Algorithm Selection for the Graph Coloring Problem

Nysret Musliu and Martin Schwengerer

Technische Universität Wien,
Institut für Informationssysteme 184/2,
Favoritenstraße 9-11, A-1040 Vienna, Austria
musliu@dbai.tuwien.ac.at mschweng@kr.tuwien.ac.at

Abstract. We present an automated algorithm selection method based on machine learning for the graph coloring problem (GCP). For this purpose, we identify 78 features for this problem and evaluate the performance of six state-of-the-art (meta)heuristics for the GCP. We use the obtained data to train several classification algorithms that are applied to predict on a new instance the algorithm with the highest expected performance. To achieve better performance for the machine learning algorithms, we investigate the impact of parameters, and evaluate different data discretization and feature selection methods. Finally, we evaluate our approach, which exploits the existing GCP techniques and the automated algorithm selection, and compare it with existing heuristic algorithms. Experimental results show that the GCP solver based on machine learning outperforms previous methods on benchmark instances.

Keywords: Algorithm Selection, Graph Coloring, Machine Learning

1 Introduction

Many heuristic algorithms have been developed to solve combinatorial optimization problems. Usually, such techniques show different behavior when solving particular instances. According to the *no free lunch* theorems [45], no algorithm can dominate all other techniques on each problem. In practice, this raises new issues, as selecting the best (or most appropriate) solver for a particular instance may be challenging. Often, the "winner-take-all" strategy is applied and the algorithm with the best average performance is chosen to solve all instances. However, this methodology has its drawbacks, because the distribution of tested instances effects the average performance, and usually in practice only a special class of instances are solved.

One possible approach to obtain better solutions on average is to select for each particular instance the algorithm with the highest expected performance. This task is known as algorithm selection (AS) and one emerging and very promising approach that is used for AS is based on *machine learning* methods. These techniques are able to learn a model based on previous observations an then predict on a new and unseen instance the best algorithm.

In this paper, we address AS using *classification* algorithms for the well-known Graph Coloring Problem (GCP). The GCP is a classical NP-hard problem in computer science. The task for this problem is to assign a color to each node of a given graph such that (a) no adjacent nodes received the same color and (b) the number of colors used is minimized. Various heuristic algorithms to solve GCP have been developed in the literature. However, recent studies [7,26] show that the performance of different heuristics highly depend on attributes of the graph like for example the density or the size. Therefore, the aim of this paper is to apply automated algorithm selection for this problem. We evaluate experimentally different heuristics and classification algorithms and show that our solver that includes algorithm selection is able to achieve much better performance than the underlying heuristics.

The rest of this paper is organized as follows: Section 2 gives a short introduction into the GCP, AS and the related work. In Section 3, we present features of a GCP instance and describe our AS approach for the GCP. The experimental results are given in Section 4 while Section 5 concludes our work and describes the future work.

2   Background and Related Work

2.1   The Graph Coloring Problem

Given a graph $G = (V, E)$, a *coloring* of G is an assignment of a color $c \leq k$ to each vertex $v \in V$ such that no vertices sharing an edge $e \in E$ receive the same color. The Graph Coloring Problem (GCP) deals with finding a coloring for G whereby it can occur as decision problem (also known as *k-coloring problem*), where the number of colors k is fixed, or as optimization problem (the *chromatic number problem*), where k has to be minimized. Instances of the k-coloring problem are, unlike other *NP-complete* problems (e.g. the *Hamilton path problem*), "hard on average" [43], meaning that also random instances tend to be difficult to solve. Moreover, approximating the chromatic number itself is very hard [14], although many different approaches for this task exist (see [36] for more details). Graph coloring itself has many applications like scheduling [25,48], register allocation [5], circuit testing [17] etc.

There exist many exact methods for solving the GCP (see [29] for more details). However, all these approaches are only usable in general on small graphs up to 100 vertices [7]. Consequently, many solvers apply heuristic algorithms. Early approaches in this context are greedy constructive heuristics (e.g. `DSATUR` [3] or `RLF` [25]) while recent algorithms use more sophisticated techniques. Especially, local search methods like tabu search [21,1] provide good results. Moreover, also several population-based and hybrid algorithms have been proposed [16,28,46]. For a survey on different heuristics, we refer to [35], [29] and [26].

2.2   Algorithm Selection

The *Algorithm Selection Problem* postulated by Rice [37] deals with this question: Given different algorithms to solve a problem, which one should be selected

for a particular instance? For this purpose, Rice identified four important components, namely

- the set of candidate algorithms A,
- the instances of a problem, the problem space P,
- measurable attributes of an instance, denoted as feature space F, and
- the performance space Y.

