Towards Objective Measures of Algorithm Performance Across Instance Space

Kate Smith-Miles*, Davaatseren Baatar, Brendan Wreford

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

Rhyd Lewis

School of Mathematics, Cardiff University, Wales

Abstract

This paper tackles the difficult but important task of objective algorithm performance assessment for optimization. Rather than reporting average performance of algorithms across a set of chosen instances, which may bias conclusions, we propose a methodology to enable the strengths and weaknesses of different optimization algorithms to be compared across a broader instance space. The results reported in a recent *Computers and Operations Research* paper comparing the performance of graph coloring heuristics are revisited with this new methodology to demonstrate i) how pockets of the instance space can be found where algorithm performance varies significantly from the average performance of an algorithm; ii) how the properties of the instances can be used to predict algorithm performance on previously unseen instances with high accuracy; and iii) how the relative strengths and weaknesses of each algorithm can be visualized and measured objectively.

Keywords: comparative analysis, heuristics, graph coloring, algorithm selection, performance prediction

* Corresponding author: kate.smith-miles@monash.edu
Phone: +61 3 99053170
Fax: +61 3 99054403

1. Introduction

Objective assessment of optimization algorithm performance is notoriously difficult [1, 2], especially when the conclusions depend so heavily on the chosen test instances of the optimization problem. The popular use of benchmark libraries of instances (e.g. the OR-Library [3]) helps to standardize the testing of algorithms, but may not be sufficient to reveal the true strengths and weaknesses of algorithms. As cautioned by Hooker [1, 2] nearly two decades ago, there is a need to be careful about the conclusions that can be drawn beyond the selected instances. It has been documented that there are some optimization problems where the benchmark library instances are not very diverse [4] and there is a danger that algorithms are developed and tuned to perform well on these instances without understanding the performance that can be expected on instances with diverse properties. Furthemore, while the peer-review process usually ensures that standard benchmark instances are used for well-studied problems, for many real-world or more unusual optimisation problems there is a lack of benchmark instances, and a tendency for papers to be published that report algorithm performance based only on a small set of instances presented by the authors. Such papers typically are able to demonstrate that the new algorithm proposed by the authors outperforms other previously published approaches (it is difficult to get published otherwise), and the choice of instances cannot be challenged due to the lack of alternative instances.

The No-Free-Lunch (NFL) Theorems [5, 6] state that all optimization algorithms have identically distributed performance when objective functions are drawn uniformly at random, and all algorithms have identical mean performance across the set of all optimisation problems. Does this idea apply also to different instances of a particular optimization problem, which give rise to only a subset of possible objective functions? Probably not [7], but it still seems unwise to believe that any one optimization algorithm will always be superior for all possible instances of a given problem. We should expect that any algorithm has weaknesses, and that some instances could be conceived where the algorithm would be less effective than its competitors, or at least instances exist where their competitive advantage disappears. Our current research culture, where negative results are seen as somehow less of a contribution than positive ones, means that the true strengths *and* weaknesses of an optimisation algorithm are rarely exposed and reported. Yet for advancement of the field, surely we must find a way to make it easier for researchers to report the strengths and weaknesses of their algorithms. On which types of instances does an algorithm outperform its competitors? Where is it less effective? How can we describe those instances?

Occasionally we find a paper that presents a well-defined class of instances where an algorithm performs well, and reports its failing outside this class (see [8] for a recent example). Such studies assist our understanding of an algorithm and its applicability. Does the class of instances where an algorithm is effective overlap real-world or other interesting instances? Is an algorithm only effective on instances where its competitors are also effective, or are there some classes where it is uniquely powerful? How do the properties of the instances affect algorithm performance? Until we develop the tools to enable researchers to quickly and easily determine the instances they need to consider to enable the boundary of effective algorithm performance to be described and quantified in terms of the properties of the instances, the objectivity of algorithm performance assessment will always be compromised with sample bias.

Recently, we have been developing the components of such a methodology [9]. Instances are represented as points in a high-dimensional feature space, with features chosen intentionally to tease out the similarities and differences between instance classes. For many broad classes of optimization problems, a rich set of features have already been identified that can be used to summarize the properties of instances affecting instance difficulty (see Smith-Miles and Lopes [10] for a survey of suitable features). Representing all available instances of an optimization problem in a single space in this manner can often reveal inadequacies in the diversity of the test instances. We can observe for some problems that benchmark instances appear to be structurally similar to randomly generated instances, eliciting similar performance from algorithms, and are not well designed for testing the strengths and weaknesses of algorithms. We have previously proposed the use of evolutionary algorithms to intentionally construct instances that are easy or hard for specific algorithms [11], thereby guaranteeing diversity of the instance set. Once we have sufficient instances covering most regions of the high-dimensional feature space, we need to be able to superimpose algorithm performance in this space and visualize the boundaries of good performance. Using dimensional reduction techniques such as principal component analysis, we have previously proposed projecting all instances to a two-dimensional "*instance space*" [9] where we can visualize the region where an algorithm can be expected to perform well based on generalization of its observable performance on the test instances. We call this region the *algorithm footprint* in instance space, and the relative size and uniqueness of an algorithm's footprint can be used as an objective measure of algorithm power. Inspection of the distribution of individual features across the instance space can also be used to generate new insights into how the properties of instances affect algorithm performance, and machine learning techniques can be employed in the feature space (or instance space) to predict algorithm performance on unseen instances [12]. Over the last few years we have applied components of this broad methodology to a series of optimization problems including the Travelling Salesman Problem [9, 11, 13], Job-Shop Scheduling [14], Quadratic Assignment Problem [15], Graph Coloring [12, 16], and Timetabling Problems [17, 18].

While our previous research has generated an initial methodology, it has raised a number of questions that need to be addressed for a more comprehensive tool to be developed: How should we select the right features to represent the instance space most effectively? How can we determine the sufficiency and diversity of the set of instances? Can we more accurately predict algorithm performance in the high-dimensional feature space or the projected two-dimensional space? How should we determine the boundary of where we expect an algorithm to perform well based on limited observations? How can we reveal the strengths and weaknesses of a portfolio of algorithms, as well as their unique strengths and weaknesses within the portfolio.

This paper extends the methodology that has been under development for the last few years by addressing these last remaining questions. We demonstrate the use of the methodology by applying it to some computational results reported recently for an extensive comparison of graph coloring heuristics [19]. This case study reveals insights into the relative powers of the chosen optimization algorithms that were not apparent by considering performance averaged across all chosen instances.

The remainder of this paper is as follows: in Section 2 we present the framework upon which our methodology rests – the Algorithm Selection Problem [20] – which considers the relationships between the instance set, features, algorithms, and performance metrics. The detailed steps of the methodology are then described in Section 3, after proposing solutions to the questions raised above. In Section 4, we present a graph coloring case study based on the computational experiments of Lewis *et al.* [19] and discuss the new insights that the methodology has generated. Our conclusions are presented in Section 5, along with suggestions for use of the methodology and future research directions.

2. Framework: The Algorithm Selection Problem

In 1976, Rice [20] proposed a framework for the Algorithm Selection Problem (ASP), which seeks to predict which algorithm from a portfolio is likely to perform best based on measurable features of problem instances. While Rice's focus was not on optimisation algorithms, instead applying this approach to predict the performance of partial differential equation solvers [21, 22], the framework is one that is readily generalizable to other domains (see the survey paper by Smith-Miles [23] for a review). There are four essential components of the model:

- the problem space $\mathcal{P}$ represents a possibly infinitely-sized set of instances of a problem;
- the feature space $\mathcal{F}$ contains measurable characteristics of the instances generated by a computational feature extraction process applied to $\mathcal{P}$;
- the algorithm space $\mathcal{A}$ is a set (portfolio) of algorithms available to solve the problem;
- the performance space $\mathcal{Y}$ represents the mapping of each algorithm to a set of performance metrics.

For performance prediction, we need to find a mechanism for generating the mapping from feature space to algorithm space. The Algorithm Selection Problem can be formally stated as: for a given problem instance $x \in \mathcal{P}$, with feature vector $f(x) \in \mathcal{F}$, find the selection mapping $S(f(x))$ into algorithm space $\mathcal{A}$, such that the selected algorithm $\alpha \in \mathcal{A}$ maximizes the performance mapping

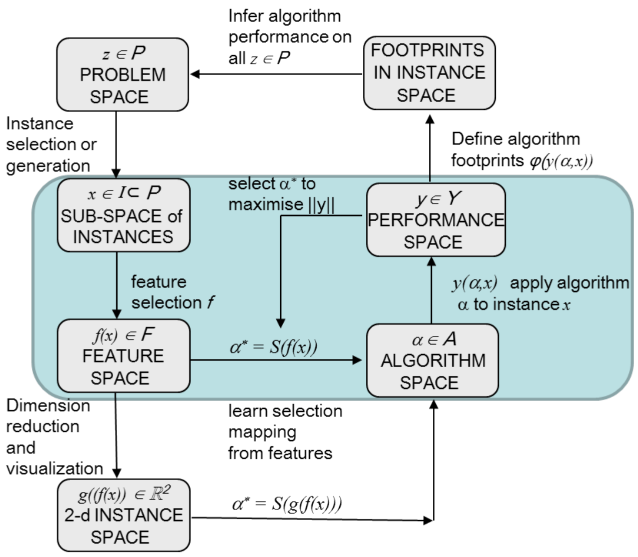
Infer algorithm performance on all $z \in P$
$z \in P$ PROBLEM SPACE
FOOTPRINTS IN INSTANCE SPACE
Instance selection or generation
Define algorithm footprints $\varphi(y(\alpha,x))$
$x \in I \subset P$ SUB-SPACE of INSTANCES
select α^* to maximise ||y||
$y \in Y$ PERFORMANCE SPACE
feature selection f
$y(\alpha,x)$ apply algorithm α to instance x
$f(x) \in F$ FEATURE SPACE
$\alpha^* = S(f(x))$
$\alpha \in A$ ALGORITHM SPACE
Dimension reduction and visualization
learn selection mapping from features
$g((f(x)) \in \mathbb{R}^2$ 2-d INSTANCE SPACE
$\alpha^* = S(g(f(x)))$

Figure 1: Methodological framework, extending the Algorithm Selection Problem of Rice (1976), shown in the box

$y(\alpha, x) \in \mathcal{Y}$. The collection of data describing $\{\mathcal{P}, \mathcal{F}, \mathcal{A}, \mathcal{Y},\}$ is known as the *meta-data*. Analysis of the meta-data, in particular using statistical and machine learning techniques to learn the mapping S, between features of instances and the performance of algorithms, has been used effectively in algorithm portfolio approaches [14, 15, 17, 24, 25, 26] to predict the algorithm likely to perform best for unseen instances.

