**Instance Space Analysis for Course Curriculum Timetabling**

Udine Timetabling, also known as the curriculum-based course timetabling problem, was the subject of track 3 of the 2007 International Timetabling Competition (ITC2007). 21 real-world instances from the University of Udine were available. In a series of studies [1], [2,], [3], we have taken the top 2 performing algorithms in ITC2007 - simulated annealing with constraint propagation (SACP) and tabu search over weighted constraint satisfaction problem (TSCS) and examined their performance on instances from a variety of sources. Besides the 21 competition real-world instances, we used instances from a random generator of Burke et al. [4], and instances we generated by using machine learning to adapt the parameters of this random generator to create instances that are i) real-world like, and ii) more discriminating of unique algorithm performance [2]. A total of 8199 instances from these three classes are contained in the meta-data, along with 32 timetabling features. Applying instance space analysis now to these previous studies supports the findings in [3] (which used decision trees and self-organising maps at that time). There are features such as slack that determine unique algorithm behaviour, other features such as properties of the teacher connectivity graph that determine tied behaviours, and features that explain the key differences between the randomly generated instances and real-world instances.

A standard performance analysis such as shown in Table 1 would conclude that both algorithms perform quite similarly, with SACP best on 26% of the instances, TSCS best on 29% of the instances, and equal performance on 45% of the instances. TSCS is better on the randomly generated instances, although most of them elicit tied performance, and SACP is slightly better on the real-world like instances [3]. Performance on the Udine real-world instances is mixed and inconclusive which algorithm is better.

| | **SACP best** | **TSCS best** | **tied** | **total** |
|---|---|---|---|---|
| **Random** | 475 | 957 | 3060 | 4492 |
| **Real-world-like** | 1613 | 1442 | 631 | 3686 |
| **Udine** | 8 | 10 | 3 | 21 |
| **total** | 2096 | 2409 | 3694 | 8199 |
| **%** | 25.56% | 29.38% | 45.05% | |

**Table 1: Standard statistical analysis of performance results**

We now show the kind of insights that can be gained through instance space analysis.

**Instance Space Analysis**

Distribution of Instances

Of the 32 features considered, 5 were selected, with the optimal projection equation to the 2-d instance space given by:

$$\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} -0.3662 & -0.0519 \\ -0.2814 & 0.4637 \\ 0.1931 & -0.2114 \\ -0.0992 & -0.4511 \\ -0.2499 & -0.1637 \end{bmatrix}^T \begin{bmatrix} sdEventSize \\ slack \\ eventDegreeTeacherConnectivity \\ oneRoomEvents \\ eventDegreeTeacherMeanWeighted \end{bmatrix}$$

The 8199 timetabling instances, projected to this instance space, are shown in the Figure 1, coloured by the source of the instances. It is clear that the 21 real-world Udine University instances have very different features compared to the randomly generated instances of Burke et al. [4]. The instances that we have previously generated [3] by adapting the random generator to be more real-world like and discriminating (shown in red), and have more similarity to the real-world instances, and are more discriminating as shown below.

