Generalising Algorithm Performance in Instance Space: A timetabling case study

Kate Smith-Miles and Leo Lopes

School of Mathematical Sciences, Monash University, Victoria 3800, Australia
{kate.smith-miles,leo.lopes}@sci.monash.edu.au

Abstract The ability to visualise how algorithm performance varies across the feature space of possible instance, both real and synthetic, is critical to algorithm selection. Generalising algorithm performance, based on "meta-learning" on a subset of instances, creates a "footprint" in instance space. This paper shows how self-organising maps can be used to visualise the footprint of algorithm performance, and illustrates the approach using a case study from university course timetabling. The properties of the timetabling instances, viewed from this instance space, are revealing of the differences between the instance generation methods, and the suitability of different algorithms.

Keywords: Algorithm Selection, Timetabling, Hardness Prediction, Phase Transition, Combinatorial optimisation, Instance Difficulty

1 Introduction

Understanding the performance of optimisation algorithms for a class of problems involves studying the behaviour of the algorithms across an instance space defined by some measurable features or characteristics of the instances. The properties of the instances that we study to test the power of an algorithm are often defined by the source of the instances. Typically, instances use for testing optimisation algorithms either come from real world optimisation problems or they are synthetically generated via some instance generation procedure. However, it is often challenging to synthetically generate instances that are real-world-like [1]. If we intend to develop algorithms for the purpose of applying them to tackle real-world problems though, we need to understand the performance of the algorithms in the part of the feature space corresponding to the real-world instances. How do we visualise if a set of synthetic instances are similar to a small set of real-world instances? In addition, we may be interested to discover the types of instances that result in an algorithm excelling or failing, regardless of whether those instances are real-world-like.

All of these concerns are addressed by developing an understanding of the generalisation performance of algorithms across instance space. Corne and Reynolds [2] have noted that when claiming a certain algorithm performance it is important to make clear the boundaries of that performance in instance space. The performance of the algorithm on studied instances can be generalised to unseen instances in the same region of instance space. They introduced the idea of a "footprint" in instance space as a means to visualise the generalisation region, and noted that "understanding these footprints, how they vary between algorithms and across instance space dimensions, may lead to a future platform for wiser algorithm-choice decisions" [2]. Using two features at a time, the footprints of algorithm performance were shown in an instance space defined by two features of the problem under study (a scheduling problem and a vehicle routing problem). For problems with more than two significant features that define classes of instances though, a different visualisation approach will be needed.

In this paper we explore further these ideas of footprints and generalisation of algorithm performance in high dimensional instance spaces. We propose the use of self-organising feature maps to visualise a high-dimensional feature space as a two-dimensional map, where the generalisation footprint of algorithm performance can be clearly seen. Using a case study of course timetabling, we demonstrate how this view of the instance space is also revealing of the relationship between the instance generation method and the resulting features of the instances. Consequently, the limitations of synthetic instance generation methods when trying to create real-world-like instances can be explained. We consider the performance of two highly competitive algorithms across three classes of instances: real-world Udine timetabling instances [3], synthetically generated instances [4], and instances that have been iteratively refined via a learning process to resemble a seed set of real-world instances [5]. We seek to understand the regions of instance space where each algorithm is superior to the other, and whether the footprint of the algorithm's strong performance includes the region of real-world instances. By providing this kind of analysis of algorithm performance we hope to enable context-specific advice to be given on algorithm selection, particularly in practical real-world settings.

The remainder of this paper is as follows: In Section 2 we describe the timetabling meta-data for our case study. In particular, we describe the three classes of instances of course timetabling and how they were generated, we provide a comprehensive list of features of the timetabling problem that we will use to define the instance space (both from the timetabling characteristics and features of the underlying graph colouring problem), we discuss the two algorithms in our algorithm portfolio, and how we measure the performance of the algorithms on the instances. Once our meta-data has been defined in this way, the analysis of the relationship between features of the instances and algorithm performance can begin. In Section 3 we present some data mining approaches to understanding the relationships in the meta-data. We begin with a self-organising feature map to visualise the high-dimensional feature space as a two-dimensional map of the instance space. The footprints of each algorithm and the regions corresponding to the types of instances seen in practise are shown. We also partition the instance space using a decision tree to provide rules describing the differences between the classes of instances, and the differences between the performance of the two algorithms in terms of the features of the instances. In Section 4 we discuss these findings and draw conclusions.

