Journal of Scheduling (2022) 25:35–58
https://doi.org/10.1007/s10951-021-00701-x

Algorithm selection and instance space analysis for curriculum-based course timetabling

Arnaud De Coster[1] · **Nysret Musliu**[2] · **Andrea Schaerf**[3] · **Johannes Schoisswohl**[1] · **Kate Smith-Miles**[4]

Accepted: 1 July 2021 / Published online: 10 September 2021

Abstract
We propose an algorithm selection approach and an instance space analysis for the well-known curriculum-based course timetabling problem (CB-CTT), which is an important problem for its application in higher education. Several state of the art algorithms exist, including both exact and metaheuristic methods. Results of these algorithms on existing instances in the literature show that there is no single algorithm outperforming the others. Therefore, a deep analysis of the strengths and weaknesses of these algorithms, depending on the instance, is an important research question. In this work, a detailed analysis of the instance space for CB-CTT is performed, charting the regions where these algorithms perform best. We further investigate the application of machine learning methods to automated algorithm selection for CB-CTT, strengthening the insights gained through the instance space analysis. For our research, we contribute new real-life instances and extend the generation of synthetic instances to better correspond to these new instances. Finally, this work shows how instance space analysis and the application of algorithm selection complement each other, underlining the value of both approaches in understanding algorithm performance.

Keywords Timetabling · Scheduling · Algorithm selection · Classification · Instance space · Instance generation

1 Introduction

Course timetabling is a combinatorial problem that all universities must solve on a regular basis. It consists of assigning rooms and timeslots to courses, avoiding conflicts and maximizing student and teacher convenience. In this work, we consider a specific version of this problem, the so-called curriculum-based course timetabling (CB-CTT) problem, which was proposed as one of the three tracks of the Second International Timetabling competition (ITC-2007) (McCollum et al. 2010) in 2007, and has since received considerable attention in the scientific literature. The distinguishing feature of this version of the problem is that conflicts are based on predefined curricula, and not on student enrollment data.

As for most optimization problems, for CB-CTT there is no single "winning" algorithm that outperforms all others on all test instances. This suggests the possibility of applying automated algorithm selection (AS) (Rice 1976; Smith-Miles 2009) to the problem, which consists of having a portfolio of "good" algorithms and learning to predict which one is likely to be best for a given problem instance based on instance-specific features. Indeed, AShas been used successfully in competitions (e.g., Lin et al. 2008) to solve collections of benchmark instances of hard combinatorial optimization problems, better than any individual algorithm can, by selecting the likely best solver for each instance.

✉ Arnaud De Coster
e11704379@student.tuwien.ac.at

✉ Johannes Schoisswohl
e1327384@student.tuwien.ac.at

Nysret Musliu
musliu@dbai.tuwien.ac.at

Andrea Schaerf
schaerf@uniud.it

Kate Smith-Miles
smith-miles@unimelb.edu.au

[1] Database and Artificial Intelligence Group, Institute of Logic and Computation, TU Wien, Vienna, Austria

[2] Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling DBAI, TU Wien, Vienna, Austria

[3] DPIA, University of Udine, Udine, Italy

[4] School of Mathematics and Statistics, The University of Melbourne, Melbourne, Australia

Therefore one aim of this paper is to investigate whether AS can be used to obtain an effective portfolio-based solver for CB-CTT that performs better than any of the individual solvers on most instances. Beyond being able to identify in advance the algorithm most likely to be best suited to solving a particular instance, our interest here extends to a second aim of seeking greater insight into algorithm strengths and weaknesses. Such insight is critical for improving timetabling algorithm design. To achieve both of these aims, we adopt a recent methodology known as *instance space analysis* (Smith-Miles et al. 2014; Muñoz et al. 2018), which also offers the opportunity to scrutinize and improve the quality of benchmark test instances currently available to support algorithm selection and understanding of algorithm performance.