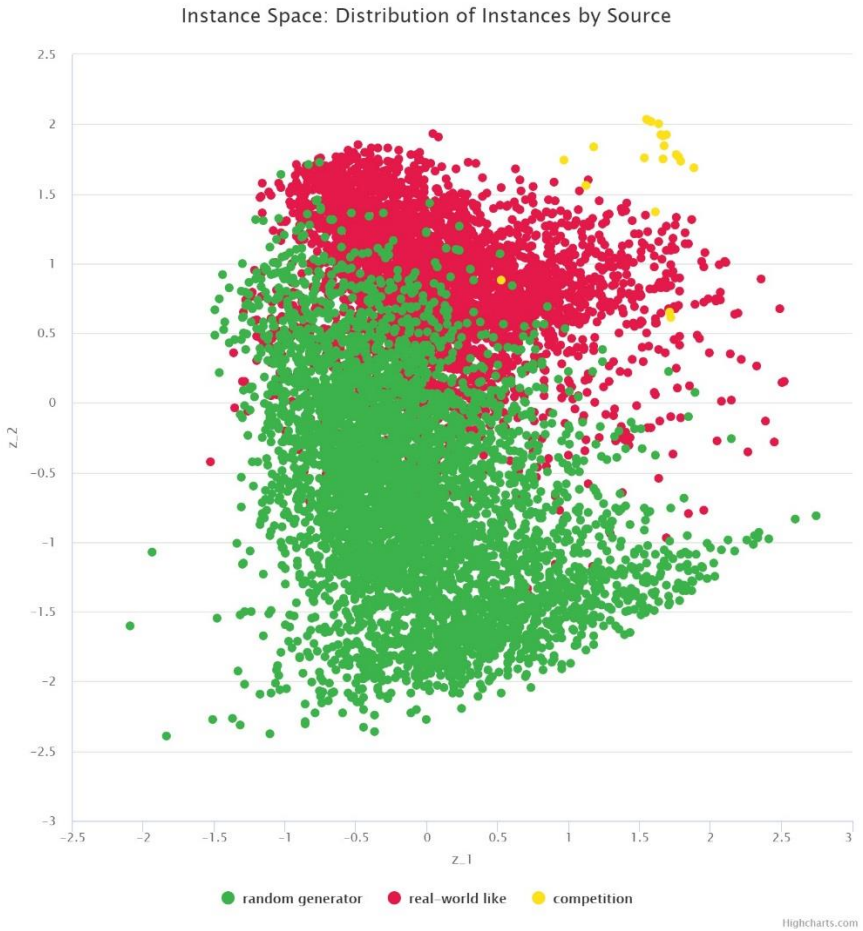


**Figure 1: Sources of instances**

Distribution of Algorithm Performance

Figures 2 and 3 show that both algorithms perform identically in the lower part of the instance space, where many of the randomly generated instances lie. These instances are not discriminating, either being equally hard or easy for both algorithms. TSCS is outperformed in the top of the instance space largely where the real-world like instances lie, and the performance of both algorithms on the real-world instances is mixed. It is clear that the instances in the bottom left corner (randomly generated) are equally hard for both algorithms, with solutions containing a large number of constraint violations, while instances in the bottom right are easy for both algorithms. The improved discrimination of the real-world like instances is clear, compared to the tied performance on the randomly generated instances.