2 Course Timetabling

Timetabling is better described as a class of problems, rather than a single problem type. This research focuses on the Udine Timetabling problem, also known as Curriculum-based Course Timetabling (heretofore CTT) problem. CTT was used for track 3 of the 2007 International Timetabling Competition (ITC2007). Our choice of CTT as a case study was motivated by several factors: the existence of instance generators as well as real-world instances; and access to two of the top five search procedures from ITC2007.

In the interest of space we describe the problem only briefly here. A detailed description of the problem can be found in [6].

CTT arises because students follow specific tracks along the coursework that leads to a degree. For example: a typical first semester engineering curriculum could be Calculus, Linear Algebra, Physics, Introduction to Computing, and Introduction to a subject of study, such as Mechanical Engineering (ME) or Chemical Engineering (ChE). Lectures for Introduction to ME can be scheduled at the same time as those for Introduction to ChE, but not at the same time as those for the first four courses. The instructor assigned to each course must also be available at the times for each lecture.

The set of conflicts within a particular instance of the CTT can be described as a conflict graph $G(V, E)$, where V is a set of vertices corresponding to "events" that need to be timetabled, and E is a set of edges connecting any two vertices when the events cannot occur at the same time. We can describe conflicts in this manner for both teachers and the curriculum.

There are also soft constraints, which incur penalties when they are violated, but do not invalidate the solution. If ME has 40 first-year students and ChE has 30, then the first four courses should be taught in a room with capacity for at least 70 students, while Introduction to ME and Introduction to ChE can be taught in rooms with capacity for 40 and 30 students respectively. All lectures for the same course should ideally be in the same room, and should be distributed throughout the week. Finally, the lectures from each curriculum should run consecutively.

This description is sufficient to describe the domain of our experiment according to a synthesis [7] of the framework in [8].

- The *problem space* $\mathcal{P}$ is the union of three sets of instances: the original 21 from the competition [3]; a set of 4500 obtained using the generator in [4]; and anther set of 4500 from [5]. The latter set was generated specifically to be differentiating of performance and similar to the original 21 instances. After excluding from $\mathcal{P}$ instances whose optimal solution violated a hard constraint (this was proved using an integer programming model), 8199 instances remained.
- The *performance space* $\mathcal{Y}$ is the sum of violations of soft constraints after 600s of computational work.

– The *algorithm space* $\mathcal{A}$ comprises two solvers: Algorithm A (TSCS [1]) is a Tabu Search over a weighed constraint satisfaction problem written in C++. Algorithm B (SACP [9]) is a constraint propagation code combined with Simulated Annealing written in Java. Of the 8199 instances in $\mathcal{P}$, 3694 resulted in draws; on 2409 instances TSCS won; and on 2096 instances SACP won.
– The *feature space* $\mathcal{F}$ is summarised in Table 1. In addition to straightforward features (like number of events), we use features related to landmarking [12] (obtained by running the DSATUR algorithm [13], which is optimal for bipartite graphs); features related to the conflicts, thus related to the underlying Graph Colouring problem and features that come from the application (Timetabling).

3 Visualising Instance Space

Now that we have assembled the meta-data for course timetabling based on two competitive algorithms, we are in a position to analyse the meta-data with a view to understanding the instance space and its properties. We seek to identify the various types of instances within the instance space and to understand the effect of instance generation method on the properties of the instances. We also seek to visualise the generalisation footprint of each algorithm's performance behaviours, and to determine the parts of instance space where one algorithm dominates the other. In order to visualise the high dimensional instance space (defined by the set of 32 candidate features in Table 1) we will be employing self-organising maps that produce a topologically-preserved mapping to a two dimensional space.

3.1 Self-organising Feature Maps

Self-Organising Feature Maps (SOFMs) are the most well known unsupervised neural network approach to clustering. Their advantage over traditional clustering techniques such as the k-means algorithm lies in the improved visualisation capabilities resulting from the two-dimensional map of the clusters. Often patterns in a high dimensional input space have a very complicated structure, but this structure is made more transparent and simple when they are clustered in a lower dimensional feature space. Kohonen [14] developed SOFMs as a way of automatically detecting strong features in large data sets. SOFMs find a mapping from the high dimensional input space to low dimensional feature space, so the clusters that form become visible in this reduced dimensionality.

