Computers & Operations Research 63 (2015) 102–113

Contents lists available at ScienceDirect

Computers & Operations Research

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

Generating new test instances by evolving in instance space

Kate Smith-Miles *, Simon Bowly

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

ARTICLE INFO

Available online 9 May 2015

Keywords:
Test instances
Benchmarking
Graph colouring
Instance space
Evolving instances

ABSTRACT

Our confidence in the future performance of any algorithm, including optimization algorithms, depends on how carefully we select test instances so that the generalization of algorithm performance on future instances can be inferred. In recent work, we have established a methodology to generate a 2-d representation of the instance space, comprising a set of known test instances. This instance space shows the similarities and differences between the instances using measurable features or properties, and enables the performance of algorithms to be viewed across the instance space, where generalizations can be inferred. The power of this methodology is the insights that can be generated into algorithm strengths and weaknesses by examining the regions in instance space where strong performance can be expected. The representation of the instance space is dependent on the choice of test instances however. In this paper we present a methodology for generating new test instances with controllable properties, by filling observed gaps in the instance space. This enables the generation of rich new sets of test instances to support better the understanding of algorithm strengths and weaknesses. The methodology is demonstrated on graph colouring as a case study.

1. Introduction

In his seminal paper of 1994, "Needed: an empirical science of algorithms", Hooker [1] challenged the OR community to augment its theoretical focus on worst-case or average-case analysis of algorithms with a more empirical approach to algorithmic analysis: one that enables better understanding of the likely performance of algorithms on diverse test instances. He identified two main directions: more rigorous experimental design to select test instances intended to expose how the characteristics of the instances affect algorithm performance; and the use of empirical results to suggest hypotheses about algorithm behaviour, creating empirically based theories that can be submitted to rigorous testing. As such, he was proposing a paradigm shift in OR towards an experimental mathematics approach [2].

In a follow-up paper in 1995, "Testing heuristics: we have it all wrong" , Hooker [3] argued that randomly generated test instances lack diversity and rarely resemble real-world instances. He also expressed concern about the usefulness of commonly studied benchmark instances and their intrinsic bias, typically well suited to the first study that chooses to report on them but not necessarily diverse or challenging. The over-tuning of algorithms to a relatively small set of aging instances has major impact on the applicability of those algorithms for real-world deployment, and for our ability to learn about the strengths and weaknesses of algorithms from empirical evidence.

As an illustrative example, consider a commercial university timetabling algorithm that has been found to outperform competitor software packages when tested on Italian university timetabling instances (from the University of Udine instances used in the International Timetabling Competition). An Australian university then purchases the software, and finds that it results in more student clashes in the timetable compared to its previous software. While the Italian and Australian universities are tackling the same optimization problem (university curriculum timetabling), their *instances* of the problem seem to have different enough properties (class sizes, room capacities, number of subjects, etc.) that the algorithm performs well on one class of instance but not another.

Ensuring that an algorithm (and choice of parameters) has been tested thoroughly under all possible conditions that could be encountered in real world deployment is a significant challenge. Exhaustive testing is usually not possible due to the potentially infinite state space. Instead, a handful of test instances are usually the focus of algorithm development efforts, from which inferences about performance on other instances are optimistically made. Obtaining a sufficient number of real-world or real-world-like instances for statistical inference can be difficult, especially since synthetically generated instances of problems often have quite different properties and underlying structure to real-world instances [4–6]. Sampling from the set of possible test instances,

* Corresponding author. Tel.: +61 3 99053170; fax: +61 3 99054403.
E-mail address: kate.smith-miles@monash.edu (K. Smith-Miles).

http://dx.doi.org/10.1016/j.cor.2015.04.022
0305-0548/

ensuring that the selected instances are real-world-like, and that no selection bias has been introduced that would affect the conclusions, is a significant challenge affecting algorithmic testing in both industrial and academic environments.

Indeed, poor research practice in academia and deployment disasters in industry can be viewed as stemming from the same problem: inadequate stress-testing of algorithms, and a consequential failure to understand an algorithm's strengths and weaknesses. The research culture often found in academic environments highlights the need for rigorous new methodologies and tools to support better research practice in algorithm testing. The OR literature, for example, is dominated by a methodology in which a new algorithm is introduced and usually claimed to be superior by showing that it outperforms previous approaches on a set of well-studied test instances. However, the weaknesses of the algorithm are rarely reported. Objective assessment of optimization algorithm performance is notoriously difficult [1,3,7], especially when the conclusions depend so heavily on the chosen test instances. The opportunity now exists to challenge and extend these benchmarks, and to generate new test instances that enable strong inferences to be made about algorithm strengths and weaknesses to support objective algorithmic testing [8].

Since Hooker's concerns were raised two decades ago, some progress has been made, and benchmark datasets have been expanding. Examples include the DIMACS challenge which created new graph colouring benchmarks [4]; efforts to generate more real-world-like instances of well-studied problems [9–12]; new instances that are more challenging for particular algorithms [13,14]; and some new instances with controlled characteristics to support experimental mathematics [15,16]. In the field of graph colouring, Culberson states on his webpage [17] about his graph generators, “my intention is to provide several graph generators that will support empirical research into the characteristics of various colouring algorithms”. Greater awareness now also exists for the importance of rigorous testing of algorithms [18].

It is not always straightforward however to generate new test instances that have the right kind of properties that will (i) challenge algorithms, enabling us to see their strengths and weaknesses and (ii) reflect real-world properties. Certainly, there is no difficulty in generating graphs that have a certain density, or other simple properties that can be easily controlled by an instance generator. But, as an example, it is much more difficult to construct a graph that has a required algebraic connectivity (2nd smallest eigenvalue of the Laplacian matrix of the graph). If we are to construct test instances with controllable properties that are thought to be significant for discriminating between algorithms or reflecting real-world properties, we must have the ability to understand which properties are important and embed their control in the instance generation process.

The design of experiments approach randomly generates instances by varying easily controlled parameters (usually just a subset of a more comprehensive feature space) to create a Latin hypercube design [15]. But more sophisticated ideas exist to enable better control over the difficulty and applicability of the instances. The idea of using a genetic algorithm (GA) to evolve instances with desirable characteristics has been around for a decade, starting with creating instances that are hard or “worst-case” for an algorithm [13,14]. Extending these ideas to create new instances that are easy or hard, and uniquely easy or hard for particular algorithms, new Travelling Salesman Problem instances have been evolved that have helped to learn the strengths and weaknesses of algorithms more effectively than relying on random or benchmark instances alone [19,20].

Another direction for instance generation has been to create instances that are real-world-like. Typically this is done by studying a small set of real-world instances and making small variations to some key parameters [11,12]. We have previously implemented a different approach using machine learning methods to identify differences between real-world instances and those produced by random instance generators, providing feedback on how to modify the generator so that instances become more real-world-like [21]. Additionally, we have imposed the condition that instances must be discriminating of algorithm performance (not uniformly easy or hard for all considered algorithms) so that the instances can be used for learning the unique strengths and weaknesses of algorithms. New timetabling instances were generated using this method, and shown to produce instances that are more similar to real-world instances than those of the seed instance generator [22], while simultaneously eliciting different performance behaviours from competitive algorithms.

Critical to the success of being able to measure how similar a synthetically generated instance is to a real-world instance, or the degree to which it exhibits a certain property, is the idea of representing the instances in a common instance space. Our recent work [8] has developed powerful new methodologies to enable objective assessment of optimization algorithm performance within such an instance space. Given a set of test instances, we can now generate a visual representation of the instance space as a topological mapping of the instance properties, and measure the region of the instance space where an algorithm can be expected to perform well, known as the algorithm footprint [23,24]. We have applied this methodology to assess the power of state-of-the-art optimization algorithms in an objective manner [8,25,26], and shown that existing benchmark instances only populate a small portion of the available instance space [8,21,24], and often do not coincide with the location of real-world instances [21].

It is clear that relying solely on existing benchmark or randomly generated instances limits our ability to understand the strengths and weaknesses of algorithms. What is required now are methods to generate a large number of new test instances for a given problem, with controllable characteristics, to enable more rigorous testing of algorithms. Where are the existing benchmark instances in the instance space, and how diverse are they? How real-world-like are they? How discriminating are they of algorithm performance or do they elicit the same response from all tested algorithms? Where should new test instances be located in the instance space that would provide the most information about the strengths and weaknesses of algorithms? The instance space provides the ideal vehicle to recognize where new test instances are needed and, combined with genetic algorithms, offers a mechanism to generate new test instances that lie at target locations in the instance space. This represents a fundamentally different approach to generating test instances compared to previous methods, which lack the ability to control for some of the sophisticated properties that are critical to explaining algorithm performance.