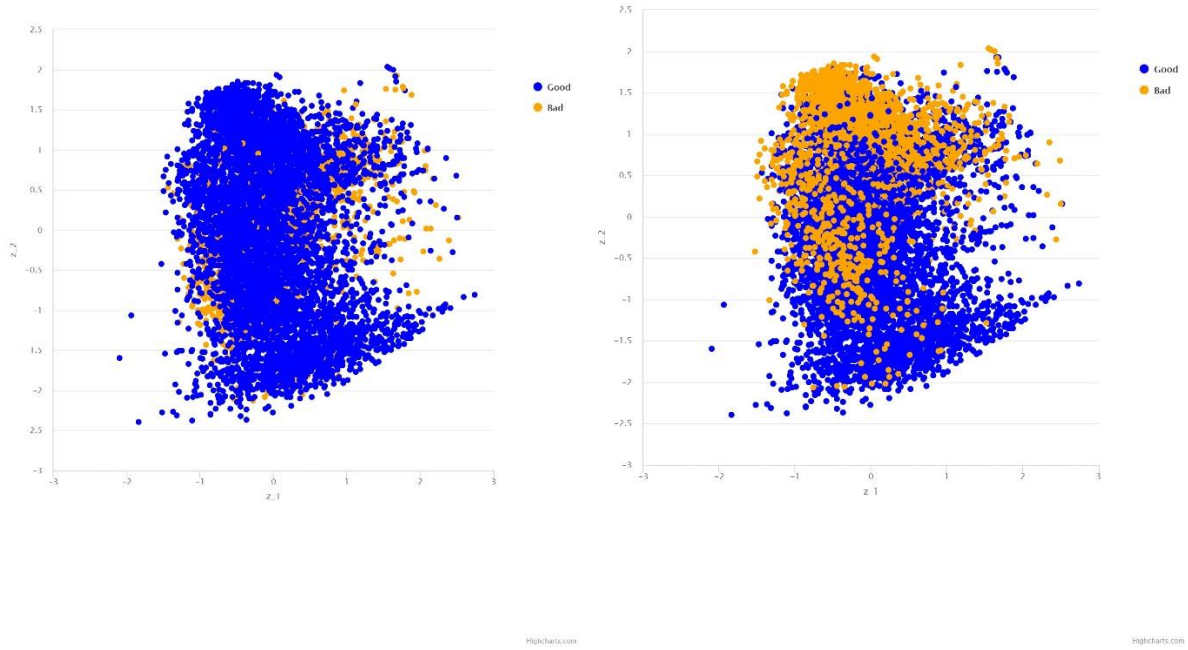


**Figure 2: Instances where each algorithm achieves best objective (minimal constraint violations) for SACP (left) and TSCS (right)**

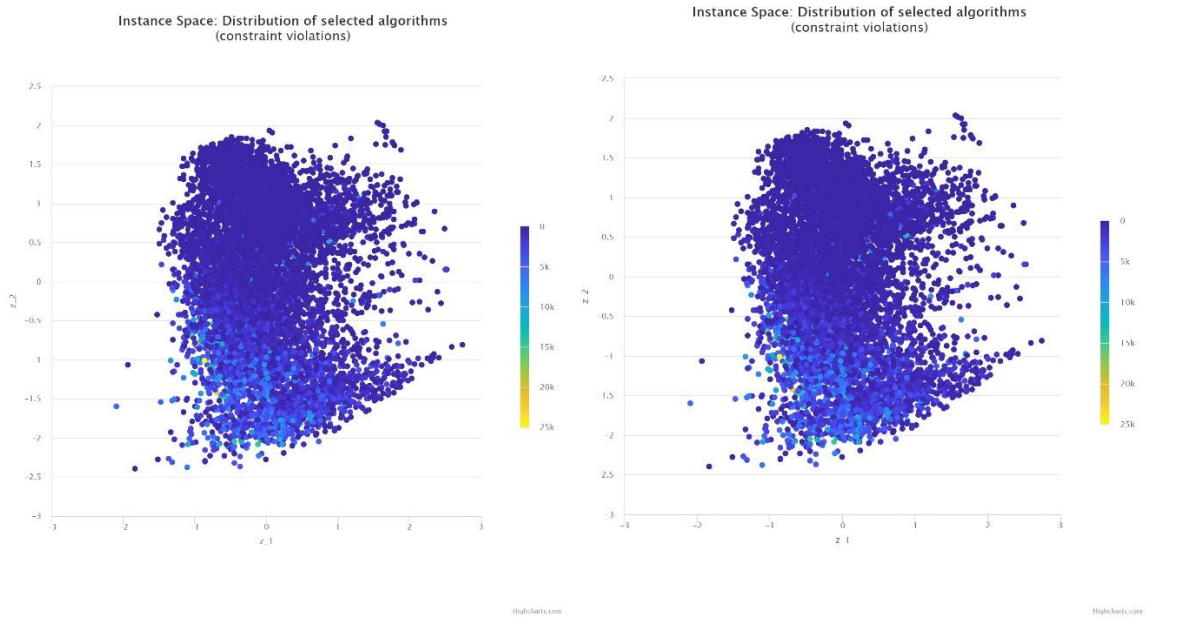


**Figure 3: Number of constraint violations achieved by each algorithm, SACP (left) and TSCS (right)**

Algorithm Selection via SVM

An SVM is used to learn in which regions of the instance space we can expect "good" performance for each algorithm. Figure 4 shows the SVM predictions, with good performance predicted from both algorithms in the bottom of the instance space, bad performance predicted on the real-world instances (insufficient evidence of consistently good performance for either algorithm in that region), and some unique expected strengths identified in the top (for SACP) and middle left (TSCS). Combining these SVM models into a single algorithm selection recommendation, Figure 5 shows which algorithm is expected to perform well, with ties broken by selecting the SVM with higher confidence. Note the large region where neither of these methods is recommended, due to lack of statistical evidence, and the region at the bottom where both methods are expected to perform

