APPLYING MACHINE LEARNING ON ALGORITHM SELECTION FOR THE GRAPH COLORING PROBLEM

Lucas de Andrade

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Gerson Zaverucha

Rio de Janeiro
Março de 2016

APPLYING MACHINE LEARNING ON ALGORITHM SELECTION FOR THE GRAPH COLORING PROBLEM

Lucas de Andrade

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Gerson Zaverucha, Ph.D.

Prof. Valmir Carneiro Barbosa, Ph.D.

Prof. Luiz Satoru Ochi, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2016

Andrade, Lucas de

Applying Machine Learning on Algorithm Selection for the Graph Coloring Problem/Lucas de Andrade. – Rio de Janeiro: UFRJ/COPPE, 2016.

XII, 62 p.: il.; 29, 7cm.

Orientador: Gerson Zaverucha

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2016.

Bibliografia: p. 54 – 59.

1. Algorithm Selection. 2. Machine Learning. 3. Empirical Performance Model. 4. Graph Coloring. I. Zaverucha, Gerson. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

To my family

Acknowledgment

First of all I would like to express my gratitude to my supervisor Gerson Zaverucha for the patience and belief along these years. His support, questions and insistence were fundamental for the development of this work.

I would also like to thank Martin Schwengerer and Nysret Musliu for sharing the materials and results of the previous research on algorithm selection for the graph coloring problem and also being available to support us and answer questions.

I would also like to thank my colleagues and managers at Vale for allowing me to pursue this objective and understanding times when I was not always available.

I must also thank Marcelo da Silva Corrêa for initiating me in the academic life during the bachelor's degree course and for supporting me to continue following this road.

Finally I would like to thank my parents who always encouraged me to endeavour in the academic life and my spouse for understanding and supporting me throughout my years of study and research.

Thank you.

Lucas de Andrade

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

APLICANDO APRENDIZADO DE MÁQUINA NA SELEÇÃO DE ALGORITMOS PARA O PROBLEMA DE COLORAÇÃO DE GRAFOS

Lucas de Andrade

Março/2016

Orientador: Gerson Zaverucha

Programa: Engenharia de Sistemas e Computação

O problema de coloração de grafos é um problema NP-Completo para o qual muitos algoritmos foram desenvolvidos ao longo das últimas décadas. Por ser um problema de natureza difícil, a ideia de seleção de algoritmos torna-se relevante para que apenas o algoritmo de coloração mais apropriado seja executado. Este trabalho apresenta uma análise sistemática da aplicação de métodos de aprendizado de máquina no problema de seleção de algoritmos, utilizando coloração de grafos como um caso de estudo. Muitos trabalhos foram realizados nesta área focando em classificação de rótulo único e visualização do espaço do problema, preterindo métodos como regressão e classificação multi-rótulo. Ademais, não houve um estudo sistemático para testar alterações na lista de propriedades de grafos disponíveis como entrada do problema. Essas lacunas são endereçadas por experimentos utilizando como ponto de partida as bases de dados e propriedades consideradas em trabalhos relacionados. A comparação dos resultados mostra que os métodos utilizados neste trabalho geram bons resultados em termos de acurácia ao selecionar os melhores algoritmos de coloração. Ficou claro que a acurácia do algoritmo de aprendizado é muito dependente do critério de desempenho e também pode ser impactada pelas propriedades de grafos. A ideia de classificação multi-rótulo neste contexto traz uma melhora relevante, pois quando há disponibilidade de tempo, alguns algoritmos de coloração alcançam valores similares para o número cromático. Por fim, de um portfólio de oito algoritmos, foi possível recomendar na média apenas quatro destes, com uma acurácia de 89%, ou apenas um, com uma acurácia de 79%.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

APPLYING MACHINE LEARNING ON ALGORITHM SELECTION FOR THE GRAPH COLORING PROBLEM

Lucas de Andrade

March/2016

Advisor: Gerson Zaverucha

Department: Systems Engineering and Computer Science

The graph coloring problem is a well-known NP-Complete problem for which many algorithms have been developed along the last decades. Because of the difficult nature of this problem, the idea of algorithm selection becomes relevant so that one can run only the most appropriate coloring algorithm. This work presents a systematic analysis of the application of machine learning methods to the algorithm selection problem using graph coloring as a case study. Much work has been done in this topic focusing on single label classification and instance space visualization, leaving sideways other methods such as regression and multi-label classification. Furthermore, no systematic approach has been adopted to test changes on the list of graph features available as input. These gaps are addressed throughout a series of experiments using as a starting point datasets and feature sets considered on previous related works. The results show that the methods employed in this work can provide good results in terms of accuracy when selecting the best coloring algorithms. It was also clear that the accuracy of the learning algorithms is highly dependent on the performance criteria and can be impacted by the graph features. The idea of multi-label classification in this context brings great improvement as after some time, some coloring algorithms seem to reach similar values for the chromatic number. In the end, from a portfolio of eight algorithms, it was possible to recommend on average just four of those with an accuracy of 89% or only one with an accuracy of 79%.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 1971, COOK [13] formalized the concept of NP-Complete: a problem is NP-Complete if it is both NP and NP-hard. If a problem belongs to NP, it means that its decision form can be verified in polynomial time by a non-deterministic Turing machine. In other words, if a solution for this problem is given, it can be verified in polynomial time. If a problem P is NP-hard then all problems in NP can be reduced in polynomial time to P, which means that P is as hard as any other problem in NP. Although it is possible to verify the correctness of a given solution quickly, the time required to locate the exact optimal solution for a NP-Complete problem usually grows exponentially as the size of the problem instance grows.

With an increasing number of algorithms being developed, RICE [51] proposed, in 1975, the Algorithm Selection Problem as *"the problem of selecting an effective or good or best algorithm"*. His goal was to find a function S that identifies the algorithm with the best performance to solve a specific instance x of a problem P, without knowing the real performance of all considered algorithms for that specific instance x. Rice's proposal was the first to consider a mathematical model for such problem.

In the eighties researchers turned their focus on the development of new algorithms with heuristics and approximate methods in search for better efficiency and feasible solutions for NP-Complete problems. In 1986 the term meta-heuristic was first adopted by GLOVER [22] in his explanation of Tabu Search and since then it has been used to name strategies that guide and modify simple algorithms to produce better quality solutions. In this period, important algorithms based on meta-heuristics have been proposed such as Simulated Annealing[34] and Ant Colony Optimization[12]. The development of these and other techniques contributed to increasing even more the number of available algorithms to solve NP-Complete problems.

One decade later, the principles and motivation that led Rice to propose the Algorithm Selection Problem were revisited. AHA [1] comes up with the idea of gen-

eralizing algorithm performance from case studies. In other words, if an algorithm presents good performance on a specific class of problems, it should also present good performance on similar classes. The term Metalearning was first used in the context of obtaining this knowledge to make it possible to identify which algorithms are suitable to solve specific classes of problems. As described by SMITH-MILES [57], although many definitions have been proposed for Metalearning, they all fall back to the concept of acquiring knowledge about algorithm performance and exploring it to improve or select algorithms.

The idea of acquiring knowledge about algorithm performance is usually tied to the need of describing a specific problem instance. Rice proposed the extraction of features from these which came to be known later as *metafeatures*. These properties seem to be closely related to the accuracy of applied machine learning methods as it is clear that, for some problems, phase transition properties have been known to dictate the performance of specific algorithms. For example, considering the SAT Problem, it is known that a phase transition property is related to the ratio of clauses to variable and that nearly all algorithms exhibit poor performance when a specific instance has this ratio near the phase transition area [65].

With a huge amount of algorithms and heuristics to solve NP-Complete problems, scientists are now relying on the use of machine learning methods to discover knowledge and hidden relationships within huge databases of algorithm performance data.

1.1 Motivation

Given the huge amount of algorithms and parameter options that can be used for performance tuning, it is a difficult task to decide which algorithm is more appropriate and how it can be tuned to reach the best results. Probably due to the recent increase in popularity of the term Big Data, many researchers have been applying machine learning methods to assess algorithm performance data and build models or portfolios that can solve the Algorithm Selection Problem. These ideas have been applied on case studies of NP-Complete problems such as: the Traveling Salesman Problem [28], SAT [28], Scheduling [43], the Graph Coloring Problem [46] and many others. An online bibliography for these researches is maintained by KOTTHOFF [36].

HUTTER *et al.* [28] introduced the term Empirical Performance Models to characterize regression models created to predict algorithm performance metrics. This expression is a generalization of the previously used term: Empirical Hardness Model, which considered only the algorithm runtime as a performance metric. In some cases, with the approximation predicted by these models, it is possible to select

the most appropriate algorithm to solve a particular instance of a problem by knowing how each algorithm is predicted to perform on that instance. HUTTER *et al.* [28] evaluated this technique to predict the runtime of algorithms that can solve instances of SAT, the Quadratic Assignment Problem and the Traveling Salesman Problem.

When considering the Graph Coloring Problem, much effort has been given to the application of classification methods to select the best algorithm and little information has been collected on how other types of recommendation perform on this problem. Even when considering only classification methods, a detailed analysis and comparison of the different case studies is not available. Currently on the algorithm selection for graph coloring problem literature there is little or no work improving findings of previous researches. As a consequence, there is little insight of which features are relevant for identifying the best algorithm or how the performance criteria impacts the overall accuracy of the machine learning method used to select the most appropriate algorithm. An evidence for this is the random selection of features, sometimes more and other times less, for each research.

1.2 Objective

The general objective of this work is to analyze and compare how different machine learning techniques perform when selecting algorithms to solve the graph coloring problem given different features and performance criteria as parameters to build various algorithm performance databases.

Many works have been done applying classification methods to identify the best algorithm to solve instances of the graph coloring problem [53] [54] [55] [56]. By exploring how other forms of recommendation perform in this case study, it is possible to confirm whether classification is indeed a good choice and if other techniques can provide accurate or relevant information that can be used to support the selection of the most appropriate algorithm.

After reviewing related works about algorithm selection for the graph coloring problem some questions remain unanswered. The current work develops a line of investigation on how changes to the algorithm selection problem spaces impact the accuracy of the selection mapping function S modeled by a machine learning method:

Table 1.1: Main questions

Question	Motivation
How does the set of features impact on the accuracy of the algorithm selection problem for the graph coloring problem?	The features evaluated in the researches are always different. It is hard to answer this question by just comparing the published researches if the set of features are not fixed.
Is it possible to accurately predict performance metrics other than algorithm runtime?	As the term Empirical Performance Model is quite recent, there has been no investigation on how such models can predict other metrics than algorithm runtime. Considering the graph coloring problem, the chromatic number seems to be the most interesting metric to predict.
Why are classification methods more appropriate to select the best algorithm?	Some researchers [7] [43] have already stated, based on experimental studies, that if the goal is to generate recommendations other than that of estimates of performance, then the problem should be addressed with classification. Aggregating the results of individual regression models does not yield the same good performance achieved by classification techniques when trying to predict the best algorithm.

1.3 Organization

In chapter 2 the fundamental concepts for this work are presented. On section 2.1 the formalism of the algorithm selection problem is detailed as described by RICE [51]. Section 2.2 provides a quick overview of machine learning and the methods used in this research. Section 2.3 introduces the concept of Empirical Performance Models and the experiments that led HUTTER *et al.* [28] to this generalization. Section 2.4 summarizes the idea of *Metalearning* considering the definitions proposed by BRAZDIL *et al.* [7].

Chapter 3 details the graph coloring problem. Section 3.1 describes a few graph types and complex network models considered in this research. Sections 3.2 and 3.3 contains information about the available algorithms and heuristics that solve the graph coloring problem. Some of these are considered in the case study explained in the next chapter.

In chapter 4 the application of machine learning methods on the algorithm selection for the graph coloring problem is presented. Related works are explained in the first section. On sections 4.2, 4.3, 4.4 and 4.5 the algorithm selection spaces considered in the experiments of this work are detailed.

At last in chapter 5, the experimental setup, results and comparison of the different forms of recommendation are presented. Section 5.1 contains details about the methodology used to run and compare the different machine learning methods applied to the algorithm selection for the graph coloring problem. The experiments results are presented and discussed on sections 5.2 to 5.4. The details that support this evaluation and the conclusion in chapter 6 are presented on appendix A.

A small program and a few scripts were developed to facilitate the feature extraction and algorithm performance data gathering. Python scripts were built for feature extraction calling NetworkX[24] functions and algorithms. A small C# application encapsulating graph coloring programs from LEWIS *et al.* [39] was used for performance data gathering. WEKA[25], R[29] and MEKA[49] were used as the source for Machine Learning algorithms that were evaluated. Detailed scripts and programs are included in Appendix B.

Chapter 2

Fundamental Concepts

This chapter introduces the main concepts that are referred to throughout this work. It starts with the first mathematical approach to the algorithm selection problem presented and then briefly describes the ideas of machine learning. Within section 2.2, the main machine learning algorithms that are used in the experiments are further detailed. Section 2.3 presents the latest ideas on instance hardness and empirical performance models, a specific view of how machine learning methods can be applied to predict algorithm performance. Finally the concept of metalearning is described in section 2.4. Although there are many interpretations to this term, it is usually related to understanding the relationship among algorithm performance and problem characteristics.

2.1 The Algorithm Selection Problem

The Algorithm Selection Problem, formalized by RICE [51] in 1975, has the main objective of selecting the most adequate algorithm to solve a specified instance of a problem.

RICE [51] proposed three models to this problem and present them in increasing order of complexity. The model referenced in this work is similar to the second model as can be seen on figure 2.1. The third and most complex model, with variable performance criteria, is capable of changing this criteria according to the problem. Despite the fact that different performance criteria are analysed in this work, these are not considered input to the algorithm selection problem. The analysis conducted here consider fixed spaces of performance criteria as will be described later in chapter 4.

The concepts presented next are part of the Algorithm Selection Problem definition:

- Problem Space (P): contains all instances of a specified Problem.

- Feature Space (F): contains all instances properties that are considered relevant to the selection of the best algorithm.
- Algorithm Space (A): contains all available algorithms to solve the problem defined in the Problem Space P. Rice considers the parameter variation of an algorithm does not qualify it as a new algorithm, although this idea could be applied in practice.
- Performance (Metrics) Space ($\mathbb{R}^n$): contains the performance metrics considering the Problem and Algorithm Spaces have been defined. It can be CPU runtime or accuracy for example.

