Discrete Applied Mathematics 176 (2014) 107–121

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Exploring the role of graph spectra in graph coloring algorithm performance

Kate Smith-Miles[*], Davaatseren Baatar

School of Mathematical Sciences, Monash University, Victoria 3800, Australia

A R T I C L E I N F O

Article history:
Received 23 November 2012
Received in revised form 5 November 2013
Accepted 9 November 2013
Available online 26 November 2013

Keywords:
Graph spectra
Graph invariants
Graph coloring
Branch and bound
DSATUR
LP-relaxation
Algorithm selection

A B S T R A C T

This paper considers the challenge of recognizing how the properties of a graph determine the performance of graph coloring algorithms. In particular, we examine how spectral properties of the graph make the graph coloring task easy or hard. We address the question of when an exact algorithm is likely to provide a solution in a reasonable computation time, when we are better off using a quick heuristic, and how the properties of the graph influence algorithm performance. A new methodology is introduced to visualize a collection of graphs in an instance space defined by their properties, with an optimal feature selection approach included to ensure that the instance space preserves the topology of an algorithm performance metric. In this instance space we can reveal how different algorithms perform across a variety of instance classes with various properties. We demonstrate the methodology using a sophisticated branch and bound exact algorithm, and the faster heuristic approaches of integer rounding of a linear programming relaxation of the graph coloring formulation, as well as a greedy degree-saturation heuristic. Our results demonstrate that spectral properties of the graph instances can provide useful descriptions of when each of these algorithms will provide the best solution for a given computational effort.

1. Introduction

It is a commonly held view that exact algorithms are only likely to solve small instances of NP-hard optimization problems within a reasonable computation time, and that heuristics are necessary for larger instances [68]. Likewise, it is frequently assumed that heuristics are unlikely to give exact solutions, but will quickly converge to near-optimal solutions. Both of these assumptions should be challenged, since instance size is not the only determinant of whether an exact algorithm will find an instance easy or hard to solve, nor whether a heuristic will find a near-optimal solution. Much research has been conducted on what makes optimization problems hard [38] and the role of certain properties of the instances that contribute to phase transition boundaries [14,23,1,28], whereby an instance becomes harder to solve as a key property of the instance is changed, such as edge density in graph coloring [1]. A recent survey has described a comprehensive collection of features that determine hardness for a wide range of common optimization problems [54], and showed that instance size is frequently not the only significant feature, with other properties like correlation coefficients playing an important role [3,30].

The No-Free-Lunch (NFL) Theorems [64,17] tell us that any general algorithm will find some instance classes difficult compared to another algorithm that exploits some prior knowledge of the characteristics of an instance class. From this idea has come an approach known as *algorithm portfolios* [37,36], whereby training data is used to build a model that predicts

[*] Corresponding author. Tel.: +61 3 99053170; fax: +61 3 99054403.
E-mail addresses: kate.smith-miles@monash.edu, kate.smithmiles@gmail.com (K. Smith-Miles).

0166-218X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.dam.2013.11.005

how each of a collection of algorithms is likely to perform for a given instance, based on measurable properties, and the model is then used on unseen data to predict the best algorithm. This approach has proven to be a powerful one, winning various competitions [66,67,43,41], and helps to circumvent the NFL Theorem by ensuring that knowledge of the instance properties is considered by a meta-algorithm.

These ideas date back to the 1970s, when the Algorithm Selection Problem (ASP) was first posed by Rice [49], and formulated as the task of learning the relationship between a set of features describing problem classes, and the performance of algorithms. The approach was then applied to the task of selecting the best solver for partial differential equations and numerical quadrature [61,32,33,47]. Independently, the same ideas have been applied in the machine learning community, to select the best algorithm for solving classification [10,2] and prediction problems [46,63]—a sub-field known as meta-learning [62] since the approach is to learn about learning algorithm performance. A discussion about how these ideas have been applied to a variety of fields of research can be found in Smith-Miles [51].