This paper is the third in a series of papers developing a methodology for providing insights into algorithm strengths and weaknesses. The first paper [27] identified the properties of instances that affect difficulty, for broad classes of combinatorial optimization problems, and provides the starting point for constructing instance spaces for a new problem. The second paper [8] demonstrated how to construct an instance space, and how visualizing and measuring the area of algorithm footprints in the instance space can provide the objective assessment of algorithm strengths and weaknesses we seek. The current paper extends these ideas to propose the use of the instance space to understand where new test instances are required, and a methodology to evolve new instances with controllable properties exploiting the mathematical relationships between the instance properties and their location in the instance space. In Section 2 we define the instance space, using graph colouring as an example, to illustrate how an instance space is constructed. Section 3 then presents the methodology for evolving new test instances within the instance space. A collection of new graph colouring test instances is presented in Section 4, and their diversity as compared to existing benchmarks is discussed. Finally, conclusions are drawn in Section 5, where we also identify opportunities for further research on this important topic.

2. Defining an instance space

How can we build a single metric space that contains all known and unknown instances of a particular problem? The first step is to devise key properties or features of the instances that summarize their similarities and differences. For a given class of optimization problem, much is already known about which features tend to correlate with difficulty of instances [27,28]. Suppose we select a set of candidate features, and through a computational feature extraction process, we are able to represent each known instance as a feature vector. Equivalently, each instance can be represented as a point in a high-dimensional feature space, with the Euclidean distance (or any other preferred similarity metric) between points providing an estimate of the similarity between instances.

Assuming that we have an adequate set of features, we can then use dimension reduction techniques to visualize the instances in a 2-d instance space, after verifying that the inevitable loss of information has not destroyed too much of the topology [29]. The choice of candidate features, the selection of an optimal subset of features, and the dimension reduction methodology are all critical to generating a valid instance space for a given set of known test instances. The process is summarized in the left hand side of Fig. 1 which generates an instance space building upon the following spaces:

- the problem space $\mathcal{P}$ represents a possibly infinitely sized set of instances of a problem;
- the instance space $\mathcal{I}$ represents a finite set of existing test instances of a problem;
- the feature space $\mathcal{F}$ contains measurable characteristics of the instances generated by a computational feature extraction process applied to $\mathcal{I}$.

This framework has been developed in our previous work [8], by extending the Algorithm Selection model of Rice [30,31] (shown in the blue box in Fig. 1) to consider visualization of instance spaces and the insights that can be learned within them. For example, suppose we have the following additional data related to algorithm performance:

- 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.

Then the instance space can be used to visualize the regions where an algorithm is predicted to perform well based on previous empirical evidence, and machine learning methods can be used to define the boundary of good performance, known as the algorithm footprint [24,8]. The ideal instance space is one that maps the available instances to a 2-d representation in such a way that the easy instances and hard instances are well separated and the topology of instance (dis)similarity is preserved [29]. 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.

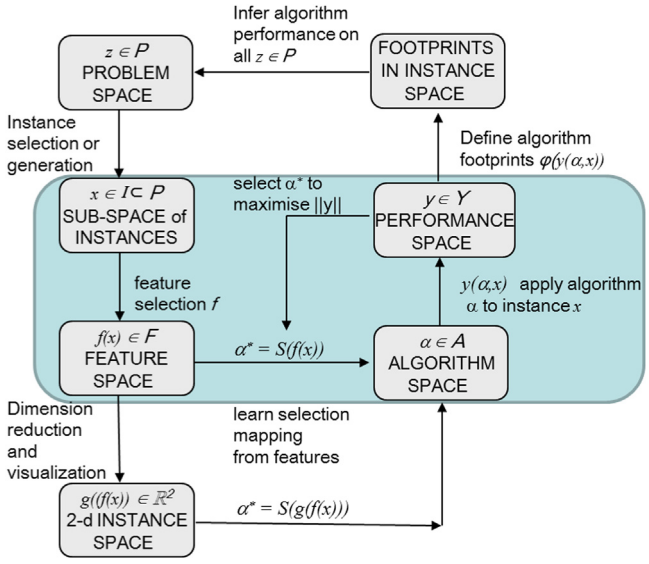

Fig. 1. Extended Algorithm Selection framework [8] based on Rice [30].

Generating a suitable instance space for a particular type of optimization problem is therefore a complicated interplay between selecting or generating diverse instances, measuring the right features that correlate with instance difficulty, and selecting the optimal subset of features that, when mapped to a 2-d 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 described here, and this could then be available as a resource for all researchers to explore how their algorithms perform in this space.

Additional insights can be revealed from the instance space as well. In particular, we can visualize where the existing test instances lie within this space. Where are there gaps, and why? What are the boundaries of the space, defined by the theoretical upper and lower bounds of each feature? Do the gaps correspond to regions where it is theoretically impossible for instances to exist, or is there an opportunity to generate instances that are located in the gaps? By studying algorithm footprints in the instance space we can also identify where new test instances could best be generated to provide the most information about the boundary of algorithm performance.

In the remainder of this section, we make concrete these abstract ideas by studying the graph colouring problem and presenting the methodology for generating an instance space, before using this instance space in Section 3 to present the methodology for new test instance generation. We start by defining the spaces $\mathcal{I} \subset \mathcal{P}$ and $\mathcal{F}$, before discussing the feature subset selection and dimension reduction method. This description is a summary of the more detailed discussion found in [8].

2.1. Graph colouring meta-data

We consider the same set of graphs, features and algorithms as reported in our previous work [8], and describe the meta-data as follows:

- The instance space $\mathcal{I}$ comprises 6948 graphs generated from a variety of sources: randomly generated bipartite graphs with random edges added (labelled class B); Joe Culberson's five graph generators – cycle-driven, geometric, girth and degree inhibited, IID, and weight-biased graphs (labelled classes C1–C5 respectively); DIMACS graphs (class D); graphs from the

real-world problems of social networks, sports scheduling and exam timetabling (labelled classes E, F and G respectively); flat graphs (class H); and random graphs (class I). The graphs range in size from 11 to 2419 nodes.

- The feature space $\mathcal{F}$ measures 18 features of the graph instances including properties relating to the nodes and edges, the cycles and paths on the graph, and the spectral properties based on the Laplacian and adjacency matrices of each graph. Namely, the number of nodes and edges, density, mean and standard deviation of the node degrees, mean path length, diameter and girth, mean and standard deviation of betweenness centrality, clustering coefficient, Szeged index, proportion of even closed walks to all closed walks, energy, standard deviation of eigenvalues of adjacency matrix, algebraic connectivity, and mean and standard deviation of eigenvector centrality.
- The algorithm performance space $\mathcal{A}$ includes the following eight graph colouring algorithms:
  - DSATUR: Brelaz's greedy algorithm (exact for bipartite graphs);
  - RandGr: Simple greedy first-fit colouring of random permutations of nodes;
  - Bktr: a backtracking version of DSATUR;
  - HillClimb: a hill-climbing improvement on initial DSATUR solution;
  - HEA: Hybrid evolutionary algorithm;
  - TabuCol: Tabu search algorithm;
  - PartCol: Like TabuCol, but does not restrict to feasible space;
  - AntCol: Ant Colony meta-heuristic.
- The performance space $\mathcal{Y}$ maps each algorithm to a real value as the percentage gap after a fixed number of iterations to the minimal number of colours required by the best performing algorithm for a given instance.

2.2. Generating the instance space

While many dimension reduction techniques can be considered, we have used Principal Component Analysis (PCA) to project the instances from the 18-d feature space to a 2-d 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 (leaving 6788 instances), 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 and mean-centred. For the feature subset selection, a 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 separability of the instance space: namely, the classification error of a Naive Bayes classifier (chosen as a simple machine learning method, and implemented with default parameters in MATLAB) using a given set of candidate features, projected to a 2-d instance space using PCA. High fitness values are awarded to feature subsets that enable the Naive Bayes classifier to correctly predict if the test instances are easy or hard. 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 in an iterative process that relied on an evolutionary algorithm to find the optimal subset of features.

Optimal subsets were found for cardinalities of the feature set from 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

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

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 Fig. 2. The grey instances show all instances, whereas the black instances show the location of particular instance classes 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 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). It should be noted that these three features with the weighting shown in Eq. (1) were the optimal subset, but there was not much difference in the error obtained when using additional features (which had smaller weights). Alternative instance spaces that we generated all took a fairly similar shape, just with some minor stretching, but the relative location of the instances within this shape did not change significantly. We therefore consider the instance space generated by Eq. (1) to be quite robust.