In this study, we have generated an extensive set of features describing CB-CTT problem instances, and constructed an instance space in which we study the performance of four state-of-the-art solvers, made available by the authors. Our instance space reveals that the ITC-2007 benchmarks, which have become the standard data set upon which algorithms are compared in the literature, as well as the instances created by the random generator proposed in Lopes and Smith-Miles (2013), occupy only a small region of the possible instance space, compared to the region that is occupied by other real-world instances we have considered. Therefore, based on our analysis, we extended the random generator to create a set of new and more diverse instances.

We decided to address specifically CB-CTT, rather than other popular timetabling problems, such as the post-enrolment course timetabling (PE-CTT) problem, due to the availability of many real-world instances, which are scarce or even totally absent for other problems (e.g., all PE-CTT instances are artificial).

This paper extends related literature where an earlier version of an instance space for CB-CTT was studied (Smith-Miles and Lopes 2011; Lopes and Smith-Miles 2010, 2013), but with a different focus and methodology. In Lopes and Smith-Miles (2010) it was pointed out that the random instances created by the generator of Burke et al. Burke et al. (2008) did not resemble the real world instances from University of Udine used in ITC-2007. In particular, it was observed that the random instances were not sufficient to distinguish the performance of the two different solvers studied in Lopes and Smith-Miles (2010, 2013). Due to this fact, the random generator was refined, as described in Lopes and Smith-Miles (2013), to generate instances that are more similar to those from ITC-2007. Subsequent work from Lopes and Smith–Miles (Smith-Miles and Lopes 2011) created an instance space (a 2D projection of the feature vector of the instances) of the following instance sets: randomly generated (Burke et al. 2008), ITC-2007, and using their modified random generator from Lopes and Smith-Miles (2013). The high-dimensional feature space was visualized using a self-organizing map (SOM), trained using a feature set described in Smith-Miles and Lopes (2011), and the performance of two heuristics (a tabu search solver[1], and a hybrid constraint propagation and simulated annealing based method (Müller 2009)) was studied.

Our current study described in this paper extends these initial investigations in light of the recently advanced instance space methodology (Smith-Miles et al. 2014; Muñoz et al. 2018), where visualizations of the instance space are based on both algorithm performance as well as instance features. This allows us to realize the aim of showing the strengths and weaknesses of solvers, while the SOM used in Smith-Miles and Lopes (2011) generated only clusters in the instance space, without taking any information from solvers into account. While in Smith-Miles and Lopes (2011) explanations of the differences between solvers were explored using a separate process (via decision trees), the visualization of those differences is built into the projection method used in the instance space analysis methodology adopted in this paper.

We propose a new method for randomly generating instances which aims to generate greater diversity in the instance space, in contrast to the random generator in Lopes and Smith-Miles (2013) which was tailored to produce instances more similar to ITC-2007and more discriminating of the two studied heuristics. Our instance generation process covers the space occupied by a broader set of real-life instances, including several new instances from diverse universities, which is a larger region than that covered by the real-life instances of ITC-2007alone. The new instances also explore more extreme regions of the instance space, to push the boundaries and algorithm testing and expose greater differences in the performance of solvers.

In addition, we consider an augmented set of features, and included more state of the art solvers in our analysis, which also include, for the first time, two exact solvers.

Our research contributions to the timetabling community can therefore be summarized as follows:

- An extensive set of features for describing CB-CTTinstances is introduced, extending previous features proposed in the literature.
- The properties of real-life and random instances based on existing generators are carefully analyzed. Based on this analysis, regions of the instance space that were not sufficiently covered by the generated instances are identified.
- A more diverse set of instances is generated to better support algorithm selection and improve the analysis of solver performance.

[1] http://www.cs.qub.ac.uk/itc2007/winner/atsuta.htm.

– Detailed experimental comparison of state of the art solvers on many new real-life and randomly generated instances is conducted.
– Algorithm selection is successfully applied for CB-CTT, and the provided features, especially probing features, are shown to be predictive.