identically, but TSCS is recommended due to the higher confidence of its SVM model. The area of the footprint (predicted good performance) for SACP is 58.4% of the instance space, and for TSCS is 65.2%, but many of the instances where TSCS is expected to perform better are randomly generated instances that have very different properties to the real-world or real-world-like instances.

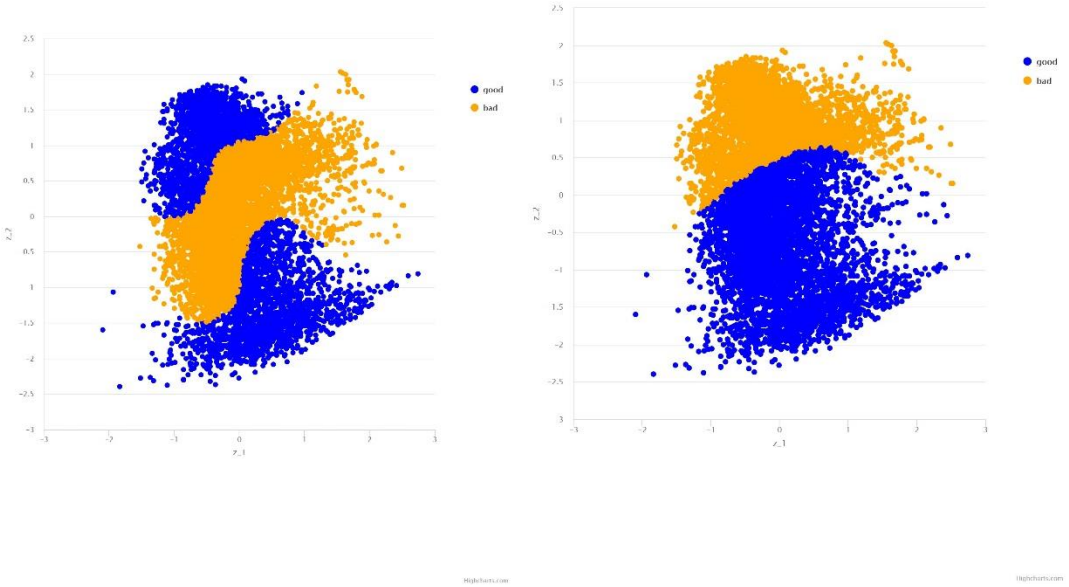


**Figure 4: SVM predicted performance of SACP (left) and TSCS (right), where good performance means relatively best (minimal constraint violations) compared to the other algorithm**