The architecture of the SOFM is a feed-forward neural network with a single layer of neurons arranged into a rectangular array. Figure 1 depicts the architecture with n inputs connected via weights to a 3×3 array of 9 neurons. The number of neurons used in the output layer is determined by the user.

[1] The 3rd-placed solver in ITC 2007, by Astuta, Nonobe, and Irabaki. The authors have not published a paper on this solver.

Feature name	Description
Size related features: those that define the dimension of the problem (3 features).	
Number of Courses	Number of courses, independently of how many lectures are in each course.
Number of Events	Sum of lectures across all courses.
Number of Rooms	Total number of rooms available.
Landmarking features: obtained from landmarking the instance by running the DSATUR algorithm [10] (2 features).	
DSATUR Solution	Upper bound on the number of colours
DSATUR Colour Sum	Sum of colour values over all nodes (DSATUR tries to minimise this quantity)
Graph Colouring features: from each of the conflict graphs $G(V, E)$, where V is a course, and E is a conflict between two courses, generated by: the curricula; the teacher availability; and the combination of both constraints (21 features):	
Edge Density	$\frac{\|E\|}{(\|V\|-1)^2}$
Node Clustering Index[11] mean and standard deviation	For each node $v \in V$, the edge density of the graph induced by v and its immediate neighbours.
Unweighted Event Degree mean and standard deviation	The degree of each node v.
Weighted Event Degree mean and standard deviation	The sum of the enrolments of all neighbours of v.
Timetabling features: features that come from the constraints unique to timetabling, as opposed to conflicts, which are more closely related to Graph Colouring (6 features) .	
Slack	Total seats in all the rooms - Total seats required by all the courses.
One Room events	Number of events that will only fit in one room
Event Size mean and standard deviation	Number of students in each course
Room Options mean and standard deviation	The number of rooms into which each course can fit without penalty.

Table 1. All features used in the meta-data. For features that are computed for every node, both the mean and standard deviation of the resulting distribution are used.