For solving this problem, it is necessary to use relevant features $f(x) \in F$ of an instance $x \in P$ that model the performance of an algorithm $a \in A$ with respect to a performance criteria Y. For a concrete application of algorithm selection, the problem space P and the performance space Y are usually given. Designing the algorithm portfolio A is also usually not so hard, because in most cases the available algorithms are limited and a good selection procedure will not use suboptimal solvers anyway. More challenging is the choice of appropriate features F and to find a good selection procedure (denoted as S).

Unfortunately, there exist no automatic way to find good features [34], as this requires usually deep domain knowledge and analytical skills. Nevertheless, some approaches seem to be useful across different problems and sometimes, even features of related problems can be reused. Regarding concrete features for the GCP, in [40] various properties of a graph that may be useful are introduced. We also adapted some other features that can be found in [47] and additionally introduced some new features.

Regarding the selection procedure, there exist different methods that for example use analytical aspects or complexity parameters [12]. One successfully and widely used solution is the application of machine learning techniques. Usually, either *classification* or *regression* techniques are used. Classification techniques classify the new instances into one category, which is the recommended algorithm. In contrast to this, regression techniques model the behavior of each algorithm by using an regression function and predict the result (e.g. runtime, solution quality) on a new instance. Based on this prediction the algorithm with the best performance is selected. Both paradigms have been successfully applied for algorithm selection. Applications of regression include [47,4,31]. Classification techniques have been used among others in [18,19,39,41,23,30]. However, none of these approaches is specially designed for the GCP and although some also consider graph properties, only one paper, [42], deals explicitly with graph coloring. This work, which was published just recently, utilizes 16 different features and a decision tree to chose between two algorithms for graph coloring: DSATUR and tabu search. Although this approach shows some similarities concerning features and algorithms, there are several differences to our work. We consider additional features (including the greedy algorithm DSATUR) and evaluate the performance of several classifiers. In addition, we focus on using multiple and more sophisticated heuristics and investigate the effect of data preparation. In this paper we present new attributes of a graph that can be calculated in polynomial time and are suitable to predict the most appropriate heuristic for GCP.

3   Algorithm Selection for the GCP

First step in algorithm selection is to identify characteristic features that can be calculated in reasonable time. Furthermore, we collect performance information about each algorithm on a representative set of benchmark instances and determine for each graph the most suited algorithm. Then, we use machine learning to train classification algorithms that act as selection procedure. To predict the best algorithm on a new instance, the proposed system extracts the features of that instance and then determines the corresponding class, which corresponds to the most appropriate algorithm.

3.1   Instance features

We identify 78 features that are grouped in eight categories: graph size, node degree, maximal clique, clustering coefficient, local search probing features, greedy coloring, tree decomposition, and lower- and upper bound. Figure 1 gives a more detailed view on the different attributes. The first two groups, *graph size*

Graph Size Features:
1: **no. of nodes:** n
2: **no. of edges:** m
3,4: **ratio:** $\frac{n}{m}, \frac{m}{n}$
5: **density:** $\frac{2 \cdot m}{n \cdot (n-1)}$

Node Degree:
6-13: **nodes degree statistics:** min, max, mean, median, $Q_{0.25}$, $Q_{0.75}$, variation coefficient, entropy

Maximal Clique:
14-20: **normalized by** n: min, max, median, $Q_{0.25}$, $Q_{0.75}$, variation coefficient, entropy
21:    **computation time**
22:    **maximum cardinality**

Clustering Coefficient
23: **global clustering coefficient** [27]
24-31: **local clustering coefficient:** min, max, mean, median, $Q_{0.25}$, $Q_{0.75}$, variation coefficient, entropy
32-39: **weighted local clustering coefficient:** min, max, mean, median, $Q_{0.25}$, $Q_{0.75}$, variation coefficient, entropy
40: **computation time**

Local Search Probing Features:
41, 42: **avg. impr.:** per iteration, per run
43: **avg no. iterations to LO**[a] per a run
44, 45: **no. conflict nodes:** at LO, at end
46, 47: **no. conflict edges:** at LO, at end
48: **no. LO found**
49: **computation time**