In this research, we utilize the framework of Rice, but extend it to consider our broader agenda: we are not simply concerned with identifying the winning algorithm, but how to use the meta-data to identify the strengths and weaknesses of algorithms, and to visualize and quantify the relative power of algorithms. While our focus is on optimisation algorithms, we will retain the generic nature of the framework, and highlight where domain-specific considerations apply. Figure 1 presents the framework for our proposed methodology.

Generating and analyzing the meta-data $\{\mathcal{P}, \mathcal{F}, \mathcal{A}, \mathcal{Y},\}$ is central to the framework, as shown in the shaded central box describing Rice's Algorithm Selection Problem framework, but we extend the framework in each direction in order to address our broader objectives.

We start by acknowledging that Rice's problem space $\mathcal{P}$ should really be considered as the set of all possible instances of a problem, and not just the subset of instances $\mathcal{I} \subset \mathcal{P}$ for which we have computational results. We therefore must consider how the subset of instances used for the meta-data are selected or generated. In order to learn the boundaries of algorithm performance, we must include instances that are easy and hard for each algorithm, and we may need to take steps through careful instance generation mechanisms to produce such instances if the diversity of the set of instances $\mathcal{I}$ is insufficient. Measuring such sufficiency will rely on being able to i) visualize the instances in a common space, and verify that they are spatially diverse and therefore dissimilar; and ii) ensuring that the instances are discriminating of algorithm performance, and are not equally easy or equally hard for all algorithms, which would tell us nothing about the relative strengths and weaknesses of algorithms.

The diversity of the instances depends on how we are measuring their properties or features. If we consider only simple properties (such as the number of cities in a Travelling Salesman Problem instance), then many genuinely diverse instances will appear similar. As the sophistication of the feature set increases we develop greater ability to discriminate between instances in a high dimensional feature space. While a large number of candidate features can be found in the literature for many common optimization problems [10], it can be somewhat of an art to select the subset of features that create the most useful feature space and resulting meta-data. Rice's framework provides no advice about the mapping from problem space $\mathcal{P}$ to the feature space $\mathcal{F}$, but acknowledges "the way problem features affect methods is complex and algorithm selection might depend in an unstable way on the features actually used" [22]. While the construction of suitable features cannot be incorporated readily into Rice's abstract model, largely due to the problem specific nature of the feature construction process, we acknowledge here the criticality of the task of constructing and selecting suitable features that adequately measure the relative difficulty of the instances for different algorithms. The feature selection process is successful if easy and hard instances are easily separable in the high dimensional feature space. Our methodology includes a feature subset selection method that achieves the desired outcomes for a useful transformation of the problem space $\mathcal{P}$ to the instance feature space $\mathcal{F}$.

Much can be learned from the meta-data $\{\mathcal{P}, \mathcal{F}, \mathcal{A}, \mathcal{Y}\}$ (or more correctly, $\{\mathcal{I}, \mathcal{F}, \mathcal{A}, \mathcal{Y}\}$) operating in this high-dimensional feature space. Machine learning methods can be employed to predict algorithm performance in terms of the feature vector describing an instance, and out-of-sample testing can be used to demonstrate predictive power. For effective visualization of the similarities and differences between instance classes, and the performance of algorithms across this common space of instances, we propose to employ dimension reduction techniques to produce a two-dimensional projection of the instances in a common space defined by the features. Many dimension reduction methods are suitable for this task, including Principal Component Analysis [27], self-organizing feature maps [28], etc. Whichever method is chosen, an instance x is projected from its location in the high-dimensional feature space (given by the vector position $f(x) \in \mathcal{F}$) to a point in $\mathbb{R}^2$ via the (linear or non-linear) mapping g. This two-dimensional instance space therefore defines the transformation of all points $z \in \mathcal{P} \xrightarrow{f} \mathcal{F} \xrightarrow{g} \mathbb{R}^2$.

The observable meta-data $\{\mathcal{I}, \mathcal{F}, \mathcal{A}, \mathcal{Y}\}$ can be visualized within this two-dimensional instance space. In particular, the location of the instances $x \in \mathcal{I} \subset \mathcal{P}$ can be inspected in $\mathbb{R}^2$ to confirm that the diversity requirement has been established by the choice of instances and the feature selection. The performance of algorithms across the instance space can also be visualized to establish the degree to which the selected instances are discriminating of algorithm performance. In fact, the separability of instances and discriminatory algorithm performance in the instance space is a criteria we will use to guide the search for an optimal subset of candidate features, and the sufficiency of the subset of instances in the meta-data.

The meta-data contains information about the performance that each algorithm exhibits on each instance in the subset $\mathcal{I}$. Measuring algorithm performance is a complex task since there are frequently many factors to consider when defining a superior algorithm: the time required to find a high quality solution; the quality of the solution found after a fixed amount of computational effort; the relative performance compared to other algorithms; the performance compared to a theoretical bound. Supposing that a user has defined how they intend to measure good performance, it is then straightforward to evaluate the meta-data and determine, for each algorithm, the set of instances where the good performance has been observed. But what about the parts of the instance space where there is no observable instance in the meta-data? What can be said about whether such regions lie within the footprint of an algorithm? This is a statistical generalization issue. Our methodology proposes an approach to identify the regions in instance space where we have sufficient evidence that good performance can be expected from an algorithm, and calculates the area of this algorithm footprint, along with its mathematical boundary $\varphi(y, \alpha, x)$. This algorithm footprint enables us to then infer the performance that we can expect from all instances $z \in \mathcal{P}$, and provides the necessary tool to report the likely strengths and weaknesses of optimization algorithms across all instances. Clearly, the confidence we have in the statistical generalization depends immensely on ensuring that our selected instances $\mathcal{I} \subset \mathcal{P}$ are unbiased and that our feature selection and dimension reduction processes have preserved most of the important topological relationships between the instances.

It is from within this framework that we propose the following methodology to develop our computational resource for the operations research community. It should be noted that the only component of the framework that is specific to optimisation problems is the choice of candidate features to summarize the instances. All other components of the framework, and consequential methodology, are generic and applicable to a wide variety of problem domains beyond optimisation.

3. Methodology

The proposed methodology comprises three stages:

1. Generating the instance space - a process whereby instances are selected, their features calculated, and an optimal subset of features is generated to create a high-dimensional summary of the instances in feature space that, when projected to the two-dimensional instance space, achieves good separation of the easy and hard instances;

2. Algorithm performance prediction - using the location of an instance within the instance space, machine learning methods are used to classify the regions where an algorithm is predicted to perform well or poorly, and to identify which algorithm is recommended for which regions of the instance space;
3. Analysis of algorithmic power - the size and location of each algorithm's footprint can be measured objectively, and conclusions can be drawn about relative algorithmic power. Insights can be gained from this visualization to explain algorithm strength or weakness by inspecting the distribution of features across the instance space.

Details of this methodology are now presented, before a case study is used in Section 4 to illustrate the methodology.

3.1. Generating the Instance Space

The ideal instance space is one that maps the available instances to a two-dimensional representation in such a way that the easy instances and hard instances are well separated. With this visualization of the instance space, we can then inspect the distribution of features across the space to understand how the features affect algorithm performance. Since we have multiple algorithms, an ideal instance space for one algorithm might not be ideal for another algorithm, so we must achieve some compromise to find a mapping of the instances to a single feature space in a way that achieves the best separability of easy and hard instances on average. Not only does the choice of features affect the resulting instance space, but the choice of instances - and their diversity - plays a major role in determining the breadth of the instance space and the variability it accommodates. If we are to learn about the boundaries of algorithm performance, then we must take steps to ensure that our instance space is as broad as possible.

Generating a suitable instance space for a particular type of optimization problem is therefore a complicated interplay between selecting diverse instances, measuring the right features that correlate with instance difficulty, and selecting the optimal subset of features that, when mapped to a two-dimensional instance space, produces the best separation of easy and hard instances averaged across all algorithms. We propose that a suitable instance space could be generated once for each class of optimization problem using the methodology presented here, and this could then be available as a resource for all researchers to explore how their algorithms perform in this space.