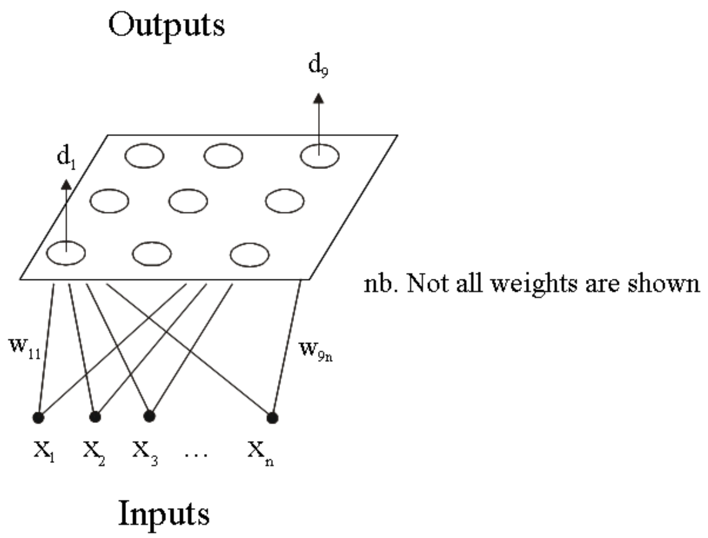

Figure 1. Architecture of Self-Organising Feature Map

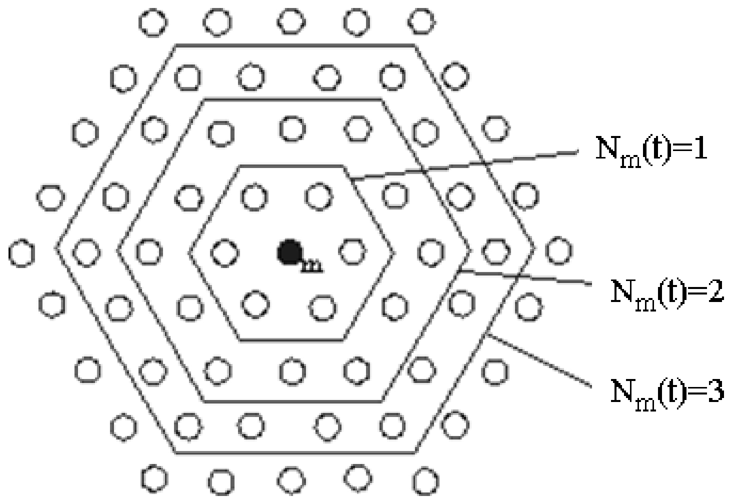

Figure 2. Varying neighbourhood sizes around winning neuron m

When an input pattern is presented to the SOFM, each neuron calculates how similar the input is to its weights. The neuron whose weights are most similar (minimal distance d in input space) is declared the winner of the competition for the input pattern, the weights of the winning neuron are strengthened to reflect the outcome, and the learning is shared with neurons in the neighbourhood of the winning neuron. This creates a process of global competition, followed by local cooperation. Figure 2 provides an example of how a neighbourhood N_m can be defined around a winning neuron m. Initially the neighbourhood size around a winning neuron is allowed to be quite large to encourage the regional response to inputs. As the learning proceeds however, the neighbourhood size is slowly decreased so that the response of the network becomes more localised. The localised response, which is needed to help clearly differentiate distinct input patterns, is also encouraged by varying the amount of learning received by each neuron within the winning neighbourhood. The winning neuron receives the most learning at any stage, with neighbours receiving less the further away they are from the winning neuron. If we denote the size of the neighbourhood around winning neuron m at time t by $N_{m(t)}$, then the amount of learning that every neuron i within the neighbourhood of m receives is determined by:

$$c = \alpha(t) e^{-\frac{\|r_i - r_m\|}{\sigma^2(t)}} \tag{1}$$

where $r_i - r_m$ is the physical distance (number of neurons) between neuron i and the winning neuron m. The two functions $\alpha(t)$ and $\sigma^2(t)$ are used to control the amount of learning each neuron receives in relation to the winning neuron. These functions are usually slowly decreased over time. The amount of learning is greatest at the winning neuron (where $i = m$ and $r_i = r_m$) and decreases the further away a neuron is from the winning neuron, as a result of the exponential function. Neurons outside the neighbourhood of the winning neuron receive no learning.

Like all neural network models, the learning algorithm for the SOFM follows the basic steps of presenting input patterns, calculating neuron outputs, and updating weights. The weight update rule, for all neurons within the neighbourhood of the winning neuron m for a given input pattern x_i is:

$$w_{ji}(t+1) = w_{ji}(t) + c[x_i - w_{ji}(t)]$$

with c as defined by equation (1). For neurons outside the neighbourhood of the winning neuron, $c = 0$. The initialisation stage involves setting the weights to small random values, setting the initial neighbourhood size $N_m(0)$ to be large (but less than the number of neurons in the smallest dimension of the array), and setting the values of the parameter functions to be between 0 and 1. The algorithm iterates through all of the input patterns repeatedly, with diminishing neighbourhood size and decaying functions $\alpha(t)$ and $\sigma^2(t)$ each time, until eventual convergence of the weights.

3.2 Visualising the Instance Space

In order to determine the features most likely to be predictive of algorithm performance, a correlation analysis was employed. Wherever a feature had a correlation greater than 0.7 with the performance metric of either algorithm, it was selected for inclusion as a feature for the SOFM. The selected features were:

- From the graph built from both curriculum and teacher conflicts:
  - the minimum colours and the colour sum (from the DSATUR algorithm); the clustering index; the edge density; the mean and standard deviation of the unweighted event degree;
- From the graph built only from curriculum conflicts; and from the graph built only from teacher conflicts:
  - the edge density; the mean and standard deviation of the unweighted event degree;
- Timetabling features:
  - The mean and standard deviation of Event Size and Room Options; slack; the number of one room events, courses, events, and rooms.

The instance space is therefore characterised by a set of 8199 course timetabling instances, each defined by a set of 21 features related to both the properties of the timetabling environment and the underlying graph colouring problem.

All features were normalised to the range [0,1] using variance. The software package Viscovery SOMine [15] was used to generate the SOFM, using a rectangular map of approximate ratio 100:52 based on the dimensions of the plane spanned by the two largest eigenvectors of the correlation matrix of the features (i.e. the first two principal components of the correlation matrix). The final map contains 2030 neurons arranged in 58 rows and 35 columns. 48 complete presentations of all 8199 instances were required to achieve convergence, with a decay factor of 0.5 applied to the functions $\alpha(t)$ and $\sigma^2(t)$. The initial neighbourhood size was 7.