Therefore, given:

- $x \in P$ an instance of the problem defined by P,
- $f(x)$ a function $P \rightarrow F$ that relates an instance x from the problem space P to properties (features) defined on the feature space F,
- $a \in A$ an algorithm to solve the problem defined by P,
- p a function $A \times P \rightarrow \mathbb{R}^n$ that determines the performance metrics of the algorithm,
- g a function that has as a result a real number $y \in \mathbb{R}$ that can be used to evaluate the performance of the algorithm.

What is the mapping function $S : F \rightarrow A$ that determines the most appropriate algorithm from A to be applied on an instance of P?

Figure 2.1 presents the schema for the Algorithm Selection Problem as originally presented by RICE [51].

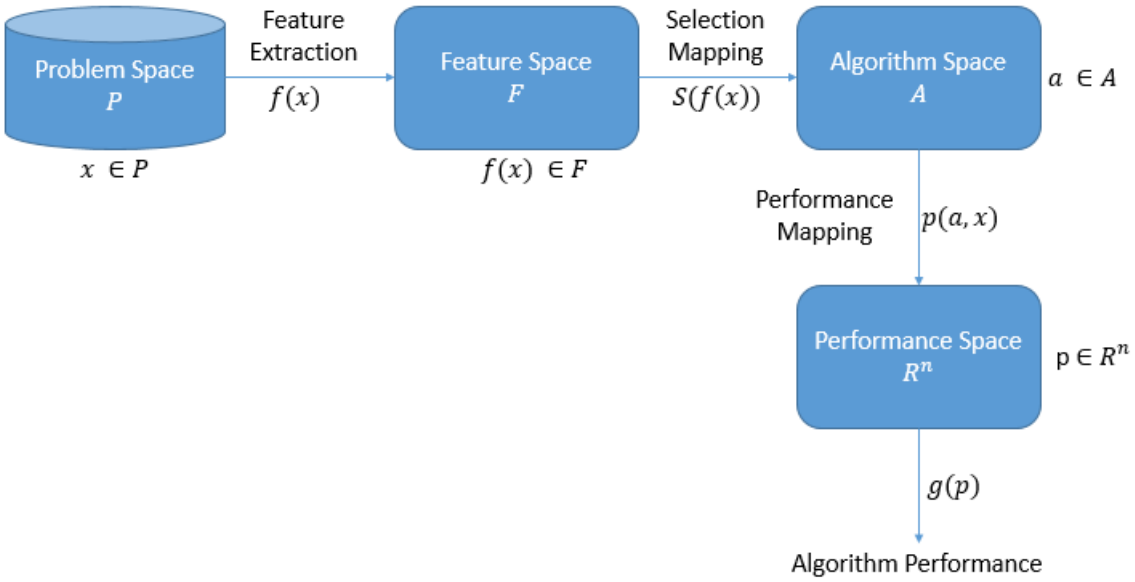


Figure 2.1: Original Schematic from Rice's Algorithm Selection Problem - RICE [51]

2.2 Machine Learning

Machine learning grew out as a sub-field in Artificial Intelligence in which the main goal is to have machines learn from data. The first ideas dates back from 1950 when TURING [60] describes the concept of *"learning machines"*, although at that time no model was proposed on how to build such entities. The general idea was that these machines would have some basic rules of operation that would tell how they would react to specific situations. This is somewhat similar to what exists today regarding trained models built with current machine learning algorithms.

The algorithms that are known today in the machine learning community had their origin in the seventies and eighties with the rise of expert systems and the return of the back-propagation algorithm. These algorithms can be grouped according to their learning method: supervised, unsupervised, semi-supervised and reinforcement learning [52].

In supervised learning, the machine is presented with some examples of what should be learned that basically consists of input-output pairs. It then learns a function $y = h(x)$ that maps values from the input x to output y. If the function $h(x)$ is discrete, then the supervised learning task is called classification, otherwise, if it is continuous, then the learning task is called regression. A specific situation occurs when there are multiple discrete values y for the same input x. The default

classification algorithm could be applied considering the union of these discrete values as a single label or a more advanced technique could be applied to allow for these multiple output values to independently related to the same input. This last method, called multi-label classification, will be further detailed on section 2.2.1.

In unsupervised learning, no labeled examples are provided from the beginning and the machine must figure out patterns within the dataset provided, grouping them accordingly. The most common algorithm for unsupervised learning is clustering, where input data is grouped by similarity of a given property.

Semi-supervised learning stays in a region between supervised and unsupervised learning. A few labeled examples are provided but the machine must figure out by itself when new labels should be created considering the similarity of some properties. This type of learning is useful when the list of labels for the instances is not exhaustive or is not known completely.

Reinforcement learning is the type of learning where a machine is either rewarded or punished for the decisions it took during the learning process.

For around 40 years many machine learning algorithms have been proposed and improved, leading us to literally hundreds of methods to choose from. Trying to understand the usability of all these algorithms in the real-world, FERNÁNDEZ-DELGADO *et al.* [18] conducted an experiment over almost all datasets from the UCI repository, evaluating 179 classifiers from 17 different families such as neural networks, support vector machines, decision trees, boosting, bagging, random forests and many others. The classifiers found with best average accuracy were Random Forest - accessed via caret package from R - and Support Vector Machine - implemented in C using LibSVM. Although FERNÁNDEZ-DELGADO *et al.* [18] acknowledge the No-Free-Lunch [64] theorem, they state that these implementations were very near to the maximum attainable accuracy for almost all the datasets evaluated. They consider the maximum attainable accuracy as the maximum accuracy achieved among the 179 classifiers. The Random Forest and Support Vector Machine algorithms are further detailed in sections 2.2.2 and 2.2.3.

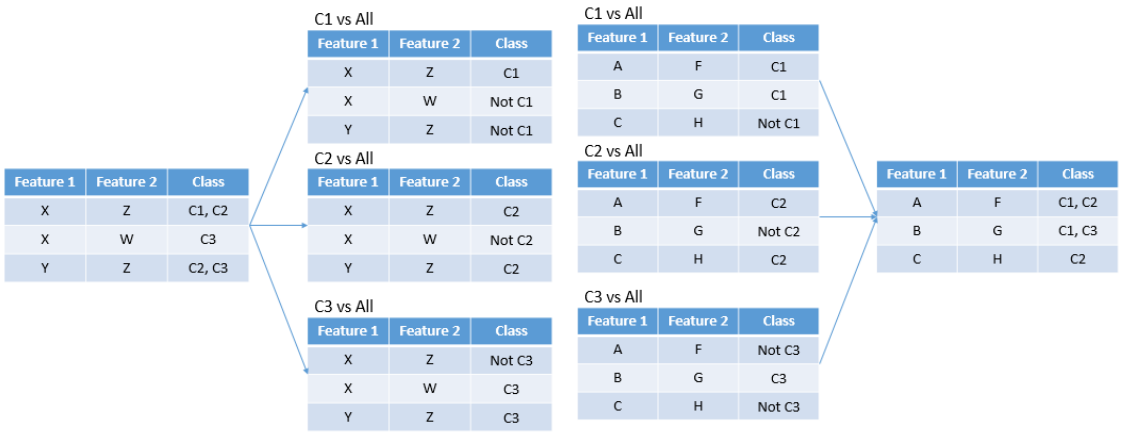
2.2.1 Multi-label Classification

Multi-label classification [58] deals with the problem of classification in a dataset that contains samples belonging to more than one class y, in other words, a single instance can have simultaneously multiple labels. This concept is often confused with multi-class classification which categorizes an instance into one class of more than two available. In the multi-class classification, all instances in the dataset still have a single label. Some methods to deal with this were proposed in the literature and can be basically categorized in two classes: problem transformation and algorithm

adaptation. The first approach transforms the problem into one or more problems of single-label classification and then standard machine learning algorithms could be applied. The second approach deals with making adjustments to known machine learning algorithms so they can also deal with multi-label classification.

Multi-label classification has been successfully used before in the context of metalearning by [32]. In their research, they considered three very simple multi-label classification methods based on the idea of problem transformation and applied these to the travelling salesman problem.

The most common problem transformation technique employed to solve multi-label classification problems is called binary relevance. In this simple method the idea is to create separate binary classification datasets to generate the predictive models, one for each label. This can be done by checking if a single label appears in the instance label. If it does, a positive symbol is assigned to that instance, otherwise the instance will be assigned to a negative symbol. The result for an unseen instance can be obtained by concatenating the results of all predictive models as one single label. Figures 2.2 shows an overview of this problem transformation method.



(a) Training (b) Prediction

Figure 2.2: Binary Relevance Multi-label Classification

The main disadvantage of the binary relevance method is that it fails to model label correlations which possibly exist in some problems. To deal with this other methods have been proposed such as Classifier Chains [50] and Random k-Labelsets (also known as RAkEL) [59].

The Classifier Chains technique also involves problem transformation exactly the same as in the binary relevance method, but instead of running a base classifier just once, the idea is to chain together a series of classifiers extending the feature space with the predictions of previously generated models. READ *et al.* [50] understand that by chaining classifiers together, it is possible to predict strong label correlations if they exist. Algorithm 2.1 shows an overview of this method.

READ *et al.* [50] further enhance this method by applying the ensemble concept that basically trains m classifier chains $h_1 ... h_m$ trained on a random selection of N training instances sampled with replacement. As with other ensembles, a voting scheme is used for the class prediction of an unseen instance.

Following a different path, TSOUMAKAS *et al.* [59] proposed the Random k-Labelsets method based on the label powerset technique of problem transformation. The idea of label powerset is to consider every possible label combination as a single label, hence the name contains the term powerset. Although this method can better model label relationships, it can only learn labels that exist on the training dataset. It is also challenged by datasets with large number of labels as there are many available combinations associated with very few training examples. The Random k-Labelsets was proposed as a solution for this problem with the approach of randomly separating the original labelsets into smaller labelsets with size k and from there apply the traditional label powerset method.

2.2.2 Random Forests

In 1995 HO [27] proposed a new classifier algorithm, named Random Decision Trees, that consists of building a collection of decision trees over a randomly selected subset of features. The combined classification results of all trees is proved to improve accuracy for both training and unseen data (test and validation). The idea of creating an ensemble of trees was the origin of the Random Forest algorithm proposed by BREIMAN [8] in 2001. The current implementation of Random Forest in R was done by LIAW and WIENER [41] which is actually a port of the original code developed in Fortran by BREIMAN [8]. Below the steps from the R implementation:

Algorithm 2.1 - Random Forest

Train

1. Create n_{tree} bootstrap trees from the training data
2. For each tree:

   Grow an unpruned classification or regression tree randomly sampling m_{try} of all features and choosing the best split among those at each step.

Test and Validate

1. Predict the values for unseen data by aggregating the predictions of n_{tree} trees according to the following methods:

   Classification: majority of values

   Regression: average

The number n_{tree} of trees necessary to achieve good performance usually grows with the number of features available. The whole idea behind Random Forests is that the result of the aggregation function converges to a single value the larger the number of trees considered. FERNÁNDEZ-DELGADO *et al.* [18] used 500 trees in their experiment, while LIAW and WIENER [41] mention that an empirical comparison can be done in order to find the number of trees enough to provide good results.

Regarding the random sample of features m_{try} to be considered when growing each tree, it is known that a reasonable value is $\sqrt{\#features}$. This is the default value in R implementation when considering the Random Forest algorithm for classification. When the task is regression, the default value for m_{try} is $\#features/3$. LIAW and WIENER [41] also suggest to test half and twice the default value in practice, picking the best among the three possible values for m_{try}.

2.2.3 Support Vector Machines

The original idea for Support Vector Machines came from the Generalized Portrait method proposed by VAPNIK [62] in 1963. The current standard is based on the optimization problem formulated by BOSER *et al.* [6], CORTES and VAPNIK [14] that enhanced the algorithm to create non-linear classifiers by applying the kernel trick.

The intuition behind Support Vector Machines is to find the maximum margin hyperplane that separates samples from the training vectors $x_i \in R^n, i = 1, \ldots, l$ according to their classes $y \in R^l, y_i \in \{-1, 1\}$. In the end, this is formulated as a constrained optimization problem.

CHANG and LIN [10] included this algorithm on their LibSVM software which is actually the classifier implementation that showed results with no significant statistical difference from the Random Forest implementation in R according to FERNÁNDEZ-DELGADO *et al.* [18].

2.3 Instance Hardness and Empirical Performance Models

The idea to use regression methods to predict the runtime of an algorithm had its origin in the nineties with the utilization of linear regression models in the areas of Parallel Processing and Artificial Intelligence [28].

LEYTON-BROWN *et al.* [40] introduced the term Empirical Hardness Models in 2002 performing a case study for the combinatorial auction problem, applying regression methods to predict algorithm runtime. From there, other case studies have been done leading to the development of the famous algorithm portfolio for the resolution of the SAT Problem: SATzilla [47] [66] [67].

The idea of LEYTON-BROWN *et al.* [40] is that the Empirical Hardness Model can be used as a foundation for the construction of an algorithm portfolio that solves the Algorithm Selection Problem automatically. The construction of this portfolio, as described in [67], has the following steps:

1. Select a representative set of instances for the problem that needs to be solved.

2. Select algorithms that have a varying performance for the chosen set of instances. If a single algorithm has superior performance when compared to all others, there is no need to build an algorithm portfolio or to treat the resolution of the problem applying the Algorithm Selection framework.

3. Identify features that characterize the chosen set of instances. Usually a specialist is required to determine these properties. For an effective result, the selected features must have a correlation with the difficulty to solve an instance while being simple to calculate.

4. Perform the feature calculation and run the algorithms to determine algorithm runtime that will be used as the training set.

5. Identify algorithms as pre-solvers when they can be used before the feature calculation. The idea here is to use these algorithms to dispose of instances considered easy to solve and focus the computational resources and the usage of the algorithm portfolio on the most difficult ones.

6. Build an Empirical Hardness Model, for each algorithm from the portfolio, to predict the algorithm runtime based on the instances features.

