm{CC_{mean}}$, $\mathrm{CC_{mean}}\cdot\mathrm{IR_{min}}$, $\mathrm{CC_{mean}}\cdot\mathrm{WCC_{min}}$, $\mathrm{CC_{mean}}/CS_{\mathrm{max}}$, $\mathrm{CC_{mean}}/CS_{\mathrm{min}}$, $\mathrm{CC_{mean}}/\mathrm{IR_{min}}$, $\mathrm{CC_{mean}}/D_{\mathrm{vc}}$, $CS_{\mathrm{max}}/\mathrm{CC_{mean}}$, $CS_{\mathrm{max}}\cdot CS_{\mathrm{mean}}$, $CS_{\mathrm{max}}\cdot\mathrm{GC_{R/D}}$, $CS_{\mathrm{max}}/\mathrm{GC_{D/R}}$, $CS_{\mathrm{max}}/D_{\mathrm{min}}$, CS_{min}, $CS_{\mathrm{min}}/\mathrm{CC_g}$, $CS_{\mathrm{min}}/\mathrm{CC_{mean}}$, $\mathrm{GC_{D/R}}$, $\mathrm{GC_{D/R}}/\mathrm{CC_{max}}$, $\mathrm{GC_{D/R}}\cdot\mathrm{LS_i}$, $\mathrm{GC_{D/R}}/\mathrm{GC_{R/D}}$, $\mathrm{GC_{D/R}}/\mathrm{WCC_e}$, $\mathrm{GC_{D/R}}/\mathrm{WCC_{min}}$, $\mathrm{GC_{R/D}}/\mathrm{GC_{D/R}}$, $\mathrm{GC_{R/D}}/\mathrm{LS_i}$, $\mathrm{IR_{min}}/\mathrm{CC_{max}}$, $\mathrm{IR_{min}}/\mathrm{CC_{mean}}$, $\mathrm{IR_{min}}/S_e$, $\mathrm{IR_{min}}/\mathrm{WCC_{min}}$, $\mathrm{LS_i}$, $\mathrm{LS_i}/\mathrm{GC_{R/D}}$, $\mathrm{LS_i}/D_{\mathrm{vc}}$, $\mathrm{LS_i}/\mathrm{WCC_e}$, $D_{\mathrm{min}}\cdot\mathrm{IR_{min}}$, $D_{\mathrm{min}}\cdot S_{\mathrm{ne}}$, $D_{\mathrm{min}}/S_{\mathrm{en}}$, $D_{\mathrm{vc}}/\mathrm{CC_{max}}$, $D_{\mathrm{vc}}/\mathrm{CC_{mean}}$, $D_{\mathrm{vc}}/\mathrm{LS_i}$, $D_{\mathrm{vc}}/\mathrm{WCC_{mean}}$, $S_e/\mathrm{IR_{min}}$, S_e/S_{ne}, $S_{\mathrm{en}}\cdot S_e$, $S_{\mathrm{en}}\cdot S_{\mathrm{ne}}$, $S_{\mathrm{en}}/CS_{\mathrm{min}}$, $S_{\mathrm{en}}/D_{\mathrm{min}}$, $S_{\mathrm{ne}}/\mathrm{CC_{max}}$, $S_{\mathrm{ne}}\cdot S_e$, S_{ne}/S_e, $\mathrm{WCC_e}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{max}}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{mean}}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{min}}$, $\mathrm{WCC_e}\cdot\mathrm{GC_{R/D}}$, $\mathrm{WCC_e}\cdot\mathrm{LS_i}$, $\mathrm{WCC_e}\cdot D_{\mathrm{vc}}$, $\mathrm{WCC_e}/\mathrm{WCC_{vc}}$, $\mathrm{WCC_{mean}}$, $\mathrm{WCC_{mean}}/\mathrm{CC_{max}}$, $\mathrm{WCC_{mean}}/D_{\mathrm{vc}}$, $\mathrm{WCC_{mean}}/\mathrm{WCC_{vc}}$, $\mathrm{WCC_{min}}\cdot\mathrm{IR_{min}}$, $\mathrm{WCC_{min}}/\mathrm{GC_{D/R}}$, $\mathrm{WCC_{min}}/\mathrm{IR_{min}}$, $\mathrm{WCC_{vc}}/\mathrm{CC_g}$, $\mathrm{WCC_{vc}}/\mathrm{CC_{max}}$, $\mathrm{WCC_{vc}}/\mathrm{CC_{mean}}$, $\mathrm{WCC_{vc}}\cdot S_e$, $\mathrm{WCC_{vc}}/\mathrm{WCC_{mean}}$
h7_b_kon_bff	$\mathrm{CC_g}$, $\mathrm{CC_{max}}$, $\mathrm{CC_{mean}}$, CS_{max}, CS_{min}, CS_{q25}, $\mathrm{GC_{D/R}}$, $\mathrm{IR_{min}}$, $\mathrm{LS_i}$, S_{en}, $\mathrm{WCC_e}$, $\mathrm{WCC_{mean}}$, $\mathrm{WCC_{min}}$, $\mathrm{WCC_{vc}}$
h7_e3_kon_bff	$\mathrm{CC_g}$, $\mathrm{CC_g}\cdot\mathrm{CC_{mean}}$, $\mathrm{CC_g}/\mathrm{WCC_{vc}}$, $\mathrm{CC_{max}}$, $\mathrm{CC_{max}}\cdot CS_{\mathrm{max}}$, $\mathrm{CC_{max}}\cdot CS_{\mathrm{min}}$, $\mathrm{CC_{max}}\cdot S_{\mathrm{en}}$, $\mathrm{CC_{max}}\cdot\mathrm{WCC_e}$, $\mathrm{CC_{max}}\cdot\mathrm{WCC_{min}}$, $\mathrm{CC_{max}}/\mathrm{GC_{D/R}}$, $\mathrm{CC_{max}}/S_{\mathrm{ne}}$, $\mathrm{CC_{max}}/\mathrm{WCC_{mean}}$, $\mathrm{CC_{mean}}$, $\mathrm{CC_{mean}}\cdot\mathrm{IR_{min}}$, $\mathrm{CC_{mean}}\cdot\mathrm{WCC_{min}}$, $\mathrm{CC_{mean}}/CS_{\mathrm{max}}$, $\mathrm{CC_{mean}}/CS_{\mathrm{min}}$, $\mathrm{CC_{mean}}/\mathrm{IR_{min}}$, $CS_{\mathrm{max}}/\mathrm{CC_{mean}}$, CS_{min}, $CS_{\mathrm{min}}/\mathrm{CC_{mean}}$, CS_{q25}, $CS_{\mathrm{q25}}\cdot S_{\mathrm{en}}$, $CS_{\mathrm{q25}}/S_{\mathrm{ne}}$, $\mathrm{GC_{D/R}}$, $\mathrm{GC_{D/R}}\cdot\mathrm{LS_i}$, $\mathrm{GC_{D/R}}/S_{\mathrm{en}}$, $\mathrm{GC_{D/R}}/\mathrm{WCC_e}$, $\mathrm{GC_{D/R}}/\mathrm{WCC_{min}}$, $\mathrm{GC_{D/R}}/\mathrm{WCC_{vc}}$, $\mathrm{GC_{R/D}}$, $\mathrm{IR_{min}}/\mathrm{CC_{mean}}$, $\mathrm{IR_{min}}\cdot S_{\mathrm{en}}$, $\mathrm{IR_{min}}/\mathrm{WCC_{min}}$, $\mathrm{IR_{min}}/\mathrm{WCC_{vc}}$, $\mathrm{LS_i}/\mathrm{WCC_e}$, $S_{\mathrm{en}}\cdot S_{\mathrm{ne}}$, $S_{\mathrm{ne}}/\mathrm{CC_g}$, $S_{\mathrm{ne}}/\mathrm{WCC_{min}}$, $\mathrm{WCC_e}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{max}}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{q25}}$, $\mathrm{WCC_e}\cdot\mathrm{LS_i}$, $\mathrm{WCC_e}/\mathrm{IR_{min}}$, $\mathrm{WCC_e}/\mathrm{WCC_{vc}}$, $\mathrm{WCC_{mean}}$, $\mathrm{WCC_{mean}}/\mathrm{LS_i}$, $\mathrm{WCC_{mean}}/\mathrm{WCC_{vc}}$, $\mathrm{WCC_{min}}\cdot\mathrm{IR_{min}}$, $\mathrm{WCC_{min}}\cdot\mathrm{LS_i}$, $\mathrm{WCC_{min}}/\mathrm{GC_{D/R}}$, $\mathrm{WCC_{vc}}/\mathrm{CC_g}$, $\mathrm{WCC_{vc}}/\mathrm{CC_{mean}}$, $\mathrm{WCC_{vc}}/\mathrm{WCC_{mean}}$
h7_e3_kon_gen	$\mathrm{CC_g}$, $\mathrm{CC_g}\cdot\mathrm{CC_{mean}}$, $\mathrm{CC_g}/CS_{\mathrm{min}}$, $\mathrm{CC_g}/\mathrm{WCC_{vc}}$, $\mathrm{CC_{max}}$, $\mathrm{CC_{max}}\cdot CS_{\mathrm{max}}$, $\mathrm{CC_{max}}\cdot CS_{\mathrm{min}}$, $\mathrm{CC_{max}}\cdot\mathrm{GC_{R/D}}$, $\mathrm{CC_{max}}\cdot\mathrm{WCC_e}$, $\mathrm{CC_{max}}\cdot\mathrm{WCC_{min}}$, $\mathrm{CC_{max}}/S_{\mathrm{ne}}$, $\mathrm{CC_{mean}}$, $\mathrm{CC_{mean}}\cdot\mathrm{WCC_{min}}$, $\mathrm{CC_{mean}}/CS_{\mathrm{max}}$, $\mathrm{CC_{mean}}/CS_{\mathrm{min}}$, $\mathrm{CC_{mean}}/\mathrm{IR_{min}}$, $\mathrm{CC_{mean}}/\mathrm{WCC_{vc}}$, $CS_{\mathrm{max}}/\mathrm{CC_{mean}}$, $CS_{\mathrm{min}}/\mathrm{CC_{mean}}$, $\mathrm{GC_{D/R}}/\mathrm{WCC_e}$, $\mathrm{GC_{D/R}}/\mathrm{WCC_{min}}$, $\mathrm{IR_{min}}/\mathrm{CC_{mean}}$, $\mathrm{LS_i}/\mathrm{WCC_e}$, $S_{\mathrm{en}}\cdot S_{\mathrm{ne}}$, $S_{\mathrm{ne}}/\mathrm{CC_g}$, $S_{\mathrm{ne}}/\mathrm{CC_{max}}$, $S_{\mathrm{ne}}/CS_{\mathrm{q25}}$, $\mathrm{WCC_e}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{max}}$, $\mathrm{WCC_e}\cdot CS_{\mathrm{q25}}$, $\mathrm{WCC_e}\cdot\mathrm{LS_i}$, $\mathrm{WCC_{mean}}/\mathrm{WCC_{vc}}$, $\mathrm{WCC_{min}}\cdot\mathrm{GC_{R/D}}$, $\mathrm{WCC_{min}}\cdot\mathrm{IR_{min}}$, $\mathrm{WCC_{min}}/\mathrm{GC_{D/R}}$, $\mathrm{WCC_{vc}}/\mathrm{CC_g}$, $\mathrm{WCC_{vc}}/\mathrm{CC_{mean}}$, $\mathrm{WCC_{vc}}/\mathrm{WCC_{mean}}$