Greedy Coloring:
50,51: **no. colors needed:** k_{DSAT}, k_{RLF}
52, 53: **computation time:** t_{DSAT}, t_{RLF}
54, 55: **ratio:** $\frac{k_{DSAT}}{k_{RLF}}, \frac{k_{RLF}}{k_{RLF}}$
56: **best coloring:** $\min(k_{DSAT}, k_{RLF})$
57-72: **independent-set size:** min, max, mean, median, $Q_{0.25}$, $Q_{0.75}$, variation coefficient, entropy

Tree Decomposition:
73: **width of decomposition**
74: **computation time**

Lower- and Upper Bound:
75, 76: **distance:** $\frac{(B_l - B_u)}{B_l}, \frac{(B_u - B_l)}{B_u}$
77, 78: **ratio:** $\frac{B_l}{B_u}, \frac{B_u}{B_l}$

---
[a] local optima

Fig. 1. Basic features for an instance of the GCP.

and *node degree* contain classical features that are also used in other systems (e.g [47]). For the *maximal clique* features, we calculate for each node a *maximal clique* by using a simple greedy algorithm and take statistical information about the size of these cliques as relevant attributes. Regarding the local clustering coefficient [44] of a node, we use, besides the classical value, a modified version

denoted as *weighted clustering coefficient*, where the coefficient of the node is multiplied with its degree. The *local search probing features* are extracted from 10 executions of a simple 1-opt best-improvement local search on the *k-coloring* problem. The *greedy coloring* attributes are based on the application of `DSATUR` and `RLF`. For these features, we take, besides the number of used colors, also the sizes of the independent sets into account and calculate statistical information like the average size or the variation coefficient. Furthermore, we consider attributes of a *tree decomposition* obtained by a *minimum-degree* heuristic. Such features have been used successfully by [32] for AS in Answer Set Programming. The last category builds on a lower bound of k, denoted as B_l, which is the cardinality of the greatest maximal clique found, and an upper bound B_u, which is the minimum number of colors needed by the two greedy algorithms. Apart from features we described above, we also take the computation times of some feature classes as additional parameters. Note that we also experimented with attributes based on the *betweenness centrality* [15] and the *eccentricity* [20] of the nodes. Unfortunately, the algorithms we implemented to calculate these features required during our tests much time, for which reasons we did not use them in our approach.

It is widely accepted that the performance of learning algorithms depend on the choice of features, and that using irrelevant features may lead to suboptimal results. Therefore, we apply a *feature subset selection* using a *forward selection* with limited backtracking and a *genetic search* to reduce the set of basic features. Both techniques are applied with the `CfsSubsetEval` criteria as evaluation function. Only features that are selected by one of these methods are used further. Additionally, for each pair of features $x_j, x_k, k > j$ we create two new features that represent the product $x_j \cdot x_k$ and the quotient x_j / x_k, respectively. This idea is based on a similar technique used in [47], where also the product of two features is included as an additional attribute. Finally, we apply feature selection on these expanded attributes to eliminate unnecessary attributes. In the end, we obtain 90 features, including 8 basic features and 82 composed attributes.

3.2   Algorithm Portfolio

To demonstrate our approach, we use six state-of-the-art (meta)heuristics for the GCP, namely: `HEA` [16], `ILS` [8], `MAFS` [46], `MMT` [28] (only the component containing the genetic algorithm), `FOO-PARTIALCOL` [1] (further abbreviated to `FPC`), and `TABUCOL` [21] (further denoted as `TABU`).

For each of these algorithms, we use parameter settings proposed in the original publications and that are suggested by their developers. The main reason for selecting the `TABU` solver is the fact that this technique is one of the most-studied heuristics and is often used as local search in various population-based algorithms for the GCP. In addition, according to a comparison by Chiarandini [6], `TABU` is besides `HEA` and `ILS` the most effective algorithm for random 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. 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.

3.3 Benchmark Instances

As training instances, we take three different publicly available sets: The first set, further denoted as `dimacs`, consists of 174 graphs from the *Graph Coloring and its Generalizations*-series (COLOR02/03/04) [1] which builds up on the well-established Dimacs Challenge [22]. This set includes instances from the *coloring* and the *clique* part of the Dimacs Challenge. The second and third set of instances, denoted as `chi500` and `chi1000`, are used by a comparative study [9] of several heuristics for the GCP and contain 520 instances with 500 nodes and 740 instances with 1000 nodes respectively[2]. These instances are created using Culberson's [10] random instance generator by controlling various parameters like the *edge density* ($p = \{0.1, 0.5, 0.9\}$) or the *edge distribution* (resulting in three groups of graphs: *uniform graphs* (G), *geometric graphs* (U) and *weight biased graphs* (W)).