The remainder of this paper is organized as follows. Section 2 describes the CB-CTT problem in detail. In Sect. 3, we present the instance space analysis methodology. Section 4 describes the meta-data we constructed for the instance space analysis and algorithm selection problem. Concretely this means we describe the instances, algorithms, features, and performance measures used for our experiments. In Sect. 5, the instance space is visualized and analyzed, and the new random generation procedure is described to create new instances to fill gaps in the instance space. Section 6 investigates the application of various machine learning models to perform algorithm selection for the CB-CTT problem. In the last section, we provide some conclusions of our work and discuss opportunities for further research.

2 Curriculum-based course timetabling

We consider the classical version of the Curriculum-Based Course Timetabling problem (CB-CTT), which is the one proposed for the ITC-2007. The detailed definition of CB-CTT is provided in McCollum et al. (2010), but in order to make the paper self-contained, we briefly recall it here. In essence, the problem consists of the following basic entities:

Days, Timeslots, and Periods. We are given a number of *teaching days* in the week. Each day is split into a fixed number of *timeslots*. Each pair (day, timeslot) represents a period.

Courses Each course consists of a fixed number of *lectures* to be scheduled in different periods. A course is attended by a number of *students*, and is taught by a *teacher*. For each course, there are a minimum number of days over which the lectures of the course should be spread. Moreover, there are some unavailable periods in which the course cannot be scheduled.

Rooms Each *room* has a *capacity*, specified as the number of available seats.

Curricula A *curriculum* is a group of courses that potentially have students in common. As a consequence, lectures of courses belonging to the same curriculum are *in conflict* and cannot be scheduled in the same period.

A solution of the problem is an assignment of a period and a room to all lectures of all courses so as to satisfy a set of *hard* constraints and to minimize the violations of *soft* constraints.

2.1 Hard constraints

There are three types of hard constraints:

Conflicts:	Lectures of courses either in the *same curriculum*, or *taught by the same teacher*, must be scheduled in distinct periods.
Availabilities:	A course may not be scheduled in an unavailable period.
RoomOccupancy:	Two lectures cannot take place in the same room in the same period.

A solution that does not satisfy the hard constraints is called infeasible. In order to rank infeasible solutions, the ITC-2007 rules count the *distance to feasibility*, which is measured as the number of violated hard constraints.

2.2 Soft constraints

There are four types of soft constraints:

RoomCapacity:	For each lecture, the capacity of the room assigned to it must be at least equal to the number of students attending the course. The cost for the violation is equal to the number of students in excess.
MinWorkingDays:	The lectures of each course must be spread into a given minimum number of days. The cost is the (positive) difference between the actual number of days and the minimum.
IsolatedLectures:	Lectures belonging to a curriculum should be adjacent to each other (i.e., be assigned to consecutive periods in the same day to minimize student idle time). There is a violation of this constraint every time, for a given curriculum, there is one lecture not adjacent to any other lecture of the same curriculum within the same day.
RoomStability:	All lectures of a course should be given in the same room. The cost is the number of distinct rooms used by the course minus one.

The total cost of a solution is the weighted sum of the costs incurred by violations of the soft constraints above. For the version of the problem that we consider, the weights are 1,

5, 2, and 1, respectively. Other versions of the problem are proposed and classified in Bonutti et al. (2012), in which this one is named UD2.

3 Framework: algorithm selection and instance space analysis

An *instance space* is a visual representation of the relationship between structural properties of problem instances and the performance of algorithms on those instances. It seeks to project instances into a 2D plane in such a way that easy and hard instances are well separated, and the distribution across the instance space of both algorithm performance metrics and instance features are mapped in a way that facilitates visual interpretation of their relationships. Instance space analysis was first proposed by Smith-Miles et al. Smith-Miles et al. (2014), building upon the algorithm selection problem framework of Rice Rice (1976). Through the analysis of the instance space it is possible to understand how the varying characteristics of problem instances affect performance of algorithms, exposing algorithm strengths, weaknesses, and suitability for various classes of instances. Such insights support a more nuanced understanding of algorithm performance than a mere average performance metric across a set of test instances can reveal. Instance space analysis also supports scrutiny of the adequacy of benchmark instances—their diversity, bias, discrimination ability and real-world-likeness. Furthermore, gaps in the instance space can reveal opportunities to generate new test instances with controllable characteristics to reach target locations in the instance space (Smith-Miles and Bowly 2015), enabling more comprehensive benchmark test suites to be developed to support stronger conclusions about algorithm performance reliability. The methodology of instance space analysis has been applied to a variety of problems including several in combinatorial optimization (Smith-Miles 2009; Smith-Miles et al. 2014), continuous optimization (Muñoz and Smith-Miles 2017), and machine learning (Muñoz et al. 2018).