Figure 3 shows that there are five natural clusters of instances in the 21-dimensional feature space when projected onto a two-dimensional map of instance space. Instances that belong to the same cluster are similar (according to Euclidean distance in 21-dimensional feature space) to each other, and significantly different from other instances in other clusters. The lower map in Figure 3 shows the location of the three classes of instances across the instance space. We find the small set of real-world Udine instances all located in the top-centre of the map (top right corner of cluster 2). Clusters 3, 4 and 5 contain predominantly the synthetic instances generated from the synthetic generator [4], and are not in the same region as the real-world instances. The instances that we have modified to be more "real-world-like" [5] (labelled as class 1 and shown as blue region on the map) surround the Udine instances and are therefore quite similar based on their features, but more diverse.

In order to determine which features make an instance more real-world like, we can inspect the distribution of features across the map. A subset of the features relating to the timetabling environment are shown in Figure 4, and some

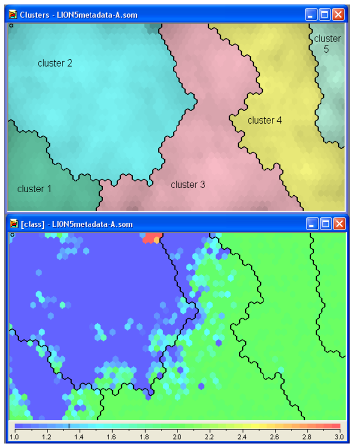

Figure 3. Five clusters in instance space (top) and the distribution of the three classes of instances across instance space (bottom). The real-world Udine instances are shown as red, the synthetic instances as green, and the refined synthetic instances as blue

of the features relating to the underlying graph colouring problem are shown in Figure 5. Here we see that one of the main differences between the Udine instances and our real-world-like instances is the mean and standard deviation of the degree of the teacher conflict graph (significantly smaller in the Udine instances). In addition, the mean and standard deviation of the event size is significantly smaller for the Udine instances. Thus we have obtained some immediate feedback on how to make our real-world-like instances more similar to the Udine instances.

The synthetic instances [4] are clearly quite different in distribution from the Udine instances. The main observations about these differences are revealed in the map by considering the boundary separating clusters 2 and 3, which correlates quite closely with the distribution of colorsum, the number of courses and mean room options.

It should be noted that no information about the class of instance was used to generate the clusters, only features of the timetabling problem and the underlying graph colouring problem. Yet the three classes of instances are clearly seen as quite distinguishable in this instance space. We now examine the performance of the two algorithms across the instance space with a view to visualising the footprint of their generalisation.

3.3 Visualising the Footprints of Algorithm Performance

Once the clusters have formed based on similarity of features of the instances, we can now superimpose additional information such as the performance of algorithms on those instances. The penalty of each algorithm for instances across the map is shown in the top row of Figure 6, with Algorithm A (TSCS) shown