7. Select the best subset of algorithms to use in the final portfolio using the validation set.

Once the portfolio has been built, it can be applied to the resolution of a new problem instance with the following steps:

1. Run the pre-solvers algorithms to a predefined time limit.

2. Perform the features calculation.

3. Predict runtime for each algorithm using the Empirical Hardness Models built previously.

4. Run the algorithm with the smallest runtime. In case it fails, try the one with the second smallest runtime and so on.

Recently HUTTER *et al.* [28] generalized the term Empirical Hardness Model to Empirical Performance Model. The idea of this generalization is that the regression models can be used to predict any performance metric and not just the algorithm runtime. This model is also extended to tune algorithm parameters. Casting this to Rice's notation, it can be said that a new space is proposed to the Algorithm Selection Problem: the Configuration Space.

The Empirical Performance Model is formalized below:

Given:

- An algorithm a with configuration space θ,

- A distribution of instances with feature space F,

An Empirical Performance Model is a function $f : \theta \times F \rightarrow \Delta(\mathbb{R})$ that defines a probability distribution over performance metrics for each combination of parameter configuration $\theta \in \Theta$ from A and instance with feature $z \in F$.

In practice, the parameters can be interpreted as features and they will be considered the same way when building the regression models.

An important aspect that must be considered depending on the data used for the experiment is the transformation necessary to better visualize the information

plotted on a graph. In the work of HUTTER *et al.* [28], logarithmic transformation is used to adjust the algorithm runtime curves as this variable can increase considerably for difficult combinatorial problems.

2.4 *Metalearning*

In the nineties the idea of generalizing algorithm performance from case studies is developed [1] [44]. In other words, if an algorithm presents good performance on a specific problem class, it may also show good performance on similar classes. AHA [1] was the first to propose a systematic empirical method to generalize case studies. His idea was to list under what conditions an algorithm is better than other. The rule proposed by AHA [1] using Rice's notation is cited below:

An algorithm $a \in A$ is better than other algorithm $b \in A$ for instances $x \in P$ with features $f(x) \in F$ considering the metric $y \in \mathbb{R}$.

The case study from AHA [1] is focused on machine learning algorithms, but the idea could be applied to any other algorithm. The term *metalearning* was first used in this context. The steps to this generalization are:

1. Gather details about the case study: algorithms, algorithm performance and problem characteristics,
2. Model the problem with artificial data,
3. Select parameters to feed the problem generator such that the algorithm performance differences observed on the artificial data is similar to the information gathered from the real data,
4. Evaluate the algorithms on the artificial data several times for each parameter configuration of the problem generator,
5. Derive a rule summarizing when the performance differences occurs.

Recently most researches related to *metalearning* have been using machine learning methods to generate models that can establish a relation between algorithm performance and instances features. BRAZDIL *et al.* [7] formalized some concepts that are fundamental to understand the practice of *metalearning*:

- *Metaknowledge*: any type of knowledge derived from the application of a Machine Learning algorithm.

- *Metafeatures*: problem properties that have correlation with algorithm performance. Three classes are proposed:
  - Statistical and Information Theory Features: calculated by applying statistic functions over properties of the problem instance. Sample functions are: median, standard deviation, variance and entropy.
  - Features based on the learned model: obtained by analyzing the model generated by the machine learning algorithm. If the machine learning algorithm is a decision tree, for example, the tree height and number of leaves could be relevant features.
  - *Landmarkers*: consisting of performance data from simple or fast algorithms. These algorithms can be executed to try and measure the instance hardness. For most problems there is always a greedy algorithm that can provide a solution even though it is not optimal. The runtime or memory consumption of such algorithm could provide hints on the instance hardness.

According to BRAZDIL *et al.* [7], "*Metalearning* is the study of principled methods that exploit *metaknowledge* to obtain efficient models and solutions adapting machine learning and data mining processes."

As explained by SMITH-MILES [57], although many definitions have been proposed for this term, all aim to explore the knowledge acquired about algorithm performance in order to improve a specific algorithm or to allow the selection of the best one.

Chapter 3

The Graph Coloring Problem

The Graph Coloring Problem, also known as Graph K-Coloring, is defined as the problem of finding a coloring with K colors for the graph G in a way that the vertices sharing a link do not have the same color. Formally the problem is stated as the following:

Given:

- An undirected graph $G = (V, E)$, and
- $K \in \mathbb{N}^+, K < |V|$

It is desired to know if exists a function $f : V \rightarrow \{1,2,...,K\}$ in a way that $f(u) \neq f(v)$ always that $(u, v) \in E$.

The statement above is related to the decision problem form of the graph K-coloring. The optimization problem associated is known as chromatic number problem, where the goal is to find the smallest value for K while still following the constraints of the Graph Coloring Problem.

This problem had its origin with the Four-Color Theorem proved in 1976 [2], despite being proposed almost a century earlier by Francies Guthrie. This theorem states that for any given separation of a plane in contiguous regions, no more than four colors are required to color all regions in a way that adjacent regions do not have the same color. A plane here can be seen as a map or planar graph. In fact, the graph coloring problem is easy to solve for specific types of graphs such as: complete graphs, cycle, wheel or planar graphs and bipartite graphs. There are known theorems and efficient algorithms to find the chromatic number for these types of graphs in polynomial time. Unfortunately, for more complex graphs, there are no strong theorems that can precisely determine the number of colors required to paint all vertices.

In 1972 KARP [33] publishes a work where it is proved that the Graph K-Coloring belongs to the NP-complete class. Later, GAREY and JOHNSON [20] gather many

results obtained until 1979 that are useful to understand the complexity of the graph coloring problem. For instance, specific graph types such as chordal graphs or graphs that do not have vertices with degree greater than three can be colored exactly in polynomial time. It is not in scope of this work to provide a detailed description of all graph types and coloring algorithms, but the next sections give a brief overview of common graph types and complex network models as well as state-of-the-art graph coloring algorithms.

3.1 Graph Types and Complex Networks

There are many different ways to classify a graph. For the objectives of this work they can be basically divided into artificially generated graphs and real-world graphs: The first are built artificially following mathematical models to simulate graphs properties that can be relevant for specific applications or domains. The second class is a mathematical representation of a real-world environment where vertices represent real-world entities and edges show relationships among them. In this work the focus is on the evaluation of artificially generated graphs, although some of these can represent, with good accuracy, properties of real-world networks.

3.1.1 Uniform Graphs

Denoted by G_{np} and proposed by GILBERT [21], ERDŐS and RÉNYI [17] in 1959. This is the simplest type of random graph where the number of vertices is predefined $|N|$ and each possible edge $e \in E$ occurs independently with probability p. The number of edges of a complete graph is $|E| = (n(n-1))/2$ and since each edge has probability p to exist in the generated graph, the expected value for M is $E[M] = (n(n-1)p)/2$ - M being the random variable representing the number of edges of G_{np}.

What is interesting about uniform graphs is that at the same time some of them area easy to color, but hard to color optimally. BOLLOBÁS [5], GRIMMETT and MCDIARMID [23] studied the chromatic number on random graphs and found the result that for a fixed probability p, the lower bound of $\chi(G_p)$ for almost all of them is:

$$\chi(G_p) = \Big(\frac{1}{2} + o(1)\Big) \log 1/(1-p) \frac{n}{\log n} \tag{3.1}$$

3.1.2 Geometric Graphs

Introduced by JOHNSON *et al.* [31], the Geometric Graph model, also known as Random Geometric Graph (RGG), is a model in which each vertex is assigned

a coordinate in a two-dimensional space and an edge is associated with a pair of vertices (u, v) if their Euclidean distance $d(u, v)$ is smaller or equal than a parameter value r.

RGGs are useful to model adhoc networks which are composed of mobile wireless devices. In a dynamic environment, every device can move independently and the links among them also change frequently. RGGs can model this behaviour quite well with the r parameter.

MCDIARMID and MÜLLER [42] also conducted some theoretical work about the chromatic number for random geometric graphs. The results are still of little practical application.

3.1.3 Weight-biased Graphs

These graphs are among the hardest to color as this model is designed to restrict the development of larger cliques while allowing smaller ones. The parameter p is used to terminate the selection of edges when the number exceeds the expected value $E[M] = (n(n-1)p)/2$. They are built by assigning each vertex pair a fixed weight W and then selecting a pair of vertices with probability $p = W(x, y)/\sum(W)$. The weights are also dynamically adjusted by two other parameters as the graph is built. When a new edge is selected γ is used to change the weight of every pair of vertices incident to this new edge. α is only applied to pairs which would form a triangle with the new edge and a previously selected edge. The adjustments done by α and γ can be done either by multiplication or addition. The graph development process terminates when the total weight becomes zero or when the expected value for M is reached.

3.1.4 Barabási-Albert Model

This model is based on the concepts of preferential attachment and growth: two ideas that are key to define the structural properties observed in real world networks. This simple idea was proposed by BARABÁSI and ALBERT [3] in 1999 when they showed that the scale-free property of many complex networks - of varied size and nature - are related to these two concepts.

The scale-free property indicates that the degree distribution of the network follows a power law. This means that the probability $P(k)$ of a vertex have a degree k follows an expression like $P(k) \sim k^{-y}$.

The proposed model is built by growing the network one vertex at a time and following the preferential attachment rule in which every vertex added to the graph is more likely to be connected to the vertex with highest degree.

Since the Seventies, many algorithms have been proposed to solve the graph coloring problem. The next sections will present these different approaches and state-of-the-art algorithms that can solve both the optimization and decision forms of this problem.

3.2 Constructive Algorithms

Constructive algorithms are based on techniques that build a solution in an iterative way. In the case of graph coloring, assigning one color to each vertex on each iteration. The advantage of these algorithms is that they are really fast and can give good results, if not exact, for basic graph structures. In the special cases of cycle and wheel graphs, for example, DSatur and RLF algorithms can find the exact result for the chromatic number. These algorithms are detailed in the next sections.

3.2.1 Greedy

The most simple constructive algorithm proposed to solve the graph coloring problem is the greedy algorithm, or *first-fit*, that determines that each vertex is assigned to an available color within a set of colors S ={1,2,...,K}, in a way that adjacent vertices do not share the same color. If this procedure is not possible, a new color is added to the set of colors $|K| = |K| + 1$ and the algorithm continues. The number of colors of the final solution will depend on the order that the vertices are evaluated and the order that the colors are tested for each vertex. Based on that, there can be many variations of this greedy algorithm.

3.2.2 DSatur - Degree Saturation

An algorithm a little bit more sophisticated and probably the most known among the graph coloring algorithms is DSatur or Degree Saturation [9]. This algorithm uses the definition of vertex degree saturation to determine dynamically during runtime the order in which vertices must be evaluated, different from the greedy algorithm previously presented where this order must be established before the algorithm starts running. The degree saturation of a vertex v is defined as the number of distinct colors of its adjacent vertices. Vertices without colors are not considered in this summation. The algorithm steps are presented below:

1. Sort the list of vertices in decreasing order of degree,
2. Assign color 1 to the vertex with greatest degree,

3. Select the vertex with the greatest saturation degree. If there is more than one option, choose any vertex with the greatest degree in the non-colored subgraph,

4. Assign the smallest color number to the chosen vertex,

5. If all vertices were colored, stop. Otherwise return to step 3.

BRÉLAZ [9] proved that this algorithm is exact for bipartite graphs.

3.2.3 RLF - Recursive Largest First

Proposed by LEIGHTON [37] in 1979, the main idea of the Recursive Largest First algorithm is to color the graph one color at a time instead of one vertex at each iteration. This algorithm leverage the concept of independent sets as it tries to identify them in the graph in order to assign the same color to all vertices within the same set. An independent set is a set of vertices such that there is no edge between any of the vertices.

After coloring an independent set, its vertices are removed from the graph and the procedure continues with the remaining sub-graph until all vertices have been colored. In order to decide which vertex to color at each step, RLF always selects the vertex with the highest degree on the remaining sub-graph that can be colored without causing a conflict.

As a greedy constructive method, RLF is a polynomial bounded algorithm with $O(n^3)$ and will always generate the same feasible solution. Even though it is not able to find the optimal solution for many types of graphs, it is known to be exact for bipartite, cycle and wheel graphs.

3.3 Heuristics and Advanced Techniques

Methods based on heuristics, also known as based on optimization, are methods that navigate within a space of candidate solutions trying to optimize an objective function defined in that space. Some algorithms may have the flexibility of navigating to the space of improper solutions, in which some of the problem constraints are not met, while others are limited to navigate only within the space of proper solutions. The algorithms presented in the next sections uses at least one of the following methods for navigating through the solution space: backtracking, hill-climbing or local search.

3.3.1 Backtracking DSatur

KORMAN [35] proposed an improved DSatur algorithm in 1979 by adding backtracking capabilities using a search tree. The idea of the backtracking procedure is to sort the vertices of a graph according to a criteria and then perform forward and backward steps according to this ordering. Forward steps would assign colors until a certain criteria is met. Then backward steps would be taken to remove colors from vertices in order to evaluate alternative feasible paths in search for an optimal solution.

3.3.2 TabuCol

TabuCol was originally developed by HERTZ and DE WERRA [26] in 1987. It is a local search based algorithm that works with improper k-colorings since when navigating the search space, it provides solutions that do not meet the constraints requirements for the graph coloring problem. In other words, when searching for a solution, this algorithm allows adjacent vertices to have the same color.

In order to find a proper solution, TabuCol needs to reduce the number of coloring clashes to zero. The objective function below counts the number of clashes, hence it is used as the objective function to be optimized by TabuCol:

$$f(S) = \sum_{\forall u,v \in E} g(u,v) \tag{3.2}$$

where

$$g(u,v) = \begin{cases} 1, & \text{if } c(u) = c(v) \\ 0, & \text{otherwise} \end{cases}$$

Many variants have been proposed for the TabuCol algorithm. The version implemented by LEWIS [38] and used in this work is based on the variant proposed by GALINIER and HAO [19]. In this version, the initial solution is built by a modified version of the greedy algorithm that allows clashes by limiting the number of colors that can be used. Starting from this candidate solution, movements in the search space are done by selecting vertices whose color assignment is in conflict and then assigning it to a new color. Moving a vertex back to the same color is disallowed for a specific number of iterations stored in the tabu list: a matrix $T_{n,k}$. For every movement of a vertex v from color i to color j, the tabu list value $T_{v,i}$ is updated with the value $l + t$, in which l represents the current iteration number and t is the *tabu tenure* parameter that controls the search behaviour of the algorithm. GALINIER and HAO [19] suggest the value $t = 0.6f(S) + r$ where r is an integer uniformly selected from the range 0 to 9 inclusive. This expression makes the parameter t proportional to the solution quality $f(S)$ which forces the algorithm to explore

different regions of the solution space when the number of color clashes is high by disallowing previously assigned colors of each vertex for long periods.