Figure 1 illustrates the framework underpinning the development of the instance space: the five core component *spaces* of the algorithm selection problem framework proposed by Rice Rice (1976). The first is the general *problem space*, $\mathcal{P}$, which contains all the relevant instances in the application problem (e.g., CB-CTT). However, we only have computational results for a subset of all possible problem instances, $\boldsymbol{I}$, which is the second space. Third is the algorithm space, $\mathcal{A}$, which is composed of a portfolio of algorithms applied to the problems in $\boldsymbol{I}$. Fourth is the performance space, $\mathcal{Y}$, which is the set of metrics $y(\alpha, x)$, measuring the performance of an algorithm $\alpha \in \mathcal{A}$ to solve a problem $x \in \boldsymbol{I}$. Fifth is the *feature space*, $\mathcal{F}$, which contains multiple measures to distinguish similarities and differences between instances in $\boldsymbol{I}$, and that may correlate with difficulty for various algorithms. These features are represented by the vector $\boldsymbol{f}(x)$. The metadata, composed of the features and algorithm performance for all the instances in $\boldsymbol{I}$, is used to learn the mapping $g(\boldsymbol{f}(x), y(\alpha, x))$ and predict the performance of an algorithm, given a feature vector summary of an instance. This was the framework proposed by Rice in 1976 (Rice 1976) for the algorithm selection problem, where simple models such as regression were proposed to learn the mapping.

Finally, in the extended framework proposed by Smith-Miles et al. Smith-Miles et al. (2014) the instances are projected from the feature space to a lower-dimensional (2-d) instance space, a sixth space in the framework, so they can be visualized and algorithm performance inspected and objectively measured. The methods used to learn the performance mapping for algorithm selection, and to project from a high-dimensional feature space to a 2-d instance space are flexible, and a self-organizing map was used in Lopes and Smith-Miles (2013). In this paper, we adopt the most recent development of the instance space methodology from Muñoz et al. Muñoz et al. (2018) to obtain an optimal projection that encourages linear trends in both features and algorithm performance to be visualized across the resulting instance space to support interpretability and insights.

Instance space analysis thereby supports the study of instances described by their location in the instance space, according to their features, and the performance of algorithms in various parts of the instance space. In particular, we are able to construct *footprints* for each algorithm, defined as the region in instance space where we statistically infer good performance of the algorithm, for a user-defined criteria of good. This inference is applied to the entire problem space $\mathcal{P}$. Furthermore, instance space analysis allows us to:

1. visualize the distribution and diversity of existing random and real-world instances;
2. assess the adequacy of the features;
3. describe the unique strengths and weaknesses of algorithms;
4. identify and measure an algorithm's footprint to objectively compare algorithms;
5. partition the instance space into recommended regions for automated algorithm selection;
6. distinguish areas of the instance space where it may be useful to generate additional instances to gain further insights.

The unique advantage of visualizing algorithm performance in the instance space, rather than as a small set of summary statistics averaged across a large collection of instances, is the nuanced analysis that becomes possible when visually exploring interesting variations in performance that may be