3.1.1. Instance selection or generation

Since our methodology involves statistical inference and machine learning, it should already be apparent that we require a large collection of instances of a problem. For many optimization problems, large collections of well-studied instances exist. If they are sufficiently diverse based on measurable features that correlate with difficulty, then that is probably sufficient for generating a useful instance space. If diversity is not achieved, then we must take steps to generate additional instances that extend the boundaries of the instance space. Instances that are extremely easy or hard for algorithms in the portfolio can be evolved using evolutionary algorithms, as we have done in our previous work [11]. Measuring the diversity of the instances as they appear in the resulting instance space requires two considerations:

1. instance dissimilarity - the instances should span a reasonable region in the instance space and not all be co-located in a small region. Suitable measures of instance dissimilarity include the average distance to the centroid of the instances, or the ratio of the area of the convex hull to the area of the rectangle containing the instances;
2. algorithmic discrimination - the instances should elicit different behaviours from the algorithms in the portfolio, with some being easy and others being hard, if we are to learn about the strengths and weaknesses of algorithms. Suitable measures of algorithmic discrimination include simple statistical metrics such as the relative difference between the best and worse performance metric averaged across all instances.

The diversity of the instances will not be apparent until the instance space has been created, which requires the features to be selected and instances to be projected to $\mathbb{R}^2$. In the event that the diversity of the selected instances is deemed insufficient, then additional instances will need to be intentionally generated, using evolutionary algorithms for example, to achieve the objective. This process is therefore iterative.

3.1.2. Feature selection

The features create our first transformation of the instances - from a collection of TSP distance matrices for example, each defining a unique TSP - to each instance being mapped to a point in a high-dimensional space. While much is known and reported about the features of instances that correlate with difficulty [10], we must consider that this list of candidate features is potentially infinite, and not all of them will be useful for our goal of creating a transformation of the instances that separates easy and hard instances. Generally, feature selection is a two-step process: firstly we need to define how we will measure the goodness of a particular subset of features, and once this metric is determined, we can utilise an optimization search strategy to find the subset $\mathcal{F}^*$ that maximizes the goodness metric. In this methodology, we define the goodness of a subset of features based on the extent to which instances that elicit similar performance of algorithms are close together in the instance space defined by the two-dimensional projection of the subset of features. In other words, for a candidate subset of features, we will measure the goodness by how well a machine learning method can discriminate between easy and hard instances in the high-dimensional feature space. We will also consider how the goodness changes if we measure discrimination in the two-dimensional projection of the instance space. We use Principal Component Analysis [27] to create the two-dimensional projection based on the candidate subset of features, and use a Naive Bayes classifer [29] as the machine learning technique, although other methods could be employed for both steps. A genetic algorithm [30] is then used to search the large space of possible subsets of m features, with this goodness measure (the classification accuracy on an out-of-sample test set) used as the fitness function to drive the search for the optimal subset of features $\mathcal{F}^* \subseteq F$. Among the best subsets for each value of m, we select the optimal subset as the one with the smallest test set classifier error. Other researchers have used similar intuitive ideas for feature subset selection for machine learning tasks [31, 32], but have utilized information theory metrics, particularly for the discrete classification task of identifying which features enable the class label defining best performing algorithm to be predicted most accurately. Certainly, a wide range of feature selection methods have been proposed in the literature (see [33] for a comprehensive review), and any of them would be a suitable alternative to the method employed in this paper.

It is an interesting question of whether the optimal subset of features is the same regardless of whether we are measuring the separation of easy and hard instances in the high-dimensional feature space or the two-dimensional instance space. We will examine this issue with our case study later, but now finish the description of the process for generating the instance space by describing the dimension reduction process for instance space visualization.

3.1.3. Visualization via dimension reduction

If our goal was just performance prediction, then, once we have selected the best m features as the subset $\mathcal{F}^*$, we can simply use machine learning methods to learn the relationship between the instance features in $\mathbb{R}^m$ and the algorithm performance labels (easy or hard). But in this research we have broader goals that include being able to visualize the instance space and the algorithm footprints, and so we need to utilize dimension reduction techniques to project the instances to $\mathbb{R}^2$, ensuring that we are not losing too much information. Here, we use Principal Component Analysis (PCA) [27], which essentially rotates the data to a new coordinate system in $\mathbb{R}^m$, with axes defined by m new features which are linear combinations of the m selected features in $\mathcal{F}^*$. These new axes are calculated as the eigenvectors of the $m \times m$ covariance matrix; we then project the instances on the two principal eigenvectors corresponding to the two largest eigenvalues of the covariance matrix. Reducing the description of each instance from a vector in $\mathbb{R}^m$ to a vector in $\mathbb{R}^2$ certainly costs us a loss of information, and we measure the loss in terms of how much of the variance found in the data cannot be explained by the first two eigenvectors. The first two eigenvalues explain a percentage of the variation in the data given by $\frac{(\lambda_1+\lambda_2)}{\sum_{i=1}^{m}\lambda_i}$. We will consider that the new two-dimensional instance space is an adequate representation of the original feature space if most of the variation in the data is explained by these two principal axes.

The iterative process involving these three steps enables a two-dimensional instance space to be created that retains most of the relationships between the instance set $\mathcal{I} \subset \mathcal{P}$ and the performance of the algorithms in the set $\mathcal{A}$ on those instances.

3.2. Algorithm performance prediction

The newly created two-dimensional instance space can be used for predicting algorithm performance, as can the high-dimensional instance space, but has advantages in terms of visualizing the results. For the task of performance prediction, we utilize standard machine learning methodologies that use a subset of the instances (the training set) to learn the relationship between the instance features (in either $\mathbb{R}^m$ or $\mathbb{R}^2$) and the label we assign each algorithm for each instance to indicate how well the algorithm performed. A variety of machine learning classification methods can be used, such as Naive Bayes classifiers or support vector machines, if our labels are binary (easy/hard); or we could use machine learning or statistical prediction methods, such as regression or neural networks, if our performance metrics are continuous valued, such as run-time to find an optimal solution, or optimality gap. Our choice of performance metric $\mathcal{Y}$ is user-defined, and machine learning methods are used to build a model to predict that performance metric, evaluated on unseen test set instances that have been randomly extracted from the available data $\mathcal{I} \subset \mathcal{P}$.

3.3. Analysis of algorithmic power

With a user-defined description of good algorithm performance, we can label each instance. In this paper we use a binary label of good or bad (easy or hard) for each algorithm in our portfolio, and then visualize the boundary in instance space between good and bad performance for each algorithm. Instances that lie within the boundary of good performance are deemed to be easy for the algorithm, even if those instance are not in the set $\mathcal{I} \subset \mathcal{P}$. The region in instance space where an algorithm is expected to perform well is called the *footprint*, and there are a number of methods we can consider to measure the relative size of one algorithm's footprint compared to another algorithm's, or compared to the span of all instances. In our previous work on this topic we used the area of the convex hull created by the points where good performance was observed to measure the size of the footprint for an algorithm [9]. We adopt an improved methodology here that considers the density of the points defining the footprint, and requires a defined level of purity (percentages of instances in the footprint that are easy for an algorithm) as summarized by Algorithm 1.

The relative size of each algorithm's footprint provides some objective measure of algorithm strength across the instance space, but it is also important to understand *where* in the instance space an algorithm is strong. If an algorithm is strong only on instances that will never be encountered in real-world applications, or is only strong on instances that all algorithms find easy, then this is important information that helps to draw a conclusion about the power of an algorithm. Extending the methodology to develop metrics that establish the degree of overlap between an algorithm's footprint and other regions of interest (such as the location of real-world instances, or the footprints of other algorithms) is easily calculated to provide this indication of the utility of a footprint.

Algorithm 1 Calculating the area of an algorithm's footprint with a minimum density and purity requirement.

```
Require: $\alpha$ (a given algorithm), $\rho$ (a density threshold), $\pi$ (a purity threshold),
  $\mathcal{I} = [\mathcal{I}_g, \mathcal{I}_b]$ (the instance set labelled as good or bad for the algorithm),
  Initialise Stage
  Randomly select a good instance $i \in \mathcal{I}_g$;
  Form a closed region (triangle) with the two closest (smallest Euclidean distance in feature space) instances to $i$ , not already part of a triangle;
  Repeat until no more triangles can be formed.
  Merge Stage
  Randomly select a closed region $J$;
  Find the closest closed region $K$ (minimum Euclidean centroid distance);
  $density = \frac{|J|+|K|}{\Xi(J \cup K)}$ where $|J|$ is the number of instances in $J$, and $\Xi$ is the area of the convex hull formed by a region;
  $purity = \frac{|J_g|+|Kg|}{|J|+|K|}$ where $|J_g|$ is the number of good instances in $J$;
  if ($density > \rho$) and ($purity > \pi$) then
    Merge closed regions $J$ and $K$ to form a new closed region
  end if
  Repeat the Merge Stage until there are no more pairs to consider.
  return $Area = \sum \Xi(j)$ for all closed regions $j$ that remain after the merge stage.
```

The final analysis that should be done to complete the methodology is to explore the instance space to gain insights into how the features affect the algorithm footprints. The distribution of each feature across the instance space can be visualized, using color coding to indicate high or low values of a feature for each instance, and conclusions can be drawn about the particular properties of the instances that are found in certain regions of the instance space, including those that define the footprint boundaries.

A case study on graph coloring will now be used to illustrate how this methodology can be applied to draw conclusions about algorithm performance in an objective and unbiased manner, and to generate new insights into the strengths and weaknesses of optimization algorithms.