Table A.12: Selected subsets of features discretized with the *KON* criteria, based on the feature selection on the data set `mixed`.

A.4 Most-Selected Features

Name	#	(%)	Name	#	(%)	Name	#	(%)
$\mathrm{GC}_{\mathrm{D/R}}$	70	95.89	$\mathrm{GC}_{\mathrm{D/R}}/\mathrm{GC}_{\mathrm{R/D}}$	25	34.25	$\mathrm{CC}_{\mathrm{mean}}/\mathrm{CC}_{\mathrm{vc}}$	19	26.03
$\mathrm{GC}_{\mathrm{R/D}}$	65	89.04	$B_{\mathrm{lu}} \cdot B_{\mathrm{ul}}$	24	32.88	IR_{e}	19	26.03
$\mathrm{LS}_{\mathrm{cne}}$	61	83.56	$CS_{\mathrm{q75}} \cdot D_{\mathrm{vc}}$	24	32.88	$B^{k\mathrm{UB}}_{\mathrm{dist}}/\mathrm{LS}_{\mathrm{cne}}$	18	24.66
$\mathrm{CC}_{\mathrm{max}}$	47	64.38	$D_{\mathrm{vc}}/\mathrm{CC}_{\mathrm{max}}$	24	32.88	$B^{k\mathrm{UB}}_{\mathrm{dist}} \cdot \mathrm{CC}_{\mathrm{g}}$	17	23.29
CS_{time}	44	60.27	$\mathrm{GC}_{\mathrm{D/R}}/\mathrm{LS}_{\mathrm{cne}}$	24	32.88	$\mathrm{CC}_{\mathrm{mean}}/\mathrm{CC}_{\mathrm{med}}$	17	23.29
D_{vc}	44	60.27	CC_{e}	23	31.51	$CS_{\mathrm{mean}} \cdot D_{\mathrm{vc}}$	17	23.29
$\mathrm{WCC}_{\mathrm{mean}}$	42	57.53	$\mathrm{CC}_{\mathrm{max}}/D_{\mathrm{vc}}$	23	31.51	$D_{\mathrm{vc}}/B^{k\mathrm{UB}}_{\mathrm{dist}}$	17	23.29
$\mathrm{WCC}_{\mathrm{vc}}$	40	54.79	$\mathrm{CC}_{\mathrm{med}}$	23	31.51	$\mathrm{LS}_{\mathrm{cne}}/\mathrm{GC}_{\mathrm{R/D}}$	17	23.29
$\mathrm{CC}_{\mathrm{vc}}$	38	52.05	$B_{\mathrm{lu}}/\mathrm{IR}_{\mathrm{vc}}$	22	30.14	$B_{\mathrm{ul}} \cdot \mathrm{IR}_{\mathrm{vc}}$	16	21.92
B_{lu}	36	49.32	CC_{g}	22	30.14	$\mathrm{CC}_{\mathrm{g}}/\mathrm{WCC}_{\mathrm{vc}}$	16	21.92
$B^{k\mathrm{LB}}_{\mathrm{dist}}$	36	49.32	$\mathrm{WCC}_{\mathrm{min}}$	22	30.14	$\mathrm{CC}_{\mathrm{med}}/\mathrm{CC}_{\mathrm{vc}}$	16	21.92
$B^{k\mathrm{UB}}_{\mathrm{dist}} \cdot \mathrm{CC}_{\mathrm{mean}}$	34	46.58	$\mathrm{WCC}_{\mathrm{vc}}/B^{k\mathrm{UB}}_{\mathrm{dist}}$	22	30.14	$\mathrm{CC}_{\mathrm{vc}}/\mathrm{CC}_{\mathrm{mean}}$	16	21.92
$B^{k\mathrm{UB}}_{\mathrm{dist}}$	33	45.21	CS_{q75}	22	30.14	CS_{mean}	16	21.92
$\mathrm{CC}_{\mathrm{mean}}$	33	45.21	$CS_{\mathrm{time}} \cdot \mathrm{LS}_{\mathrm{cee}}$	22	30.14	$B^{k\mathrm{LB}}_{\mathrm{dist}}/\mathrm{LS}_{\mathrm{cne}}$	15	20.55
ID_{e}	31	42.47	$\mathrm{GC}_{\mathrm{R/D}}/\mathrm{GC}_{\mathrm{D/R}}$	22	30.14	$\mathrm{CC}_{\mathrm{e}}/D_{\mathrm{vc}}$	15	20.55
$\mathrm{CC}_{\mathrm{mean}}/D_{\mathrm{vc}}$	29	39.73	$\mathrm{IR}_{\mathrm{vc}}$	22	30.14	$\mathrm{CC}_{\mathrm{e}}/\mathrm{LS}_{\mathrm{cee}}$	15	20.55
$B^{k\mathrm{UB}}_{\mathrm{dist}}/\mathrm{WCC}_{\mathrm{vc}}$	27	36.99	$D_{\mathrm{vc}}/\mathrm{CC}_{\mathrm{mean}}$	21	28.77	CS_{e}	15	20.55
$\mathrm{GC}_{\mathrm{D/R}} \cdot \mathrm{LS}_{\mathrm{cne}}$	27	36.99	$B^{k\mathrm{UB}}_{\mathrm{dist}}/D_{\mathrm{vc}}$	20	27.4	$D_{\mathrm{vc}} \cdot \mathrm{IR}_{\mathrm{e}}$	15	20.55
B_{ul}	26	35.62	$B^{k\mathrm{UB}}_{\mathrm{dist}} \cdot \mathrm{CC}_{\mathrm{max}}$	20	27.4	$\mathrm{ID}_{\mathrm{e}}/\mathrm{LS}_{\mathrm{cne}}$	15	20.55
$\mathrm{WCC}_{\mathrm{mean}} \cdot \mathrm{LS}_{\mathrm{cne}}$	26	35.62	$B^{k\mathrm{UB}}_{\mathrm{dist}} \cdot \mathrm{CC}_{\mathrm{med}}$	20	27.4			
$\mathrm{LS}_{\mathrm{cee}}$	26	35.62	$\mathrm{CC}_{\mathrm{g}}/D_{\mathrm{vc}}$	19	26.03			

Table A.13: Selected features for the instance set chi500 which have been selected in at least 20% of the subsets.