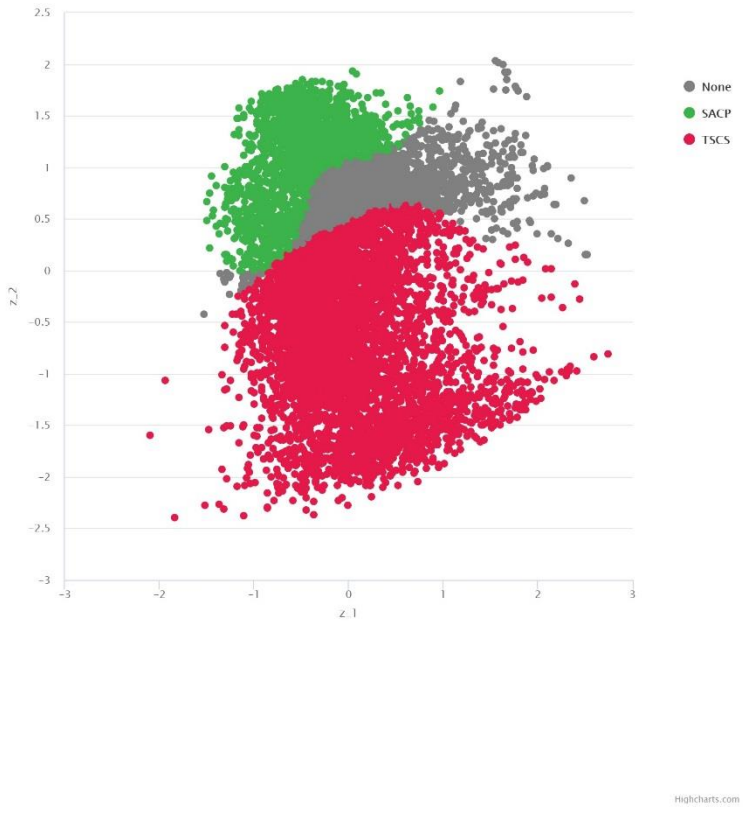


**Figure 5: Recommended algorithms from SVM prediction models**

Distribution of Features

Inspection of the distribution of features across the instance space in Figure 6 helps reveal the properties of instances in different regions and from different sources. We note that high slack values define the top region, from the real-world-like generator, where SACP has an advantage. Low slack defines the non-discriminating instances at the bottom, with easier instances having a high value ofeventDegreeTeacherConnectivity, and lower value of sdEventSize. Harder non-discriminating instances have a larger value of sdEventSize. The random generator produces instances that have a much larger number of oneRoomEvents compared to the real-world instances, and also higher values of eventDegreeTeacherConnectivity. Finally, we note that the real-world instances have very low values of eventDegreeTeacherMeanWeighted compared to either synthetic generator, so this provides useful feedback to further refine the generator parameters.