For the final evaluation of our algorithm selection approach with the underlying algorithms, we use a `test set` comprising complete new instances of different size, density and type, generated with Culberson's instance generator. We constructed *uniform* (G), *geometric* (U) and *weight biased* (W) graphs of different sizes $n = \{500, 750, 1000, 1250\}$ and density values $p = \{0.1, 0.5, 0.9\}$. For each parameter setting we created 5 graphs, leading to a total of 180 instances.

In order to ensure practicable results and prevent excessive computational effort, we use a maximal time limit per color $t_{max} = \min(3600, \sqrt{|E|} \cdot x)$ 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 are obtained experimentally. In this context, we want to note that the average time needed for the best solution on the *hard* instances is only 21.58% of the allowed value t_{max} and 90% of the best solutions are found within 62.66% of t_{max}.

Regarding the feature computation, we do not use any time limitations except for the local search probing, although this might be reasonable for practical implementations. However, for our test data the median calculation time is 2 s, the 95th percentile is 18 s and the 99th percentile is 53 s.

In total, we collected 1434 graphs of variable size and density as training data. We removed instances where an optimal solution has been found by one of the two greedy algorithms or where the heuristics did not find better colorings than obtained by the greedy algorithms. We further excluded all instances where at least four heuristics (more than 50%) yield the best solution in less than five seconds. These seem to be *easy* instances which can be solved efficiently by most

---

[1] available at `http://mat.gsia.cmu.edu/COLOR04/`, last visited on 22.10.2012
[2] available at `www.imada.sdu.dk/~marco/gcp-study/`, last visited on 28.10.2012

heuristics. Therefore, they are less interesting for algorithm selection. In the end, our training data consist of 859 *hard* instances.

Note that during our experiments, we discovered instances where several heuristics obtain best result. For machine learning, this is rather uncomfortable, as the training data should contain only one recommended algorithm per instance. One solution for this issue is using multi-labeled classification [23]. However, we follow a different strategy where we prioritize the algorithms according to their average rank on all instances. Thus, in case of a tie, we prefer the algorithm with the lower rank. Concerning the performance evaluation of the classifiers, we have to take into account that there might be several "best" algorithms. For that reason, we introduce a new performance measurement, called *success rate* (*sr*), that is defined as follows: Given for each instance $i \in I$ a set of algorithms B^i that obtains best result on i. Then, the *sr* of a classifier c on a set of instances I is $sr = \frac{|\{i \in I : c(i) \in B^i\}|}{|I|}$ where $c(i)$ is the predicted algorithm for the instance i. Furthermore, the success rate of a solver is the ratio between the number of instances for which the solver achieves the best solution and the total number of instances.

3.4 Classification Algorithms

For the selection procedure itself, we test six popular classification algorithms: Bayesian Networks (BN), C4.5 Decision Trees (DT), k-Nearest Neighbor (kNN), Multilayer Perceptrons (MLP), Random Forests (RF), and Support-Vector Machines (SVM). For all these techniques, we use the implementation included in the *Weka* software collection [2], version *3.6.6*. Furthermore, we manually identify important parameters of these learning algorithms and experimented with different settings. We refer the reader to [38] for more details regarding different parameter settings that we used for classification algorithms.

3.5 Data Discretization

Apart from selection of relevant features, a different, but also important issue is whether to use the original numeric attributes or to apply a *discretization* step to transform the values into nominal attributes. Besides the fact that some classification algorithms can not deal with numeric features, research has clearly shown that some classifiers achieve significant better results when applied with discretized variables [11]. In this work, we experimented with two different supervised discretization techniques. The first one is the classical minimum-descriptive length (MDL) method [13], while the second method is a derivation of MDL using a different criteria [24] (further denoted as Kononenko's criteria (KON)).

4 Experimental Results

All our experiments have been performed on 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.

Concerning the heuristic for the GCP, we execute each algorithm $n = 10$ times ($n = 20$ for the `dimacs` instances) using different random seeds. The result of each algorithm is the lowest number of colors that has been found in more than 50% of the trials. Furthermore, we take the median time needed within the n executions as required computation time. In cases of a timeout, we take t_{max} as computation time. Detailed results of the experiments can be found at `http://www.kr.tuwien.ac.at/staff/mschweng/gcp/`.

4.1 Parameter Configuration and Discretization

Regarding the effect of data discretization, we compare the success rate of the best parameter configuration for each of the three methods on several data sets (e.g. using different feature subsets). The experimental results clearly show that most classifiers achieve a higher accuracy on data with nominal attributes.