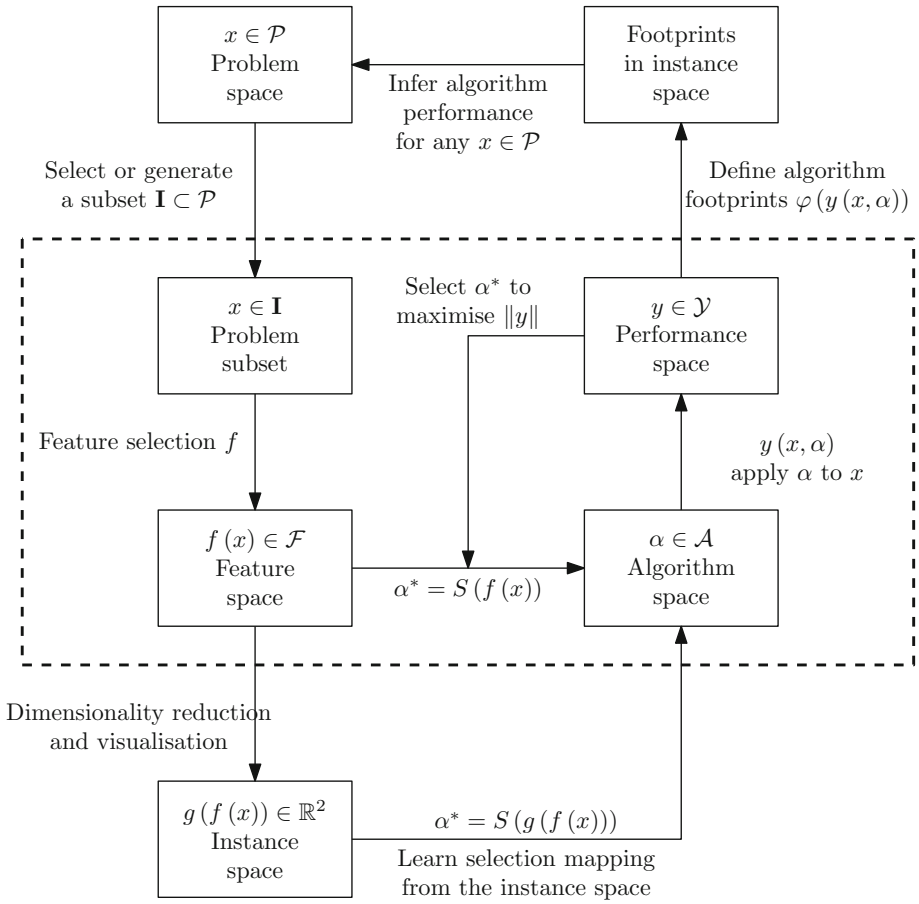


Fig. 1 Summary of the instance space methodology proposed by Smith-Miles et al. (2014), underpinned by the algorithm selection framework (in the dotted box) by Rice (1976)

hidden by tables of summary statistics, or too insignificant to attract attention from automated machine learning methods.

4 Methodology

In this section, we provide details of the meta-data we have generated for CB-CTT to enable the construction of the instance space.

4.1 Problem subset *I*

Since the ITC-2007(McCollum et al. 2010) was the first competition involving a set of real-world instances for CB-CTT, the set of 21 instances[2] from University of Udine became the benchmark set on which algorithms are compared (Achá and Nieuwenhuis 2014; Banbara et al. 2019; Bellio et al. 2016; Müller 2009). We used the competition instances and additionally used a different set of 54 real world instances, including instances from the University of Erlangen, the company EasyStaff, the University of Pristina, and some other Italian universities. Of these, the instances obtained from the company EasyStaff and the University of Pristina are used for the first time in an instance space analysis. The competition instances will be referred to as **ITC**, and the set of all real world instances as **Real**.

Since these 83 real-life instances is a rather limited amount of data for applying machine learning methods, in the first phase, we additionally generated 495 instances using an open source random generator[3]. There are two main versions of the random generator: Originally, it was developed by Burke et al. Burke et al. (2008). Later the random generator was revised by Lopes and Smith-Miles Lopes and Smith-Miles (2013, 2010) to produce instances more similar to **ITC**. In our paper, we used this latest version to generate this first batch of 495 random instances. We will refer to these instances as $\mathbf{Rand_0}$. The set $\mathbf{Real} \cup \mathbf{Rand_0}$ will also be called the initial set of instances.