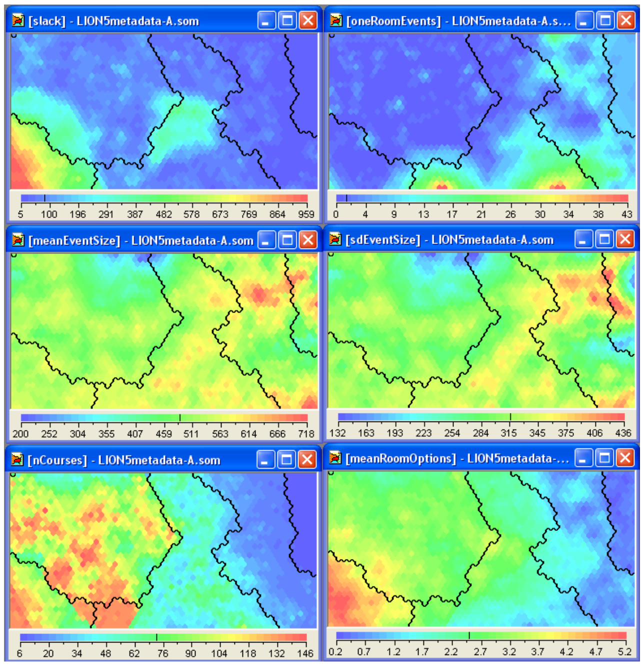

Figure 4. The distribution of timetabling based features across instance space

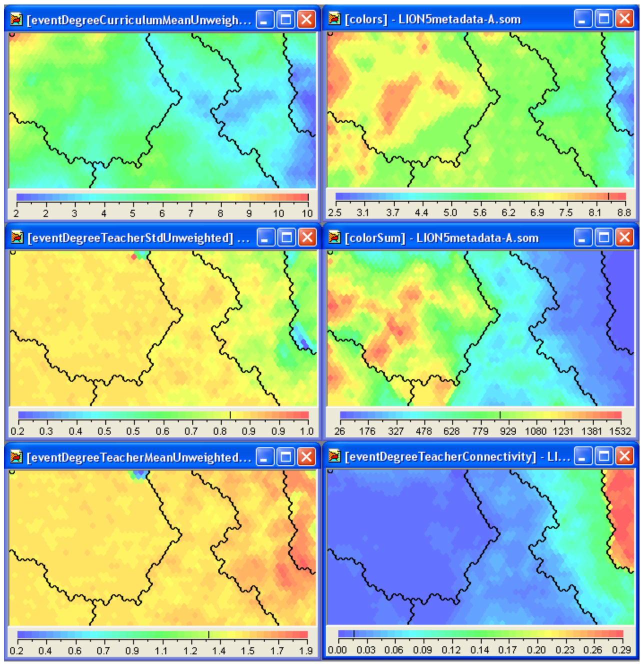

Figure 5. the distribution of graph based features across instance space

on the left and Algorithm B (SACP) on the right. Visually, these two algorithms appear to be very competitive with each other, producing low penalty solutions to instances in cluster 1, high penalty solutions to the difficult instances at the bottom of cluster 3, and similar penalties to each other across the map. The difference in penalty (Algorithm A minus Algorithm B) is shown in the lower left of Figure 6, and reveals that there is little difference in the performance of the algorithms on the synthetic instances in clusters 3, 4 and 5. Only instances in clusters 1 and 2 provide the opportunity for each algorithm to show its relative power. Since a difference of 1 penalty point is less meaningful for a high penalty solution than a low penalty solution, we also show the relative difference in algorithm performance on the bottom right of Figure 6. The relative difference is calculated as the ratio of the difference in penalty to the mean penalty for each instance, which reduces the impact of small differences in high penalty solutions.

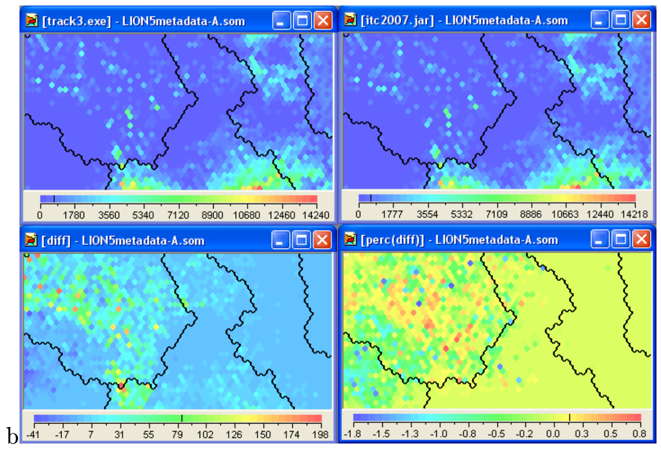

Figure 6. The performance of each algorithm across instance space. The penalty obtained by Algorithm A (TSCS) is shown top left, and Algorithm B (SACP) is shown top right. The difference in penalty between Algorithm A and B is shown bottom left (penalty of Algorithm A minus penalty of Algorithm B), and the relative difference is shown bottom right

Clearly, the boundary between cluster 2 and 3 provides some kind of partition across instance space to separate those instances that elicit identical performance from both algorithms from those instances that present unique challenges to each algorithm. It is also clear that there are some instances in clusters 1 and 2 where Algorithm A outperforms Algorithm B, and others where the reverse is true. Unfortunately, the regions where one algorithm clearly outperforms the other are not so well defined. There is a region spreading from the top left corner of the map diagonally down through cluster 2 where the relative difference is large and positive (i.e. Algorithm A has a much higher penalty than Algorithm B, and therefore Algorithm B is the superior algorithm for such instances). Returning to the features however, it is difficult to see a single feature that explains this diagonal pattern (possibly colorsum and number of courses, but the superior performance of Algorithm B does not continue into cluster 1 where colorsum and number of courses are also high). Likewise, the superior performance of Algorithm A in cluster 1 (and a small region just over the boundary into cluster 2) is not well explained by any of the features of the instances.