Name	#	(%)	Name	#	(%)	Name	#	(%)
CC_{g}	40	83.33%	CC_{max}	19	39.58%	$\text{CC}_{\text{g}} \cdot \text{CC}_{\text{mean}}$	11	22.92%
CC_{mean}	36	75.00%	$\text{CC}_{\text{min}} \cdot CS_{\text{time}}$	19	39.58%	$\text{CC}_{\text{min}} \cdot CS_{\text{q25}}$	11	22.92%
CC_{min}	34	70.83%	$\text{CC}_{\text{g}} \cdot CS_{\text{time}}$	17	35.42%	CS_{vc}	11	22.92%
WCC_{mean}	32	66.67%	CS_{q25}	17	35.42%	GC_{best}	11	22.92%
D_{mean}	28	58.33%	D_{vc}	17	35.42%	$\text{CC}_{\text{mean}}/\text{WCC}_{\text{e}}$	10	20.83%
CC_{vc}	26	54.17%	$\text{GC}_{\text{R/D}}$	16	33.33%	$\text{CC}_{\text{mean}} \cdot CS_{\text{vc}}$	10	20.83%
WCC_{e}	26	54.17%	ID_{e}	15	31.25%	$\text{WCC}_{\text{e}}/\text{CC}_{\text{min}}$	10	20.83%
WCC_{med}	26	54.17%	IR_{min}	15	31.25%	LS_{i}	10	20.83%
CS_{time}	24	50.00%	$\text{CC}_{\text{min}}/\text{WCC}_{\text{e}}$	12	25.00%	S_e	10	20.83%
$\text{GC}_{\text{D/R}}$	23	47.92%	$\text{WCC}_{\text{vc}} \cdot \text{IR}_{\text{min}}$	12	25.00%	TD_{width}	10	20.83%
WCC_{vc}	21	43.75%	D_{q75}	12	25.00%	$TD_{\text{width}}/\text{CC}_{\text{g}}$	10	20.83%
LS_{ce}	20	41.67%	S_d	12	25.00%			

Table A.14: Selected features for the instance set chi1000 which have been selected in at least 20% of the subsets.

Name	#	(%)	Name	#	(%)	Name	#	(%)
CC_{max}	68	86.08%	S_{en}	25	31.65%	$\text{CC}_{\text{max}}/S_{\text{ne}}$	19	24.05%
CC_{mean}	65	82.28%	$\text{GC}_{\text{D/R}}/\text{WCC}_{\text{e}}$	24	30.38%	$\text{WCC}_{\text{min}}/\text{GC}_{\text{D/R}}$	19	24.05%
WCC_{e}	64	81.01%	$\text{IR}_{\text{max}}/D_{\text{vc}}$	24	30.38%	$\text{IR}_{\text{max}}/\text{WCC}_{\text{vc}}$	19	24.05%
$\text{GC}_{\text{D/R}}$	64	81.01%	$\text{CC}_{\text{mean}}/CS_{\text{min}}$	22	27.85%	$S_{\text{en}} \cdot S_{\text{ne}}$	19	24.05%
WCC_{mean}	61	77.22%	$\text{CC}_{\text{mean}} \cdot \text{WCC}_{\text{min}}$	22	27.85%	$\text{CC}_{\text{e}}/\text{WCC}_{\text{e}}$	18	22.78%
CC_{g}	51	64.56%	WCC_{vc}	22	27.85%	$\text{CC}_{\text{mean}} \cdot \text{IR}_{\text{min}}$	18	22.78%
D_{min}	46	58.23%	$B_{\text{lu}} \cdot \text{CC}_{\text{max}}$	21	26.58%	WCC_{min}	18	22.78%
$\text{GC}_{\text{R/D}}$	40	50.63%	$B_{\text{lu}} \cdot \text{CC}_{\text{mean}}$	21	26.58%	$CS_{\text{min}}/\text{CC}_{\text{mean}}$	18	22.78%
$\text{CC}_{\text{max}} \cdot D_{\text{min}}$	34	43.04%	$\text{CC}_{\text{g}}/\text{WCC}_{\text{vc}}$	21	26.58%	LS_{i}	18	22.78%
CS_{mean}	31	39.24%	$\text{CC}_{\text{mean}}/\text{WCC}_{\text{vc}}$	21	26.58%	IR_{max}	18	22.78%
CS_{min}	30	37.97%	$\text{WCC}_{\text{mean}}/D_{\text{vc}}$	21	26.58%	S_e	18	22.78%
$D_{\text{min}}/S_{\text{en}}$	30	37.97%	$D_{\text{vc}}/\text{IR}_{\text{max}}$	21	26.58%	$\text{CC}_{\text{max}} \cdot CS_{\text{min}}$	17	21.52%
D_{vc}	29	36.71%	$\text{CC}_{\text{max}}/\text{GC}_{\text{D/R}}$	20	25.32%	$\text{WCC}_{\text{e}} \cdot CS_{\text{q25}}$	17	21.52%
$\text{CC}_{\text{max}} \cdot CS_{\text{max}}$	28	35.44%	$\text{CC}_{\text{max}} \cdot CS_{\text{mean}}$	20	25.32%	$\text{WCC}_{\text{vc}}/\text{CC}_{\text{mean}}$	17	21.52%
$\text{WCC}_{\text{e}} \cdot CS_{\text{max}}$	28	35.44%	CC_{vc}	20	25.32%	$\text{WCC}_{\text{vc}}/\text{IR}_{\text{max}}$	17	21.52%
$D_{\text{min}} \cdot \text{IR}_{\text{min}}$	28	35.44%	$\text{WCC}_{\text{vc}}/\text{CC}_{\text{g}}$	20	25.32%	$D_{\text{vc}} \cdot \text{ID}_{\text{e}}$	17	21.52%
ID_{e}	28	35.44%	CS_{max}	20	25.32%	$\text{WCC}_{\text{e}}/\text{CC}_{\text{e}}$	16	20.25%
$\text{CC}_{\text{g}} \cdot \text{CC}_{\text{mean}}$	27	34.18%	$\text{GC}_{\text{D/R}}/\text{WCC}_{\text{min}}$	20	25.32%	$D_{\text{min}} \cdot \text{IR}_{\text{max}}$	16	20.25%
$S_{\text{en}}/D_{\text{min}}$	26	32.91%	$\text{GC}_{\text{D/R}}/\text{GC}_{\text{R/D}}$	20	25.32%	$D_{\text{min}} \cdot S_{\text{ne}}$	16	20.25%
CS_{q25}	25	31.65%	IR_{min}	20	25.32%			

Table A.15: Selected features for the instance set mixed which have been selected in at least 20% of the subsets.

Bibliography

[1] R. Abbasian and M. Mouhoub. An efficient hierarchical parallel genetic algorithm for graph coloring problem. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 521–528, New York, NY, USA, 2011. ACM.

[2] D. Achlioptas, A. Naor, and Y. Peres. Rigorous location of phase transitions in hard optimization problems. *Nature*, 435:759–764, 2005.

[3] D. W. Aha. Generalizing from case studies: A case study. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 1–10. Morgan Kaufmann, 1992.

[4] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, Jan. 1991.