3.3.3 PartialCol

PartialCol was proposed by BLÖCHLIGER and ZUFFEREY [4] 20 years after the original idea of TabuCol, yet it has a similar search strategy. Instead of working with improper solutions, PartialCol deals with incomplete solutions by separating vertices that cannot be colored without causing a conflict into a set of uncolored vertices U. The objective function for this algorithm is very simple as it consists of the size of set U which needs to be reduced to zero:

$$f(S) = |U| \tag{3.3}$$

The initial solution for PartialCol is developed the same way as in TabuCol with the difference that when a clash of color occurs, the conflicting vertex is included in set U instead of randomly being assigned to a color. Movements in the search space of this algorithm are performed by selecting a vertex $v \in U$ and assigning it to a color $S_j \in S$. After that, all vertices $u \in S_j$ adjacent to v are transferred to U, in other words, all vertices adjacent to v that have the same color S_j are discolored. PartialCol also uses a *tabu tenure* to manage the search behaviour. BLÖCHLIGER and ZUFFEREY [4] proposed a mechanism that changes the parameter t based on the search progress. This was implemented in the variant known as FOO-PartialCol, in which FOO stands for Fluctuation Of the Objective function.

3.3.4 HEA - Hybrid Evolutionary Algorithm

The Hybrid Evolutionary Algorithm was proposed by GALINIER and HAO [19] in 1999 using ideas of evolutionary algorithms to evolve a population of candidate solutions with a specific recombination operator and local search methods.

The initial solution for HEA is provided by a modified version of DSatur with fixed number k of colors to induce conflicts. Evolution occurs by randomly selecting two parent solutions S_1 and S_2 to generate a child solution S' using the recombination operator. S' is mutated with a local search method, in this case TabuCol and inserted back into the population replacing the weaker of its two parents. The recombination operator proposed by GALINIER and HAO [19] is called Greedy Partition Crossover (GPX) as it selects the largest color classes from the parents to generate the new child candidate solution. The procedure is done by selecting the vertices with the largest color class from the parent solution S_1 and include them in the child solution S'. Then remove the vertices already selected from both parents

and select the ones from S_2 with the largest color class. This process is repeated until k color classes are formed.

3.3.5 AntCol

AntCol is another graph coloring metaheuristic-based method. It was proposed by DOWSLAND and THOMPSON [16] in 2008. As other ant colony optimization algorithms, AntCol leverage the capability that ants have to find the shortest path between food sources and their colonies. In this case, the pheromone trail that indicates the nearest food source guides the ants to solutions that present lower number of colors.

Each ant produce a candidate solution based on the RLF algorithm, but with some modifications to incorporate the pheromone trail parameter. At each step, the ants try to build a solution with a limited number of k colors. The first vertex to be colored is chosen randomly from the set of uncolored vertices X. After that, the ant is guided by the pheromone trail and pick the next vertex according to the probability below:

$$P_{vi} = \begin{cases} \frac{\tau_{vi}^{\alpha} \times \eta_{vi}^{\beta}}{\sum_{u \in X} (\tau_{ui}^{\alpha} \times \eta_{ui}^{\beta})}, & \text{if } v \in X \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

where τ_{vi} is:

$$\tau_{vi} = \frac{\sum_{u \in S_i} t_{uv}}{|S_i|} \tag{3.5}$$

Vertices that can't be colored without causing a clash are included in a separated set Y. t is the pheromone trail matrix. α and β are parameters that control the influence of τ and η in equation 3.4. η is the known as the heuristic rule on ant colony optimization algorithms. In the case of AntCol, η is the degree of vertex v in the graph formed by the current set of uncolored vertices $X \cup Y$. When all vertices have been evaluated, the ant comes up with a proper, possibly partial, solution. If there are still vertices remaining to be colored, they are assigned randomly to one of the k colors and then TabuCol is applied to form a complete feasible solution.

Chapter 4

Algorithm Selection for The Graph Coloring Problem

In this chapter the domain spaces that compose the algorithm selection framework used within this work are further detailed from sections 4.2 to 4.5. Section 4.1 presents the related works since when the focus was just on performance data gathering and analysis until the most recent researches where machine learning techniques have been applied in the context of the algorithm selection problem.

4.1 Related Works

It is common practice in academic research when proposing a novel or enhanced algorithm to compare it with state-of-the-art algorithms. It is not different with the graph coloring problem. As new algorithms were proposed, many empirical and theoretical analysis have been made comparing such algorithms and special cases where it was known that the they would perform well [61]. CULBERSON *et al.* [15] were the first to consider the comparison of many algorithms trying to relate properties of graph instances with the idea of instance hardness. Although not mentioning the suggestion made by AHA [1], CULBERSON *et al.* [15] followed the same idea by developing a random generator to create graph instances so they could be used to identify regions of the problem space where algorithms perform better.

CHIARANDINI and STÜTZLE [11] conducted another significant empirical analysis comparing many graph coloring algorithms and trying to relate algorithm performance with graph instances features such as graph size, density and type. In their research they analyze 3 constructive algorithms and 9 stochastic local search algorithms running over 1260 artificial graph instances, leveraging the same random generator developed by CULBERSON *et al.* [15]. Their findings indicate that RLF is the best constructive algorithm when it comes to solution quality, while DSatur is

nice and fast but sometimes it does not generate good solutions. Their implementation of TabuCol performed well for Uniform and Weight-biased graphs of density 0.5 while it did not dominate those of density 0.1 and 0.9. Guided Local Search (GLS) was found to be the best for Geometric graphs and since this type has a clique size close to the chromatic number, they suggested GLS works best for graphs with this property.

SMITH-MILES *et al.* [55] were the first to apply data mining and machine learning algorithms to predict performance of graph coloring algorithms. In their work, they compared two graph coloring algorithms with very different approaches: DSatur and TabuCol. Table 4.1 below summarizes the conducted experiment casting the terminology of the algorithm selection framework.

Table 4.1: Algorithm Selection domain spaces from SMITH-MILES *et al.* [55]

P	F	A	S	Y
5000 graphs	16 features	DSatur TabuCol	Self-organising Feature Maps and Decision Tree	Minimum number of colors

Self-organising Feature Maps were used to produce a visual representation of the instance space so that later the information about algorithm performance (minimum number of colors) could be superimposed. As expected, DSatur was superior to TabuCol only in a few number of instances. By viewing the generated content as a dataset having the label as a tag identifying if TabuCol outperforms DSatur, they split the generated dataset randomly extracting 80% of the data to use as a training set with the remaining 20% to be used for testing. Using the decision tree algorithm from IBM SPSS software, they claim that with 93.7% accuracy the generated model can predict whether TabuCol will be better than DSatur.

SMITH-MILES and BAATAR [54] continued the work by investigating if graph spectra features could improve the predictions of graph coloring algorithm performance. By graph spectra, they consider a series of features extracted from the adjacency and laplacian matrix representations of a graph. The table 4.2 summarizes the domain spaces considered in the research.

Following the same approach as in the previous work, SMITH-MILES and BAATAR [54] first build a visual representation from the instance space - but this time using Principal Component Analysis (PCA) - and then they apply a machine learning method to predict which graph coloring algorithm would be best for an unseen graph. In this experiment, they formulated a novel performance metric combining the CPU runtime (β) with the gap in the number of colors found and the best

Table 4.2: Algorithm Selection domain spaces from SMITH-MILES and BAATAR [54]

P	F	A	S	Y
800 graphs	21 features	Branch & Bound DSatur LPround	Principal Component Analysis and Naive Bayes	$\beta + 1800\alpha$

known lower bound (α). The class label used this time is the actual name of the graph coloring algorithm, while the features are the two-dimensional coordinates of the instance space projection. The training set was obtained by randomly extracting 70% of the instances with stratified sampling and the remaining 30% was used for out-of-sample testing to get the classification accuracy. The machine learning method considered was a Naive Bayes classifier from MATLAB, which achieved a test set accuracy of 72%. SMITH-MILES and BAATAR [54] attribute the lower accuracy obtained in this work to the difficulty that the machine learning method would have in *"separating instances based on class labels"*.

The last work by SMITH-MILES *et al.* [56] involved lots of different instances and algorithms. They kept using the PCA tactic to generate a $\mathbb{R}^2$ visualization of the instance space and then superimposing the information about algorithm performance to understand regions where coloring algorithms are better than others. The main novelty of this paper is related to the performance metric and how the label is generated for the dataset so a machine learning method can be applied to predict whether a graph coloring algorithm is recommended for an unseen instance. The new metric considered the possibility of graph coloring algorithms finding solutions with the equal or approximately the same quality. In other words, an algorithm is now "ϵ-good" if it finds a number of color K within ϵ% of the minimum color found. The percentages considered are 0% - a tie - and 5%. Table 4.3 summarizes the information about this work.

SMITH-MILES *et al.* [56] first tried Naive Bayes (as they did in the previous work) to predict the regions where each algorithm would be "ϵ-good". This initial try was a failure as it predicted that HEA would be the best across the whole instance space. Then they turned to Support Vector Machine, using a 50-50 split for training and out-of-sample testing and generating eight models to predict the boundaries for each graph coloring algorithm. The accuracies on the testing set ranged from 90% for DSatur and Backtracking DSatur down to 73% for AntCol.

Around the same period, SCHWENGERER and MUSLIU [53] conducted an

Table 4.3: Algorithm Selection domain spaces from SMITH-MILES *et al.* [56]

P	F	A	S	Y
6712 graphs	18 features	DSatur Backtracking DSatur HillClimb HEA TabuCol PartialCol AntCol Random Greedy	Principal Component Analysis and Naive Bayes / Support Vector Machine	Time cap of 5×10^{10} Constraint Checks and concept of "ϵ-good"

extensive research focusing on the algorithm selection problem applied to graph coloring.

Table 4.4: Algorithm Selection domain spaces from SCHWENGERER and MUSLIU [53]

P	F	A	S	Y
1265 graphs	79 features	GLS PartialCol HEA ILS MACOL MAFS MMT TabuCol	Bayesian Network Decision Tree k-Nearest Neighbor Muti-layer Perceptron Random Forest Support Vector Machine	Minimum number of colors Risk Median time

SCHWENGERER and MUSLIU [53] propose a novel performance metric named risk which is how often the same coloring has been found. They consider such metric in their performance metric space Y by comparing the minimum number of colors found with a risk above 50% over 10 executions for the same graph coloring algorithm starting with different randomization seeds. In other words, in this experiment an algorithm must find the same minimum number of colors at least 5 times. They break ties by considering the median time needed to find that number of colors.

They also test other techniques to manipulate the feature set by using discretization and basis function expansion. When dealing with classification problems, discretization techniques can improve the overall accuracy of the machine learning algorithm, while basis function expansion is more commonly used on regression problems to generate new features. The results showed that such techniques could improve the final accuracy of the selection mapping function slightly.

Finally they conclude that the idea of using a portfolio of graph coloring algorithms to build a predictive model to select the best one can solve more instances than every single coloring algorithm used in the portfolio. The best result achieved on the test set was obtained using Random Forest as the machine learning algorithm which led to 70.39% classification accuracy.

In the next sections the domain spaces of algorithm selection are detailed for the set of experiments conducted in this work.

4.2 Problem Space

In this work a straightforward methodology was adopted that could be used on future comparisons with the results presented here. All datasets are either public available or can be generated by random graph generators. We emphasize that such methodology, although simple, is relevant to be followed as it truly shows the impacts of changing variables for this problem and allows a fair comparison among the many methods proposed to select the best graph coloring algorithm.

For all the experiments conducted in this work, the datasets used in the machine learning tasks were built by extracting features and running the selected graph coloring algorithms to gather performance data on the following sets of instances:

- CHI500: 520 graphs of size 500 randomly generated with Culberson's random generator with variable density and types used on the original work of CHIARANDINI and STÜTZLE [11].
- CHI1000: 740 graphs of size 1000 randomly generated with Culberson's random generator with variable density and types used on the original work of CHIARANDINI and STÜTZLE [11].
- DIMACS Graph Coloring: 29 graphs of different size and density from the Graph Coloring Benchmark collection of the DIMACS Challenge.
- BA500: 200 graphs of size 500 randomly based on Barabási-Albert model for complex networks.
- BA1000: 200 graphs of size 1000 randomly based on Barabási-Albert model for complex networks.

CHI500 and CHI1000 datasets were used on a research that had the same objective of generalizing from case studies[11]. It could be considered the first metalearning analysis for the graph coloring problem, in which the authors evaluated algorithm performance to determine which algorithm performs best for specific classes

of graphs. The graph types within these datasets were explained earlier in chapter 3 and consists on: uniform, geometric and weight-biased graphs.

DIMACS graph coloring instances were obtained from the second DIMACS challenge [30] that occurred in 1992 and 1993. Named after the Center for Discrete Mathematics and Theoretical Computer Science, this challenge aims to promote the search for effective algorithm implementations to solve complex problems.

BA500 and BA1000 were generated using NetworkX random generator for Barabási-Albert model. After generating 1000 instances, many were eliminated by selecting those with approximate graph density belonging to CHI500 and CHI1000 datasets in an effort to keep the final dataset balanced. Listing B.1 on appendix B contains the script used for this procedure.

4.3 Algorithm Space

The graph coloring algorithms considered in this work were implemented by Rhyd Lewis [38]. There are eight algorithms available: Random Greedy, Recursive Largest First (RLF), DSatur, Backtracking DSatur, TabuCol, PartialCol, Hybrid Evolutionary Algorithm (HEA) and AntCol.