Table 1 gives an overview regarding the impact of discretization. The column *avg* shows the improvement regarding the average success rate, whereas the column *best* represents the gap between the best value obtained using numerical values and the best value achieved with the discretized data sets. Both

Method	BN		C4.5		kNN	
	avg	best	avg	best	avg	best
MDL	+2.40	+2.30	+6.34	+7.15	+9.41	+7.00
KON	+4.93	+4.85	+5.78	+6.23	+11.09	+8.92
	MLP		RF		SVM	
MDL	+4.16	+5.42	+2.25	+2.25	+4.57	+1.75
KON	−20.33	+4.37	+3.94	+4.38	+5.33	+4.20

Table 1. Improvements of *sr* (in percent) when using discretized data in relation to results achieved with non-discretized data on the training set using cross validation.

discretization variants improve the best reached success rate. The classical MDL method improves *sr* on average by 5.14%, while Kononenko's criteria by 4.35%. However, for some classifiers, the benefits of discretized values are up to +9.41% with MDL and even +11.09% using KON. The only classifier which does not benefit from a discretization is MLP. Its training time increases dramatically (up to several hours). Even more, when using KON, the average success rate decreases by 20.33%. Nevertheless, as KON provides for the most classifiers slightly better results than MDL, we decided to use Kononenko's criteria for all further experiments.

As mentioned before, we experimented with different parameter configurations for each classifier. Based on these tests, we selected for the remaining tests the most successful configuration. In detail, the maximum number of parent nodes that we used for BN is 3. For the DT the minimum number of objects per leave was set to 3. Regarding the kNN, the size of the neighborhood is set to 5

and for the RF, we set the number of trees to 15. For the MLP and SVM, and other remaining parameters, we used the default settings from the *Weka* system.

4.2 Results on the Training Data

To show the performance on the training data set, we tested each classifier 20 times using a 10-fold cross validation. The results of these experiments are given in Figure 2, which shows the average number of correct predictions for each classifier and instance set. The figure also gives a comparison with the existing solvers for the GCP regarding the number of instances on which the best solution is achieved. The diagram shows that 5 of 6 tested classifiers achieve good results.
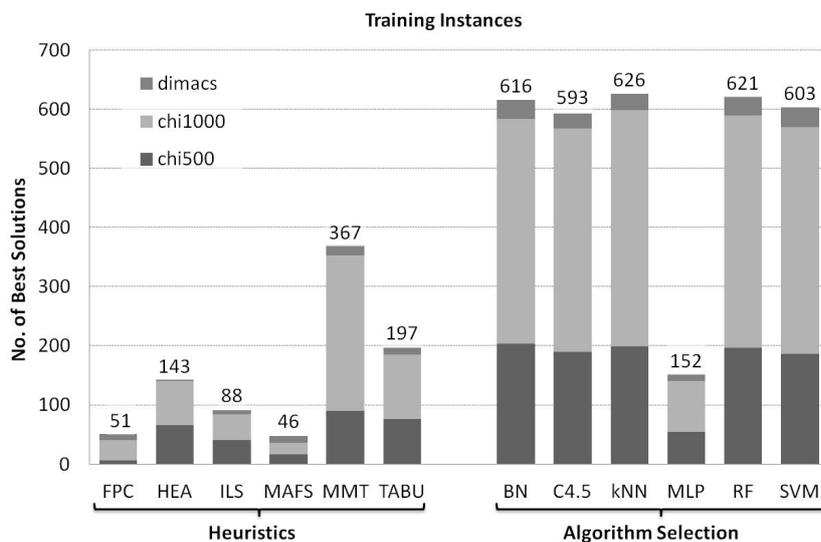

Fig. 2. Prediction of the best algorithm by different classifiers on the training data and their comparison with the existing (meta)heuristics.

Only the MLP gives very weak results. This method requires more than 24 hours for one run of cross-validation and its results are even below those of the existing heuristics. Nevertheless, other approaches show very good performance by obtaining on up to 625.9 (72.86%) instances the best solution. Compared with `MMT`, which is the best heuristic for the GCP, an improvement on 259 instances (30.15%) is reached. Even more, this performance increase can be observed on all three instance sets.

For a more detailed statistical analysis, we applied a *corrected resampled T-test* [33] on the results of the 10-fold cross-validation (except the MLP). These experiments, applied with a level of significance of $\alpha = 0.05$, reveal that BN, kNN and RF are significant better than DT while all other pairwise comparisons do not show significant differences.