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

[6] J. Akbari Torkestani and M. R. Meybodi. A cellular learning automata-based algorithm for solving the vertex coloring problem. *Expert Systems with Applications*, 38(8):9237–9247, Aug. 2011.

[7] S. Ali and K. A. Smith. On learning algorithm selection for classification. *Applied Soft Computing*, 6(2):119–138, Jan. 2006.

[8] E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[9] E. Angel and V. Zissimopoulos. On the classification of np-complete problems in terms of their correlation coefficient. *Discrete Applied Mathematics*, 99(1-3):261–277, Feb. 2000.

[10] W. Armstrong, P. Christen, E. McCreath, and A. P. Rendell. Dynamic algorithm selection using reinforcement learning. In *Proceedings of the International Workshop on on Integrating AI and Data Mining*, AIDM '06, pages 18–25, Washington, DC, USA, 2006. IEEE Computer Society.

[11] C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2):379 – 388, 2003.

[12] V. C. Barbosa and R. G. Ferreira. On the phase transitions of graph coloring and independent sets. *Physica A: Statistical Mechanics and its Applications*, 343:401–423, Nov 2004.

[13] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. Resende, and W. R. Stewart. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1:9–32, 1995.

[14] R. S. Barr, B. L. Goldeny, J. Kellyz, W. R. Stewart, and M. G. C. Resende. Guidelines for Designing and Reporting on Computational Experiments with Heuristic Methods. Technical report, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX, 2001.

[15] C. J. Beck and E. C. Freuder. Simple rules for low-knowledge algorithm selection. In *Proceedings of 1st CPAIOR*, pages 50–64. Springer, 2004.

[16] M. R. Berthold and D. J. Hand, editors. *Intelligent Data Analysis, An Introduction, 2nd editon*. Springer, 2003.

[17] M. Bessedik, R. Laib, A. Boulmerka, and H. Drias. Ant colony system for graph coloring problem. In *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-1 (CIMCA-IAWTIC'06) - Volume 01*, CIMCA '05, pages 786–791, Washington, DC, USA, 2005. IEEE Computer Society.

[18] M. Bessedik, B. Toufik, and H. Drias. How can bees colour graphs. *International Journal of Bio-Inspired Computation*, 3(1):67–76, Feb. 2011.

[19] M. Birattari. On the estimation of the expected performance of a metaheuristic on a class of instances. How many instances, how many runs? Technical Report TR/IRIDIA/-2004-001, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2004.

[20] I. Blöchliger and N. Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operations Research*, 35(3):960–975, Mar. 2008.

[21] C. Blum and R. Battiti, editors. *Learning and Intelligent Optimization, 4th International Conference, LION 4, Venice, Italy, January 18-22, 2010. Selected Papers*, volume 6073 of *Lecture Notes in Computer Science*. Springer, 2010.

[22] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

[23] R. R. Bouckaert, E. Frank, M. Hall, R. Kirkby, P. Reutemann, A. Seewald, and D. Scuse. *Weka manual (3.6.6)*, Oct. 2011.

[24] N. Bouhmala and O.-C. Granmo. Solving graph coloring problems using learning automata. In *Proceedings of the 8th European conference on Evolutionary computation in combinatorial optimization*, EvoCOP'08, pages 277–288, Berlin, Heidelberg, 2008. Springer-Verlag.

[25] H. Bouziri, K. Mellouli, and E.-G. Talbi. The k-coloring fitness landscape. *Journal of Combinatorial Optimization*, 21(3):306–329, Apr. 2011.

[26] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[27] P. Brazdil and C. Soares. A comparison of ranking methods for classification algorithm selection. In *Proceedings of the 11th European Conference on Machine Learning*, ECML '00, pages 63–74, London, UK, UK, 2000. Springer-Verlag.

[28] P. B. Brazdil, C. Soares, and J. P. Da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, Mar. 2003.

[29] D. Brélaz. New methods to color the vertices of a graph. *Commun. ACM*, 22:251–256, apr 1979.

[30] C. E. Brodley. Addressing the selective superiority problem: Automatic algorithm/model class selection. In *10th International Machine Learning Conference(ICML'93)*, pages 17–24, 1993.

[31] K. L. Brown, E. Nudelman, G. Andrew, J. Mcfadden, and Y. Shoham. Boosting as a Metaphor for Algorithm Design. In F. Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 899–903. Springer, 2003.

[32] K. L. Brown, E. Nudelman, and Y. Shoham. Empirical hardness models: Methodology and a case study on combinatorial auctions. *J. ACM*, 56(4):1–52, 2009.

[33] E. K. Burke, J. Marecek, A. J. Parkes, and H. Rudová. On a clique-based integer programming formulation of vertex colouring with applications in course timetabling. *CoRR*, abs/0710.3603, 2007.

[34] E. K. Burke, J. Mareček, A. J. Parkes, and H. Rudová. On a clique-based integer programming formulation of vertex colouring with applications in course timetabling. Technical Report NOTTCS-TR-2007-10, The University of Nottingham, Nottingham, 2007.

[35] E. K. Burke, B. Mccollum, A. Meisels, S. Petrovic, and R. Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176:177–192, 2007.

[36] M. Campelo, V. A. Campos, and R. C. Corrêa. On the asymmetric representatives formulation for the vertex coloring problem. *Discrete Applied Mathematics*, 156(7):1097 – 1111, 2008.

[37] M. Caramia and P. Dell'Olmo. A fast and simple local search for graph coloring. In *Proceedings of the 3rd International Workshop on Algorithm Engineering*, WAE '99, pages 316–329, London, UK, UK, 1999. Springer-Verlag.

[38] M. Caramia and P. Dell'Olmo. Bounding vertex coloring by truncated multistage branch and bound. *Networks*, 44(4):231–242, Dec. 2004.

[39] M. Caramia and P. Dell'Olmo. Coloring graphs by iterated local search traversing feasible and infeasible solutions. *Discrete Applied Mathematics*, 156(2):201–217, Jan. 2008.

[40] M. Caramia and P. Dell'Olmo. Embedding a novel objective function in a two-phased local search for robust vertex coloring. *European Journal of Operational Research*, 189(3):1358 – 1380, 2008.

[41] M. Caramia, P. Dell'Olmo, and G. F. Italiano. Checkcol: Improved local search for graph coloring. *Journal of Discrete Algorithms*, 4(2):277 – 298, 2006.

[42] T. Carchrae and J. C. Beck. Low-knowledge algorithm control. In D. L. McGuinness and G. Ferguson, editors, *AAAI*, pages 49–54. AAAI Press / The MIT Press, 2004.

[43] R. Caruana, N. Karampatziakis, and A. Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 96–103, New York, NY, USA, 2008. ACM.

[44] G. Chaitin. Register allocation and spilling via graph coloring. *SIGPLAN Not.*, 39(4):66–74, Apr. 2004.

[45] D. Chalupa. Population-based and learning-based metaheuristic algorithms for the graph coloring problem. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 465–472, New York, NY, USA, 2011. ACM.

[46] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of the 12th international joint conference on Artificial intelligence - Volume 1*, IJCAI'91, pages 331–337, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[47] M. Chiarandini. *Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems*. PhD thesis, TU Darmstadt, aug 2005.

[48] M. Chiarandini, I. Dumitrescu, and T. Stützle. Stochastic local search algorithms for the graph colouring problem. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, Computer & Information Science Series, pages 63.1–63.17. Chapman & Hall/CRC, Boca Raton, FL, USA, 2007.

[49] M. Chiarandini, I. Dumitrescu, and T. Stützle. Very large-scale neighborhood search: Overview and case studies on coloring problems. In C. Blum, M. J. B. Aguilera, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 114 of *Studies in Computational Intelligence*, pages 117–150. Springer, 2008.

[50] M. Chiarandini, G. Galbiati, and S. Gualandi. Efficiency issues in the RLF heuristic for graph coloring. In L. D. Gaspero, A. Schaerf, and T. Stützle, editors, *Proceedings of the 9th Metaheuristics International Conference, MIC 2011*, pages 461–469, Udine, Italy, 2011. Dipartimento di Ingegneria Elettrica, Gestionale e Meccanica, Università di Udine.

[51] M. Chiarandini and T. Stützle. An application of iterated local search to graph coloring. In D. S. Johnson, A. Mehrotra, and M. A. Trick, editors, *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, pages 112–125, Ithaca, New York, USA, sep 2002.