The view of the instance sets in Fig. 2 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 see that the real world instances (classes E, F and G) fall in small regions, quite distinct from each other. The randomly generated instances (classes H and I) are located in narrow bands within the instance space. Most interesting, we see that the existing comprehensive set of 6788 instances we have used to generate the instance space have failed to fill the entire instance space. If we seek to generate the most diverse and useful set of instances, then we must ask do graphs exist in these gaps in the instance space and beyond the boundary of the shape where the existing graphs lie? If it is theoretically possible for such a graph to exist, can we generate it? The methodology we present in the next section relies on the instance space to answer these questions, and to generate new test instances at targeted locations in the instance space. It should be noted however that the instance space has broader application than guiding the design of new test instances. We refer the reader to our earlier work [8] where we have used the instance space to overlay algorithm performance information, showing the regions where each algorithm is deemed to have performed well (according to some performance criteria), and the learned boundaries in the instance space where each algorithm's footprint can be inferred. Genuine insights into algorithm strengths and weaknesses can be revealed from the instance space, but this representation and potential for insights is fully dependent on the choice of instances that are used to generate the instance space. The methodology presented in the following section provides the

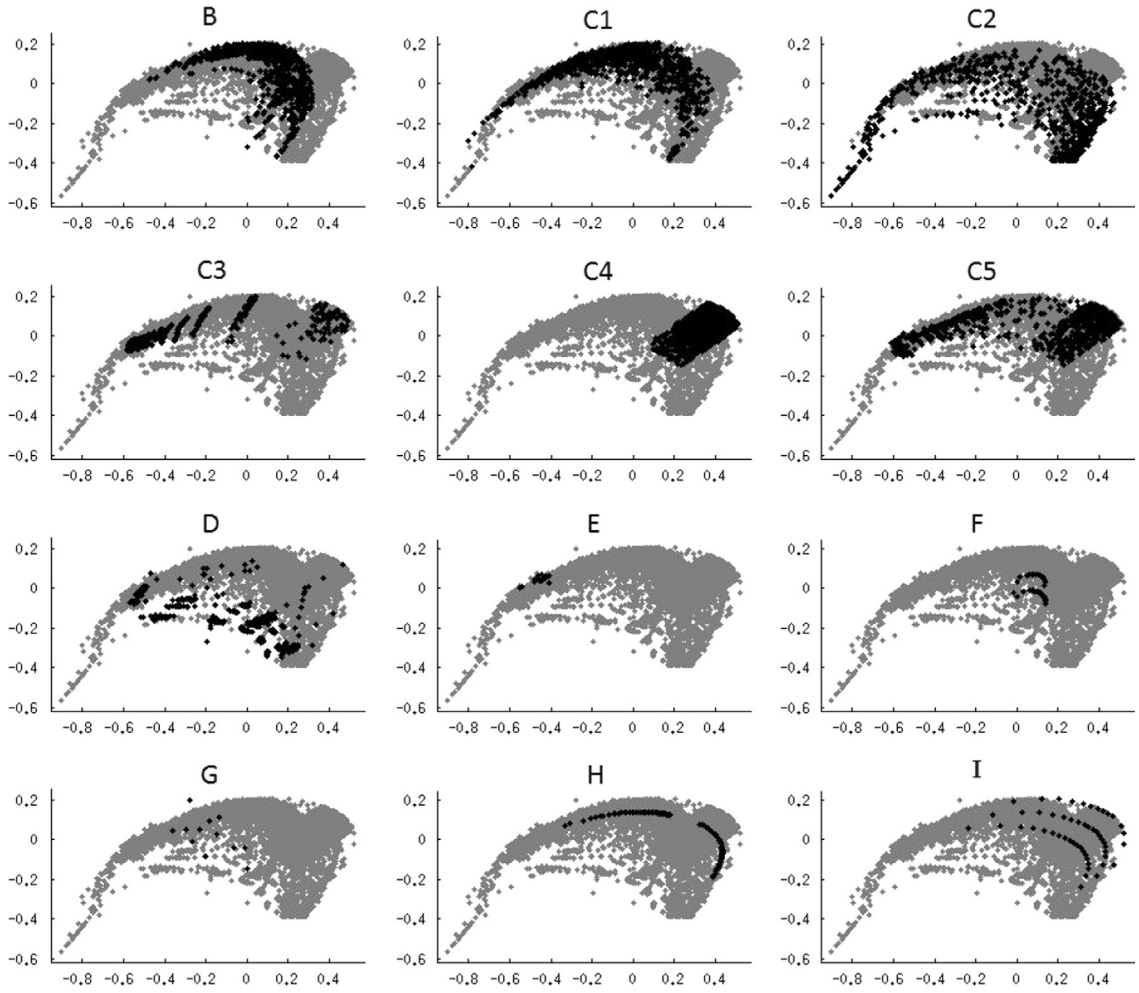

Fig. 2. Twelve views of the graph colouring instance space [8] showing location (in black) of each instance class.

means to generate new test instances with controllable characteristics and locations in the instance space to support insights.

3. Test instance generation in instance space

In this section we propose a new method to generate test instances with controllable characteristics. Clearly, random instance generators fail to create sufficient diversity of characteristics, as shown by the location of random graphs in class H and I of Fig. 2. Our study of the Travelling Salesman Problem has shown similar limitations of randomly generated and benchmark instances [24], with the TSPLIB instances being located in the middle of the instance space (corresponding to average features), alongside randomly generated instances, and failing to push the boundaries of the instance space. Intentionally creating instances with controllable structure, as shown by the locations of instances from Culberson's five instance generators (C1-C5) in Fig. 2, enables the instances to occupy a greater region of the instance space. This diversity is critical if we are to obtain insights into how test instance characteristics illicit different behaviours from algorithms.

Given the complex nature of some instance characteristics (such as the algebraic connectivity of a graph), it is not always possible to know how to adjust a random instance generator to ensure that instances with a particular value for a measurable feature are created. In the following section we propose the use of genetic algorithms to adapt and evolve random instances until they are located at target regions in the instance space, representing desired features for the instances.

3.1. Proposed methodology

Before selecting a target region in the instance space at which we would like to generate new test instances, we must first ask if it is theoretically possible for an instance to live at the target location in instance space. There is little point asking a genetic algorithm to try to find an instance that lives at a target location if no such instance with the required combination of features can exist. Utilizing tightest known lower and upper bounds of each feature, we can define a region in the high-dimensional feature space as a set of inequalities that contains all valid instances. Projecting these inequalities into the 2-d instance space using the same linear transformation from the PCA that created the instance space, we can define the region of the instance space $\mathcal{V} \subseteq \mathbb{R}^2$ within which all valid instances must lie.

Once the valid instance space is defined, we can inspect the location of existing instances within the valid instance space, and identify the gaps we wish to fill by generating new instances at target locations. Alternatively, if we wish to construct instances with specific measurable features, then we can use the linear transformation that created the PCA space to locate the desired coordinates in the instance space. Regardless of how we arrive at a target point $v_T \in \mathcal{V}$, we can define a fitness function for an instance $v \in \mathcal{V}$ inversely proportional to its Euclidean distance to the target point, with the fitness function maximized when $v = v_T$. One such fitness function is $F(v) = 1/(\|v - v_T\|^2 + \epsilon)$ for some small value of ϵ, although any function with similar properties will suffice. Of course, maximizing this function is equivalent to minimizing the distance to the target $\|v - v_T\|^2$, but we refer to the maximization problem to

maintain the genetic algorithm analogy with the survival of the fittest premise [32]. Utilizing the mechanisms of genetic algorithms, we can start with a population of randomly generated instances (all likely to be located within the region of our existing graphs), and rely on the process of crossover, mutation and the survival of the fittest premise to improve the average fitness of instances in each generation and move towards a targeted new region. The final generation should contain instances that are very close to the target point, and the process can be repeated for different target points. If the computation time is prohibitive, then the initial population could be chosen as existing instances that are closer to the target points, although this may result in some loss of diversity. The proposed methodology is summarized in Algorithm 1, which relies on a standard genetic algorithm implementation EvolveNewGeneration, taking the current population of instances and using suitable crossover, mutation and selection strategies to produce a new generation with improved average fitness. In addition to this genetic algorithm engine, the algorithm requires as input the linear transformation matrix generated from the PCA construction of the instance space ($\mathbf{A}$), the feature vector $\mathbf{f}$ for any instance, the upper ($\mathbf{f}^u$) and lower ($\mathbf{f}^l$) bounds on each feature (which may be expressed algebraically, as will be illustrated in the next section), the target point in the instance space ($\mathbf{v}_T$), the number of instances in the population (P), the number of generations (τ), and the parameter (ϵ) in the fitness function. This generic statement of the methodology is now made more concrete in the following section by revisiting the graph colouring case study to generate new instances.

Algorithm 1. Evolving new instances near a target point in instance space.