4.3   Evaluation on the Test Set

In the next step, we trained the classifiers with the complete training set and evaluate the performance of them on the test set. The corresponding results are shown in Figure 3, which shows the number of instances on which the solvers show the best performance. From this figure, we can see that all learning strategies except MLP accomplish a higher number of best solutions than any existing solver for the GCP. The most successful classifiers are RF, BN and kNN which predict on up to 71.71% of the 152 graphs the most appropriate algorithm.
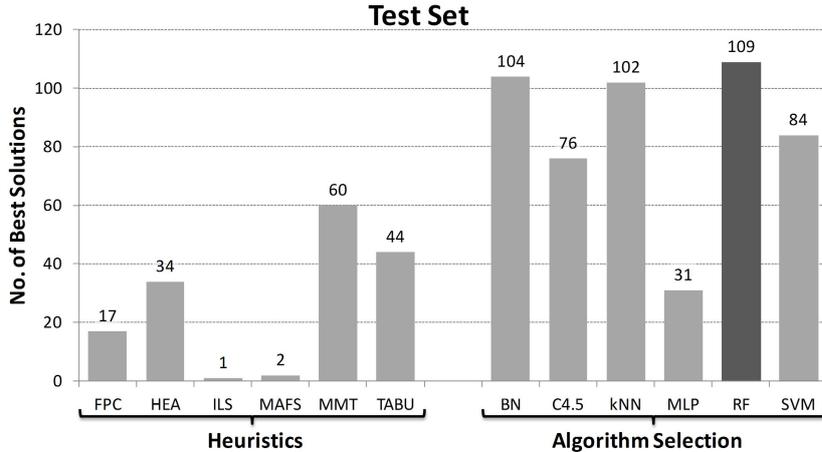

Fig. 3. Number of instances from the `test set` on which a solver shows best performance.

A more detailed view on the results using different metrics is given in Table 2. Besides the success rate, we also consider the distance to the best known solution, $err(\widehat{\chi}, G)$ [6], and the average rank. The figures point out that `MMT` is the best single heuristic with respect to the number of best solutions. Moreover, it accomplishes the lowest average distance $err(\widehat{\chi}, G)$ 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. Thus, it seems that although `MMT` obtains often solution with a low number of colors (resulting in a low $err(\widehat{\chi}, G)$), it is not always ranked first. One possible explanation for this is that `MMT` is a method which is powerful, but rather slow. Consequently, on instances where other heuristics (e.g. `TABU` or `HEA`) find equal colorings, `MMT` requires more computation time and is therefore, ranked behind its competitors.

Compared with our solver that applies all algorithms and an automated algorithm selection mechanism, we can see that for all considered metrics except $err(\widehat{\chi}, G)$ at least one system shows a stronger performance than the best single heuristic. The best selection mechanism provides clearly RF, which is on all

Solver	No. Best Solution	sr (%)	$err(\widehat{\chi}, G)$ (%)	Rank avg	σ
	Heuristics (H)				
FPC	17	11.18	25.42	3.29	1.42
HEA	34	22.37	14.91	2.66	1.38
ILS	1	0.66	21.73	3.82	1.36
MAFS	2	1.32	30.17	4.62	1.52
MMT	60	39.47	**3.78**	2.76	1.84
TABU	44	28.95	19.23	2.58	1.29
	Algorithm Selection (AS)				
BN	104	68.42	5.16	1.59	1.08
C4.5	76	50.00	5.86	2.21	1.50
kNN	102	67.11	3.82	1.52	0.91
MLP	31	20.39	24.90	3.14	1.66
RF	**109**	**71.71**	5.44	**1.41**	**0.78**
SVM	84	55.26	8.32	1.97	1.38
Best (H)	60	39.47	**3.78**	2.58	1.29
Best (AS)	**109**	**71.71**	3.82	**1.41**	**0.78**

Table 2. Performance metrics of the algorithm selection and the underlying heuristics on the *test set*.

criteria except $err(\widehat{\chi}, G)$ better than the other classifiers. In detail, this system achieves a success rate of 71.71% (+32.24% compared with MMT) and an average rank of 1.41 (−1.17 compared with TABU). Only on the metric $err(\widehat{\chi}, G)$, MMT shows with 3.78% a lower value than RF, which colorings of the choosen algorithms have an average distance of 5.44% to the best known solution. The best classifier in this context is kNN, which achieves with 3.82 a slightly higher value than MMT. The worst performance among the classifiers shows clearly MLP, which results concerning the number of instances where it finds the best solution are even below those of MMT or TABU. These data confirm that this machine learning technique in combination with KON is not suited for the GCP. This does not imply that MLP is in general inappropriate for AS. The results using data sets with continuous attributes show that this classifier can achieve competitive results compared to the other tested classifiers. However, when using nominal features, its training time usually increases dramatically while the accuracy decreases.