A nice feature from these implementations is that all algorithms are limited by a number of constraint checks. As defined by Lewis: *"a constraint check occurs whenever an algorithm requests some information about a graph"*. The following tasks increase the number of constraint checks when running the graph coloring algorithms:

- Accessing an element A_{uv} from the adjacency matrix of a graph adds one unit to the number of checks.
- When the graph is represented as an adjacency list, accessing all vertices adjacent to a vertex Adj_v adds $|Adj_v|$ units to the number of checks.

  A specific case of this situation occurs when the algorithm needs to know the degree of a vertex.
- Accessing an element C_{vi} from an additional matrix to count the number of vertices that belong to color class $S_i \in S$ and are adjacent to v. Such access also counts as one unit.

No parameter tuning was done for the graph coloring algorithms listed here. The default values defined in the implementation of LEWIS [38] are maintained. These are based on suggested values from each original publication about the coloring algorithm.

4.4 Feature Space

The feature space varies in the experiments conducted in this work in order to identify the impact of adding or removing features from the space when it comes to measuring the accuracy of the selection mapping function S or, in other words, the model built by the machine learning algorithm. What was interesting on related works is that there was no correlation on why some features were added and others were removed and the feature set not always increase.

Considering this, four different feature sets from related works on algorithm selection for the graph coloring problem were evaluated. A fifth feature set combining previously used features plus adding novel ones is also analysed. FS1 and FS5 have some sets of features that are based on statistical aggregate functions and these are presented in the table 4.5.

Table 4.5: Aggregate functions for features

Name	Description
minimum	minimum value of the population
maximum	maximum value of the population
mean	arithmetic mean of a population
standard deviation	measure of dispersion of a population
variation coefficient	a normalized measure of dispersion population
first quartile	value that splits lowest 25% of the population
median	value that splits lowest 50% of the population
third quartile	value that splits lowest 75% of the population

The list of features for each feature set are presented next from table 4.6 to table 4.10.

Table 4.6: Feature Set 1 - 79 features from SCHWENGERER and MUSLIU [53]

1. Number of vertices
2. Number of edges
3. Density
4. Ratio between number of vertices and number of edges and its inverse
5. Node degree: 8 aggregate functions
6. Maximal clique size: 10 aggregate functions
7. Clustering coefficient: 10 aggregate functions
8. Weighted clustering coefficient: 8 aggregate functions
9. Landmarking
   Local search: 9 features
   Greedy coloring: 23 features
10. Tree decomposition: 2 features
11. Lower and upper bound: 4 features

Table 4.7: Feature Set 2 - 16 features from SMITH-MILES *et al.* [55]

1. Number of vertices
2. Number of edges
3. Diameter: the greatest distance between any pair of vertices. Can also be interpreted as the maximum eccentricity of a graph.
4. Density
5. Average path length: average number of steps along the shortest paths for all possible pair of vertices.
6. Girth: length of the shortest cycle in the graph
7. Mean vertex degree
8. Standard deviation of vertex degree
9. Clustering coefficient: a measure of degree to which nodes in a graph tend to cluster together. This is a ratio of the closed triplets to the total number of triplets in a graph. A closed triplet is a triangle, while an open triplet is a triangle without one side.
10. Mean eigenvector centrality
11. Standard deviation of eigenvector centrality
12. Mean betweenness centrality
13. Standard deviation of betweenness centrality
14. Mean spectrum: mean of the set of eigenvalues of adjacency matrix
15. Standard deviation of set of eigenvalues of adjacency matrix
16. Algebraic connectivity: second smallest eigenvalue of laplacian matrix

Table 4.8: Feature Set 3 - 21 features from SMITH-MILES and BAATAR [54]

1. Number of vertices
2. Number of edges
3. Diameter: The greatest distance between any pair of vertices. Can also be interpreted as the maximum eccentricity of a graph.
4. Density
5. Average path length: average number of steps along the shortest paths for all possible pair of vertices.
6. Mean vertex degree
7. Standard deviation of vertex degree
8. Clustering coefficient: a measure of degree to which nodes in a graph tend to cluster together. This is a ratio of the closed triplets to the total number of triplets in a graph. A closed triplet is a triangle, while an open triplet is a triangle without one side.
9. Energy: sum of absolute values of eigenvalues of adjacency matrix
10. Standard deviation of the set of eigenvalues of the adjacency matrix
11. Algebraic connectivity: second smallest eigenvalue of laplacian matrix
12. Smallest non-zero eigenvalue of the Laplacian matrix
13. Second smallest non-zero eigenvalue of the Laplacian matrix
14. Largest eigenvalue of the Laplacian matrix
15. Second largest eigenvalue of the Laplacian matrix
16. Smallest eigenvalue of the adjacency matrix
17. Second smallest eigenvalue of the adjacency matrix
18. Largest eigenvalue of the adjacency matrix
19. Second largest eigenvalue of the adjacency matrix
20. Gap between the largest and second largest eigenvalues of the adjacency matrix
21. Gap between the largest and smallest non-zero eigenvalues of the Laplacian matrix

Table 4.9: Feature Set 4 - 18 features from SMITH-MILES *et al.* [56]

1. Number of vertices
2. Number of edges
3. Density
4. Mean vertex degree
5. Standard deviation of vertex degree
6. Average path length: average number of steps along the shortest paths for all possible pair of vertices.
7. Diameter: the greatest distance between any pair of vertices. Can also be interpreted as the maximum eccentricity of a graph.
8. Girth: length of the shortest cycle in the graph
9. Mean betweenness centrality
10. Standard deviation of betweenness centrality
11. Clustering coefficient: a measure of degree to which nodes in a graph tend to cluster together. This is a ratio of the closed triplets to the total number of triplets in a graph. A closed triplet is a triangle, while an open triplet is a triangle without one side.
12. Szeged index:
13. Beta:
14. Energy: sum of absolute values of eigenvalues of adjacency matrix
15. Standard deviation of set of eigenvalues of adjacency matrix
16. Algebraic connectivity: second smallest eigenvalue of laplacian matrix
17. Mean eigenvector centrality
18. Standard deviation of eigenvector centrality

Table 4.10: Feature Set 5 - 102 features

1. Number of nodes
2. Number of edges
3. Ratio between number of nodes and number of edges and its inverse
4. Density
5. Node degree: 8 aggregate functions
6. Betweenness centrality: 8 aggregate functions
7. Closeness centrality: 8 aggregate functions
8. Eigenvector centrality: 8 aggregate functions
9. Eccentricity: 8 aggregate functions
10. Clustering coefficient: 8 aggregate functions
11. Global clustering coefficient
12. Weighted clustering coefficient: 8 aggregate functions
13. Graph spectra: 16 features
14. Wiener index
15. Szeged index
16. Beta
17. Girth: length of shortest cycle in a graph
18. Average path length
19. Degeneracy
20. Connected components
21. Rank: number of vertices minus the number of connected components
22. Corank: number of edges minus the graph rank
23. Maximal clique size normalized by the number of vertices: 8 aggregate functions
24. Maximum maximal clique size and time to generate it
25. Tree decomposition: 2 features
26. Lower and upper bound: 4 features

4.5 Performance Criteria

Related works have mentioned that the results published are strongly tied to the performance criteria chosen, but no emphasis is given on such metric. The performance criteria discussed here determines how to evaluate when a graph coloring algorithm a is better than another algorithm b.

Different researches considered distinct time caps to run the graph coloring algorithms in order to gather performance metrics such as minimum number of colors achieved and CPU runtime. In this work the impact of different criteria when comparing graph coloring algorithms is evaluated. The upper bound for the time cap considered in this work is not directly related to CPU runtime, instead the concept of constraint checks is used - which seems to be fair when comparing algorithms. The maximum number of constraint checks allowed is 5×10^{10}, in other words, each algorithm is given the chance to access data from the graph coloring problem instance 5×10^{10} times. By changing the performance criteria also requires to change the machine learning method employed as when allowing ties, the class for a specific instance could be multi-label instead of having a single label. This is detailed in the list below:

- Classification and Regression:

  Performance criteria 1: Minimum color achieved. Ties are resolved by selecting the algorithm that spent less CPU runtime.

  Performance criteria 2: Minimum color achieved. Ties are resolved by selecting the average best algorithm.

  Performance criteria 3: Minimum color achieved. Ties are resolved by selecting the algorithm that had the minimum number of constraint checks.

- Multi-label classification:

  Performance criteria 4: Minimum color achieved allowing ties.

  Performance criteria 5: Minimum color achieved allowing ties with a tolerance of 5%. In other words, algorithms that reach a coloring within 5% of the minimum color achieved are added to the final class label.

  Performance criteria 6: Minimum color achieved allowing ties with a tolerance of 10%. In other words, algorithms that reach a coloring within 10% of the minimum color achieved are added to the final class label.

Chapter 5

Experimental Setup and Results

This chapter discuss the results from the experiments configured according to the details presented on section 5.1. Section 5.2 shows the impact of changing some of the algorithm selection framework parameters on the selection mapping function accuracy. The regression results are presented on section 5.3 where it is possible to see that the concept of empirical performance models is feasible for the graph coloring problem, although it cannot be used as a mean to select the best algorithm. Another quick lookup on the graph coloring performance database revealed that some algorithms were reaching the same value for the chromatic number and that by ignoring the CPU runtime, the classification dataset could be turned into a multi-label classification dataset. The results of applying this method are detailed in section 5.4.

5.1 Methods and Materials

As previously mentioned, the first step in the algorithm selection process is to extract features from the instances of the problem space. In related works, many sets of features have been proposed but no comparison has been made among them. The idea here is to compare these feature sets and propose new features to try improve the selection mapping function accuracy to predict which is the best algorithm for an unseen instance. After comparing the accuracy obtained by using each of these feature sets, the best set was selected to be used on the experiments to support the investigation of other questions from this work.

These feature sets were evaluated without any type of preprocessing or calculation. Neither discretization, nor basis function expansion have been applied to these sets as the goal was to evaluate the original impact that each feature had on the accuracy of the learned model. Listing A.2 on appendix A contains a small part of the source code used for feature extraction. A summary of these sets is given below:

- 79 features from SCHWENGERER and MUSLIU [53] denoted from now on by FS1.
- 16 features from SMITH-MILES *et al.* [55] denoted from now on by FS2.
- 21 features from SMITH-MILES and BAATAR [54] denoted from now on by FS3.
- 18 features from SMITH-MILES *et al.* [56] denoted from now on by FS4.
- 102 features as a novel set introduced by this work denoted from now on by FS5.

The next step in the algorithm selection process is to gather performance data about the algorithms being evaluated. To automate this process, a small C# program [1] was developed to run the graph coloring algorithms from the algorithm space and populate a database with the number of constraint checks and cpu runtime necessary to reach a specific coloring. These performance metrics were gathered running the algorithms on a Intel(R) Core(TM) i7-3612QM CPU @ 2.10GHz.

Finally, the last step is to build the dataset to be used by machine learning algorithms. Attribute-Relation File Format (ARFF) [48] is a simple format that can be used by many machine learning software including WEKA, MEKA, R and Azure ML[45]. Using this standard the following datasets were generated:

- 15 datasets for classification experiments: with a fixed problem and algorithm space, 5 different feature spaces were used as well as 3 different performance criteria: breaking ties comparing CPU runtime, number of constraint checks or selecting the average best coloring algorithm.
- 16 datasets for the regression experiment: by using the best feature set as a result of the comparison done in the classification experiment, 16 regression models were built, one for each graph coloring algorithm in order to predict the minimum number of colors and runtime to achieve that result.
- 3 datasets for the multi-label classification experiment: using the best feature set available and performing the comparison by changing the multi-label performance criteria: allowing ties and including a tolerance margin of 5% and 10%.

Following the recommendation of FERNÁNDEZ-DELGADO *et al.* [18], Random Forest and Support Vector Machine were used to build the predictive models. WEKA toolkit was used to simplify the experimentation process since no significant

[1] available at: https://github.com/ldandrade/gcol-alg-selection

differences have been observed on its results and the ones obtained when running the experiments on R. The default parameters from WEKA were used for the Support Vector Machine algorithm, while for Random Forest, the number of trees used was 500, the same value used in the experiments of [18]. For the multi-label classification experiment three algorithms were compared: binary relevance, ensemble of classifier chains and random k-labelsets. All of those are implemented in MEKA: a multi-label classification toolkit extended from WEKA. The base learner for all multi-label classification algorithms is J48 - WEKA's implementation of the C4.5 decision tree algorithm - with default parameter values from MEKA. The statistical analysis of the model accuracy was performed using WEKA's t-test function after 10 runs of 10-fold cross-validation. Tables 5.1, 5.2 and 5.3 summarizes the experiments conducted.

In order to compare the application of regression methods with classification, sixteen models were built to estimate the chromatic number and algorithm runtime for each graph coloring algorithm. By having the results of this prediction the algorithms were ranked according to a unified performance metric and then the best one is selected for each instance. With this approach it is possible to check if the selected algorithm is indeed the best one and also have an accuracy metric which can be compared with the results from the classification method.

Table 5.1: Algorithm selection domains used in this work for classification experiment

P	F	A	S	Y
1687 graphs	FS1 or FS2 or FS3 or FS4 or FS5	Random Greedy DSatur Backtracking DSatur RLF PartialCol TabuCol HEA AntCol	Random Forest and SVM	Minimum color and break ties comparing: 1 - CPU Runtime 2 - Constraint Checks or 3 - Selecting average best

The unified performance metric has an important role on the final results of this ranking. The most logical combination of algorithm runtime and chromatic number was used. Even though the regression models predict real values for the chromatic numbers, these were rounded to integers. This way the predicted chromatic number is compared first - choosing the minimum - and in case of ties, the algorithm with smallest predicted runtime is selected.

After running the graph coloring algorithm performance evaluation for all eight algorithms, the final database ended up with more than 100000 records of per-

Table 5.2: Algorithm selection domains used in this work for multi-label classification experiment

P	F	A	S	Y
1687 graphs	Best Feature Set	Random Greedy DSatur Backtracking DSatur RLF PartialCol TabuCol HEA and AntCol	Binary Relevance, Ensemble of Classifier Chains and Random k-Labelsets	Allow ties with 0%, 5% and 10% of tolerance

Table 5.3: Algorithm selection domains used in this work for regression experiment

P	F	A	S	Y
1687 graphs	Best Feature Set	Random Greedy DSatur Backtracking DSatur RLF PartialCol TabuCol HEA and AntCol	Random Forest and SVM	Minimum color in less time

formance data as whenever an algorithm found a feasible solution, the number of constraint checks and the time needed to reach that specific number of colors were stored. The database model can be seen in figure A.1 on appendix A. As expected, no single algorithm is best to solve all instances, regardless of the performance criteria.