Require$(\mathbf{A}, \mathbf{f}, \mathbf{f}^u, \mathbf{f}^l, \mathbf{v}_T, P, \tau, \epsilon)$
  STEP 1: Define with inequalities $\mathcal{V} \subseteq \mathbb{R}^2$:
    $\mathbf{v} = \mathbf{A}\mathbf{f}(x) : \in \mathcal{V}, : \forall \mathbf{f}(x) \in \mathbb{R}^m : \mathbf{f}^l \le \mathbf{f}(x) \le \mathbf{f}^u$
  STEP 2: Select target point $\mathbf{v}_T \in \mathcal{V}$
  STEP 3: Randomly generate initial population of P instances as set I_0
  STEP 4: for $t=0$ to τ
    STEP 4a: For each instance $x \in I_t$ project into 2-d instance space:
      $\mathbf{v} = \mathbf{A}\mathbf{f}(x) \in \mathcal{V}$
    STEP 4b: Evaluate fitness functions of current generation:
      $F(\mathbf{v}) = \frac{1}{\|\mathbf{v}-\mathbf{v}_T\|^2+\epsilon}$, $\forall x \in I_t$ and $\mathbf{v} = \mathbf{A}\mathbf{f}(x)$
    STEP 4c: $I_{t+1} \leftarrow$ **Evolve New Generation** (I_t)
  Return I_τ

3.2. Defining the valid instance space

The existing graph colouring instances shown in the instance space in Fig. 2 have a very distinctive structure, with curved boundaries and some gaps in the interior of the embedding shape. We start by considering the reasons for this shape and to formally construct the valid instance space to ascertain if there are indeed additional instances that could be generated to increase diversity. Our feature selection process originally considered 18 properties of graphs, but settled on three that had the most explanatory power of how these graphs elicit different performance from graph colouring algorithms: density, algebraic connectivity and the energy of a graph. Each graph can therefore be first represented as a point in a 3-d space given by these three axes, before projecting to 2-d for ease of visualization. While there is no particular difficulty visualizing in 3-d, we are working on the assumption that our feature selection process could well have selected $m > 3$ features as optimal, and so we must find a generic method for understanding the boundaries of the valid instances in the 2-d instance space. We do this by considering the upper and lower bounds of each axis in the high dimensional feature space, which creates a bounding box of all valid instances. We then project each of the $2^{m-1}m$ edges of this bounding box using the linear transformation given by equation (1), before selecting target points within the valid instance space and evolving new graphs from initial random graph seeds.

3.2.1. Upper and lower bounds of the instance features

The density (ρ) of a graph $G(V,E)$ with a set of nodes or vertices V and edges E is bounded by [0,1] since $\rho = 2|E|/(|V|(|V|-1))$, and the maximum number of edges possible is $(|V|(|V|-1))/2$ for a fully connected graph. But what do we know of the range of permissible values of the algebraic connectivity and energy of a graph? Drawing on theoretical analysis of these graph properties and their best known upper and lower bounds, we can derive some inequalities that bound all graphs in the 3-d feature space. Let e be the energy of a graph [33], defined as the sum of absolute values of the eigenvalues of the graph's adjacency matrix. The mean energy $\bar{e} = e/|V|$ is a scale invariant metric. Let c be the algebraic connectivity of a graph [34], defined as the second smallest eigenvalue of the Laplacian matrix of a graph. The following known results help us construct suitable upper and lower bounds for these features:

The first upper bound for the energy of a graph is due to McClelland in 1971 [35] and gives $e \le \sqrt{2|V||E|}$, which can be simplified to give an upper bound on mean energy as a function of density and graph size (number of nodes), by expressing the number of edges also as a function of density:

$$\bar{e} \le \sqrt{\rho}\sqrt{(|V|-1)} \le \sqrt{(|V|-1)} \tag{2}$$

Additionally, Koolen and Moulton [36] provided an alternative upper bound on e, independent of ρ, which states

$$\bar{e} \le \tfrac{1}{2}(1+\sqrt{|V|}) \tag{3}$$

For lower bounds of the energy of a graph, Yu et al. [37] provide $e \ge 2\sqrt{|E|}$ which simplifies to

$$\bar{e} \ge \sqrt{\frac{2\rho(|V|-1)}{|V|}} \tag{4}$$

Further bounds are provided in Yu et al. [37], but inequalities (2)–(4) are the only ones that directly relate the properties of interest, namely the energy or its mean in terms of the density and size of a graph.

Similarly, we can derive inequalities for the bounding region of algebraic connectivity in terms of the density of the graphs. An upper bound on algebraic connectivity is $2|E|/(|V|-1)$ [38], which simplifies to provide upper and lower bounds in terms of density as

$$0 \le c \le \rho|V| \le |V| \tag{5}$$

since the second smallest eigenvalue of the Laplacian matrix (which is positive semi-definite) will always be non-negative. In fact, the algebraic connectivity will only be zero if the graph is disconnected [38]. The accuracy of these upper and lower bounds is shown in Fig. 3, where the bounding lines utilize the bounds (2) and (3) (selecting the tighter upper bound for a given density) and (4) for mean energy, and (5) for algebraic connectivity, as a function of density. The location of all instances from the 12 classes of graphs in our dataset are plotted separately in the $(\rho, \bar{e})$ and (ρ, c) spaces, and can be seen to lie within the bounding region, with randomly generated instances (shown in red) occupying a thin region of this space. We can also observe that it appears there is room for tightening the upper bounds for mean energy and lower bounds for algebraic connectivity for higher density graphs, since our empirical evidence suggests that the

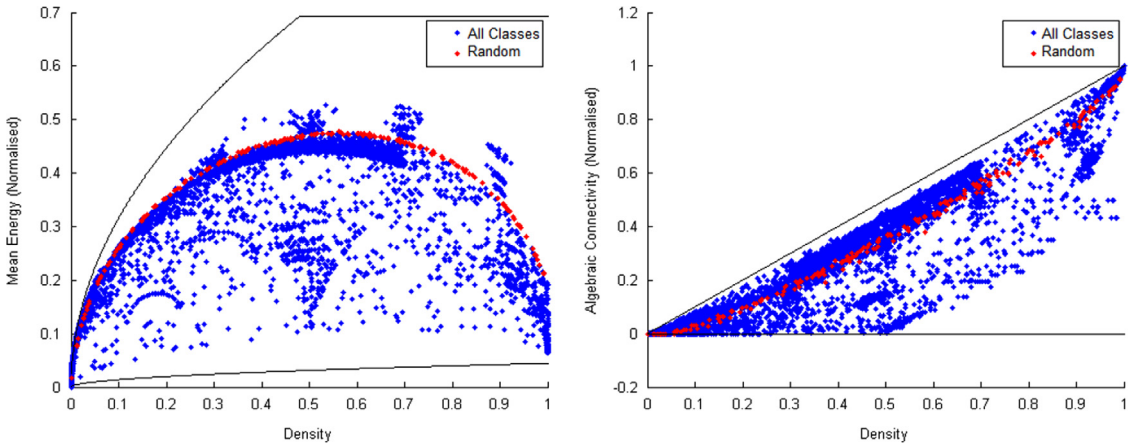

Fig. 3. Upper and lower bounds of normalized mean energy (left) and algebraic connectivity (right) for given graph density, and location of existing graphs within these bounds. (For interpretation of the references to colour in this figure caption, the reader is referred to the web version of this paper.)

current bounds are too loose. We should reserve judgement however until we have attempted to evolve instances to fill those gaps in the instance space before concluding if such graphs are likely to exist.

3.2.2. *Projecting the bounding box to determine the boundaries of valid instances*

These upper and lower bounds, on mean energy and algebraic connectivity in terms of density, together with the density bound [0, 1], provide us with the means to construct a wireframe for each of the $2^{m-1}m$ edges in the m-dimensional feature space. Our 12 edges can be parameterized with a variable t, that spans the extremes of each edge as t varies, as described in Table 1. Each of these edges is then projected using the PCA transformation to transcribe a curve in the 2-d instance space, providing our best estimate of the bounding region for the valid instances within the instance space, as shown in Fig. 4.

An improved PCA projection was adopted, rather than Eq. (1) based on our earlier work [8]. Firstly, it uses normalized features to allow fairer comparison of graphs of all sizes, and to bound the instance space on all sides. Secondly, it employs a more rigorous approach to determining a robust linear transformation: it uses 50% of the graphs to determine the linear transformation that maximizes variance preserved in 2-d; this process is repeated 10 times for different random subsets of graphs, and the resulting matrices of coefficients are then averaged. The final, more robust, linear transformation is provided by

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0.638 & 0.604 & 0.478 \\ -0.211 & -0.460 & 0.862 \end{bmatrix} \begin{bmatrix} \rho \\ c/|V| \\ \overline{e}/\sqrt{|V|-1} \end{bmatrix} \quad (6)$$

This PCA projection to 2-d retains 95% of the variation of all instances (all sizes $|V|$) found in the 3-d feature space, since the top two eigenvalues truly are dominant compared to the 3rd eigenvalue. Furthermore, a scatterplot of mean pairwise distances from each point to all others in the 3-d feature space versus the 2-d instance space shows a correlation coefficient of 0.99 supporting the view that the 2-d instance space has not compromised the topographical relationships amongst the instances compared to the 3-d feature space. While there has been no significant advantage created by visualizing in the 2-d instance space, compared to the optimally selected 3-d feature space, we are describing a more generic methodology here that provides considerable advantages if more features are required to adequately define the instance space.