4. Graph Coloring Case Study

In this section we introduce the graph coloring problem and a set of algorithms studied by Lewis et al. [19]. We start by defining the meta-data for our study in Section 4.1, including the set of graph coloring instances, the features we use to summarize the instances, the set of algorithms, and the performance metric chosen to measure algorithm performance. We then demonstrate the methodology introduced in the previous section by generating the instance space in Section 4.2, and visualizing the algorithm footprints in Section 4.3. Performance prediction using machine learning methods is demonstrated in Section

4.4, but fails to completely identify the pockets of unique strength we can see in the footprints. We then return to the algorithm footprints in Section 4.5 to objectively measure the power of each algorithm, and to gain insights into the condition under which each has demonstrable stregnths and weaknesses.

4.1. Graph Coloring Meta-Data

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 represents the time period assigned to events [34, 35, 36].

4.1.1. Graph Coloring Problem Instances ($\mathcal{P}$)

The problem instances we have used in this research consist of the graph instances used by Lewis et al. [19], others that we have sourced from the instance generators on Joe Culberson's website [37], the well-studied DIMACS instances [38], NetworkX generated instances [39] and some additional instances that we have generated by starting with bipartite graphs and adding edges in a controlled manner to produce graphs that are less bipartite. All instance sets are described below:

- B (Bipartivity-controlled): Starting with 50 randomly generated bipartite graphs, with the number of nodes $|V|$ randomly generated in the range $[100, 1000]$, random edges are added to produce 20 graphs of each size, with the number of added edges being $\{0, 100, 1000, 2000, 3000, 4000, 5000, 7500, 10000, 15000, 20000, 30000, 40000, 50000, 75000, 100000, 250000, 500000, 750000, 1000000\}$

- C1 (Culberson - cycle driven graphs): This generator creates graphs with cycles of a specified length. K color partitions are created, the algorithm then generates a cycle by randomly generating a path with each vertex belonging to a different color partition than the last.

- C2 (Culberson - geometric graphs): These graphs are generated by choosing a radius r and uniformly distributing n pairs of numbers (x, y) in the range of $0 \leq x, y < 1$. Vertices in the graph correspond to the points (x, y) in the plane, with edges included if the distance between a pair of vertices is less than the radius r.

- C3 (Culberson - girth and degree inhibited graphs): Each graph is assigned a probability p, girth limit g, and a degree limit δ. The girth limit indicates that no cycle will be created with girth less than g. Hence if an edge (v, w)

is being considered as a new edge, every pair of vertices (x, y) will have a distance of less than g after the addition is blocked, and will never be selected as a possible new edge. p is the probability that a possible edge will be used. δ is a hard limit on the difference between the average node degree and the maximum degree of any vertex.

- C4 (Culberson - uniform or IID graphs): edges are assigned to vertex pairs with a fixed probability p

- C5 (Culberson - weight-biased graphs): These graphs contain cliques limited to a given size. Each clique is generated by randomly creating K color partitions, then randomly selecting one of the vertices in each partition and joining every pair by an edge. Each clique is generated independently.

- D (DIMACS and Networkx): DIMACS instances are 125 benchmark instances consisting of different type of graphs such as Leighton, flat, Mycielski, queen, miles, game, register, insertions etc. A description of DIMACS instances can be obtained from http://mat.gsia.cmu.edu/COLOR03/. The Networkx generator was used to produce 675 instances: 125 instances of 30, 50 and 70 vertex graphs (totalling 375 instances) were generated using the *random graph* generator; and 100 instances of 100, 150 and 200 vertex graphs (totalling 300 instances) were generated using the *geometric graph* generator. Further information about the Networkx generators can be found at http://networkx.lanl.gov/reference/generators.html

- E (Social Network graphs): These graphs are based on the social networks of school–friends, compiled as part of the USA–based National Longitudinal Study of Adolescent Health project [40], and were used in the study of Lewis *et al.* [19].

- F (Sports Scheduling graphs): These graphs represent round–robin tournaments used in sports leagues [41, 19]. In such problems, we are given an even number of teams n, and each team is required to participate in a match against all other teams m times in $m(n-1)$ rounds. These graphs were used in the study of Lewis *et al.* [19].

- G (Exam Timetabling graphs): These graphs represent the conflicts in real-world exam timetabling problems ranging in size from $|V| = 81$ to 2419 [42], and were used in the study of Lewis *et al.* [19].

- H (Flat graphs): These are the flat graphs used by Lewis et al. [19], constructed by partitioning the vertices into K almost equi-sized sets. Edges are then added between pairs of vertices in different sets with probability p in such a way that the variance in vertex degrees is kept to a minimum. Flat graphs were generated for $K \in \{10, 50, 100\}$. In each case we used $|V| = 500$, implying 50, 10, and 5 vertices per colour, respectively. We generated 41 instances for the $K = 10$ and $K = 100$ graphs, and 21 instances for $K = 50$ graphs, corresponding to p values in and around the

Instance Set	Source	Number of Instances (after outlier removal)
B	Our generator	1000 (991)
C1	Culberson [37]	1000 (1000)
C2	Culberson [37]	932 (806)
C3	Culberson [37]	1000 (1000)
C4	Culberson [37]	1000 (987)
C5	Culberson [37]	1000 (946)
D	DIMACS [38] and Networkx [39]	743 (731)
E	Social Network [19]	20 (20)
F	Sports Scheduling [19]	80 (64)
G	Timetabling [19]	13 (12)
H	Flat [19]	103 (103)
I	Random [19]	57 (52)

Table 1: Graph Coloring Instances

phase transition regions (ensuring the instances are quite hard to K-color in general).

- I (Random graphs): These are the random graphs used by Lewis et al. [19]. Each pair of vertices are made adjacent with probability p ranging from 0.05 (sparse) to 0.95 (dense), incrementing in steps of 0.05,with $|V| \in \{250, 500, 1000\}$.

Table 1 summarizes the instances generated for this case study and also shows how many instances remain in each instance set after outliers are removed using a procedure described in Section 4.2.

Empirical studies have already shown how the performance of some algorithms depends on the source of the instances [43, 44]. Culberson [37] states about his resources for graph coloring, "My intention is to provide several graph generators that will support empirical research into the characteristics of various coloring algorithms. Thus, I want generators that will exhibit variations of various characteristics of graphs, such as degree (expectation and variation), hidden colorings, girth, edge distributions etc.". It should be noted that the study of Lewis *et al.* [19] considered only a small subset of the instances we consider here, and it will be interesting to see if the conclusions of that study are generalizable to a more diverse instance set.

4.1.2. Graph properties or features ($\mathcal{F}$)

Our recent survey of what makes optimization problem instances difficult [45] shows that there are many features or properties of a graph that can be calculated in polynomial time and can be used to shed some light on the relationships between graph instances and algorithm performance. Many of these features are based on properties of the adjacency matrix A and Laplacian matrix L of the graph $G(V, E)$ defined as follows: $A_{i,j\neq i} = 1$ if an edge connects vertices i and j, and 0 otherwise; $L_{i,i} = degree(V_i)$, $L_{i,j \neq i} = -1$ if an edge connects vertices i and j, and 0 otherwise.

Graph theory researchers have long collected interesting graphs [46] which are used to develop new conjectures, and often provide counter-examples to existing theorems. Recently, several exciting developments have occured where graphs with specific properties can be generated from an extendable database - known as the House of Graphs [47]. These graphs have been used successfully to generate new conjectures by generating linear inequalities that describe the relationships between specific graph properties known as *invariants* [48]. In this study we consider many of the invariant graph properties referred to by House of Graphs, as well as some additional spectral features based on the eigenvalues of the adjacency and Laplacian matrices of the graph. For the graph $G = (V, E)$, we consider the following 18 features relating to a) the nodes and edges (features 1-5); b) the cycles and paths on the graph (features 6-13); and c) the spectral properties of the graph (features 14-18):

1. The number of vertices in a graph: $n = |V|$
2. The number of edges in a graph: $|E|$
3. The density of a graph: the ratio of the number of edges to the number of possible edges $\rho = \frac{2|E|}{n(n-1)}$.
4. Mean vertex degree: the number of edges from a vertex, averaged across all vertices.
5. Standard deviation of vertex degree: the average vertex degree and its standard deviation can give us an idea of how connected a graph is.
6. Average path length: the average number of steps along the shortest paths for all possible pairs of vertices. It is a measure of the efficiency of traveling between vertices.
7. The diameter of a graph: the greatest distance between any pair of vertices. To find the diameter we find the shortest path between each pair of vertices and take the greatest length of these paths.
8. The girth of a graph: the length of the shortest cycle.
9. Mean betweenness centrality: average fraction of all shortest paths connecting all pairs of vertices that pass through a given vertex.
10. Standard deviation of betweenness centrality: with the mean, the SD gives a measure of how central the vertices are in a graph.
11. The clustering coefficient: a measure of degree to which vertices 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 [49].
12. The Szeged index: generalisation of Wiener number to cyclic graphs, providing a measure that correlates with bipartivity [50].
13. Beta: proportion of even closed walks to all closed walks, providing an alternative measure of the bipartivity of a graph [51].
14. Energy: the mean of the absolute values of the eigenvalues of the adjacency matrix [52].

15. Standard deviation of the set of eigenvalues of the adjacency matrix:
16. Algebraic connectivity: the second smallest eigenvalue of the Laplacian matrix [53]. This reflects how well connected a graph is. Cheeger's constant, another important graph property, is bounded by half the algebraic connectivity [54].
17. Mean eigenvector centrality: the Perron-Frobenius eigenvector of the adjacency matrix, averaged across all components.
18. Standard deviation of eigenvector centrality: together with the mean, the standard deviation of eigenvector centrality gives us a measure of the importance of a vertex inside a graph.