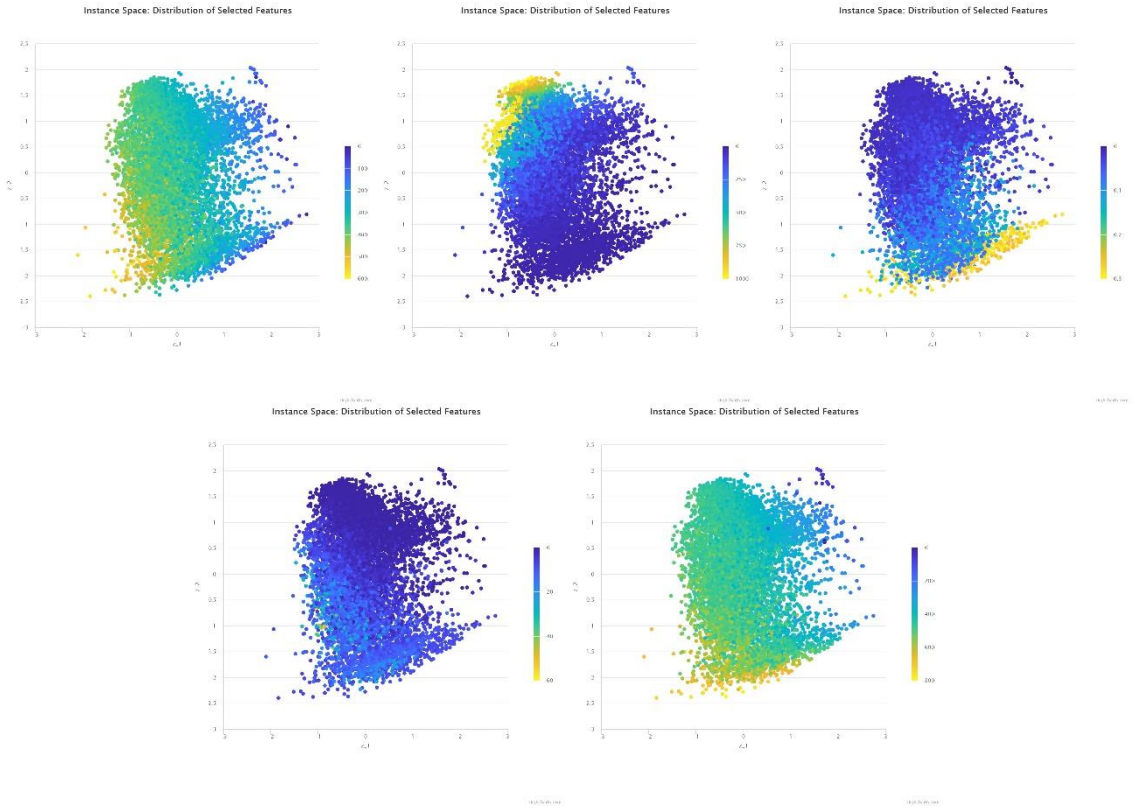


**Figure 9: Feature distributions for sdEventSize, slack, eventDegreeTeacherConnectivity, oneRoomEvents, eventDegreeTeacherMeanWeighted**

Insights into Algorithm Strengths and Weaknesses

Putting all of this analysis together we reach the following conclusions:

- Udine instances have very different properties to the random generator: much lower number of oneRoomEvents, and lower values of node degree for the two graphs TeacherConnectivity and TeacherMeanWeighted;
- Udine instances have lower slack values compared to the real-world-like generator [2];
- Both algorithms perform similarly on instances defined by higher values of node degree for the two graphs, with lower values of node degree making instances more discriminating;

- SACP performs better when eventDegreeTeacherConnectivity is slow, and when slack is high;
- TSCS performs better for mid-range slack values, with both algorithms performing similarly for low slack values;
- Harder instances (more constraint violations) are created by larger sdEventSize, but these are not discriminating nor real-world (like Udine instances);
- SACP is good when slack is low (less than 200) and best when slack is high (above 400);
- TSCS is good when slack is less than 400, and best for slack in the range [200-400];
- There are insufficient instances around the real-world Udine instances for the SVM model to have confidence in the prediction of which method is best, although both methods could find low cost solutions to these instances;
- Compared to the real-world-like instances, the Udine instances do not allow us to identify unique strengths and weaknesses of the two algorithms studied (they are easy for both, and if one doesn't beat the other, it is very close).

These conclusions allow a more insightful understanding of how algorithm performance depends on the instances, compared to the machine learning approach used to generate rules in [3]. While the decision tree rules in [3] are supported by this analysis, they are not as comprehensive, with much subtle nuance being overlooked by machine learning methods if it only affects a small subset of instances. The conclusions are also very different from those generated by an initial statistical analysis of the performance data, which would have concluded that, on average, both methods are similar, with TSCS having a slight advantage.

**References**:

[1] Lopes, L. B. and Smith-Miles, K. A., "Pitfalls in Instance Generation for Udine Timetabling", Proceedings of the 4th Learning and Intelligent Optimization conference, LION 4, Lecture Notes in Computer Science, vol. 6073, pp. 299-302, Springer, 2010. https://link.springer.com/chapter/10.1007/978-3-642-13800-3_31

[2] Lopes, L. and Smith-Miles, K., "Generating Applicable Synthetic Instances for Branch Problems", Operations Research, vol. 61, no. 3, pp. 563-577, 2013. https://www.researchgate.net/profile/Kate_Smith-Miles/publication/260162490_Generating_Applicable_Synthetic_Instances_for_Branch_Problems/links/57e1dbad08ae9e25307d4af2.pdf

[3] Smith-Miles, K. A., and Lopes, L. B., "Generalising Algorithm Performance in Instance Space: A timetabling case study", Proceedings of the 5th Learning and Intelligent Optimization conference, LION 5, Lecture Notes in Computer Science, vol. 6683, pp. 524-539, Springer, 2011. https://link.springer.com/chapter/10.1007/978-3-642-25566-3_41

[4] Burke, E. K., Marecek, J., Parkes, A. J. and Rudova, H., "A supernodal formulation of vertex colouring with applications in course timetabling", Annals of Operations Research, vol. 179, no. 1, pp. 105-130, 2010.