3.3. *Identifying target points and evolving new instances*

Now that we have defined the valid instance space for graph colouring, we can identify promising gaps within which to target new instances. To illustrate the methodology, we consider only 100-node graphs, with the aim of identifying which regions of the instance space are occupied by existing 100-node graphs, and where we should be trying to evolve new 100-node graphs. It should be noted that our methodology for evolving new instances requires the size of the instance to be specified for encoding an instance within a genetic algorithm, but this process can be repeated for different sizes. Selecting only the 100-node graphs from classes C1–C5 (Culberson), class I (random), and all other 100-node graphs (shown in green, red and blue respectively in Fig. 5), we observe vast areas of the valid instance space where no 100-node instances currently exist. The same is true for every node size, so we focus without loss of generality on $|V| = 100$.

We apply our proposed Algorithm 1 with the following inputs: $|V| = 100$, $\mathbf{A}$ is the PCA linear transformation matrix given by Eq. (6), the feature vector $\mathbf{f}$ calculates $(\rho, c/|V|, \overline{e}/\sqrt{|V|-1})$, the valid instance space $\mathcal{V}$ is defined by the parametric bounding region shown in Table 1, the number of instances in the population is $P=200$, the number of generations is $\tau = 200$, the parameter $\epsilon = 0.2$ in the fitness function.

Graphs were encoded by unrolling the upper triangular part of the adjacency matrix of the graph (see Fig. 6). Given the context of graph colouring, where the interpretation of nodes as 'connected' is independent of edge direction, the adjacency matrix is symmetric and the upper triangular part fully describes the graph. Furthermore edge weights have no relevance in this context, so the encoded chromosome is a binary vector of length $0.5|V|(|V|-1)$ which is the total number of possible edges in the graph. A population of instances was stored in MATLAB as an $M \times N$ matrix, where M is the number of instances, and N is the length of the chromosome. This choice of encoding restricts only graphs of a single size to be evolved in a single run of the algorithm. Progression of instances to the next generation is carried out using the standard GA operators of selection, crossover and mutation. The selection method used is Linear Rank selection, using the maximum selection pressure parameter of 2. A variation on single point crossover is used to combine the selected instances into new instances for the next generation. The method used cuts the adjacency matrix at a random point on the diagonal and exchanges the 4 components (see Fig. 7). This has the structural effect of dividing each parent into two subgraphs, and producing a child using the internal connections of the subgraphs from one parent,

Table 1
Derivation of parametric edges of bounding regions for graphs with $|V|$ nodes in feature space $\mathbb{R}^m$.

Edge $(\rho, c, \overline{e})$	Parametric form	Parametric range
(min, min, free)	$(0,0,t)$	$t=0$
(min, free, min)	$(0,t,0)$	$t=0$
(min, max, free)	$(0,0,t)$	$t=0$
(min, free, max)	$(0,t,0)$	$t=0$
(max, min, free)	$(1,t,0)$	$\sqrt{\frac{2(\lvert V\rvert-1)}{\lvert V\rvert}} \le t \le \min\left(\sqrt{\lvert V\rvert-1}, \frac{1}{2}(1+\sqrt{\lvert V\rvert})\right)$
(max, free, min)	$(1,t,0)$	$0 \le t \le \lvert V\rvert$
(max, max, free)	$\left(1,t,\sqrt{\frac{2(\lvert V\rvert-1)}{\lvert V\rvert}}\right)$	$\sqrt{\frac{2(\lvert V\rvert-1)}{\lvert V\rvert}} \le t \le \min\left(\sqrt{\lvert V\rvert-1}, \frac{1}{2}(1+\sqrt{\lvert V\rvert})\right)$
(max, free, max)	$(1,\lvert V\rvert,t)$	$0 \le t \le \lvert V\rvert$
(free, min, min)	$\left(t,0,\sqrt{\frac{2t(\lvert V\rvert-1)}{\lvert V\rvert}}\right)$	$0 \le t \le 1$
(free, min, max)	$\left(t,0,\min\left(\sqrt{t}\sqrt{\lvert V\rvert-1}, \frac{1}{2}(1+\sqrt{\lvert V\rvert})\right)\right)$	$0 \le t \le 1$
(free, max, min)	$\left(t,t\lvert V\rvert,\sqrt{\frac{2t(\lvert V\rvert-1)}{\lvert V\rvert}}\right)$	$0 \le t \le 1$
(free, max, max)	$\left(t,t\lvert V\rvert,\min\left(\sqrt{t}\sqrt{\lvert V\rvert-1}, \frac{1}{2}(1+\sqrt{\lvert V\rvert})\right)\right)$	$0 \le t \le 1$

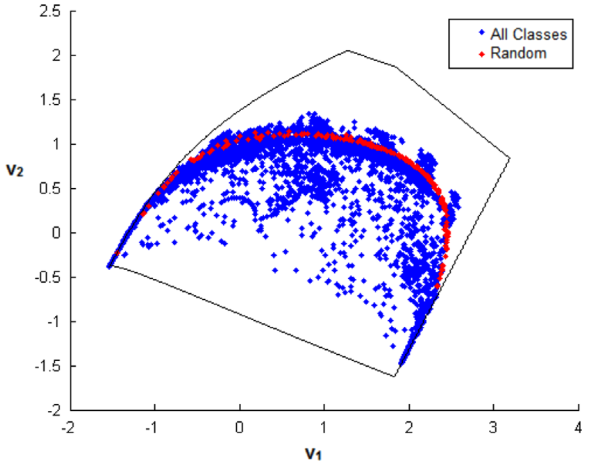

Fig. 4. Valid instance space boundary and graph instances for the normalized PCA space, for all 12 classes of graphs, with random graphs shown in red. (For interpretation of the references to colour in this figure caption, the reader is referred to the web version of this paper.)

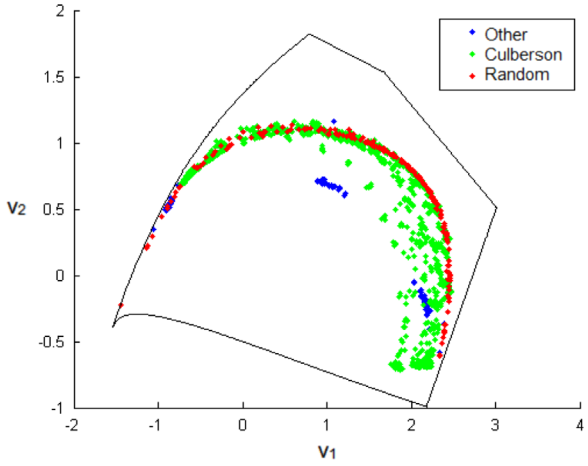

Fig. 5. 100-node graphs from our dataset in the instance space. (For interpretation of the references to colour in this figure caption, the reader is referred to the web version of this paper.)

and the connections between subgraphs from the other parent. Mutation of the instances is performed by randomly changing up the 3% of the elements in the chromosome (adding or removing random edges). Elitism is also applied in the progression, carrying the top 10% of the instances unchanged to the next generation. Crossover is performed on 100% of the non-elite instances, and mutation performed on 3% of instances. The initial population $\mathcal{I}_0$ comprised $P=200$ graphs: half of which were selected randomly from classes I and C1–C5 with $|V| = 100$, and the other half were existing instances with the highest fitness function (closest to target point). Each run of the GA was terminated after 200 generations.

Four strategies were explored, varying both the fitness function and method for selecting target points ($\mathbf{v}_T$), to determine the most efficient manner in which to fill the instance space with new graphs:

- Strategy 1: Apply a custom fitness function which rewards increased distance from the set of known instances. This was defined by considering known graphs as points of repulsion in the space.
- Strategy 2: Use a grid of target points across the valid instance space, with the fitness of a graph defined to increase with decreasing distance to a target point.
- Strategy 3: Identify target points along the theoretical boundary of the valid instance space.
- Strategy 4: Identify target points arbitrarily close to known instances (to accelerate convergence) but well spread across the instance space, and then include these to select new targets for the next iteration. Target points within the valid instance space were selected by requiring at least 10 neighbours in the known set to be within a radial distance of 0.2 in the instance space. A subset of these points was selected as target points by maximizing a weighted sum of distance from known graphs and distance from other target points, to give a spread of target points across the space.

For each run, 20 fittest graphs were saved from the final generation of the algorithm, as well as graphs produced in intermediate generations which occupied unique areas of the space (defined as uniquely located with a 0.2 radial distance).