More recently, we have been applying these ideas to understand how optimization algorithm performance is affected by instance properties, considering the Travelling Salesman Problem [56], Scheduling [52], timetabling [53], quadratic assignment problems [50], and graph coloring [57]. The latter paper considered two heuristics: a degree saturation graph coloring heuristic called DSATUR [65] and a tabu search heuristic downloaded from Culberson's web resources for graph coloring [18]. The analysis showed that the best algorithm for the five sets of instance classes from Culberson's instance generators [18] could be predicted quite accurately (more than 93% accuracy on out-of-sample testing) using some simple properties of graphs. The region in instance space where each algorithm outperforms the other is known as the *algorithm footprint* [16], and we have previously provided a methodology for quantifying the area of each algorithm's footprint as a comparative measure of performance [55].

In this paper, we extend this work to consider how such an approach can be used to determine when an exact algorithm can be expected to deliver a solution within a reasonable computation time, and when a quick heuristic might be expected to be more productive. If a heuristic can obtain the same solution as an exact approach in a much quicker time, then the exact algorithm is unnecessary and not worth the wait. We ask the question, *under what conditions of a graph instance can we expect that a sophisticated exact algorithm will be worth the computational effort*? Our exact algorithm is a branch-and-bound approach [39] which utilizes the DSATUR heuristic and involves solving the linear programming (LP) relaxation at the root node. So we can consider that this sophisticated exact algorithm has some exit points where we could terminate with a heuristic solution. In this paper we also extend the collection of properties of the graph to consider many spectral features, as well as other invariant graph properties that have been proposed recently to enable new conjectures to be formed about graphs [12]. We augment our previous methodology to include, not just machine learning of the relationships between features and algorithm performance, but a visual exploration of a well-defined instance space. This instance space is created by performing optimal feature selection in a manner that projects the instances to a two-dimensional space where the algorithm performance has been topologically preserved. Inspection of the distribution of features across this instance space permits insights to be formed about how certain features are influencing algorithm performance in a way that is hidden from standard machine learning approaches to predicting the winning algorithm.

In Section 2 we briefly describe the task of graph coloring and discuss the branch-and-bound algorithm and heuristics we consider in this paper. In Section 3 we then present the framework of the Algorithm Selection Problem [49,51] that enables our graph coloring algorithm investigation to be described, including the instances, features, algorithms, and performance metrics. Section 4 proposes the methodology for generating a topology preserving instance space via optimal feature selection to enable clear visualization of the algorithm footprints and the impact of the chosen features. Experimental results are presented in Section 5 to demonstrate the methodology, and Conclusions and future research directions are discussed in Section 6.

2. Graph coloring algorithms

A graph $G = (V, E)$ comprises a set of vertices V and a set of edges E that connect certain pairs of vertices. The graph coloring problem (GCP) is to assign colors to the vertices, minimizing the number of colors used, subject to the constraint that two vertices connected by an edge (called adjacent vertices) do not share the same color. The optimal (minimal) number of colors needed to solve this NP-complete problem is called the chromatic number of the graph. Graph coloring finds important applications in problems such as timetabling, where events to be scheduled are represented as vertices, with edges representing conflicts between events, and the color representing the time period [13,19].

Due to the NP-completeness of this problem, many heuristics have been designed [45,22]. One of the earliest heuristics was *DSATUR* [11,65], which was shown to be exact for bipartite graphs. The saturation degree of a vertex is defined to be the number of different colors to which it is adjacent. The DSATUR (degree saturation) heuristic is a simple approach to coloring a graph is shown in Algorithm 1.

For non-bipartite graphs though, the performance of DSATUR is not optimal, and many more sophisticated search strategies have been employed to provide effective heuristics for general graphs, including tabu search [29], simulated annealing [34], iterated local search [15], scatter search [26], genetic algorithms [21], and hybrid approaches [20,9]. Just as the performance of DSATUR depends on the bipartivity of the graph, we should also expect the performance of any method to depend on various properties of the graph, but the relationship between the properties of the graph and the performance