For a more detailed analysis, we group the graphs according to their density and graph class and compare the performance of RF, which is the best classifier, with the existing heuristics. Figure 4 presents the number of graphs on which the different methods show the best performance. The figures show that our solver based on algorithm selection is on 5 of the 9 subsets better or equal compare to the best heuristic. On the group U-0.9 the algorithm selection fails by predicting on only 6 of 10 graphs the correct algorithm. The reason for this bad results might be that for almost all graphs with high density, either TABU or MMT is the
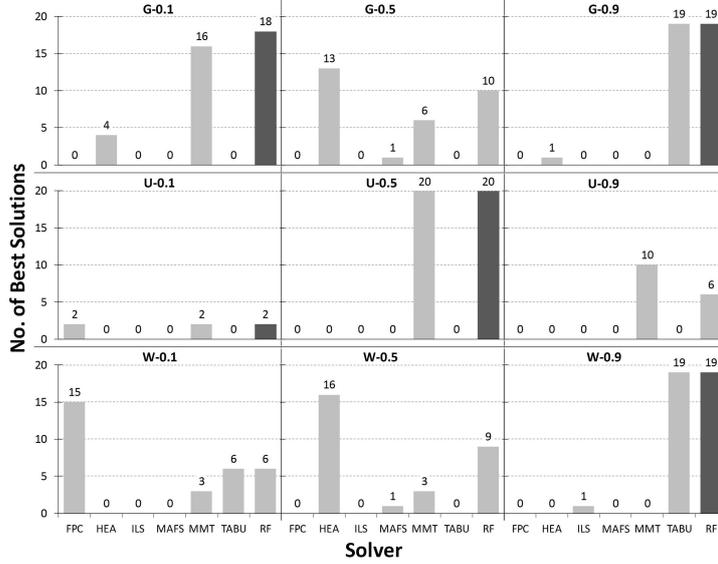
Fig. 4. Number of instances of the test set on which a solver shows the best performance, grouped by the graph type (edge distribution) and the density. The dark bar denotes that our approach is at least as successful as the best single solver.

best heuristic whereby the decision criteria is based on the *edge distribution*. In detail, TABU is to prefer on *uniform* and *weight biased* graphs while MMT is better on *geometric* graphs. Unfortunately this information is not part of the feature space and it seems that the classifier is not able to distinguish this based on other features, which leads to mispredictions. The suboptimal prediction rate on W-0.1 is hard to explain, as FPC is also in the related subset W-0.1 of the training data the best algorithm. Thus, is seems that the classifier is just not able to learn this pattern correctly. On the groups G-0.5 and W-0.5 our approach is also not able to achieve competitive results compare to the best single solver. This is surprising as the best heuristic on these instances is HEA, which shows also on the corresponding training data good results. One possible explanation is that, as for instances with high density, the classifier is unable to detect the type of edge distribution. Nevertheless, we can see that in many cases, the classifier is able to predict the most appropriate algorithm, which leads to a better average performance compare to any single heuristic.

5   Conclusion

In this paper, we presented a novel approach based on machine learning to automate algorithm selection for the GCP. Given a set of algorithms and a set of specific features of a particular instance, such a system selects the algorithm which is predicted to show the best performance on that instance. Our proposed

approach applies a classification algorithm as selection procedure that assigns a new instance to one of the available algorithms based on a previously learned model. For this purpose, we identified 78 attributes for the GCP that can be calculated in reasonable time and that have impact on solving of the GCP.

To demonstrate our approach, we evaluated the performance of six state-of-the-art (meta)heuristics on three publicly available sets of instances and showed that no algorithm is dominant on all instances. We further applied machine learning to build an automated selection procedure based on the obtained data. For that purpose, we experimented with six well known classification algorithms that are used to predict on a new instance the most appropriate algorithm. Our experiments clearly showed that a solver that applies machine learning yield a significant better performance compared with any single heuristic. We further demonstrated that using data discretization increases the accuracy of most classifiers.

Regarding future work, we plan to investigate 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. Also worth considering is a hybridization of our method with *automated parameter selection* and the combination of heuristic and exact techniques for the GCP in a system that applies automated algorithm selection.