Figure 5.1 shows the class distribution considering the performance criteria 1 in which the best algorithm is the one that reached the minimum color in less time. The algorithm with best average performance on the dataset was Backtracking DSatur, being able to solve 37% of the instances in less time than other algorithms. An interesting result is that all graphs from Barabási-Albert model (BA500 and BA1000) were best solved by the Backtracking DSatur algorithm. This may have happened because graphs of this type are built incrementally following a rule that generates a scale-free network. Since Backtracking DSatur is also a constructive method tuned with backtracking capabilities, it was able to reach a minimum coloring much faster than other heuristics could. TabuCol and HEA, the best known graph coloring heuristics, were practically tied with 20% and 23% of the instances solved respectively.

Random Greedy and RLF solved a minimal percentage of the dataset in less time

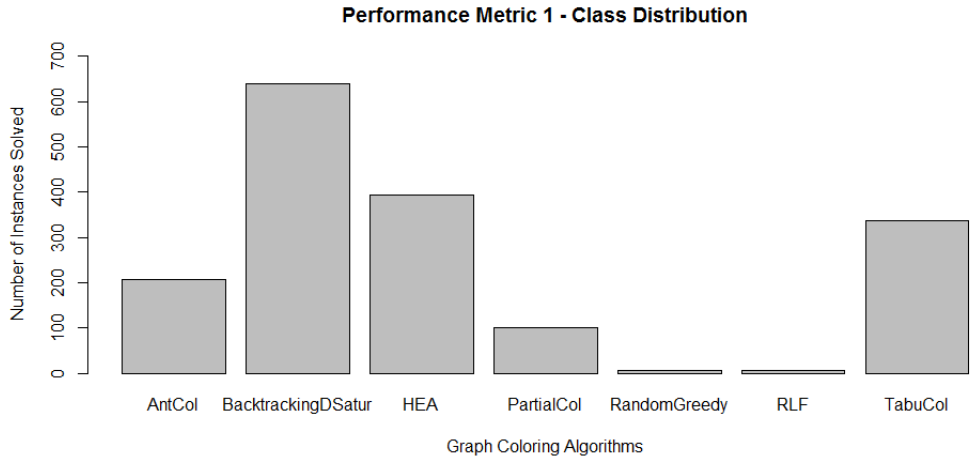


Figure 5.1: Class Distribution for Performance Criteria 1 - Minimum color reached in less time

than other algorithms. As they are very simple constructive algorithms and always generate a feasible solution, they seem to be appropriate to solve either very simple or very hard instances that no other algorithm is able to solve within the same time. For this dataset, RLF would be selected as best algorithm for just 5 instances, all of them being Uniform graphs with high density. Random Greedy solved 7 instances from the DIMACS benchmark challenge that were also optimally solved by all other 7 algorithms despite requiring more time. DSatur did not appear as best algorithm for this dataset as it did not reach the performance criteria considered for any instance, even though for some graphs it was able to reach a minimum coloring equally to other algorithms.

For the second experiment, the performance criteria is modified to select the average best algorithm if more than one algorithm reaches the same minimum coloring for a graph instance. Figure 5.2 shows the class distribution for this dataset.

In this new configuration, HEA is by far the best algorithm, being able to solve 77% of the instances. Random Greedy has disappeared in this case as it was only able to solve very few instances and therefore could not be selected as best on average for any dispute. TabuCol, AntCol and BacktrackingDSatur obtained similar performance being selected for solving 138, 137 and 94 instances respectively.

Finally figure 5.3 shows the class distribution when breaking ties by selecting the algorithm that obtained the minimum coloring with the smallest number of constraint checks. It can be seen that the overall distribution is changed again. Still HEA seems to be the best on average algorithm considering this performance criteria, being able to solve 38% of the instances and reaching similar performance

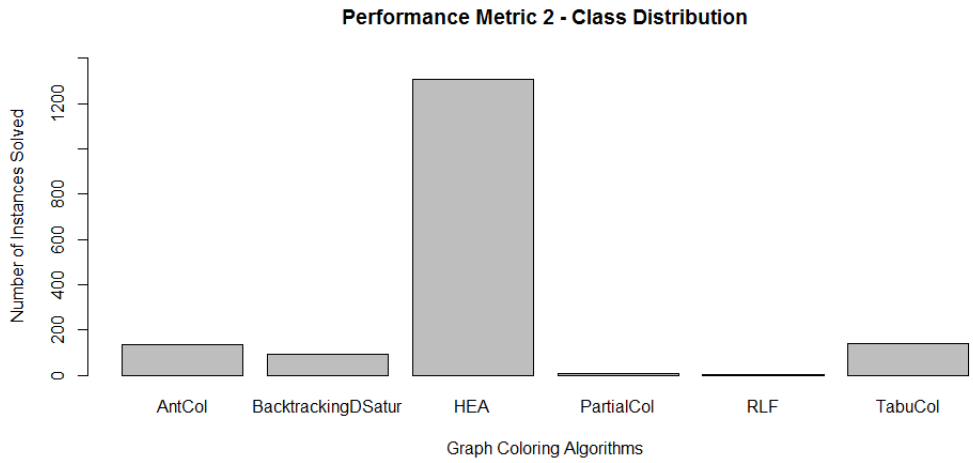


Figure 5.2: Class Distribution for Performance Criteria 2 - Minimum color and Average Best Algorithm

to Backtracking DSatur as on the first experiment. Backtracking DSatur stepped down the ranking as it requires many more constraint checks when compared to other algorithms. AntCol increased a little bit its participation on the number of instances solved. Indeed it was possible to observe during the experiments that AntCol was one of the slowest algorithms to reach the constraint checks limit number. Therefore we can assume that it spends much time processing the same information it requested about the problem instance.

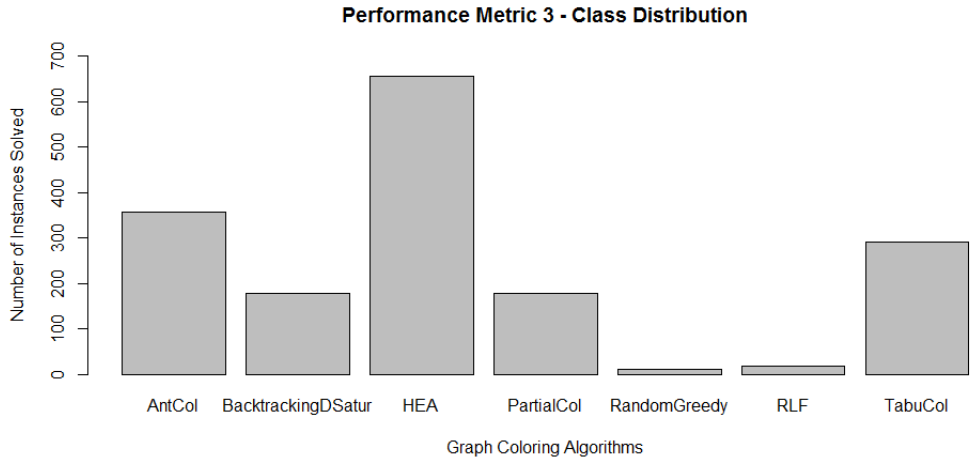


Figure 5.3: Class Distribution for Performance Criteria 3 - Minimum color reached with less constraint checks

By analysing the performance database it was possible to notice that some algo-

rithms were reaching the same number of colors for some graph instances, confirming what SMITH-MILES *et al.* [56] observed in their experiment. This led to the assumption that multi-label classification could perform well if ties were allowed when defining the label class for each instance. The idea of including a tolerance margin if other algorithms were close enough to the minimum number of colors achieved was also tested. On the next sections the details and analysis of these experiments are presented.

5.2 Features and Performance Criteria Impact on Classification Accuracy

Tables 5.4, 5.5 and 5.6 shows the average accuracy of 10 runs of 10-fold cross-validation for the 15 classification experiments. ZeroR and OneR accuracies are also included in order to have a baseline comparison of the improvement obtained when using more sophisticated algorithms [63]. ZeroR always predicts the majority if the class is nominal and if it is numeric, the average is calculated. OneR is another very simple algorithm that generates a one-level decision tree by selecting the feature that can split the dataset providing the highest accuracy. The best combination of feature set, performance metric and selection mapping function (machine learning algorithm) are highlighted in the table. When there is more than one accuracy highlighted for the same machine learning algorithm, it means there is no significant statistical difference among those. T-test was executed with a two tailed confidence level of 0.05 analysing the percent correct metric of the experiments conducted.

Table 5.4: Accuracy for Color and CPU Runtime Performance Criteria Datasets

Dataset	FS5	FS1	FS2	FS3	FS4
ZeroR	37.82± 0.21	37.82±0.21	37.82± 0.21	37.82± 0.21	37.82± 0.21
OneR	**70.40±3.01**	50.48±3.50	68.08± 3.12	**71.45±3.00**	**71.45±3.00**
SVM	37.82± 0.21	45.88±2.46	**50.32±2.95**	39.41± 0.85	37.82± 0.21
Random Forest	**78.73±2.29**	61.17±2.95	74.69± 2.53	76.82± 2.41	75.89± 2.44

Table 5.5: Accuracy Average Best Performance Criteria Datasets

Dataset	FS5	FS1	FS2	FS3	FS4
ZeroR	77.36± 0.27	77.36± 0.27	77.36± 0.27	77.36± 0.27	77.36±0.27
OneR	**83.09±1.86**	81.09± 2.20	81.07± 2.65	**83.17±2.06**	81.57±1.89
LibSVM	77.36± 0.27	**78.74±1.36**	**78.22±0.67**	77.35± 0.49	77.36±0.27
Random Forest	**85.98±2.08**	**85.89±1.88**	82.82± 2.16	83.34± 2.05	83.06±2.14

It is interesting to see that a simple machine learning algorithm like OneR is capable of increasing the algorithm selection accuracy by a lot when comparing to the baseline ZeroR for the first performance criteria. Whenever available in the

Table 5.6: Accuracy Color and Constraint Check Performance Criteria Datasets

Dataset	FS5	FS1	FS2	FS3	FS4
ZeroR	38.89± 0.27	38.89± 0.27	38.89± 0.28	38.89± 0.28	38.89± 0.28
OneR	**55.49±3.33**	50.48± 3.50	49.54± 3.10	**54.62±3.25**	**54.68±3.21**
SVM	38.89± 0.27	**45.88±2.46**	**45.15±2.45**	40.47± 1.35	38.89± 0.28
Random Forest	**60.08±3.14**	**61.17±2.95**	56.41± 3.36	**57.97±3.04**	56.40± 2.99

feature set, the energy property was always selected as the root node, leading to the highest accuracies among the OneR set of experiments: above 70%. Indeed, this feature has been selected among others by the PCA technique used in [56] and [54]. A more complex algorithm like Random Forest could still improve the accuracy beyond that as it develops the tree starting from the same root node but having other features to further enhance the predictive model. Surprisingly, SVM does not perform well on these datasets, achieving results equal to or worse than OneR.

Because of the unbalanced characteristic of the dataset generated with the average best performance criteria, the percentage of correct predictions is much higher. One can notice that by checking the accuracies obtained by the ZeroR algorithm that always select the class with most instances on the dataset, in this case, the HEA class representing 77% of the total number of instances.

The last performance metric verified was to break ties using the number of constraint checks instead of CPU Runtime. It is by comparing this metric with the first one that we can see the big impact that this parameter has on the prediction accuracy of the model. ZeroR showed very similar accuracies for both performance metrics meaning that these datasets are equally balanced. OneR had just a slight improvement, but below the level reached using the first criteria. Then Random Forest showed a difference of almost 20% when comparing to the accuracies reached by using the first performance criteria. It must be emphasized that the only difference between these datasets is the performance metric used to populate the class attribute.

Another takeaway from this set of experiments is that the datasets with the most number of features are the ones with the highest accuracies when applying the Random Forest algorithm. This behaviour is expected as the more information this algorithm has about an instance, easier it should be to differentiate one instance from others. This capability of an embedded feature selection process is characteristic of the Random Forest algorithm since it tests only a subset of features randomly extracted from the whole feature set at each node for each random tree. As the number of trees in the forest increases, the features that aggregate the most value are often selected. On the other hand it can be seen that SVM does not yield good results when increasing the number of features. Table 5.7 presents the statistical significance tests comparing the different machine learning algorithms and showing

that Random Forest is the most appropriate to be used to build the predictive model.

Table 5.7: Accuracy Comparison for all Machine Learning Algorithms

Dataset	Random Forest	ZeroR	OneR	SVM
graphcoloring-avgbest-fs1	**85.89±1.88**	77.36±0.27	81.09± 2.20	78.74±1.36
graphcoloring-avgbest-fs2	**82.82±2.16**	77.36±0.27	**81.07±2.65**	78.22±0.67
graphcoloring-avgbest-fs3	**83.34±2.05**	77.36±0.27	**83.17±2.06**	77.35±0.49
graphcoloring-avgbest-fs4	**83.06±2.14**	77.36±0.27	**81.57±1.89**	77.36±0.27
graphcoloring-avgbest-fs5	**85.98±2.08**	77.36±0.27	83.09± 1.86	77.36±0.27
graphcoloring-colorchecks-fs1	**61.17±2.95**	38.89±0.27	50.48± 3.50	45.88±2.46
graphcoloring-colorchecks-fs2	**56.41±3.36**	38.89±0.28	49.54± 3.10	45.15±2.45
graphcoloring-colorchecks-fs3	**57.97±3.04**	38.89±0.28	54.62± 3.25	40.47±1.35
graphcoloring-colorchecks-fs4	**56.40±2.99**	38.89±0.28	**54.68±3.21**	38.89±0.28
graphcoloring-colorchecks-fs5	**60.08±3.14**	38.89±0.27	55.49± 3.33	38.89±0.27
graphcoloring-colortime-fs1	**61.17±2.95**	37.82±0.21	50.48± 3.50	45.88±2.46
graphcoloring-colortime-fs2	**74.69±2.53**	37.82±0.21	68.08± 3.12	50.32±2.95
graphcoloring-colortime-fs3	**76.82±2.41**	37.82±0.21	71.45± 3.00	39.41±0.85
graphcoloring-colortime-fs4	**75.89±2.44**	37.82±0.21	71.45± 3.00	37.82±0.21
graphcoloring-colortime-fs5	**78.73±2.29**	37.82±0.21	70.40± 3.01	37.82±0.21

In order to obtain some insight on what features were contributing most to the predictive model, we counted the number of times each feature appeared on every tree of a single run of 10-fold cross-validation using FS5 as the feature set. If one feature is always selected, its count should be 5000 as the experiment generates 500 trees for each run of the 10-fold cross-validation. This analysis also indicated that some features did not appear in any of the trees. These were: beta, number of connected components, girth, mean spectrum and smallest eigenvalue of laplacian matrix. With the exception of beta, these features did not present variations in their values for this dataset, hence they were not selected for the predictive model. They could, however, prove useful for other datasets with other types of graphs. Table 5.8 shows the top 10 selected features with their corresponding count and percentage considering the total of 5000 trees.