[52] M. Chiarandini and T. Stützle. An analysis of heuristics for vertex colouring. In P. Festa, editor, *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 326–337. Springer Berlin / Heidelberg, 2010.

[53] M. Chiarandini and T. Stützle. Online compendium to the article: An analysis of heuristics for vertex colouring. `http://www.imada.sdu.dk/~marco/gcp-study/`, 2010.

[54] N. Christofides. An algorithm for the chromatic number of a graph. *The Computer Journal*, 14(1):38–39, 1971.

[55] G. F. Cooper and E. Herskovits. A bayesian method for constructing bayesian belief networks from databases. In *Proceedings of the seventh conference (1991) on Uncertainty in artificial intelligence*, pages 86–94, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[56] G. F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, Oct. 1992.

[57] C. Cortes and V. N. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[58] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, Jan. 1967.

[59] J. Culberson and A. Beacham. Hiding our colors. In *CP'95 Workshop on Studying and Solving Really Hard Problems*, pages 31–42, 1995.

[60] J. C. Culberson. On the futility of blind search: An algorithmic view of “no free lunch”. *Evolutionary Computation*, 6(2):109–127, June 1998.

[61] J. C. Culberson and F. Luo. Exploring the k-colorable landscape with iterated greedy. In *Dimacs Series in Discrete Mathematics and Theoretical Computer Science*, pages 245–284. American Mathematical Society, 1995.

[62] J. C. Culberson and F. Luo. Exploring the k-colorable landscape with iterated greedy. In *Dimacs Series in Discrete Mathematics and Theoretical Computer Science*, pages 245–284. American Mathematical Society, 1995.

[63] V. Cutello, G. Nicosia, and M. Pavone. A hybrid immune algorithm with information gain for the graph coloring problem. In *Proceedings of the 2003 international conference on Genetic and evolutionary computation: PartI*, GECCO'03, pages 171–182, Berlin, Heidelberg, 2003. Springer-Verlag.

[64] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1:131–156, 1997.

[65] L. Davis. Order-based genetic algorithms and the graph coloring problem. In *Handbook of Genetic Algorithms*, pages 72–90. Van Nostrand Reinhold; New York, 1991.

[66] A. de Carvalho and A. Freitas. A tutorial on multi-label classification techniques. In A. Abraham, A.-E. Hassanien, and V. Snášel, editors, *Foundations of Computational Intelligence Volume 5*, volume 205 of *Studies in Computational Intelligence*, pages 177–195. Springer Berlin / Heidelberg, 2009.

[67] C. Demetrescu and G. F. Italiano. What do we learn from experimental algorithmics? In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, MFCS '00, pages 36–51, London, UK, UK, 2000. Springer-Verlag.

[68] A. Dermaku, T. Ganzow, G. Gottlob, B. J. McMahan, N. Musliu, and M. Samer. Heuristic methods for hypertree decomposition. In A. F. Gelbukh and E. F. Morales, editors, *MICAI*, volume 5317 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2008.

[69] I. Devarenne, H. Mabed, and A. Caminada. Intelligent neighborhood exploration in local search heuristics. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '06, pages 144–150, Washington, DC, USA, 2006. IEEE Computer Society.

[70] A. Di Blas, A. Jagota, and R. Hughey. Energy function-based approaches to graph coloring. *Transactions on Neural Networks*, 13(1):81–91, Jan. 2002.

[71] A. Di Blas, A. Jagota, and R. Hughey. A range-compaction heuristic for graph coloring. *Journal of Heuristics*, 9(6):489–506, Dec. 2003.

[72] M. Dorigo and T. Stützle. *Ant colony optimization.* MIT Press, 2004.

[73] R. Dorne and J.-K. Hao. A new genetic local search algorithm for graph coloring. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, PPSN V, pages 745–754, London, UK, UK, 1998. Springer-Verlag.

[74] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *MACHINE LEARNING: PROCEEDINGS OF THE TWELFTH INTERNATIONAL CONFERENCE*, pages 194–202. Morgan Kaufmann, 1995.

[75] K. A. Dowsland and J. M. Thompson. An improved ant colony optimisation heuristic for graph colouring. *Discrete Applied Mathematics*, 156(3):313–324, Feb. 2008.

[76] S. Droste, T. Jansen, and I. Wegener. Optimization with randomized search heuristics - the (a)nfl theorem, realistic scenarios, and difficult functions. *Theoretical Computer Science*, 287(1):131 – 144, 2002.

[77] A. E. Eiben, J. K. Van der Hauw, and J. Van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46, June 1998.

[78] D. Eppstein. Small maximal independent sets and faster exact graph coloring. In *Proceedings of the 7th International Workshop on Algorithms and Data Structures*, WADS '01, pages 462–470, London, UK, UK, 2001. Springer-Verlag.

[79] A. E. Eraghi, J. A. Torkestani, and M. R. Meybodi. Cellular learning automata-based graph coloring problem. In *Machine Learning and Computing: Selected, Peer Reviewed Papers from the 2009 International Conference on Machine Learning and Computing (ICMLC 2009)*, Perth, Australia, 2011. IPCSIT.

[80] R. Ewald. Experimentation methodology. In *Automatic Algorithm Selection for Complex Simulation Problems*, pages 203–246. Vieweg+Teubner Verlag, 2012.

[81] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In R. Bajcsy, editor, *IJCAI*, pages 1022–1029. Morgan Kaufmann, 1993.

[82] U. Feige and J. Kilian. Zero knowledge and the chromatic number. *Journal of Computer and System Sciences*, 57(2):187–199, Oct. 1998.

[83] E. Fink. How to solve it automatically: Selection among problem solving methods. In R. G. Simmons, M. M. Veloso, and S. F. Smith, editors, *AIPS*, pages 128–136. AAAI, 1998.

[84] D. Fotakis, S. D. Likothanassis, and S. K. Stefanakos. An evolutionary annealing approach to graph coloring. In *Proceedings of the EvoWorkshops on Applications of Evolutionary Computing*, pages 120–129, London, UK, UK, 2001. Springer-Verlag.

[85] L. C. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40(1):35–41, Mar. 1977.

[86] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997.

[87] M. Gagliolo and J. Schmidhuber. Dynamic algorithm portfolios, jan 2006. *AI&MATH '06 — Ninth International Symposium on Artificial Intelligence and Mathematics*.

[88] M. Gagliolo and J. Schmidhuber. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, 47(3):295–328, aug 2006.

[89] M. Gagliolo and J. Schmidhuber. Algorithm selection as a bandit problem with unbounded losses. In Blum and Battiti [21], pages 82–96.

[90] P. Galinier and J.-K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3:379–397, 1999.

[91] P. Galinier and A. Hertz. A survey of local search methods for graph coloring. *Computers and Operations Research*, 33(9):2547–2562, Sept. 2006.

[92] P. Galinier, A. Hertz, and N. Zufferey. An adaptive memory algorithm for the k-coloring problem. *Discrete Applied Mathematics*, 156(2):267–279, Jan. 2008.

[93] A. Gamst. Some lower bounds for a class of frequency assignment problems. *Vehicular Technology, IEEE Transactions on*, 35(1):8 – 14, feb 1986.

[94] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[95] M. R. Garey, D. S. Johnson, and S. C. Hing. An application of graph coloring to printed circuit testing. *Circuits and Systems, IEEE Transactions on*, 23(10):591 – 599, oct. 1976.

[96] D. Gassen and J. Carothers. Graph color minimization using neural networks. In *Neural Networks, 1993. IJCNN '93-Nagoya. Proceedings of 1993 International Joint Conference on*, volume 2, pages 1541 – 1544 vol.2, oct. 1993.

[97] C. Gebruers, A. Guerri, B. Hnich, and M. Milano. Making choices using structure at the instance level within a case based reasoning framework. In *CPAIOR*, pages 380–386. Springer Verlag, 2004.

[98] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, M. T. Schneider, and S. Ziller. A portfolio solver for answer set programming: preliminary report. In *Proceedings of the 11th international conference on Logic programming and nonmonotonic reasoning*, LPNMR'11, pages 352–357, Berlin, Heidelberg, 2011. Springer-Verlag.

[99] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[100] S. Ghodke and T. Baldwin. An investigation into the interaction between feature selection and discretization: Learning how and when to read numbers. In M. Orgun and J. Thornton, editors, *AI 2007: Advances in Artificial Intelligence*, volume 4830 of *Lecture Notes in Computer Science*, pages 48–57. Springer Berlin / Heidelberg, 2007.