References

1. I. Blöchliger and N. Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operations Research*, 35(3):960–975, 2008.
2. R. R. Bouckaert, E. Frank, M. Hall, R. Kirkby, P. Reutemann, A. Seewald, and D. Scuse. *Weka manual (3.6.6)*, Oct. 2011.
3. D. Brélaz. New methods to color the vertices of a graph. *Commun. ACM*, 22:251–256, Apr. 1979.
4. 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.
5. G. Chaitin. Register allocation and spilling via graph coloring. *SIGPLAN Not.*, 39(4):66–74, Apr. 2004.
6. M. Chiarandini. *Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems*. PhD thesis, TU Darmstadt, Aug. 2005.
7. 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*. Chapman & Hall/CRC, 2007.

8. 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*, 2002.

9. M. Chiarandini and T. Stützle. An analysis of heuristics for vertex colouring. In *Experimental Algorithms*. Springer Berlin / Heidelberg, 2010.

10. 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.

11. 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.

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

13. U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In R. Bajcsy, editor, *IJCIA*. Morgan Kaufmann, 1993.

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

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

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

17. M. R. Garey, D. S. Johnson, and S. C. Hing. An application of graph coloring to printed circuit testing. *IEEE Transactions on Circuits and Systems*, 1976.

18. A. Guerri and M. Milano. Learning techniques for automatic algorithm portfolio selection. In R. L. de Mántaras and L. Saitta, editors, *Conference on Artificial Intelligence, ECAI'2004*, pages 475–479. IOS Press, 2004.

19. 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.

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

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

22. D. J. Johnson and M. A. Trick, editors. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993*. American Mathematical Society, 1996.

23. J. Kanda, A. Carvalho, E. Hruschka, and C. Soares. Selection of algorithms to solve traveling salesman problems using meta-learning. *Neural Networks*, 8(3), 2011.

24. I. Kononenko. On biases in estimating multi-valued attributes. In *IJCAI*, San Francisco, CA, USA, 1995. Morgan Kaufmann.

25. 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.

26. 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.

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

28. 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.

29. E. Malaguti and P. Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, pages 1–34, 2009.
30. Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. Non-model-based algorithm portfolios for SAT. In *Proceedings of the 14th international conference on Theory and application of satisfiability testing*, SAT'11. Springer-Verlag, 2011.
31. T. Messelis and P. De Causmaecker. An algorithm selection approach for nurse rostering. In *Proceedings of BNAIC 2011*, pages 160–166. Nevelland, Nov. 2011.
32. 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, *LION*, Lecture Notes in Computer Science. Springer, 2012.
33. C. Nadeau and Y. Bengio. Inference for the generalization error. *Machine Learning*, 52(3):239–281, Sept. 2003.
34. E. Nudelman. *Empirical approach to the complexity of hard problems.* PhD thesis, Stanford University, Stanford, CA, USA, 2006.
35. P. Pardalos, T. Mavridou, and J. Xue. *The graph coloring problem: A bibliographic survey*, volume 2, pages 331–395. Kluwer Academic Publishers, Boston, 1998.
36. V. T. Paschos. Polynomial approximation and graph-coloring. *Computing*, 70(1):41–86, Mar. 2003.
37. J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.
38. M. Schwengerer. Algorithm selection for the graph coloring problem. Master's thesis, Vienna University of Technology, Oct. 2012.
39. K. Smith-Miles. Towards insightful algorithm selection for optimisation using meta-learning concepts. In *IEEE International Joint Conference on Neural Networks*. IEEE, 2008.
40. K. Smith-Miles and L. Lopes. Measuring instance difficulty for combinatorial optimization problems. *Computers & OR*, 39(5):875–889, 2012.
41. K. Smith-Miles, J. van Hemert, and X. Y. Lim. Understanding TSP difficulty by learning from evolved instances. In C. Blum and R. Battiti, editors, *LION*, Lecture Notes in Computer Science. Springer, 2010.
42. K. Smith-Miles, B. Wreford, L. Lopes, and N. Insani. Predicting metaheuristic performance on graph coloring problems using data mining. In *Hybrid Metaheuristics*. Springer Berlin Heidelberg, 2013.
43. 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.
44. D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.
45. D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.
46. X.-F. Xie and J. Liu. Graph coloring by multiagent fusion search. *Journal of Combinatorial Optimization*, 18(2):99–123, Aug. 2009.
47. L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32, 2008.
48. 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.