4.1.3. Algorithms ($\mathcal{A}$)

We consider a portfolio of the same six high-performing algorithms described in the computational comparison of graph coloring algorithms by Lewis et al. [19], augmented with a random greedy heuristic and the degree saturation heuristic DSATUR [55], which are used as components of the six algorithms. Thus we have eight algorithms representing a wide range of solution strategies:

- DSATUR: Brelaz's greedy algorithm based on saturation degree of the vertices [55], which has been shown to be exact for bipartite graphs [56].
- Bktr: a backtracking version of the DSATUR heuristic which also includes an operator for dynamically re–ordering the vertices when a node in the search tree is revisited. This algorithm was implemented by Culberson and is available for download [37].
- HillClimb: A hill-climbing method [57] that operates in the space of feasible solutions with the initial solution being formed using the DSATUR heuristic.
- HEA: A hybrid evolutionary algorithm of Galinier and Hao [58] that operates by maintaining a (typically small) steady–state population of candidate solutions which are evolved via a problem–specific recombination operator and a local search method.
- TabuCol: Galinier and Hao's variant of the Tabu search algorithm for graph colouring [58].
- PartialCol: The algorithm of Blochliger and Zufferey [59] that operates in a similar fashion to TABUCOL but with a different neighborhood operator that does not allow improper solutions to be considered.
- AntCol: Ant Colony meta heuristic-based method of Dowsland and Thompson [60] combining global and local search operators.
- RandGr: This is a simple greedy algorithm that takes a random permutation of the vertices and then colours them using a first-fit greedy algorithm. This is the heuristic used to provide the initial number of colours for both the PartCol and TabuCol algorithms.

All algorithms are run in the same environment with the same number of function evaluations. All algorithms were run in the same environment (Windows XP using a 3.0 GHz processor with 3.18 GB of RAM) for a fixed computation limit of 5×10^{10} constraint checks. This led to run times of approximately 5-15 minutes per instance, depending on the structures of the graph considered. Further details on the algorithms' implementations - together with descriptions on what constitutes a constraint check - can be found in Lewis *et al.* [19].

4.1.4. Performance metric ($\mathcal{Y}$)

The performance of an algorithm can be defined in many different ways depending on the goals. We might say that one algorithm is better than another if it finds the same quality solution in a faster time, or if it obtains a much better quality solution for the same run-time. How we define "same quality" may involve some tolerance, and we can define this more specifically if we introduce concepts like "two algorithms give the same result in their solution costs are within ϵ% of each other". Acknowledging the arbitrariness of this decision, and without loss of generality, in this paper we measure algorithm performance as either easy or hard, based on the gap between the best solution (number of colors) produced by the portfolio and the number of colors needed by an algorithm to properly color the graph within the stated computation limit of 5×10^{10} constraint checks. We say that an algorithm is "ϵ-good" if the number of colors needed by the algorithm is no more than ϵ% greater than the best algorithm in the portfolio on that instance. We will consider both $\epsilon = 0\%$ and $\epsilon = 5\%$.

4.1.5. Summary of experimental meta-data

The meta-data for our experimental case study on graph coloring can now be summarized using the framework proposed in Rice [20], and adapted for the study of optimization algorithm performance by Smith-Miles [23]:

- The *problem sub-space* $\mathcal{I}$ is a set of 6948 graph coloring instances, from benchmark instances and random and geometric graph generators, with the number of vertices in the range $[11, 2419]$, as described in Section 4.1.1.
- The *feature space* $\mathcal{F}$ is defined by the 18 invariant graph properties listed in section 4.1.2.
- The *algorithm space* $\mathcal{A}$ comprises eight heuristics as described in Section 4.1.3.
- The *performance space* $\mathcal{Y}$ is a binary measure labelling algorithm performance as "ϵ-good" or not, defined after a maximum of 5×10^{10} constraint checks for a given value of ϵ.

4.2. Generating the Instance Space

From the raw meta-data we first removed any outliers that were more than three standard-deviations from the mean of any feature, and then applied a log transform to all features to reign in the effect of any remaining outliers in the instance space in order that our instance space not be unduly distorted by outliers. All features were then normalized to $[0, 1]$ using min-max normalization. For the feature subset selection, the genetic algorithm was implemented using the MATLAB optimization toolbox. Each selection of subsets (individuals) was represented using a binary vector to indicate if a feature was a member of the candidate subset of a given cardinality m. In order to determine the utility of a subset of features, we recognize that a useful instance space is one that has created an easy-hard partitioning of the instance space to support our visualization. The chosen fitness function was related to the classification error that a Naive Bayes classifier (chosen as a simple machine learning method, and implemented with default parameters in MATLAB) could obtain using those candidate features to project to a 2-dimensional instance space using PCA and correctly predict if an algorithm could obtain the best result for each instance. The mean average error (averaged across the eight Naive Bayes classifiers trained for each algorithm) was used as the reciprocal of the fitness function, with the error based on out-of-sample testing using a randomly extracted 50% of each instance set not used for learning the Naive Bayes classifiers. Thus, the feature selection process involved performing PCA for each individual subset being evaluated by the genetic algorithm, and using a Naive Bayes classifier in the PCA space (either in $\mathbb{R}^2$or in $\mathbb{R}^m$) in an iterative process that relied on an evolutionary process to find the optimal subset of features. We also experimented with classification in the m-dimensional feature space, but found that the resulting errors were marginally reduced in $\mathbb{R}^2$. We additionally experimented with selecting the feature subset that minimized the maximum error from any of the Naive Bayes classifiers, but found that this was not as effective compared to seeking the subset that minimized the average error. Optimal subsets were found for cardinalities 2 to 18, with the minimum average error (0.146) attained for a set of $m = 3$ features: namely the density, algebraic connectivity and energy of the graph. PCA on these three features creates three new axes - linear combinations of these features - by which to describe the meta-data. Projection onto only the two principal axes (with the largest eigenvalues) retains 98.4% of the variation in the data. These two axes define the instance space and are algebraically described as:

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0.559 & 0.614 & 0.557 \\ -0.702 & -0.007 & 0.712 \end{bmatrix} \begin{bmatrix} \textit{density} \\ \textit{algebraic connectivity} \\ \textit{energy} \end{bmatrix} \tag{1}$$

Rotating all instances into this new coordinate system and plotting (v_1, v_2) for all instances $x \in \mathcal{I} \subset \mathcal{P}$, we generate the instance space shown in Figure 2. The grey instances show all instances, whereas the black instances show the location of particular instance sets within this instance space. The centre of this instance space, $(v_1, v_2) = (0, 0)$, corresponds to an average instance with average values of density, algebraic connectivity and graph energy. Instances

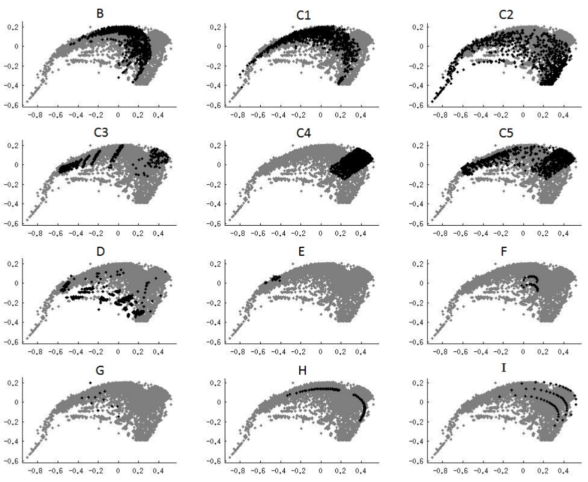

Figure 2: Each instance sets shown as black points in the instance space, ordered alphabetically from set B (shown at top left) to set I (shown at bottom right). The grey points define the entire instance space.

that are near each other in the instance space have similar values of these three features (chosen optimally from the set of all possible combinations of features).

This view of the instance sets enables us to form conclusions about the diversity of each of the instance generators. We see clearly that the five types of Culberson generators are indeed serving their purpose of generating instances that are diverse. Collectively, instance sets C1-C5 help to define much of the shape of the instance space. We can also see that the instances used in the study of Lewis *et al.* [19], instance sets E, F, G, H and I, tend to fall in narrow bands within the broader instance space defined by the Culberson generators. While Lewis *et al.*'s chosen instances fall in different regions of the instance space their diversity is not as extensive, and large regions of the instance space would be left unexplored if we had not augmented the instance set with additional instances.

4.3. Algorithm Footprints

Now that we have generated an instance space defined by the axes given by equation (1), we can examine algorithm performance within this space. Suppose we define an algorithm's performance on an instance to be good if it attains the minimum number of colors of all algorithms. That is, it is $\epsilon - good$ with $\epsilon = 0$, since its performance is within 0% of the best solution provided by the portfolio of algorithms. Figure 3 shows the footprint of each algorithm, with blue instances showing good performance with $\epsilon = 0$. We can measure the area of each footprint using the method proposed in Section 3.3 (Algorithm 1), with parameters $\rho = 50,000$ and $\pi = 0.95$, as shown in the first column of Table 2. Clearly, the algorithm with the largest footprint is HEA, since there are only

Algorithm	Area ($\epsilon = 0\%$)	Area ($\epsilon = 5\%$)
AntCol	19.35%	34.9%
Bktr	11.63%	14.17%
DSATUR	7.11%	12.84%
HEA	41.17%	57.14%
HillClimb	32.97%	52.08%
PartialCol	30.86%	51.84%
RandGr	0.90%	3.13%
TabuCol	36.05%	48.7%

Table 2: Relative areas of algorithm footprints for $\epsilon = 0\%$ and $\epsilon = 5\%$, expressed as a percentage of the total area of the instance space, calculated using Algorithm 1.

small regions of the instance space where it is not $\epsilon - good$ with $\epsilon = 0$ (i.e. where its solution is worse that the best in the portfolio). It is interesting to note these regions though, and to observe that HEA is not best everywhere. We will return to explore how the features define these regions later in Section 4.5.