[101] K. Giaro, M. Kubale, and P. Obszarski. A graph coloring approach to scheduling of multiprocessor tasks on dedicated machines with availability constraints. *Discrete Applied Mathematics*, 157(17):3625–3630, 2009.

[102] C. Glass. Bag rationalisation for a food manufacturer. *Journal of the Operational Research Society*, 53(5):544–551, 2002.

[103] F. Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, Jul 1990.

[104] F. Glover, M. Parker, and J. Ryan. Coloring by tabu branch and bound. In Johnson and Trick [136], pages 285–307.

[105] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[106] C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, Feb. 2001.

[107] S. Gualandi and F. Malucelli. Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, 2011. Published online.

[108] A. Guerri and M. Milano. Learning Techniques for Automatic Algorithm Portfolio Selection. In R. L. de Mántaras and L. Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 475–479. IOS Press, 2004.

[109] H. Guo. *Algorithm selection for sorting and probabilistic inference: a machine learning-based approach*. PhD thesis, Kansas State University, Manhattan, KS, USA, 2003.

[110] H. Guo and W. H. Hsu. A learning-based algorithm selection meta-reasoner for the real-time mpe problem. In *Proceedings of the 17th Australian joint conference on Advances in Artificial Intelligence*, AI'04, pages 307–318, Berlin, Heidelberg, 2004. Springer-Verlag.

[111] H. Guo and W. H. Hsu. A machine learning approach to algorithm selection for np-hard optimization problems: a case study on the mpe problem. *Annals of Operations Research*, 156:61–82, 2007.

[112] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, Mar. 2003.

[113] P. Hage and F. Harary. Eccentricity and centrality in networks. *Social Networks*, 17(1):57 – 63, 1995.

[114] R. Halin. S-functions for graphs. *Journal of Geometry*, 8:171–186, 1976.

[115] M. A. Hall. *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, Hamilton, New Zealand, 1998.

[116] J.-P. Hamiez and J.-K. Hao. Scatter search for graph coloring. In *Selected Papers from the 5th European Conference on Artificial Evolution*, pages 168–179, London, UK, UK, 2002. Springer-Verlag.

[117] P. Hansen, M. Labbé, and D. Schindl. Set covering and packing formulations of graph coloring: Algorithms and first polyhedral results. *Discrete Optimization*, 6(2):135 – 147, 2009.

[118] J.-k. Hao and D. Porumbel. Recent advances in graph vertex coloring. In *Handbook of Optimization: From Classical to Modern Approach*, 2012.

[119] A. K. Hartmann and M. Weigt. Statistical mechanics of the vertex-cover problem. *Journal of Physics A: Mathematical and General*, 36(43):11069–11093, Oct 2003.

[120] D. Heckerman. A tutorial on learning with bayesian networks. Technical report, Learning in Graphical Models, 1996.

[121] S. Held, W. Cook, and E. C. Sewell. Safe lower bounds for graph coloring. In *Proceedings of the 15th international conference on Integer programming and combinatoral optimization*, IPCO'11, pages 261–273, Berlin, Heidelberg, 2011. Springer-Verlag.

[122] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, Dec. 1987.

[123] A. Hertz, M. Plumettaz, and N. Zufferey. Variable space search for graph coloring. *Discrete Applied Mathematics*, 156(13):2551–2560, July 2008.

[124] A. Hertz and M. Widmer. Guidelines for the Use of Meta-Heuristics in Combinatorial Optimization. *European Journal of Operational Research*, 151:247–252, 2003.

[125] T. K. Ho. Random decision forests. In *ICDAR*, pages 278–, 1995.

[126] T. K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, Aug. 1998.

[127] J. N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1:33–42, 1995.

[128] E. Horvitz and J. Breese. Ideal partition of resources for metareasoning. Technical report, Knowledge Systems Laboratory, Stanford University, February 1990.

[129] E. N. Houstis, A. C. Catlin, J. R. Rice, V. S. Verykios, N. Ramakrishnan, and C. E. Houstis. Pythia-ii: a knowledge/database system for managing performance data and recommending scientific software. *ACM Trans. Math. Softw.*, 26(2):227–253, June 2000.

[130] B. A. Huberman, R. M. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 27:51–53, 1997.

[131] G. H. John. *Enhancements to the data mining process*. PhD thesis, Stanford University, Stanford, CA, USA, 1997.

[132] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *MACHINE LEARNING: PROCEEDINGS OF THE ELEVENTH INTERNATIONAL*, pages 121–129. Morgan Kaufmann, 1994.

[133] D. J. Johnson and M. A. Trick, editors. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993*. American Mathematical Society, Boston, MA, USA, 1996.

[134] D. S. Johnson. A theoretician's guide to the experimental analysis of algorithms. *Data structures, near neighbor searches, and methodology: fifth and sixth dimacs implementation challenges*, 59:215–250, 1996.

[135] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, May 1991.

[136] D. S. Johnson and M. Trick, editors. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, Providence, RI, USA, Boston, MA, USA, 1996.

[137] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. Algorithm selection and scheduling. In J. H.-M. Lee, editor, *CP*, volume 6876 of *Lecture Notes in Computer Science*, pages 454–469. Springer, 2011.

[138] J. Kanda, A. Carvalho, E. Hruschka, and C. Soares. Selection of algorithms to solve traveling salesman problems using meta-learning. *Neural Networks*, 8(3):117–128, 2011.

[139] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[140] S. Kirkpatrick, D. C. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, May 1983.

[141] D. Kirovski and M. Potkonjak. Efficient coloring of a large spectrum of graphs. In *Proceedings of the 35th annual Design Automation Conference*, DAC '98, pages 427–432, New York, NY, USA, 1998. ACM.

[142] D. E. Knuth. Big omicron and big omega and big theta. *SIGACT News*, 8(2):18–24, Apr. 1976.

[143] D. E. Knuth. *The art of computer programming*. Number Bd. 3 in Addison-Wesley series in computer science and information processing. Addison-Wesley, 1981.

[144] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, Dec. 1997.

[145] I. Kononenko. On biases in estimating multi-valued attributes. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*, IJCAI'95, pages 1034–1040, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[146] L. Kotthoff. Algorithm selection for combinatorial search problems literature summary. `http://www.cs.st-andrews.ac.uk/~larsko/assurvey/`, 2012. [Online; accessed 03-October-2012].

[147] L. Kotthoff. *On Algorithm Selection, with an Application to Combinatorial Search Problems*. PhD thesis, University of St Andrews, 2012.

[148] L. Kotthoff, I. P. Gent, and I. Miguel. A preliminary evaluation of machine learning in algorithm selection for search problems. In D. Borrajo, M. Likhachev, and C. L. López, editors, *SOCS*. AAAI Press, 2011.

[149] F. Krzakala and J. Kurchan. Landscape analysis of constraint satisfaction problems. *Physical Review E*, 76(2), Aug. 2007.

[150] M. Kubale. *Graph Colorings*. Contemporary mathematics - American Mathematical Society. American Mathematical Society, 2004.

[151] M. Kuramochi and G. Karypis. Gene classification using expression profiles: A feasibility study. In *Proceedings of the 2nd IEEE International Symposium on Bioinformatics and Bioengineering*, BIBE '01, pages 191–, Washington, DC, USA, 2001. IEEE Computer Society.

[152] M. G. Lagoudakis and M. L. Littman. Algorithm selection using reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 511–518, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[153] M. G. Lagoudakis and M. L. Littman. Learning to Select Branching Rules in the DPLL Procedure for Satisfiability. *Electronic Notes in Discrete Mathematics*, 9:344–359, June 2001.

[154] M. G. Lagoudakis, M. L. Littman, and R. E. Parr. Selecting the right algorithm. In C. Gomes and T. Walsh, editors, *Proceedings of the 2001 AAAI Fall Symposium Series: Using Uncertainty within Computation, Cape Cod, MA, USA, November 2001*, 2001.

[155] M. Laguna and R. Martí. A grasp for coloring sparse graphs. *Computational Optimization and Applications*, 19(2):165–178, July 2001.

[156] P. Langley and S. Sage. Scaling to domains with irrelevant features. In R. Greiner, T. Petsche, and S. J. Hanson, editors, *Computational learning theory and natural learning systems: Volume IV*, pages 51–63. MIT Press, Cambridge, MA, USA, 1997.