So while we have features that are clearly capable of distinguishing between the classes of instances (real world versus synthetic), and whether the instances will be discriminating of algorithm performance (or just elicit a tied outcome), it would appear that the selected features are not ideal for explaining the conditions under which Algorithm A outperforms Algorithm B and vice versa.

3.4 Partitioning the Instance Space via Decision Trees

Decision trees can be very powerful tools for elucidating rules that can help explain performance differences between algorithms. However, extraordinary care must be taken in designing the experiment, especially to ensure class balance and avoid bias.

In these experiments, we perform training on a random subset of each class of cardinality equal to the cardinality of the class with the fewest elements to help control bias.

Figure 7 summarises a decision tree experiment run on a subset of $\mathcal{P}$ obtained by randomly sampling 21 instances (the number of real instances from the ITC2007 competition) from each of the *synthetic* and *real-world-like* instances, then trying to separate all three types of instances. While the clustering index of the teacher conflict graph can be easily used to separate the real instances from the synthetic ones, such separation is inconsequential, at least if the goal is to compare these two solvers. The tree in Figure 8, obtained by randomly sampling an equal number of instances on which each solver won or there was a draw, illustrates that while it is possible to learn some aspects of the relationships between the features and which solver will win, the most important features are distinct from those in Figure 7.

Node	competition	real-world-like	synthetic
1. Root	21	21	21
2. Teacher Clustering Index Mean$<$ 0.4	21 (100%)	0	0
3. Teacher Clustering Index Mean$\geq$ 0.4	0	21 (50%)	21 (50%)
4. One Room Events $<$ 2.5	0	20 (87%)	3 (13%)
5. One Room Events $\geq$ 2.5	0	1 (5%)	18 (95%)

Figure 7. A decision tree that describes features that can be used to determine the origin of the instance. Unfortunately, these features do not help predict algorithm performance.

Node	SACP wins	Tie	TSCS wins
1. Root	2096	2096	2096
2. DSATUR colour sum ≤ 393	339 (17%)	1486 (76%)	126 (7%)
3. DSATUR colour sum > 393	1757 (41%)	610 (14%)	1970 (45%)
4. Slack$<$ 112	1337 (63%)	216 (10%)	567 (27%)
5. Slack ≥ 112	420 (19%)	394 (18%)	1403 (63%)

Figure 8. A decision tree that describes features that can be used to determine which algorithm will win. The results are insightful but inconclusive.

Another word of caution illustrated by Figure 8 is that simply counting wins can be a dangerous way to decide which solver is superior to the other. These data show that in problems in which the DSATUR colour sum is low, there is not a significant difference between the solvers, and that in the remaining instances, SACP wins on those instances in which there is little slack. This tree, therefore, supports the argument that SACP is the better solver on hard instances, since it provides evidence that SACP wins where there is little slack and heuristics have difficulty reducing conflicts. In contrast, if only raw performance data excluding feature information is used, a statistically strong conclusion would be reached that TSCS wins more often overall in $\mathcal{P}$, and that the difference in mean performance is greater than 0. This conclusion would hold even in the subset of $\mathcal{P}$ composed exclusively of real-world-like instances.

It is up to the reader of a specific research paper to decide whether or not it is important to them that a solver wins on a subset of instances that is harder. However, having that information – as opposed to relying on statistics over an entire instance set – is valuable independent of any subjective consideration. Furthermore, the information is supported by a repeatable (and challengeable), concrete experiment.