Table 5.8: Number of times each feature was used on the Random Forest using 10-fold cross-validation and FS5 dataset - top 10 features

Feature	Count	Percentage
Number of edges	4926	99%
Density	4520	90%
Standard deviation of degree	4416	88%
Ratio between number of edges and nodes	4384	88%
Ratio between number of nodes and edges	4164	83%
Minimum degree	4000	80%
Variation coefficient of degree	4000	80%
Mean degree	3690	74%
Maximum betweenness centrality	3460	69%
Variation coefficient of betweenness centrality	3392	68%

5.3 Multi-label Classification

For the multi-label classification experiments only the novel feature set FS5 was used to compare the prediction performance of three selection mapping functions: binary relevance, ensemble of classifier chains and random k-labelsets. Table 5.9 shows the average accuracy of 10 runs of 10-fold cross-validation obtained by each of the algorithms when evaluated using three different performance criteria with tolerance margins of 0%, 5% and 10%:

Table 5.9: Classification accuracy for different performance criteria using multi-label classification methods as selection mapping functions

Selection Mapping	0% Tol.	5% Tol.	10% Tol.
Binary Relevance	76%	87%	93%
Random k-Labelsets	76%	89%	94%
Ensemble of Classifier Chains	79%	89%	94%

It is possible to see that just allowing ties with a 0% tolerance margin for the number of colors does not yield better results than Random Forest on a single label multi-class classification scenario. However, increasing the tolerance margin to 5% improved the average accuracy by 10%. As this tolerance margin increases, the average accuracy is also improved reaching up to 94% in this set of experiments.

Another relevant metric for multi-label classification is the average label cardinality predicted by the selection mapping algorithm. This indicates the average number of graph coloring algorithms within the same recommendation. Table 5.10 shows that out of 8 algorithms, 3 graph coloring algorithms are usually recommended as they reach the exact same number of colors. When adding a 5% tolerance, almost 5 algorithms are often recommended and with 10% tolerance more than 5 algorithms can actually be selected. This metric shows how the chromatic number found by an algorithm is close to one found by another algorithm, thus confirming the hardness of estimating graph coloring algorithm performance in order to select the best one.

Table 5.10: Label cardinality for different performance criteria using multi-label classification methods as selection mapping functions

Selection Mapping	0% Tol.	5% Tol.	10% Tol.
Binary Relevance	3.171	4.821	5.511
Random k-Labelsets	3.373	5.015	5.568
Ensemble of Classifier Chains	3.236	4.865	5.484

Since multi-label classification work with individual predictions for each class and

then concatenate all results to have a unified label as a result, it is possible that the predictions for each individual algorithm have a negative outcome and therefore the final result is an empty label vector. For example, suppose the predictive model has already been trained - the same way it has been generated in this work - and a new instance will be evaluated to check what would be most recommended graph coloring algorithm to find its chromatic number. When using the multi-label classification, eight predictive models, one for each graph coloring algorithm will be evaluated based on this instance's features. Since this is a prediction, there may be the case where each model predicts that the graph coloring algorithm being tested is not recommended, hence the final result will be an empty label vector.

Fortunately, empty label vectors hardly ever occur, but still it is an important measure of the quality of the predictive model. Table 5.11 below shows the percentage of empty label vectors for each multi-label classification algorithm and performance criteria. Note that only when the average number of recommended algorithms is close to 3 according to table 5.10, that some empty label vectors appear in the results of table 5.11.

Table 5.11: Empty label vectors percentage for different performance criteria using multi-label classification methods as selection mapping functions

Selection Mapping	0% Tol.	5% Tol.	10% Tol.
Binary Relevance	0.01%	0%	0%
Random k-Labelsets	0.01%	0%	0%
Ensemble of Classifier Chains	0.003%	0%	0%

5.4 Prediction of Graph Coloring Algorithm Performance

Based on the results of the classification experiment, feature set 5 was selected as the feature set for the graph coloring algorithm performance prediction. The idea is to use the regression form of Random Forest to predict the chromatic number of each coloring algorithm as well as the CPU runtime. The chromatic number is the key metric to define which coloring algorithm is the best one, but CPU runtime is also a relevant metric to be considered in the regression experiments since it has been originally used by other researches in the area[28]. It is expected that with this information it would be possible to rank the algorithms according to the chromatic number and then transform the regression problem into a classification problem by selecting the best algorithm. Depending on the performance criteria used for the

ranking, the dataset could assume different formats: if a criteria is used to break ties between algorithms that are predicted to have the same chromatic number, the problem is a single label classification task, otherwise if ties are allowed, the problem is then a multi-label classification task. There are also multiple criteria that can be used to break the ties explained previously: CPU runtime would be the most logical choice, but other options are available such as the error metrics for each regression model like mean squared error (MSE), root mean squared error (RMSE) or correlation coefficient.

Table 5.12 below shows the correlation coefficient calculated using 10-fold cross-validation in WEKA for all sixteen predictive models: eight models to predict the chromatic number for each coloring algorithm and eight models to predict their respective CPU runtime. The correlation coefficient evaluation measure ranges from 1 to -1 representing perfectly positive or negative correlated results respectively and 0 when there is no correlation.

Table 5.12: Correlation coefficient for the sixteen graph coloring algorithm performance predictive models generated with Random Forest

Graph Coloring Algorithm	Chromatic Number	CPU Runtime
DSatur	0.9901	0.9882
Backtracking DSatur	0.9902	0.4576
Random Greedy	0.9934	0.9341
RLF	0.9904	0.9894
HEA	0.9935	0.9395
PartialCol	0.9931	0.8687
TabuCol	0.9924	0.7969
AntCol	0.9932	0.6832

It is possible to notice that for the chromatic number performance metric all models achieved a correlation coefficient above 0.99. On the other hand, the CPU runtime predictive models did not reach such a high level of correlation. This might be explained with the fact that the range of CPU runtime for some algorithms to complete is much higher than others. Backtracking DSatur, for example, required up to 30 minutes to complete processing some graph instances, while for others it did not register a significant amount of CPU runtime: zero milliseconds. The same characteristic can be observed with TabuCol and AntCol ranges of CPU Runtime: the first having a maximum of 15 minutes and the last requiring up to 529 minutes. The CPU runtime predictive models with highest correlation coefficient had a much lower maximum CPU runtime: DSatur with 1390 milliseconds and RLF with 3 minutes.

The predictions on test data from the cross-validation experiments were gath-

ered and used to check if it was possible to rank the algorithms according to their performance criteria in order to select the best algorithm. In the first experiment, the coloring algorithms were ranked according to the actual chromatic number and the rounded predicted chromatic number. Thus the dataset class is converted from a numeric type to a multi-label type (just like the multi-label classification experiment dataset). In this case, only 33% of the predicted values match the actual best coloring algorithms. Table 5.13 shows examples of two instances with the actual chromatic number and their predictions.

Table 5.13: Actual and Predicted Values for Chromatic Number from instances 1 and 4 from fold 1

Inst.	DSatur	BkDSatur	Greedy	RLF	HEA	PartCol	TabuCol	AntCol
Actual Chromatic Number								
#1	26	24	31	24	21	**20**	**20**	21
#4	**149**	**149**	188	152	**149**	**149**	**149**	151
Predicted Chromatic Number								
#1	24	23	30	24	**21**	**21**	**21**	22
#4	152	152	188	154	152	**151**	152	158

After browsing the results of the whole ranking, it can be seen that most predictions are similar to the ones presented in table 5.13. The best algorithms often reach close values for the chromatic number, sometimes with the difference of a single unit. In the case of instance #1, the actual best algorithms are PartialCol and TabuCol that reached a chromatic number value of 20. However, the chromatic number predicted by the regression models indicate that HEA could also be one of the best algorithms, besides PartialCol and TabuCol. Since the interest is on the exact match of actual and predicted values, instance #1 count as an incorrectly classified instance. The same thing occurs with instance #4 but with a different order of magnitude for the chromatic number values.

This is a challenge for the regression methods because they need to be extremely precise to be able to maintain this relative performance difference between the algorithms. One could also argue that since the regression models generated are independent, they are not appropriate to generate this ranking.

On the second experiment, the usage of the predicted CPU runtime values to break ties and select only the best of all 8 coloring algorithms improved the accuracy a little bit - 38% - despite still being very low when compared to the results obtained by employing the original single label classification algorithm.

Table 5.14 summarizes the accuracy of these two experiments conducted.

Table 5.14: Accuracy obtained by ranking the results of regression models and selecting the best algorithms

Performance Criteria	Dataset Type	Accuracy
Ranking chromatic number allowing ties	multi-label	33%
Ranking chromatic number and breaking ties with CPU Runtime	single label	38%

5.5 Results Comparison

Even though it is not fair to directly compare the results obtained by this work to others from related works because of the different objectives and experiments configurations, it is possible to see some improvement in the accuracy obtained by the application of machine learning algorithms in this case study. Using Random Forest as the single label classification algorithm, it was possible to obtain an average accuracy of 79% using 10 runs of 10-fold cross-validation. Applying more sophisticated methods of multi-label classification resulted in a 10-fold cross-validation average accuracy of 79% when not adding a tolerance margin to build the label vector for each instance. By adding a reasonable tolerance of 5% margin to the chromatic number, the accuracy goes up to 89%.

SCHWENGERER and MUSLIU [53] obtained an average accuracy of 73% using 10-fold cross-validation and performing the experiments on a single label classification dataset comparing 8 coloring algorithms. SMITH-MILES and BAATAR [54] reached an accuracy of 74% using out-of-sample testing on a single label classification dataset comparing the performance of 3 coloring algorithms. At last, SMITH-MILES *et al.* [56] used a different approach by predicting the boundaries of a coloring algorithm's footprint. In their comparison of 8 coloring algorithms using one SVM model for each one of them, the out-of-sample test accuracies ranged from 90% for DSatur down to 73% for AntCol. The approach adopted by SMITH-MILES *et al.* [56] is somewhat similar to a multi-label classification since more than one algorithm can be recommended for the same graph instance. However, a final accuracy for the aggregated result is not provided neither the detailed results for each individual SVM model is available.

Chapter 6

Conclusion

The application of machine learning methods to the algorithm selection problem have been further explored in this work considering the specific domain of graph coloring. Related works have not followed a systematic approach to study the impact of different parameters, methods and options that can be used to select the best graph coloring algorithm from a portfolio: graph features have been added and removed without too much explanation and performance criteria have been selected empirically[53] [54] [55] [56].

Much emphasis is given on the importance of domain knowledge when trying to select the best algorithm to solve a specific problem. Even though this is still essential to the algorithm selection problem, it can be seen that machine learning algorithms can model quite well the relationship among problem features and algorithm performance given a specific criteria. While the classification methods can capture a direct relation of features and classes, the regression algorithms can provide complementary information by predicting relevant metrics.

During the experiments conducted it was possible to see that the Random Forest machine learning algorithm can benefit from the increased number of features as it has embedded a feature selection function in its training procedure. The highest classification accuracies have been reached by this algorithm when applied to the datasets that contained most information about graphs. The best result was obtained by the specific case of Random Forest applied to the novel feature set of 102 graph properties.

Another important result from this work and one that can be further explored is the impact of performance criteria on the dataset construction and learning process. There are many different ways to evaluate how one graph coloring algorithm is better than another and depending on the criteria selected, the dataset will assume a shape that can make the learning task easier or more difficult. It is amazing how the simple change of breaking ties can significantly impact the learning task: when considering the CPU runtime it is much easier for the learning algorithm to find association rules

between the features and classes than when considering the number of constraint checks.

Still regarding the performance criteria, the impact of different time limits when running the graph coloring algorithms can be further explored. Related works have considered different time limits along years in which the computational resources have changed significantly. By having a single work analysing different time limit options running the experiments with the same computer configuration it would be possible to see if there are differences in the chromatic numbers found by coloring algorithms as more time is allowed for them to run.

In the situation where ties are allowed according to the performance criteria, the dataset to be used for the machine learning task has a multi-label class. There are many multi-label classification algorithms available in the literature and within this work some of the most known have been tested to generate the recommendations for the algorithm selection problem. When allowing ties with no tolerance margin on the chromatic number, the classification accuracy obtained was similar to the one reached by the single label classification experiment. Because of the similar performance for some of the coloring algorithms, it seems that there is not enough data yet to model these small differences of the chromatic number found by the best heuristics. Although, when adding a tolerance margin of 5% and 10% in order to accept these small differences, the accuracy obtained by the multi-label classification algorithms increased by 10% and 15% respectively. Out of a portfolio with 8 algorithms, it was possible to recommend on average just 4 of those with an accuracy of 89%.

Another possibility explored within this work did not generate good results when it comes to the accuracy metric: the prediction of graph coloring algorithm performance using regression methods. The initial expectation was that by ranking the algorithms according to the predicted chromatic number it would be possible to select the best algorithm. Despite the disappointment of the accuracy obtained using this approach - only 33% and 38% - it was possible to see that the regression methods are able to estimate the chromatic number that would be found by a coloring algorithm.