[157] F. T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506, 1979.

[158] R. Leite, P. Brazdil, and J. Vanschoren. Selecting classification algorithms with active testing. In *Proceedings of the 8th international conference on Machine Learning and Data Mining in Pattern Recognition*, MLDM'12, pages 117–131, Berlin, Heidelberg, 2012. Springer-Verlag.

[159] D. Ler, I. Koprinska, and S. Chawla. A proposed meta-learning framework for algorithm selection utilising. Technical report, University of Sydney, School of Information Technologies, 2005.

[160] R. Lewis, J. Thompson, C. L. Mumford, and J. W. Gillard. A wide-ranging computational comparison of high-performance graph colouring algorithms. *Computers & Operations Research*, 39(9):1933–1950, Sept. 2012.

[161] H. Liu and R. Setiono. A probabilistic approach to feature selection - a filter solution. In *13th International Conference on Machine Learning*, pages 319–327, 1996.

[162] L. Lobjois and M. Lemaître. Branch and bound algorithm selection by performance prediction. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pages 353–358, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[163] S. Loudni. Intensification/diversification-driven ils for a graph coloring problem. In *Proceedings of the 12th European conference on Evolutionary Computation in Combinatorial Optimization*, EvoCOP'12, pages 160–171, Berlin, Heidelberg, 2012. Springer-Verlag.

[164] Z. Lü and J.-K. Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1):241 – 250, 2010.

[165] D. R. Luce and A. D. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14:95–116, 1949.

[166] C. Lucet, F. Mendes, and A. Moukrim. An exact method for graph coloring. *Computers & Operations Research*, 33(8):2189–2207, 2006.

[167] D. Mahjoub and D. W. Matula. Constructing efficient rotating backbones in wireless sensor networks using graph coloring. *Computer Communications*, 35(9):1086–1097, May 2012.

[168] E. Malaguti. *The Vertex Coloring Problem and its Generalizations*. PhD thesis, University of Bologna, may 2007.

[169] E. Malaguti, M. Monaci, and P. Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2):302–316, Apr. 2008.

[170] E. Malaguti and P. Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, pages 1–34, 2009.

[171] Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. Non-model-based algorithm portfolios for sat. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6695 LNCS:369–370, 2011.

[172] M. Maratea, L. Pulina, and F. Ricca. Applying machine learning techniques to asp solving. In A. Dovier and V. S. Costa, editors, *ICLP (Technical Communications)*, volume 17 of *LIPIcs*, pages 37–48. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[173] M. Maratea, L. Pulina, and F. Ricca. The multi-engine asp solver me-asp. In L. F. del Cerro, A. Herzig, and J. Mengin, editors, *JELIA*, volume 7519 of *Lecture Notes in Computer Science*, pages 484–487. Springer, 2012.

[174] S. McConnell. *Code Complete, Second Edition*, chapter 28. Microsoft Press, Redmond, WA, USA, 2004.

[175] A. Mehrotra and M. A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8:344–354, 1995.

[176] I. Méndez-Díaz and P. Zabala. A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154(5):826–847, Apr. 2006.

[177] I. Méndez-Díaz and P. Zabala. A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics*, 156(2):159–179, Jan. 2008.

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

[179] M. Mézard, M. Palassini, and O. Rivoire. Landscape of solutions in constraint satisfaction problems. *CoRR*, abs/cond-mat/0507451, 2005.

[180] K. Mizuno and S. Nishihara. Toward ordered generation of exceptionally hard instances for graph 3-colorability. In D. S. Johnson, A. Mehrotra, and M. A. Trick, editors, *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, pages 1–8, Ithaca, New York, USA, 2002.

[181] K. Mizuno and S. Nishihara. Constructive generation of very hard 3-colorability instances. *Discrete Applied Mathematics*, 156(2):218–229, Jan. 2008.

[182] M. Morak, N. Musliu, R. Pichler, S. Rümmele, and S. Woltran. Evaluating tree-decomposition based algorithms for answer set programming. In Y. Hamadi and M. Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 130–144. Springer Berlin Heidelberg, 2012.

[183] B. M. E. Moret. Towards a discipline of experimental algorithmics. *DIMACS Monographs in Discrete Mathematics and Theoretical Computer Science*, 2002.

[184] C. Morgenstern. Distributed coloration neighborhood search. In Johnson and Trick [136], pages 335–357.

[185] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms. Technical report, California Institute of Technology, 1989.

[186] N. Musliu. An iterative heuristic algorithm for tree decomposition. In C. Cotta and J. I. van Hemert, editors, *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, volume 153 of *Studies in Computational Intelligence*, pages 133–150. Springer, 2008.

[187] N. Musliu and W. Schafhauser. Genetic algorithms for generalised hypertree decompositions. *European Journal of Industrial Engineering*, 1(3):317–340, 2007.

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

[189] M. Nikolic, F. Maric, and P. Janicic. Simple algorithm portfolio for sat. *Artificial Intelligence Review*, pages 1–9, 2011. 10.1007/s10462-011-9290-2.

[190] E. Nudelman. *Empirical approach to the complexity of hard problems*. PhD thesis, Stanford University, Stanford, CA, USA, 2006.

[191] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *19th Irish Conference on AI*, 2008.

[192] A. Pahlavani and K. Eshghi. A hybrid algorithm of simulated annealing and tabu search for graph colouring problem. *International Journal of Operational Research*, 11(2):136, 2011.

[193] L. Paquete and T. Stützle. An experimental investigation of iterated local search for coloring graphs. In *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN*, pages 122–131, London, UK, UK, 2002. Springer-Verlag.

[194] P. M. Pardalos and J. Xue. The maximum clique problem. *Journal of Global Optimization*, 4:301–328, 1994.

[195] V. T. Paschos. Polynomial approximation and graph-coloring. *Computing*, 70:41–86, 2003. 10.1007/s00607-002-1468-7.

[196] B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 743–750. Morgan Kaufmann, 2000.

[197] W. J. M. Philipsen and L. Stok. Graph coloring using neural networks. In *Circuits and Systems, 1991., IEEE International Sympoisum on*, pages 1597 –1600 vol.3, jun 1991.

[198] E. Pitzer, M. Affenzeller, and A. Beham. A closer look down the basins of attraction. In *Computational Intelligence (UKCI), 2010 UK Workshop on*, pages 1 –6, sept. 2010.

[199] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in kernel methods*, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[200] M. Plumettaz, D. Schindl, and N. Zufferey. Ant local search and its efficient adaptation to graph colouring. *JORS*, 61(5):819–826, 2010.

[201] D. C. Porumbel, J.-K. Hao, and P. Kuntz. Diversity control and multi-parent recombination for evolutionary graph coloring algorithms. In *Proceedings of the 9th European Conference on Evolutionary Computation in Combinatorial Optimization*, EvoCOP '09, pages 121–132, Berlin, Heidelberg, 2009. Springer-Verlag.

[202] D. C. Porumbel, J.-K. Hao, and P. Kuntz. Position-guided tabu search algorithm for the graph coloring problem. In T. Stützle, editor, *Learning and Intelligent Optimization*, pages 148–162. Springer-Verlag, Berlin, Heidelberg, 2009.

[203] D. C. Porumbel, J.-K. Hao, and P. Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers and Operations Research*, 37(10):1822–1832, Oct. 2010.

[204] D. C. Porumbel, J.-K. Hao, and P. Kuntz. A search space "cartography" for guiding graph coloring heuristics. *Computers and Operations Research*, 37(4):769–778, Apr. 2010.

[205] P. Prakasam, M. Toulouse, G. T. Crainic, and R. Qu. Design of a multilevel cooperative heuristic for the graph coloring problem. In *LION 2009 (III)*, Lecture Notes in Computer Science. Springer, 2009.

[206] L. Pulina and A. Tacchella. A multi-engine solver for quantified boolean formulas. In *Proceedings of the 13th international conference on Principles and practice of constraint programming*, CP'07, pages 574–589, Berlin, Heidelberg, 2007. Springer-Verlag.

[207] L. Pulina and A. Tacchella. Aqme'10. *JSAT*, 7(2-3):65–70, 2010.

[208] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, Mar. 1986.