Once we relax the definition of good to consider any algorithm that achieves a performance within 5% of the best algorithm, we obtain larger footprints for all algorithms, shown in column 2 of Table 2, and we can see that HEA loses its competitive advantage somewhat. Measuring the area of the footprint of each algorithm, for various definitions of goodness, provides an objective measure of the power of each algorithm across the instance space, and the margin of its competitive advantage.

4.4. Predicting algorithm performance for automated algorithm selection

From these footprints we can also develop automated methods to predict the performance of algorithms on untested instances. Certainly, we can visually explore the footprint of an algorithm in the location of an untested instance, and use simple methods such as a k-nearest neighbor algorithm to determine the likelihood that an algorithm will perform well on an untested instance. But more powerful machine learning methods can provide a more objective and robust approach to algorithm performance prediction.

Our first attempt to use a Naive Bayes classifier resulted in a model that predicted that HEA would be best across the whole instance space, and failed to detect the small regions where other algorithms are best. We then employed a more powerful machine learning method: support vector machines (SVMs), trained in MATLAB. Using a random extaction of 50% of the instances, with the remaining 50% reserved for out-of-sample testing, we built eight SVMs to predict the boundaries of each algorithm's footprint, achieving better results, although still not perfect due to the many contradictory instances within a region. The out-of-sample test set accuracies ranged from 90% for the DSATUR and Bktr predictions (easier to predict since they tend to be quite consistent in regions where they don't perform well) down to 73% accuracy for the AntCol prediction. The SVM prediction model for HEA was 82% accurate. Combining

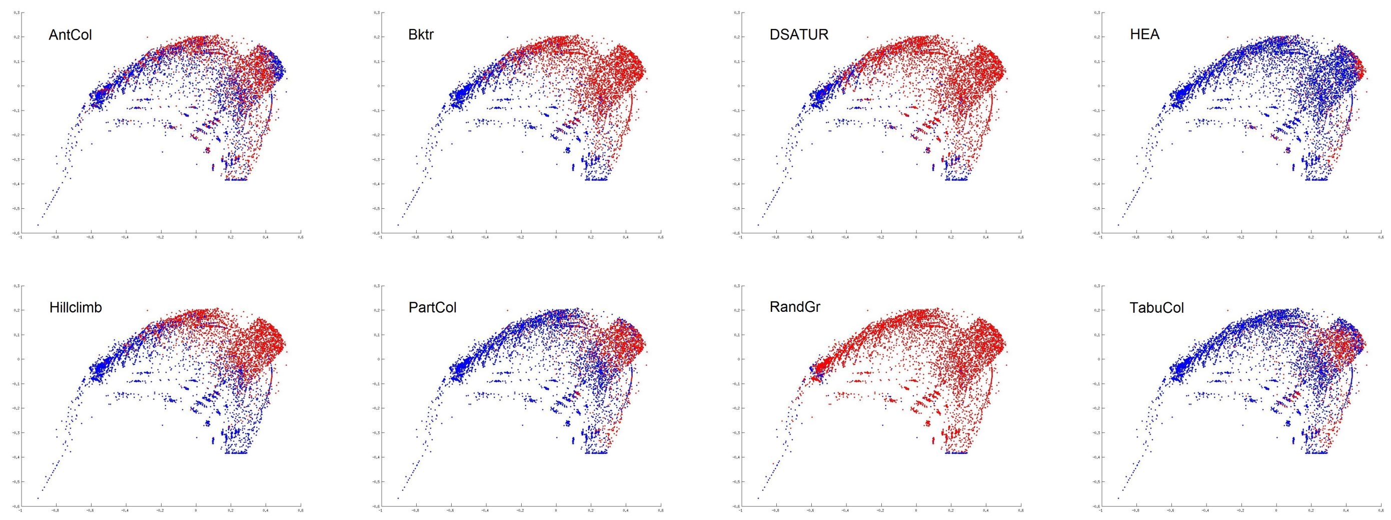

Figure 3: Algorithm Footprints showing in blue where an algorithm achieves $\epsilon - \textit{good}$ performance, with $\epsilon = 0$. Red instances are not within the algorithm footprint.

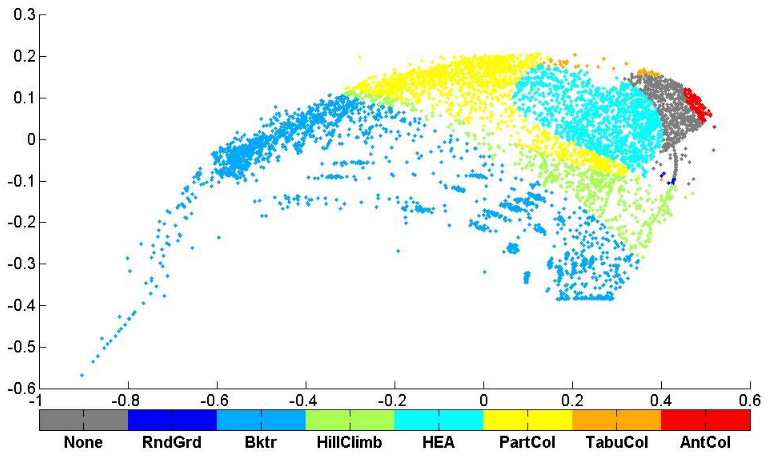

Figure 4: Machine learning (SVM) recommendations about which algorithm to use in each region.

these eight SVM predictions, we can identify for each instance the algorithm that is predicted to be best. In the event that multiple algorithms are predicted to be best, we recommend adopting the algorithm which has the highest model accuracy (although shortest run-time could be another criterion). This approach leads to the algorithm recommendations shown in Figure 4, including a region near the upper right portion where no SVM model predicted any algorithm to be best. Since at least one algorithm is best, by definition, this is clearly a failing of a sophisticated machine learning method in this region. While this depiction of algorithm strength across the instance space is interesting and somewhat enlightening, it should be used with caution, since it is only as accurate as the machine learning models we are relying upon.

4.5. Insights into Algorithm Strengths and Weaknesses

The instance space affords us the opportunity to explore more than algorithm footprints, but also to develop a good understanding of where the unique strengths and weaknesses of each algorithm lie. If an algorithm is only good where many other algorithms are good, then this is useful information to assess the relative power of algorithms. We wish to visualize where each algorithm offers a unique advantage, and where it might struggle where other algorithms succeed. These kinds of insights are critical to inform better algorithm design, and to help automated algorithm selection where machine learning methods may not be accurate enough.

For each instance, we now count how many of the eight algorithms in the portfolio are $\epsilon - good$ with $\epsilon = 0$. Figure 5 shows the location of the instances that are easily solved by all algorithms (shown as red on the color scale, with

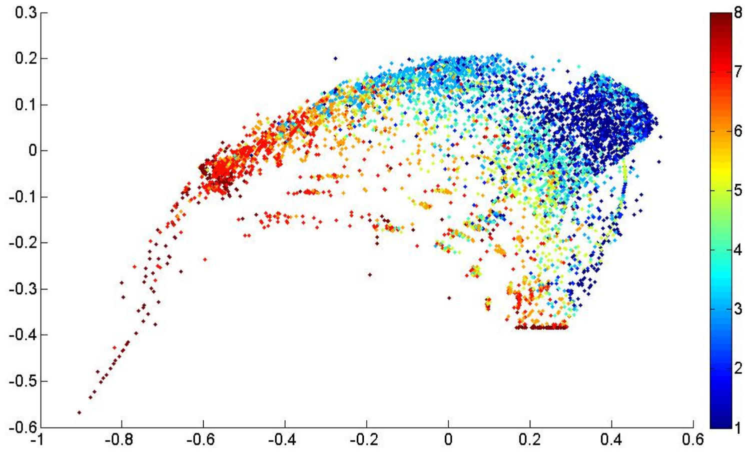

Figure 5: The number of algorithms that achieved $\epsilon - good$ performance, with $\epsilon = 0$.

8 algorithms finding the best number of colors of the graph), and the instances that are more challenging since only one algorithm attains the best result (shown as dark blue in the upper right portion). For these harder instances, we are interested to know which algorithm provides the unique advantage over others, and this is shown in Figure 6. Only three algorithms show clearly consistent regions where they are uniquely best: AntCol (red), HEA (blue) and HillClimb (green). These are all methods that combine local search strategies with global operators that allow much larger changes to be made to a solution. Understandably, DSATUR and RandGr are never uniquely best, since the other algorithms can be considered extensions of them. The remaining algorithms are sometimes uniquely best but the types of instances that they are best suited to are not well co-located in the instance space. These results are interesting in the context of Lewis *et al.*'s conclusions that HEA was the best performing algorithm. Certainly on the instances that were considered in their study, this conclusion is most likely correct. But the view of the instance space generated from a broader set of instances shows clearly that there are pockets of instances where HEA is not the best algorithm, and these instances are co-located in regions of the instance space that might enable us to infer under what conditions HEA is not as effective as other algorithms.