4. Results and discussion

This section presents an analysis of the evolved 100-node graphs, and the insights they provide into the quality of the theoretical upper

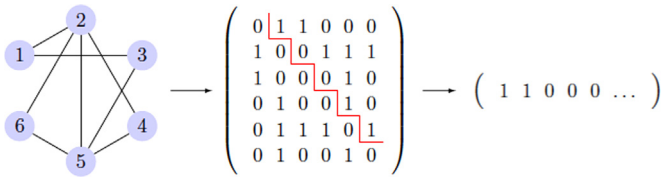

Fig. 6. String encoding of a graph for use by genetic algorithm.

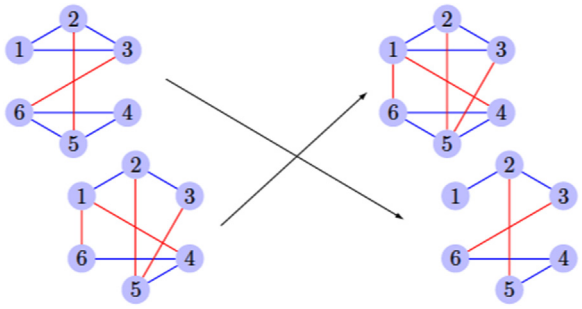

Fig. 7. Crossover operator applied to two parent graphs to produce two offspring graphs.

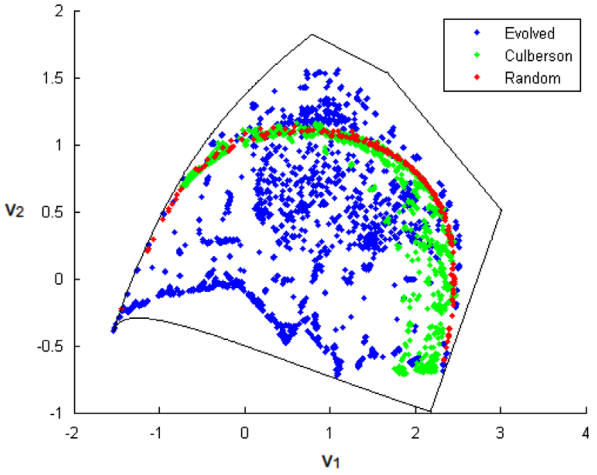

Fig. 8. Graph instances and expected boundary for the normalized PCA space, for 100 node graphs, showing the location of random and Culberson graphs and our newly evolved graphs. (For interpretation of the references to colour in this figure caption, the reader is referred to the web version of this paper.)

and lower bounds used to define the valid instance space, as well as the insights they offer about algorithm strengths and weaknesses. The effectiveness of the various strategies for selecting target points across the instance space is also discussed, and computational experiments are presented for smaller and larger graphs to provide an indication of the computational resources that may be required to fill the instance space with graphs of different sizes beyond the 100-node graphs used here to demonstrate the methodology.

4.1. Evolved graphs

Using the proposed methodology, a diverse set of 8278 new 100-node graphs was generated. The location of the evolved graphs within the valid instance space is shown in blue in Fig. 8, alongside the 100-node random graphs (shown in red) and Culberson's five classes of instances (shown in green). It should be noted that our evolved instances come much closer to the edges of the valid instance space than the random or Culberson instances. The evolved graphs clearly have very different combinations of density, algebraic connectivity and energy compared to existing graphs used as benchmarks for graph colouring, and the proposed methodology has successfully enabled us to control the generation of instances with desired characteristics in the chosen feature space. Recall that while we may be able to construct a graph with a required density, the other two features are much harder to control. This collection of 8278 graphs was the result of applying each of the target selection strategies discussed in Section 3.3. Strategy 1, with no target point required, was reasonably effective at generating new instances that were repelled away from existing instances, but repeating this process after adding the new graphs to the collection saw a tendency for the evolution to become trapped in a limited set of regions after a few runs. Strategies 2 and 3 resulted in many target points which were very far from any known instances. The performance of the GA was quite limited in many of these runs as diversity was lost well before any instances approached the target point. Strategy 4 achieved a good compromise, allowing individual runs of the GA to produce graphs slightly further away from the set of known instances each time. By continually updating the set of existing instances and generating new target points at each stage this method was able to progressively generate instances over a large area. Typically, selecting a target point with 10 nearest neighbours within a radius of 0.2 managed to achieve convergence to the target point within 100 generations. The computational issue in scaling up to larger graph sizes will be discussed in Section 4.4.

4.2. Empirical analysis of upper and lower bounds of graph features

The accuracy of the upper and lower bounds utilized to define the valid instance space can also be inspected by studying where, despite our best efforts to evolve new graphs, there is still a gap between the graphs we were able to create and the limits suggested by the best available theoretical upper and lower bounds. Fig. 9 shows that some of the bounds are quite accurate, but also identifies where there is likely to be further improvements that can be made, especially to try to reduce the upper bound for mean energy. Of course, caution must be applied when using this empirical evidence to suggest that no graphs exist in a certain region because we could not succeed in evolving a graph that lives at that location. Our evolutionary search is indeed a heuristic process, not exact, and failure to converge to a graph at a target point does not mean that such a graph does not exist – merely that our GA could not find it and perhaps needs parameter tuning or more attempts at the stochastic search. However, the empirical evidence can certainly motivate the effort to reconsider the theoretical foundations of bounds and develop new conjectures. The use of inequalities in feature spaces to empirically support theoretical analysis of upper and lower bounds has already been proposed [39], and our proposed methodology of evolving new instances at target locations in the instance space can clearly lend weight to this effort, with augmented collections of instances strengthening empirical evidence.

4.3. Insights from new instances

Besides demonstrating that we are able to evolve new graphs that lie in regions of the instance space that were not reachable using existing graph generation methods, we would also like to determine if the evolved graphs enable new insights into the strengths and weaknesses of algorithms. To this end, we have evaluated the performance of two common graph colouring heuristics, DSATUR (degree saturation) and MAXIS (maximal independent set) on two collections of 100-node graph instances: an original set using existing generators and an augmented set that adds our newly evolved 100-node graphs. The original set consisted of 1000 random graphs and 1500 Culberson graphs (300 each from the five generators) to produce an even larger set of

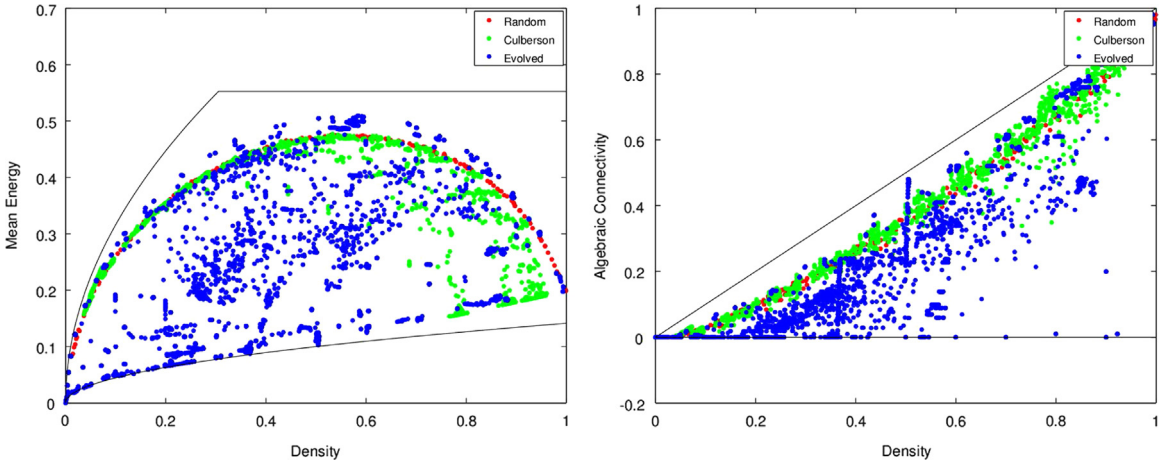

Fig. 9. Upper and lower bounds of mean energy (left) and algebraic connectivity (right) for given graph density, and location of 100-node graphs within these bounds.

Table 2
DSATUR versus MAXIS algorithm performance on original and augmented graph instance sets.

	DSATUR wins	MAXIS wins	Tied
Original set (2500 instances)	599 (24%)	762 (30%)	1139 (46%)
Augmented set (10 778 instances)	2875 (27%)	2313 (21%)	5591 (52%)

100-node graphs than we used before. The augmented set added our 8278 evolved graphs to the original set of 2500 graphs.

For both heuristics we used the implementation available for download from Culberson's website [17]. DSATUR was configured to use its standard ordering of breaking ties by colouring vertices of highest degree first. MAXIS was configured to use a branching factor of 3, no backtrack limit at the second pruning, and selection of next vertex in the independent set by highest degree vertex first. Both algorithms were given a maximum run time of 2 s before being terminated.