4 Conclusions

In this paper we have shown, through a case study of university course timetabling, that data mining techniques like self-organising feature maps and decision trees can be used to explore the high-dimensional feature space that defines an instance space. Specifically, the instance space can be visualised with a view to understanding the applicability of synthetic instance generators to real-world instances, and examining the generalisation footprint of algorithm performance. For our case study, we have utilised a comprehensive set of features based on both timetabling and graph colouring properties. We have demonstrated that these features create an instance space where the differences between real-world and synthetically generated instances are readily visualised. The effectiveness of different algorithms can then be superimposed across the instance space and the footprint can be visualised. For our chosen algorithms we have been able to partition the instance space to separate instances that elicit tied performance from these two highly competitive algorithms, from those instances where one algorithm outperforms the other. The chosen features have proven to be insufficient, however, for discovering the properties of instances that make SACP outperform TSCS, or vice versa. The footprint of both algorithms, where they performs well, includes the Udine real-world instances (which is why both algorithms performed well in the competition), but as we move away from the Udine instances in the instance space, we find some regions where one algorithm dominates. The boundaries of these regions are less well defined based on the current features. It remains for future research to consider additional features of instances that could distinguish between these two competitive algorithms, and the relationship between landscape metrics [16,17,18] and algorithm performance should also be considered.

The ability to generate instances that are both discriminating of algorithm performance and real-world-like is critical for progress in understanding the strengths and weaknesses of various algorithms, and ensuring that the right algorithm is being selected to avoid deployment failures for practical applications. Analysis of the kind presented in this paper provides a starting point to examine the characteristics of a set of instances, and enables feedback into the instance generation process [5] to develop a meaningful set of instances to drive future research developments.

References

1. Hill, R., Reilly, C.: The effects of coefficient correlation structure in two-dimensional knapsack problems on solution procedure performance. Management Science (2000) 302–317
2. Corne, D., Reynolds, A.: Optimisation and Generalisation: Footprints in Instance Space. Parallel Problem Solving from Nature–PPSN XI (2010) 22–31
3. McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A.J., Gaspero, L.D., Qu, R., Burke, E.K.: Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS JOURNAL ON COMPUTING **22**(1) (January 2010) 120–130
4. Burke, E.K., Mareček, J., Parkes, A.J., Rudová, H.: Decomposition, reformulation, and diving in university course timetabling. Computers & Operations Research **37**(3) (March 2010) 582–597
5. Lopes, L., Smith-Miles, K.: Generating applicable synthetic instances for branch problems. (2011)
6. Gaspero, L.D., McCollum, B., Schaerf, A.: The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3). Technical report, DIEGM, University of Udine (2007)
7. Smith-Miles, K.A., Lopes, L.B.: Measuring Combinatorial Optimization Problem Difficulty for Algorithm Selection. Annals of Mathematics and Artificial Intelligence, under review (2009)
8. Rice, J.: The Algorithm Selection Problem. Advances in computers (1976) 65
9. Müller, T.: Itc2007 solver description: A hybrid approach. In: Proceedings of the Seventh PATAT Conference. (2008)
10. Culberson, J., Luo, F.: Exploring the k-colorable landscape with iterated greedy. Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993 (1996) 245

11. Beyrouthy, C., Burke, E., Landa-Silva, D., McCollum, B., McMullan, P., Parkes, A.: Threshold effects in the teaching space allocation problem with splitting. European Journal of Operational Research (EJOR) (2008)
12. Pfahringer, B., Bensusan, H., Giraud-Carrier, C.: Meta-learning by landmarking various learning algorithms. In: Proceedings of the Seventeenth International Conference on Machine Learning table of contents, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA (2000) 743–750
13. Wood, D.: An algorithm for finding a maximum clique in a graph. Operations Research Letters **21**(5) (January 1997) 211–217
14. Kohonen, T.: Self-organized formation of topologically correct feature maps. Biological Cybernetics **43**(1) (January 1982) 59–69
15. SOMine, V.: Eudaptics software Gmbh
16. Knowles, J., Corne, D.: Towards landscape analyses to inform the design of a hybrid local search for the multiobjective quadratic assignment problem. Soft computing systems: design, management and applications (2002) 271–279
17. Bierwirth, C., Mattfeld, D., Watson, J.: Landscape regularity and random walks for the job-shop scheduling problem. Lecture notes in computer science **3004** (2004) 21–30
18. Schiavinotto, T., Stützle, T.: A review of metrics on permutations for search landscape analysis. Comput. Oper. Res. **34**(10) (2007) 3143–3153