Table 3 shows the relative unique strengths of each algorithm, focusing on how many instances were uniquely solved well by each algorithm, and the area of the footprints for a definition of ϵ-good with $\epsilon = 0\%$ and $\epsilon = 5\%$. For 82.48% of the instance space there is no uniquely best algorithm, but for the 1176 instances with a unique winning algorithm, it is HEA in 67.77% of the instances (shown as light blue in Figure 6). AntCol is uniquely best in 11.22% of the 1176 discriminating instances (shown as red in Figure 6). HillClimb is uniquely

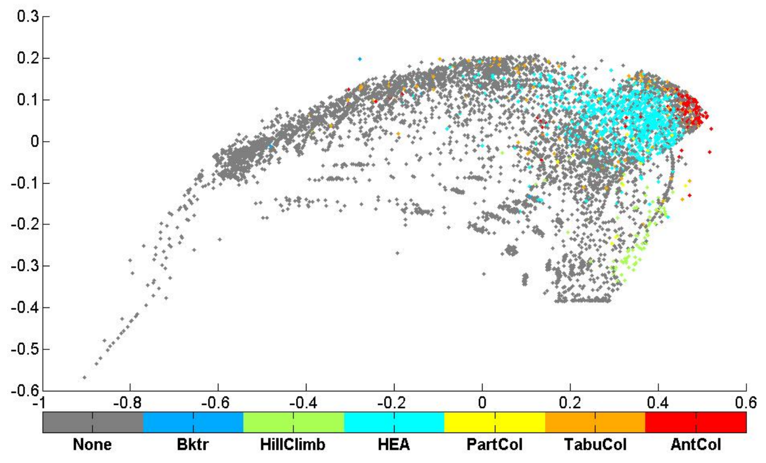

Figure 6: Instances that have a uniquely $\epsilon - good$ ($\epsilon = 0$) algorithm, color-coded by the algorithm. Grey instances are those for which there are multiple algorithms that achieved the best performance.

best for 6.72% of the discriminating instances. While TabuCol is uniquely good for 12% of the discriminating instances, its footprint is not very contiguous, and we cannot easily predict where it will be the best algorithm. The regions of unique strength of HEA, AntCol and HillClimb are clear however, and the areas of their unique footprints are 15.38%, 2.82%, and 2.78% respectively. Clearly, with $\epsilon = 0$ HEA has the largest unique footprint, in addition to the largest footprint overall. However, there is a small but identifiable part of the instance space where HEA is not the best.

If we now permit an algorithm to be considered good if it is within 5% of the best algorithm's performance, ($\epsilon = 5\%$), we find that 98.18% of the instances are well solved by more than one algorithm. The unique footprint of each algorithm diminishes dramatically, suggesting that there is not much difference between the results of many of the algorithms. Only a small group of instances elicit a stand-out performance from one algorithm with this definition of good, and none of them are in a region consistent enough to attract a footprint that is dense or pure enough with the parameters $\rho = 50,000$ and $\pi = 0.95$ using Algorithm 1, apart from TabuCol which finds a few small triangles of instances where it is uniquely good. .

Considering again the unique footprints shown in Figure 6, beyond measuring their area, we are also interested to gain insight into what the location in the instance space tells us about the conditions under which an algorithm has a unique strength. To achieve this insight, we inspect the distribution of each of the three features across the instance space, looking in particular for clues about why HEA seems to struggle in the upper right portion of the instance space where AntCol excels, and why HillClimb is so effective on the instances

	$\epsilon = 0\%$		$\epsilon = 5\%$	
Algorithm	Instances (%)	Footprint Area	Instances (%)	Footprint Area
AntCol	132 (11.22%)	2.82%	5 (4.10%)	0%
Bktr	7 (0.06%)	0%	1 (0.82%)	0%
DSATUR	0 (0%)	0%	0 (0%)	0%
HEA	797 (67.77%)	15.38%	54 (44.26%)	0%
HillClimb	79 (6.72%)	2.78%	12 (9.84%)	0%
PartialCol	20 (1.70%)	0%	6 (4.92%)	0%
RandGr	0 (0%)	0%	0 (0%)	0%
TabuCol	141 (11.99%)	0.86%	44 (36.07%)	0.14%
TOTAL	1176 (17.52%)		122 (1.82%)	

Table 3: Number (Percentage) of instances, and relative area of footprints where each algorithm is uniquely $\epsilon - good$, for $\epsilon = 0\%$ and $\epsilon = 5\%$

found on the thin right-most edge of the instance space. Figure 7 shows that these interesting regions of unique strength of some algorithms correspond to extreme values of the features defining the instances. In particular, AntCol appears to outperform all algorithms (including HEA) when the energy of the graph is exceptionally high; HillClimb is the most effective method for graphs with high algebraic connectivity and high density; and HEA is uniquely best when the graph has moderate energy and moderate density. In all other cases, there is no single stand-out algorithm. This kind of visualization enables us to identify the unique strengths and weaknesses of algorithms in a way that is not possible if we don't consider the right features and a diverse enough set of instances.

5. Conclusions

This paper has proposed a new methodology for objective assessment of the relative power of algorithms, in general, and has focused on optimization algorithms in particular. It is a methodology based on representing instances to optimization problems as points in a two-dimensional plane, which opens up the opportunity to visualize instance diversity and observe any sample bias; to identify the regions of instance space where algorithms have unique strengths and weaknesses; and to generate new insights based on the instance features defining those instances that lie within an algorithm's footprint.

It has been nearly two decades since Hooker [1, 2] called for a more empirical approach to testing algorithm performance. By building upon the framework of Rice [20], we have proposed a pathway to develop the tools required for such an empirical approach. This paper has addressed some of the remaining questions in the methodology we have been developing over recent years, and has demonstrated the usefulness of the approach by revisiting the case study of Lewis *et al.* [19]. While our methodology supports the broad conclusion of their study - that HEA was the best algorithm in their portfolio on average - we have

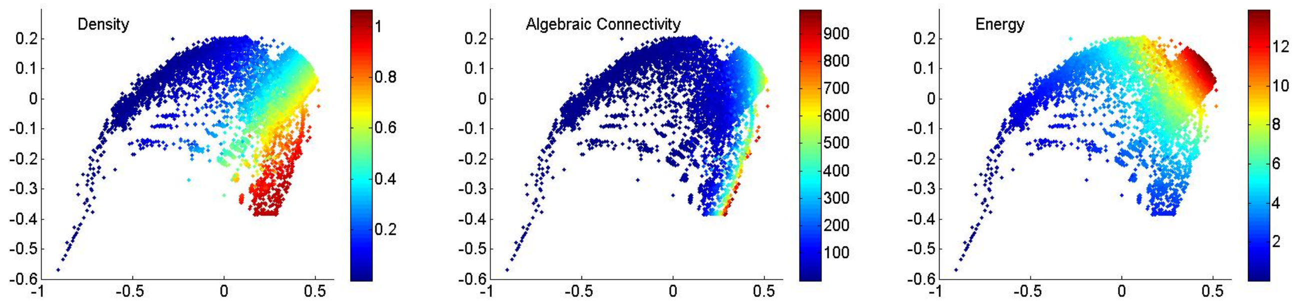

Figure 7: Distribution of the three selected features (density, algebraic connectivity and energy) across the instance space

also been able to provide additional insights that were not observable from their study, which considered only a subset of the instances used in our meta-data and did not have access to the tools that we have developed in this research. In particular, we have been able to identify the regions in instance space where HEA is not the best algorithm, and to define classes of instances where other algorithms, such as AntCol and HillClimb, consistently outperform HEA. Most significantly, we are able to relate the instance features to these classes, and show that HEA (or at least the implementation of HEA that we used) has a relative weakness on instances with extreme values of the instance features. Of course, an algorithm is really just a particular implementation of an algorithmic strategy, with parameter values that are tunable and can affect the performance of the algorithm. We are not attempting to draw any conclusions about any particular algorithm's potential with this analysis. Indeed, the same algorithm with a different parameter combination can be considered as a new algorithm, with its own footprint, within this methodology. We are simply demonstrating that, using exactly the same algorithmic implementations as used in the study of Lewis *et al.* [19] we can obtain additional insights that were not otherwise apparent using a standard analysis of computational results.

Of course, our analysis also included a much broader set of instances than those used by Lewis *et al.*, and the shape of the instance space we create is certainly dependent upon these instances and their distributions. A very different shape for the instance space would have been generated had we only used instances (from classes E, F, G, H, and I) used by Lewis *et al.*, since it is primarily the Culberson instances (C1-C5) that define the shape of our instance space, with other types of instances lying within the boundaries of this space. The shape and density of the instance space is also dependent on the distribution of the instances. Adding additional instances from one class, for example, would potentially change the mean value of certain features, creating a rescaling of the space, and while the density of the instance space would change if new instances from one class were added, the locations of the instances relative to the new mean would not change significantly, and the relative area of the footprint should be largely unaffected. We believe that the instance space we have generated is a good representation of the set of all graphs that are typically studied for graph coloring. Nevertheless, it is an interesting question for our future research to see if we can evolve instances that lie outside the boundary of the presented instance space, and if we can't then to conjecture why such graphs might not exist in terms of their features [48].