Table 2 shows the performance of both heuristics on both sets of instances. The performance is also visually illustrated in the instance space in Fig. 10, with the grey shadow showing the location of the original graphs, and colour-coded evolved instances to show where one algorithm achieved a better solution than the other (smaller chromatic number) or tied performance.

Based on the original set of known graphs, we would conclude that MAXIS on average has better performance (wins 30% of the time compared to DSATUR's 24%) and performs well across a wide range of graphs within the shadowed area in Fig. 10. However, the new set of evolved graphs exposes a previously unexplored region where DSATUR performance is better than MAXIS for many more graphs. In fact, across the augmented set of instances, the performance of DSATUR can be viewed as stronger (winning 27% of the time, compared to 21% for MAXIS). The evolved instances clearly have challenged our perception of algorithm performance that was formed when only examining graphs from the existing graph generators. Additionally, a new set of graphs in the upper region of the valid instance space have been created, where MAXIS is clearly better suited. Inspecting the distribution of our three features across the instance space in Fig. 11 reveals that MAXIS tends to perform better for graphs with high mean energy, while DSATUR prefers moderate mean energy graphs. Graphs with normalized mean energy below about 0.2 tended to elicit tied performance from both heuristics. These tend to be quite sparse graphs with low chromatic number that are easily solved by most algorithms [40]. Consequently, it is probably not worth the effort trying to generate further graphs along the lower boundary of the instance space, since these are easy for all algorithms and therefore uninteresting.

4.4. Computational issues

While the proposed methodology was applied to graphs of 100 nodes only, due to the string restrictions of a single run, there is no reason why the same algorithm could not produce graphs of different sizes. Just as we observe in Fig. 5 for 100-node graphs, we see a similar pattern of empty space around the currently available sets of graphs at each node size, so a tremendous opportunities exist for filling these spaces with a diverse collection of graphs of various sizes. What are the computational resource implications for scaling this up to larger graphs?

For reference, the time taken to perform a single run of the genetic algorithm (using a target point selected using strategy 4) has been measured for graphs of size 30, 100 and 300 nodes, requiring 13 s, 76 s and 15 min respectively resulting in at least 20 new graphs per run. The dimension of the encoding increases with the square of the number of nodes, and therefore generation progression time (dictated by the computationally expensive operation of calculating the graph features) increases at a similar rate. There are computational efficiencies that can be gained if necessary by locking in certain features, such as density for example, and evolving graphs that have a pre-defined value for density, but it would be best to enable the GA to search the entire feature space without such restraint. Of course much of the calculation can be parallelized to simultaneously fill the space with different target points, and we do not expect the computational resources required to generate thousands of new graphs of various sizes to be prohibitive. This computational effort is underway, and the resulting graphs will be made available through our project's website (monash.edu/matilda).

5. Conclusions

This paper is the third in a series that completes the methodology for understanding the strengths and weaknesses of optimization algorithms. Our previous work [8,27] has enabled objective assessment of algorithm performance by creating an instance space with all known instances of an optimization problem, within which the footprint of each algorithm can be viewed and measured. Recognizing, however, that our ability to gain insights into the strengths and weaknesses of algorithms in this manner is completely dependent on (i) the choice of features and (ii) the choice of test instances, this paper has proposed a new methodology for generating test instances that are located at target regions of the instance space. This enables us to generate new test instances that can clarify algorithm footprint boundaries, as well as generate test instances with controllable characteristics in a manner that has not

been achievable before apart from controlling for the simplest features. The methodology also provides a solution for the quest to generate more real-world-like instances, since the location of existing real-world instances can be readily identified, and new test instances can be evolved around that target location in instance space, while controlling for subtle perturbation of features.

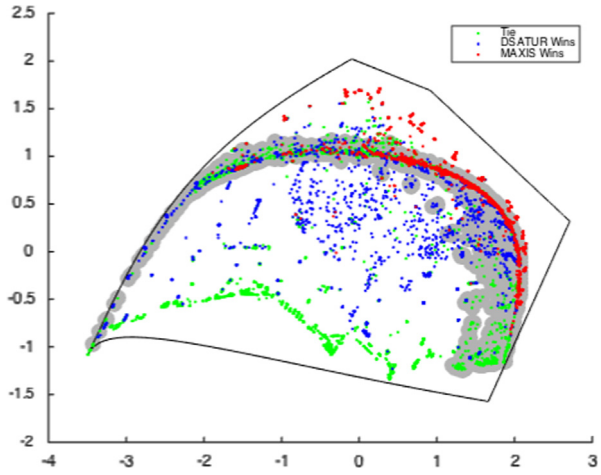

Fig. 10. Performance of DSATUR and MAXIS on evolved graph instances, with grey shadowing showing the location of the original instance set of 2500 graphs. (For interpretation of the references to colour in this figure caption, the reader is referred to the web version of this paper.)

The visualization power of the proposed methodology has been emphasized in this paper, but it should be noted that we were only able to represent the instances effectively in a 2-d instance space because the graph colouring features we selected appear to do an excellent job at explaining algorithm performance, and could be effectively reduced to two linear combinations of the features (the eigenvectors used as the axes for our instance space) without any significant loss of information or without significantly altering the topological relationships between instances. This will not always be possible for other problems we could study. In any case, there is nothing in this methodology (apart from the 2-d plots) that cannot be done in the higher dimensional feature space: the machine learning methods can determine the subset of features that best separates easy and hard instances in the higher dimensional space; target points can be identified in the higher dimensional space, and the GA can evolve new instances calculating distance to the target point also in the higher dimensional space. Of course, insights are more likely if we can benefit from the power of visualization, and we also benefit from faster computation times working in a 2-d space.

The methodology has been demonstrated in this paper on the graph colouring problem, and a large collection of over 8000 new 100-node graphs has been generated. The test instances are very diverse from each other, and completely different in structure (at least as measured by the three chosen features) from the existing 100-node graphs produced by Culberson's five graph generators. The utility of the instances to elicit different types of behaviours from graph colouring algorithms has also been illustrated by studying the algorithm footprints of two popular heuristics, DSATUR and MAXIS, on our augmented set of test instances. We

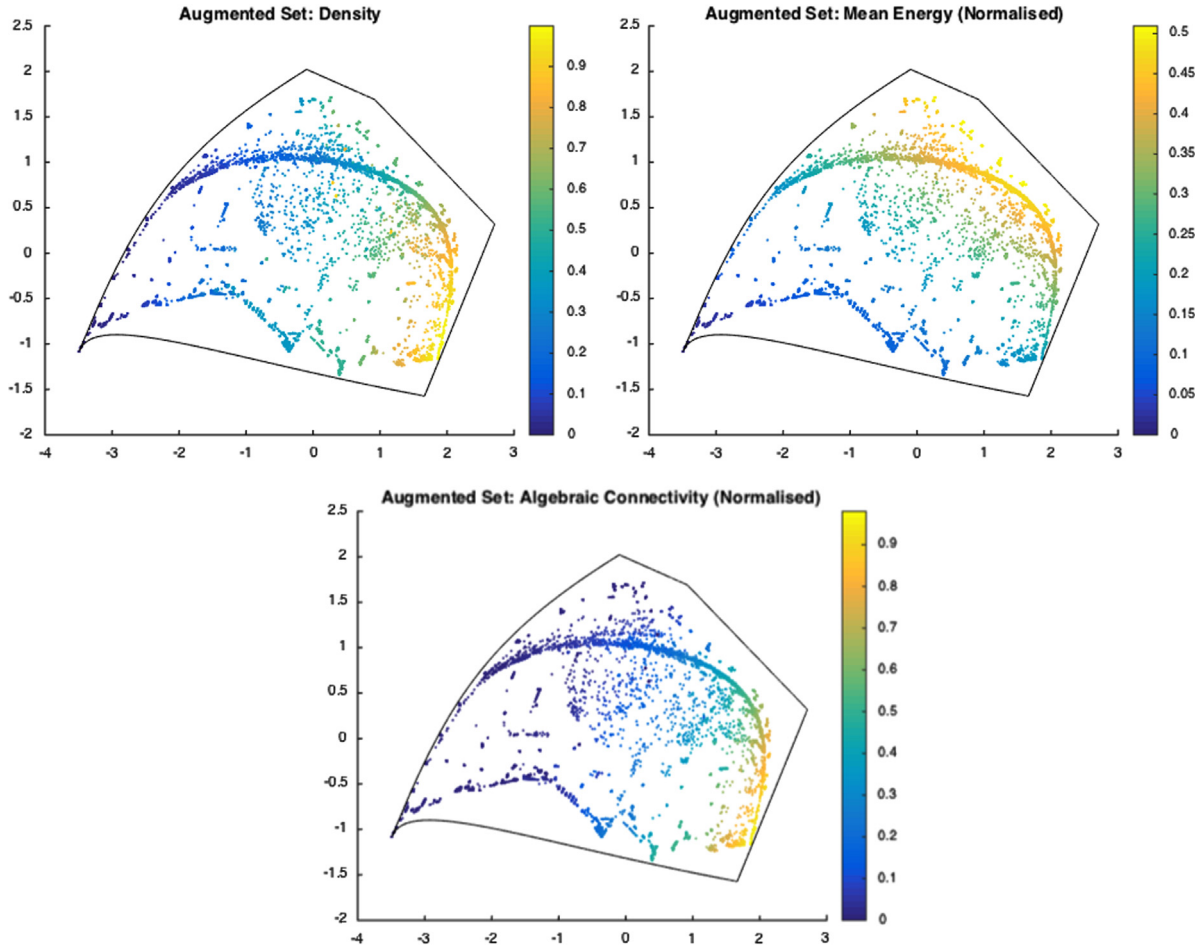

Fig. 11. Distribution of features across instance space for augmented set.

have generated new instances that enable each algorithm to show its strengths and weaknesses providing a different conclusion than if we relied only on studying graphs from existing generators.