[209] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[210] N. J. Radcliffe and P. D. Surry. Fundamental Limitations on Search Algorithms: Evolutionary Computing in Perspective. In *Lecture Notes in Computer Science 1000*, volume 1000, pages 275–291, 1995.

[211] R. L. Rardin and R. Uzsoy. Experimental evaluation of heuristic optimization algorithms: A tutorial. *Journal of Heuristics*, 7(3):261–304, May 2001.

[212] I. Rebollo-Ruiz and M. G. Romay. Further results of gravitational swarm intelligence for graph coloring. In *NaBIC*, pages 183–188. IEEE, 2011.

[213] I. Rebollo-Ruiz and M. G. Romay. Gravitational swarm approach for graph coloring. In D. A. Pelta, N. Krasnogor, D. Dumitrescu, C. Chira, and R. I. Lung, editors, *NICSO*, volume 387 of *Studies in Computational Intelligence*, pages 159–168. Springer, 2011.

[214] J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[215] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2010.

[216] E. Salari and K. Eshghi. An aco algorithm for graph coloring problem. In *Computational Intelligence Methods and Applications, 2005 ICSC Congress on*, pages 1–5, 2005.

[217] H. Samulowitz and R. Memisevic. Learning to solve qbf. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 1*, AAAI'07, pages 255–260. AAAI Press, 2007.

[218] J. Silberholz and B. Golden. Comparison of metaheuristics. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 625–640. Springer US, 2010.

[219] B. Silverthorn and R. Miikkulainen. Latent Class Models for Algorithm Portfolio Methods. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[220] D. Singh, N. Mehta, and P. Purohit. Text based image recognition using multilayer perceptron. *Special issues on IP Multimedia Communications*, (1):143–146, oct 2011. Published by Foundation of Computer Science, New York, USA.

[221] S. N. Sivanandam, S. Sumathi, and T. Hamsapriya. A hybrid parallel genetic algorithm approach for graph coloring. *International Journal of Knowledge-based and Intelligent Engineering Systems*, 9(3):249–259, Aug. 2005.

[222] K. Smith-Miles and L. Lopes. Measuring instance difficulty for combinatorial optimization problems. *Computers & OR*, 39(5):875–889, 2012.

[223] K. Smith-Miles, J. I. van Hemert, and X. Y. Lim. Understanding tsp difficulty by learning from evolved instances. In Blum and Battiti [21], pages 266–280.

[224] K. A. Smith-Miles. Towards insightful algorithm selection for optimisation using meta-learning concepts. In *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 4118–4124. IEEE, june 2008.

[225] K. A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):6:1–6:25, Jan. 2009.

[226] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[227] S. B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. D. Jong, S. Dzeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, W. V. D. Welde, W. Wenzel, J. Wnek, and J. Zhang. The monk's problems a performance comparison of different learning algorithms. Technical report, Computer Science Department, Carnegie Mellon University, Pittsburgh, 1991.

[228] O. Titiloye and A. Crispin. Quantum annealing of the graph coloring problem. *Discrete Optimization*, 8(2):376–384, 2011.

[229] J. A. Torkestani and M. R. Meybodi. Graph coloring problem based on learning automata. In *Information Management and Engineering, 2009. ICIME '09. International Conference on*, pages 718 –722, april 2009.

[230] J. A. Torkestani and M. R. Meybodi. A new vertex coloring algorithm based on variable action-set learning automata. *Computing and Informatics*, 29(3):447–466, 2010.

[231] M. A. Trick and H. Yildiz. A large neighborhood search heuristic for graph coloring. In *Proceedings of the 4th international conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, CPAIOR '07, pages 346–360, Berlin, Heidelberg, 2007. Springer-Verlag.

[232] B. Uestuen, W. Melssen, and L. Buydens. Facilitating the application of Support Vector Regression by using a universal Pearson VII function based kernel. *Chemometrics and Intelligent Laboratory Systems*, 81:29–40, 2006.

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

[234] V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[235] V. Vasilikos and M. G. Lagoudakis. Optimization of heuristic search using recursive algorithm selection and reinforcement learning. *Annals of Mathematics and Artificial Intelligence*, 60:119–151, 2010. 10.1007/s10472-010-9217-7.

[236] M. N. Velev. Exploiting hierarchy and structure to efficiently solve graph coloring as sat. In *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, ICCAD '07, pages 135–142, Piscataway, NJ, USA, 2007. IEEE Press.

[237] R. Venkatesan and L. Levin. Random instances of a graph coloring problem are hard. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 217–222, New York, NY, USA, 1988. ACM.

[238] C. Voudouris and E. Tsang. Guided local search. Technical report, European Journal of Operational Research, 1995.

[239] C. C. Wang. An algorithm for the chromatic number of a graph. *J. ACM*, 21(3):385–391, July 1974.

[240] D. J. Watts and S. H. Strogatz. Collective dynamics of /'small-world/' networks. *Nature*, 393(6684):440–442, June 1998.

[241] D. J. A. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 1967.

[242] I. Witten and E. Frank. *Data mining: practical machine learning tools and techniques with Java implementations*. The Morgan Kaufmann series in data management systems. Morgan Kaufmann, 2000.

[243] D. H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, Oct. 1996.

[244] D. H. Wolpert and W. G. Macready. No free lunch theorems for search. *JPL*, pages 1–38, 1996.

[245] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, 1(1):67–82, 1997.

[246] C. W. Wu. Graph coloring via synchronization of coupled oscillators. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 45(9):974 –978, sep 1998.

[247] H. Wu and P. v. Beek. On portfolios for backtracking search in the presence of deadlines. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence - Volume 01*, ICTAI '07, pages 231–238, Washington, DC, USA, 2007. IEEE Computer Society.

[248] Q. Wu and J.-K. Hao. Coloring large graphs based on independent set extraction. *Computers and Operations Research*, 39(2):283–290, Feb. 2012.

[249] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, Dec. 2007.

[250] X.-F. Xie and J. Liu. Graph coloring by multiagent fusion search. *Journal of Combinatorial Optimization*, 18(2):99–123, Aug. 2009.

[251] L. Xu, H. Hoos, and K. Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In M. Fox and D. Poole, editors, *AAAI*. AAAI Press, 2010.

[252] L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown. Evaluating component solver contributions to portfolio-based algorithm selectors. In *Proceedings of the 15th international conference on Theory and Applications of Satisfiability Testing*, SAT'12, pages 228–241, Berlin, Heidelberg, 2012. Springer-Verlag.

[253] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32:565–606, jun 2008.

[254] L. Xu, F. Hutter, J. Shen, H. Hoos, and K. Leyton-Brown. SATzilla2012: Improved algorithm selection based on cost-sensitive classification models. Solver description, SAT Challenge 2012, 2012.

[255] S. R. Yadav, R. R. M. R. Muddada, M. K. Tiwari, and R. Shankar. An algorithm portfolio based solution methodology to solve a supply chain optimization problem. *Expert Systems with Applications*, 36(4):8407–8420, May 2009.

[256] Y. Yang and G. Webb. On why discretization works for naive-bayes classifiers. In T. Gedeon and L. Fung, editors, *AI 2003: Advances in Artificial Intelligence*, volume 2903 of *Lecture Notes in Computer Science*, pages 440–452. Springer Berlin / Heidelberg, 2003. 10.1007/978-3-540-24581-0_37.

[257] C.-W. Yeh and K.-R. Wu. A novel dna-based parallel computation for solving graph coloring problems. In *Software Engineering, 2009. WCSE '09. WRI World Congress on*, volume 2, pages 213 –217, may 2009.

[258] G. Zäpfel, R. Braune, and M. Bögl. *Metaheuristic Search Concepts: A Tutorial with Applications to Production and Logistics*. Springer, 2010.

[259] E. Zemel. Measuring the Quality of Approximate Solutions to Zero-One Programming Problems. *Mathematics of Operations Research*, 6:319–332, 1981.

[260] M. Zlochin and M. Dorigo. Model-based search for combinatorial optimization: A comparative study. In *Parallel Problem Solving from Nature - PPSN VII: 7th International Conference*, volume 2439 of *Lecture Notes in Computer Science*, pages 651–661, Granada, Spain, Sept. 2002. Springer Berlin / Heidelberg.

[261] N. Zufferey, P. Amstutz, and P. Giaccari. Graph colouring approaches for a satellite range scheduling problem. *Journal of Scheduling*, 11(4):263–277, Aug. 2008.