The next steps for this research include developing the instance spaces for a variety of optimization problems, and making them available to the operations research community via a web tool. Researchers will then be able to generate footprints for their chosen algorithm in this instance space, and can compare the area and location of algorithm footprints to draw conclusions about the power of new algorithms they are proposing, without concern about sample bias. Understanding if an algorithm's footprint overlaps regions of interest, such as real-world instances or challenging benchmark instances, is a key step in designing algorithms that are applicable and can avoid deployment disasters

[18]. With new insights into the true strengths and weaknesses of optimization algorithms possible through this methodology, enabling us to match algorithm choice to the characteristics of the problem instances, we hope to take a step closer to a free lunch for optimization, at least within a given problem domain.

Acknowledgements

This research is funded by the Australian Research Council under grant DP120103678. The authors are grateful to Gordon Royle for provided useful discussions about the energy of a graph. They are also grateful to the two reviewers who made useful suggestions to improve the clarity of the paper, and to Simon Bowly for his assistance with some of the figures.

References

[1] J. Hooker, Needed: An empirical science of algorithms, Operations Research (1994) 201–212.

[2] J. Hooker, Testing heuristics: We have it all wrong, Journal of Heuristics 1 (1) (1995) 33–42.

[3] J. Beasley, OR-Library: Distributing test problems by electronic mail, Journal of the Operational Research Society (1990) 1069–1072.

[4] R. Hill, C. Reilly, The effects of coefficient correlation structure in two-dimensional knapsack problems on solution procedure performance, Management Science (2000) 302–317.

[5] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, IEEE transactions on evolutionary computation 1 (1) (1997) 67–82.

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

[7] C. Igel, M. Toussaint, A no-free-lunch theorem for non-uniform distributions of target functions, Journal of Mathematical Modelling and Algorithms 3 (4) (2005) 313–322.

[8] S. Margulies, J. Ma, I. Hicks, The Cunningham-Geelen method in practice: Branch-decompositions and integer programming, INFORMS Journal on Computing, Published online before print November 27, 2012, doi: 10.1287/ijoc.1120.0524.

[9] K. Smith-Miles, T. Tan, Measuring algorithm footprints in instance space, in: IEEE Congress on Evolutionary Computation (CEC), IEEE, 2012, pp. 1–8.

[10] K. Smith-Miles, L. Lopes, Measuring instance difficulty for combinatorial optimization problems, Computers & Operations Research 39 (5) (2012) 875–889.

[11] K. Smith-Miles, J. van Hemert, Discovering the suitability of optimisation algorithms by learning from evolved instances, Annals of Mathematics and Artificial Intelligence 61 (2) (2011) 87–104.

[12] K. Smith-Miles, B. Wreford, L. Lopes, N. Insani, Predicting metaheuristic performance on graph coloring problems using data mining, in: E. G. Talbi (Ed.), Hybrid Metaheuristics, Springer, 2013, pp. 417–432.

[13] K. Smith-Miles, J. van Hemert, X. Lim, Understanding TSP Difficulty by Learning from Evolved Instances, Learning and Intelligent Optimization, LNCS 6073 (2010) 266–280.

[14] K. Smith-Miles, R. James, J. Giffin, Y. Tu, A knowledge discovery approach to understanding relationships between scheduling problem structure and heuristic performance, Learning and Intelligent Optimization (2009) 89–103.

[15] K. Smith-Miles, Towards insightful algorithm selection for optimisation using meta-learning concepts, in: IEEE International Joint Conference on Neural Networks, 2008, pp. 4118–4124.

[16] K. Smith-Miles, v. n. p. y. p. Baatar, Davaatseren journal=Discrete Applied Mathematics, in press, Exploring the role of graph spectra in graph coloring algorithm performance.

[17] K. Smith-Miles, L. Lopes, Generalising Algorithm Performance in Instance Space: A timetabling case study, Lecture notes in computer science 6683 (2011) 524–539.

[18] L. Lopes, K. A. Smith-Miles, Generating applicable synthetic instances for branch problems, Operations Research, in press, 2013.

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

[20] J. Rice, The Algorithm Selection Problem, Advances in computers 15 (1976) 65–117.

[21] S. Weerawarana, E. N. Houstis, J. R. Rice, A. Joshi, C. E. Houstis, Pythia: a knowledge-based system to select scientific algorithms, ACM Trans. Math. Softw. 22 (4) (1996) 447–468.

[22] N. Ramakrishnan, J. Rice, E. Houstis, GAUSS: An online algorithm selection system for numerical quadrature, Advances in Engineering Software 33 (1) (2002) 27–36.

[23] K. Smith-Miles, Cross-disciplinary perspectives on meta-learning for algorithm selection, ACM Computing Surveys 41 (1).

[24] L. Xu, F. Hutter, H. Hoos, K. Leyton-Brown, SATzilla-07: The design and analysis of an algorithm portfolio for SAT, Lecture Notes in Computer Science 4741 (2007) 712.

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

[26] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, B. O'Sullivan, Using case-based reasoning in an algorithm portfolio for constraint solving, in: Irish Conference on Artificial Intelligence and Cognitive Science, 2008.

[27] I. Jolliffe, Principal Component Analysis, Springer, 2002.

[28] T. Kohonen, Self-organized formation of topologically correct feature maps, Biological Cybernetics 43 (1) (1982) 59–69.

[29] T. M. Mitchell, Machine Learning (Mcgraw-Hill Series in Computer Science), McGraw-Hill Higher Education, New York, NY, USA, 1997.

[30] D. Goldberg, Genetic Algorithms in Search and Optimization (1989).

[31] A. Tsymbal, M. Pechenizkiy, P. Cunningham, Diversity in ensemble feature selection, The University of Dublin: Technical Report TCD-CS-2003-44.

[32] Y. Bengio, N. Chapados, Extensions to metric based model selection, The Journal of Machine Learning Research 3 (2003) 1209–1227.

[33] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, The Journal of Machine Learning Research 3 (2003) 1157–1182.

[34] E. Burke, B. McCollum, A. Meisels, S. Petrovic, R. Qu, A graph-based hyper-heuristic for educational timetabling problems, European Journal of Operational Research 176 (1) (2007) 177–192.

[35] D. de Werra, An introduction to timetabling, European Journal of Operational Research 19 (2) (1985) 151–162.

[36] R. Lewis, A survey of metaheuristic-based techniques for university timetabling problems, OR Spectrum 30 (1) (2008) 167–190.

[37] J. Culberson, Graph coloring page, URL: http://www. cs. ualberta. ca/˜ joe/Coloring.

[38] D. S. Johnson, M. A. Trick, Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993, Vol. 26, American Mathematical Society, 1996.

[39] A. Hagberg, D. Schult, P. Swart, Networkx library developed at the los alamos national laboratory labs library (doe) by the university of california, Code available at https://networkx. lanl. gov.

[40] J. Moody, D. R. White, Structural cohesion and embeddedness: A hierarchical concept of social groups, American Sociological Review (2003) 103–127.

[41] D. De Werra, Some models of graphs for scheduling sports competitions, Discrete Applied Mathematics 21 (1) (1988) 47–65.

[42] M. W. Carter, G. Laporte, S. Y. Lee, Examination timetabling: Algorithmic strategies and applications, Journal of the Operational Research Society (1996) 373–383.

[43] D. Johnson, C. Aragon, L. McGeoch, C. Schevon, Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning, Operations research (1991) 378–406.

[44] B. A. Culberson, J., D. Papp, Hiding our colors, in: In CP95 Workshop on Studying and Solving Really Hard Problems, 1995.

[45] K. A. Smith-Miles, L. B. Lopes, Measuring instance difficulty for combinatorial optimization problems, Computers and Operations Research 39 (5) (2012) 875–889.

[46] R. Read, R. Wilson, An atlas of graphs, Oxford University Press, USA, 1998.

[47] G. Brinkmann, K. Coolsaet, J. Goedgebeur, H. Melot, House of graphs: a database of interesting graphs, Discrete Applied Mathematics 161 (2013) 311–314.

[48] H. Mélot, Facet defining inequalities among graph invariants: The system graphedron, Discrete Applied Mathematics 156 (10) (2008) 1875–1891.

[49] S. Soffer, A. Vázquez, Network clustering coefficient without degree-correlation biases, Physical Review E 71 (5) (2005) 057101.

[50] T. Pisanski, M. Randić, Use of the szeged index and the revised szeged index for measuring network bipartivity, Discrete Applied Mathematics 158 (17) (2010) 1936–1944.

[51] E. Estrada, J. A. Rodríguez-Velázquez, Spectral measures of bipartivity in complex networks, Physical Review E 72 (4) (2005) 046105.

[52] R. Balakrishnan, The energy of a graph, Linear Algebra and its applications 387 (2004) 287–295.

[53] B. Mohar, The laplacian spectrum of graphs, Graph theory, combinatorics, and applications 2 (1991) 871–898.

[54] N. Biggs, Algebraic graph theory, Vol. 67, Cambridge Univ Pr, 1993.

[55] D. Brélaz, New methods to color the vertices of a graph, Communications of the ACM 22 (4) (1979) 251–256.

[56] D. Wood, An algorithm for finding a maximum clique in a graph, Operations Research Letters 21 (5) (1997) 211–217.

[57] R. Lewis, A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing, Computers & Operations Research 36 (7) (2009) 2295–2310.

[58] P. Galinier, J. Hao, Hybrid evolutionary algorithms for graph coloring, Journal of Combinatorial Optimization 3 (4) (1999) 379–397.

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

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