Our future research will address the issue of how to select target points and fitness functions to ensure that the new test instances add significant value to the benchmark collection. That is to say, we should be evolving new instances that shed light on state-of-the-art algorithm strengths and weaknesses, and so discrimination power is an important criteria for selecting target locations in instance space. We will need to examine how the fitness function can be adapted to ensure evolved test instances are discriminating, challenging and include real-world-like instances, adapting ideas from our previous work [21]. For a wide range of optimization problems, we plan to generate instance spaces to identify where the existing benchmarks lie, and where we need additional new test instances that will enable new insights to be formed. Beyond the field of optimization, of course, the ideas presented in this series of three papers extend to any field that utilizes test instances to assess the relative power of algorithms (e.g. machine learning, forecasting, software testing, etc.) and we look forward to seeing these ideas extend to impact these other fields also in need of improved methodologies for stress-testing of algorithms [41,42].

Acknowledgements

This research is funded by the Australian Research Council under Grant DP120103678 and Australian Laureate Fellowship FL140100012. We gratefully acknowledge the two anonymous reviewers whose comments and suggestions strengthened the paper, and Joe Culberson's excellent web resources for graph colouring which made this research possible.

References

[1] Hooker J. Needed: an empirical science of algorithms. Oper Res 1994:201–12.
[2] Borwein JM, Bailey DH, Girgensohn R. Experimentation in mathematics: computational paths to discovery, vol. AMC 10; 2004. p. 12.
[3] Hooker J. Testing heuristics: we have it all wrong. J Heuristics 1995;1 (1):33–42.
[4] Johnson DS, Trick MA. Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11–13, 1993, vol. 26. American Mathematical Society; Providence, RI: 1996.
[5] Leyton-Brown K, Pearson M, Shoham Y. Towards a universal test suite for combinatorial auction algorithms. In: Proceedings of the 2nd ACM conference on Electronic Commerce. ACM; New York, NY: 2000. p. 66–76.
[6] Reilly CH. Synthetic optimization problem generation: show us the correlations!. INFORMS J Comput 2009;21(3):458–67.
[7] Hall N, Posner M. Generating experimental data for computational testing with machine scheduling applications. Oper Res 2001:854–65.
[8] Smith-Miles K, Baatar D, Wreford B, Lewis R. Towards objective measures of algorithm performance across instance space. Comput Oper Res 2014;45:12–24.
[9] Koch T, Martin A, VoßS. SteinLib: an updated library on Steiner tree problems in graphs. In: Xiu Zhen Cheng, Ding- Zhu Du, editors, Steiner trees in industry, volume 11 of combinatorial optimization, . Springer: Boston, MA, US, 2001, p. 285–325.
[10] Rosseti I, de Aragão MP, Ribeiro CC, Uchoa E, Werneck RF. New benchmark instances for the Steiner problem in graphs. In: Metaheuristics. Kluwer Academic Publishers; Dordrecht, the Netherland: 2004. p. 601–14.
[11] Schiavinotto T, Stützle T. The linear ordering problem: instances, search space analysis and algorithms. J Math Model Algorithms 2004;3(4):367–402.
[12] Stützle T, Fernandes S. New benchmark instances for the QAP and the experimental analysis of algorithms. In: Lecture notes in computer science, vol. 3004; 2004. p. 199–209.
[13] Cotta C, Moscato P. A mixed evolutionary-statistical analysis of an algorithm's complexity. Appl Math Lett 2003;16(1):41–7.
[14] van Hemert J. Evolving combinatorial problem instances that are difficult to solve. Evol Comput 2006;14(4):433–62.
[15] Asahiro Y, Iwama K, Miyano E. Random generation of test instances with controlled attributes. DIMACS Ser Discrete Math Theor Comput Sci 1996;26 (1):377–94.
[16] Culberson J. Hidden solutions tell-tales heuristics and anti-heuristics. In: IJCAI 2001 workshop on empirical methods in AI. Citeseer; 2001. p. 9–14.
[17] Culberson J. Graph coloring page ⟨http://www.cs.ualberta.ca/joe/Coloring⟩.
[18] McGeoch CC. Toward an experimental method for algorithm simulation. INFORMS J Comput 1996;8(1):1–15.
[19] Smith-Miles K, van Hemert J, Lim X. Understanding TSP difficulty by learning from evolved instances. In: Learning and intelligent optimization. Lecture Notes in Computer Science, vol. 6073; 2010. p. 266–80.
[20] Smith-Miles K, van Hemert J. Discovering the suitability of optimisation algorithms by learning from evolved instances. Ann Math Artif Intell 2011;61(2):87–104.
[21] Lopes L, Smith-Miles KA. Generating applicable synthetic instances for branch problems. Oper Res 2013;61(3):563–77.
[22] Burke EK, Mareček J, Parkes AJ, Rudová H. A supernodal formulation of vertex colouring with applications in course timetabling. Ann Oper Res 2010;179 (1):105–30.
[23] Corne D, Reynolds A. Optimisation and generalisation: footprints in instance space. In: Parallel problem solving from nature-PPSN XI. Lecture Notes in Computer Science, New York, NY: vol. 6238. 2010. p. 22–31.
[24] Smith-Miles K, Tan T. Measuring algorithm footprints in instance space. In: IEEE congress on evolutionary computation (CEC). IEEE; 2012. p. 1–8.
[25] Smith-Miles K, Wreford B, Lopes L, Insani N. Predicting metaheuristic performance on graph coloring problems using data mining. In: Talbi EG, editor. Hybrid metaheuristics. Boston, MA: Springer; 2013. p. 417–32.
[26] Smith-Miles K, Baatar D. Exploring the role of graph spectra in graph coloring algorithm performance. Discrete Appl Math 2014;176:107–21.
[27] Smith-Miles K, Lopes L. Measuring instance difficulty for combinatorial optimization problems. Comput Oper Res 2012;39(5):875–89.
[28] Macready W, Wolpert D. What makes an optimization problem hard. Complexity 1996;5:40–6.
[29] Goodhill GJ, Sejnowski TJ. Quantifying neighbourhood preservation in topographic mappings. In: Proceedings of the 3rd joint symposium on neural computation, vol. 6. Citeseer; 1996. p. 61–82.
[30] Rice J. The algorithm selection problem. Adv Comput 1976;15:65–117.
[31] Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput Surv 2008;41(1).
[32] Goldberg D. Genetic algorithms in search and optimization; 1989.
[33] Balakrishnan R. The energy of a graph. Linear Algebra Appl 2004;387:287–95.
[34] Biggs N. Algebraic graph theory. 67. Cambridge University Press; 1993.
[35] McClelland B. Properties of the latent roots of a matrix: the estimation of π-electron energies. J Chem Phys 2003;54(2):640–3.
[36] Koolen JH, Moulton V. Maximal energy bipartite graphs. Graphs Comb 2003;19(1):131–5.
[37] Yu A, Lu M, Tian F. New upper bounds for the energy of graphs. Commun Math Comput Chem/MATCH 2005;53(2):441–8.
[38] de Abreu NMM. Old and new results on algebraic connectivity of graphs. Linear Algebra Appl 2007;423(1):53–73.
[39] Mélot H. Facet defining inequalities among graph invariants: the system graphedron. Discrete Appl Math 2008;156(10):1875–91.
[40] Akbari S, Ghorbani E, Zare S. Some relations between rank, chromatic number and energy of graphs. Discrete Math 2009;309(3):601–5.
[41] Hyndman R. It's time to move from what to why. Int J Forecast 2001;17:567–70.
[42] Salzberg SL. On comparing classifiers: pitfalls to avoid and a recommended approach. Data Min Knowl Discov 1997;1(3):317–28.