The results comparison show that the methodology employed in this work, specially the experiments related to multi-label classification, can provide good results in terms of accuracy when selecting the best algorithms from a portfolio of graph coloring algorithms - up to 89%. The systematic approach proposed here can be reused to continue the experimentation process of adding new performance metrics, features and even coloring algorithms. The programs and scripts used can be easily extended and provide a common foundation for comparison of different approaches and configurations of algorithm selection for the graph coloring problem.

Bibliography

[1] AHA, D. W., 1992, "Generalizing from Case Studies: A Case Study". In: *Proceedings of the Ninth International Conference on Machine Learning*, pp. 1–10. Morgan Kaufmann.

[2] APPEL, K., HAKEN, W., OTHERS, 1977, "Every planar map is four colorable. Part I: Discharging", *Illinois Journal of Mathematics*, v. 21, n. 3, pp. 429–490.

[3] BARABÁSI, A.-L., ALBERT, R., 1999, "Emergence of scaling in random networks", *science*, v. 286, n. 5439, pp. 509–512.

[4] BLÖCHLIGER, I., ZUFFEREY, N., 2008, "A graph coloring heuristic using partial solutions and a reactive tabu scheme", *Computers & Operations Research*, v. 35, n. 3, pp. 960–975.

[5] BOLLOBÁS, B., 1988, "The chromatic number of random graphs", *Combinatorica*, v. 8, n. 1, pp. 49–55.

[6] BOSER, B. E., GUYON, I. M., VAPNIK, V. N., 1992, "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152. ACM.

[7] BRAZDIL, P., CARRIER, C. G., SOARES, C., et al., 2008, *Metalearning: applications to data mining*. Springer.

[8] BREIMAN, L., 2001, "Random forests", *Machine learning*, v. 45, n. 1, pp. 5–32.

[9] BRÉLAZ, D., 1979, "New methods to color the vertices of a graph", *Communications of the ACM*, v. 22, n. 4, pp. 251–256.

[10] CHANG, C.-C., LIN, C.-J., 2011, "LIBSVM: A library for support vector machines", *ACM Transactions on Intelligent Systems and Technology (TIST)*, v. 2, n. 3, pp. 27.

[11] CHIARANDINI, M., STÜTZLE, T., 2010, "An analysis of heuristics for vertex colouring". In: *Experimental Algorithms*, Springer, pp. 326–337.

[12] COLORNI, A., DORIGO, M., MANIEZZO, V., et al., 1991, "Distributed optimization by ant colonies". In: *Proceedings of the first European conference on artificial life*, v. 142, pp. 134–142. Paris, France.

[13] COOK, S. A., 1971, "The complexity of theorem-proving procedures". In: *Proceedings of the third annual ACM symposium on Theory of computing*, pp. 151–158. ACM.

[14] CORTES, C., VAPNIK, V., 1995, "Support-vector networks", *Machine learning*, v. 20, n. 3, pp. 273–297.

[15] CULBERSON, J., BEACHAM, A., PAPP, D., 1995, "Hiding our colors". In: *CP'95 Workshop on Studying and Solving Really Hard Problems*, pp. 31–42. Citeseer.

[16] DOWSLAND, K. A., THOMPSON, J. M., 2008, "An improved ant colony optimisation heuristic for graph colouring", *Discrete Applied Mathematics*, v. 156, n. 3, pp. 313–324.

[17] ERDŐS, P., RÉNYI, A., 1959, "On random graphs", *Publicationes Mathematicae Debrecen*, v. 6, pp. 290–297.

[18] FERNÁNDEZ-DELGADO, M., CERNADAS, E., BARRO, S., et al., 2014, "Do we need hundreds of classifiers to solve real world classification problems?" *The Journal of Machine Learning Research*, v. 15, n. 1, pp. 3133–3181.

[19] GALINIER, P., HAO, J.-K., 1999, "Hybrid evolutionary algorithms for graph coloring", *Journal of combinatorial optimization*, v. 3, n. 4, pp. 379–397.

[20] GAREY, M. R., JOHNSON, D. S., 1990, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA, W. H. Freeman & Co. ISBN: 0716710455.

[21] GILBERT, E. N., 1959, "Random graphs", *The Annals of Mathematical Statistics*, v. 30, n. 4, pp. 1141–1144.

[22] GLOVER, F., 1986, "Future paths for integer programming and links to artificial intelligence", *Computers & Operations Research*, v. 13, n. 5, pp. 533–549.

[23] GRIMMETT, G. R., MCDIARMID, C. J., 1975, "On colouring random graphs". In: *Mathematical Proceedings of the Cambridge Philosophical Society*, v. 77, pp. 313–324. Cambridge Univ Press.

[24] HAGBERG, A., SCHULT, D., SWART, P., et al., 2004, "Networkx. High productivity software for complex networks", *Webová strá nka https://networkx. lanl. gov/wiki*.

[25] HALL, M., FRANK, E., HOLMES, G., et al., 2009, "The WEKA data mining software: an update", *ACM SIGKDD explorations newsletter*, v. 11, n. 1, pp. 10–18.

[26] HERTZ, A., DE WERRA, D., 1987, "Using tabu search techniques for graph coloring", *Computing*, v. 39, n. 4, pp. 345–351.

[27] HO, T. K., 1995, "Random decision forests". In: *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, v. 1, pp. 278–282. IEEE.

[28] HUTTER, F., XU, L., HOOS, H. H., et al., 2014, "Algorithm runtime prediction: Methods & evaluation", *Artificial Intelligence*, v. 206, pp. 79–111.

[29] IHAKA, R., GENTLEMAN, R., 1996, "R: a language for data analysis and graphics", *Journal of computational and graphical statistics*, v. 5, n. 3, pp. 299–314.

[30] JOHNSON, D. S., TRICK, M. A., 1996, *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*, v. 26. American Mathematical Soc.

[31] JOHNSON, D. S., ARAGON, C. R., MCGEOCH, L. A., et al., 1991, "Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning", *Operations research*, v. 39, n. 3, pp. 378–406.

[32] KANDA, J., CARVALHO, A., HRUSCHKA, E., et al., 2011, "Selection of algorithms to solve traveling salesman problems using meta-learning", *International Journal of Hybrid Intelligent Systems*, v. 8, n. 3, pp. 117–128.

[33] KARP, R. M., 1972, *Reducibility among combinatorial problems*. Springer.

[34] KIRKPATRICK, S., 1984, "Optimization by simulated annealing: Quantitative studies", *Journal of statistical physics*, v. 34, n. 5-6, pp. 975–986.

[35] KORMAN, S. M., 1979, "The graph-colouring problem", *Combinatorial Optimization. N. Christofides. A. Mingozzi, P. Toth, and C. Sandi. Eds., Wiley, New York*, pp. 211–235.

[36] KOTTHOFF, L., 2015. "Algorithm Selection literature summary". Disponível em: <http://larskotthoff.github.io/assurvey/>.

[37] LEIGHTON, F. T., 1979, "A graph coloring algorithm for large scheduling problems", *Journal of research of the national bureau of standards*, v. 84, n. 6, pp. 489–506.

[38] LEWIS, R., 2016, *A Guide to Graph Colouring - Algorithms and Applications*. Springer.

[39] LEWIS, R., THOMPSON, J., MUMFORD, C., et al., 2012, "A wide-ranging computational comparison of high-performance graph colouring algorithms", *Computers & Operations Research*, v. 39, n. 9, pp. 1933–1950.

[40] LEYTON-BROWN, K., NUDELMAN, E., SHOHAM, Y., 2002, "Learning the empirical hardness of optimization problems: The case of combinatorial auctions". In: *Principles and Practice of Constraint Programming-CP 2002*, pp. 556–572. Springer.

[41] LIAW, A., WIENER, M., 2002, "Classification and regression by randomForest", *R news*, v. 2, n. 3, pp. 18–22.

[42] MCDIARMID, C. J., MÜLLER, T., 2005, "Colouring random geometric graphs". In: *2005 European Conference on Combinatorics, Graph Theory and Applications (EuroComb'05), volume AE of DMTCS Proceedings*, pp. 1–4.

[43] MESSELIS, T., DE CAUSMAECKER, P., 2014, "An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem", *European Journal of Operational Research*, v. 233, n. 3, pp. 511–528.

[44] MICHIE, D., SPIEGELHALTER, D. J., TAYLOR, C., 1994. "Machine Learning, Neural and Statistical Classification". .

[45] MUND, S., 2015, *Microsoft Azure Machine Learning*. Packt Publishing.

[46] MUSLIU, N., SCHWENGERER, M., 2013, "Algorithm selection for the graph coloring problem". In: *Learning and Intelligent Optimization*, Springer, pp. 389–403.

[47] NUDELMAN, E., LEYTON-BROWN, K., DEVKAR, A., et al., 2004, "Satzilla: An algorithm portfolio for SAT", *Solver description, SAT competition*, v. 2004.

[48] OF WAIKATO, T. U., 2015. "WEKA Wiki: ARFF". Disponível em: <http://weka.wikispaces.com/ARFF>.

[49] READ, J., REUTEMANN, P., 2012, "MEKA: a multi-label extension to WEKA", *URL http://meka. sourceforge. net*.

[50] READ, J., PFAHRINGER, B., HOLMES, G., et al., 2011, "Classifier chains for multi-label classification", *Machine learning*, v. 85, n. 3, pp. 333–359.

[51] RICE, J. R., 1975, *The algorithm selection problem*. Relatório Técnico 99, Department of Computer Science at Purdue University. Report Number 75-152. This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

[52] RUSSEL, S., NORVIG, P., 2010, *Artificial Intelligence: A Modern Approach*. Pearson Education.

[53] SCHWENGERER, M., MUSLIU, N., 2012, *Algorithm Selection for the Graph Coloring Problem*. M.Sc. dissertation, Faculty of Informatics at Vienna University of Technology.

[54] SMITH-MILES, K., BAATAR, D., 2013, "Exploring the role of graph spectra in graph coloring algorithm performance", *Discrete Applied Mathematics*.

[55] SMITH-MILES, K., WREFORD, B., LOPES, L., et al., 2013, "Predicting metaheuristic performance on graph coloring problems using data mining". In: *Hybrid Metaheuristics*, Springer, pp. 417–432.

[56] SMITH-MILES, K., BAATAR, D., WREFORD, B., et al., 2014, "Towards objective measures of algorithm performance across instance space", *Computers and Operations Research*, v. 45, n. 0, pp. 12 – 24.

[57] SMITH-MILES, K. A., 2008, "Cross-disciplinary perspectives on meta-learning for algorithm selection", *ACM Computing Surveys (CSUR)*, v. 41, n. 1, pp. 6.

[58] TSOUMAKAS, G., KATAKIS, I., 2006, "Multi-label classification: An overview", *Dept. of Informatics, Aristotle University of Thessaloniki, Greece*.

[59] TSOUMAKAS, G., KATAKIS, I., VLAHAVAS, I., 2011, "Random k-labelsets for multilabel classification", *Knowledge and Data Engineering, IEEE Transactions on*, v. 23, n. 7, pp. 1079–1089.

[60] TURING, A. M., 1950, "Computing machinery and intelligence", *Mind*, pp. 433–460.

[61] TURNER, J. S., 1988, "Almost all k-colorable graphs are easy to color", *Journal of Algorithms*, v. 9, n. 1, pp. 63–82.

[62] VAPNIK, V., 1963, "Pattern recognition using generalized portrait method", *Automation and remote control*, v. 24, pp. 774–780.

[63] WITTEN, I. H., FRANK, E., 2005, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

[64] WOLPERT, D. H., 1996, "The lack of a priori distinctions between learning algorithms", *Neural computation*, v. 8, n. 7, pp. 1341–1390.

[65] XU, K., LI, W., 1999, "The SAT phase transition", *Science in China Series E: Technological Sciences*, v. 42, n. 5, pp. 494–501.

[66] XU, L., HUTTER, F., HOOS, H. H., et al., 2007, "SATzilla-07: The design and analysis of an algorithm portfolio for SAT". In: *Principles and Practice of Constraint Programming–CP 2007*, Springer, pp. 712–727.

[67] XU, L., HUTTER, F., HOOS, H. H., et al., 2008, "SATzilla: Portfolio-based Algorithm Selection for SAT." *J. Artif. Intell. Res.(JAIR)*, v. 32, pp. 565–606.

Appendix A

Programs and Scripts

Listing A.1: Script to generate and convert Barabási-Albert models to DIMACs format

```
import networkx as nx

for count in range(1,101):
    G=nx.barabasi_albert_graph(500,1)
    G=nx.convert_node_labels_to_integers(G,1)
    f = open("barabasi_albert_500_1-"+str(count)+".col",'w')
    f.write("p edge "+str(nx.number_of_nodes(G))+" "+str(nx.
       number_of_edges(G))+"\n")
    for line in nx.generate_edgelist(G, data=False):
        f.write("e "+line+"\n")
    f.close()
```

Listing A.2: Partial source code to extract features from a graph

```
    import networkx as nx
    import numpy as np
    from decimal import *

    class Features(object):

        def __init__(self, graphFilePath):
            self.graphFilePathDimacs = graphFilePath
            self.graphFilePath = graphFilePath+'.edl'
            self.G=nx.read_edgelist(graphFilePath+'.edl',
               nodetype=int)
            self.MAXDECIMAL = float
```

```
            ('99999999999999999.9999999999')

    ## Nodes and Edges Features ###
        def NumberOfNodes(self) :
            return nx.number_of_nodes(self.G)

        def NumberOfEdges(self) :
            return nx.number_of_edges(self.G)

        def RatioNodesEdges(self) :
            return float(nx.number_of_nodes(self.G))/nx.
               number_of_edges(self.G)

        def RatioEdgesNodes(self) :
            return float(nx.number_of_edges(self.G))/nx.
               number_of_nodes(self.G)

        def Density(self) :
            return nx.density(self.G)

    ### Degree Features ###

        def ComputeDegree(self):
            self.degreeList = np.array(nx.degree(self.G).
               values())

        def MinDegree(self) :
            return min(self.degreeList)

        def MaxDegree(self) :
            return max(self.degreeList)

        def MeanDegree(self) :
            return np.mean(self.degreeList)

        def StdDegree(self) :
            return np.std(self.degreeList)
```

Figure A.1: Graph Coloring Algorithm Performance Database Model