Additionally, based on our initial instance space analysis, we created a new version of the random generator (see Sect.5) which we used to generate 3852 additional instances. Those will be referred to as $\mathbf{Rand_{PCA}}$. $\mathbf{Rand_0} \cup \mathbf{Rand_{PCA}}$ will be called **Rand**.

[2] These instances are actually 20, given that for our formulation two of them, namely `comp03` and `comp15`, are identical.

[3] https://code.google.com/archive/p/udinettgen/.

4.2 Algorithm space $\mathcal{A}$

Our objective for the selection of solvers is twofold. On the one hand, we want to use solvers that achieve the best results in the literature; on the other hand, we want to have a balanced mix between meta-heuristicsand exact methods. According to these aims, we consider the four solvers described in the following, whose source code was kindly supplied to us by their authors.

HYBR The first solver is the winner of ITC-2007, which is the hybrid solver developed by Tomáš Müller Müller (2009). This is a constraint-based local search method, in which hard constraints are never violated, but some lectures might remain unassigned. It uses a combination of several complex neighborhoods, each one targeting the improvement of one specific type of soft constraint. For guiding the search, it uses three different meta-heuristicsin turn, namely Hill Climbing, Simulated Annealing, and Great Deluge.

SA The second solver is another meta-heuristicsapproach based on local search, and it is proposed by Bellio *et al* Bellio et al. (2016). This is a simulated annealing procedure that solves hard and soft constraints together in one single stage. It uses the classical metropolis acceptance criterion and geometric cooling, plus a cutoff mechanism to reduce the computational cost on the early stages of the search. The neighborhood employed is a relatively small one, based on the union of two atomic ones: move a single lecture and swap two lectures.

SAT The third solver is an exact one, which is based on the encoding into MaxSAT and on the use of a general-purpose MaxSAT solver. In detail, it uses the ad hoc encoding proposed by Asín Achá and Nieuwenhuis Achá and Nieuwenhuis (2014) into a *Partial* MaxSAT formula, where Partial means that only some clauses can be violated (the soft one), whereas the others must be strictly satisfied. For the validation of the formulas, we use the solver developed by Berg *et al* Berg et al. (2019).

ASP The fourth and final solver is based on *answer set programming* (ASP) which is also an exact technique, based on Boolean constraint propagation and *nogoods* learning techniques. A sharp ASP model of CB-CTT has been developed by Banbara et al. Banbara et al. (2019), and we use this specific model. For solving the model, we run *clasp* (Gebser et al. 2012), a popular state-of-the-art ASP solver.

Table 1 Base features $\mathbf{X}$ of which we computed $\mathbf{stat}(\mathbf{X})$

	Base feature	Special values
Rooms	Capacity	None.
Courses	# lectures	None.
	Min working days constraint	1, # teaching days
	# students	0
	# unavailable periods	0

4.3 Feature space $\mathcal{F}$

The problem space of CB-CTT has already been studied in Lopes and Smith-Miles (2010, 2013); Smith-Miles and Lopes (2011). Furthermore there have been studies of the instance hardness for a related course timetabling problem in Chiarandini and Stützle (2003); Rossi-Doria et al. (2002); Kostuch and Socha (2004). In addition, feature-based parameter-tuning has been studied in Bellio et al. (2016). In this study, we use the CB-CTT features found to be most significant in other studies and adjusted some of the most important features from Chiarandini and Stützle (2003); Rossi-Doria et al. (2002); Kostuch and Socha (2004) so that they can be used for CB-CTT.

As explained in detail later, our full set of features is computed based on a graph-based representation of more basic features, extending the original formulation of the features in previous work. Therefore, we will point out in which way the most important features from other studies are included, and in which way they have been generalized in the course of this section.

Statistical Features We extracted different base features from each instance. For each base feature $\mathbf{X}$, we computed the set of features $\mathbf{stat}(\mathbf{X})$, which contains the count, sum, mean, variance, standard deviation, variation coefficient, entropy, quantiles, moments, central moments, skewness, and kurtosis. Additionally for some base features, we defined "special values". An example of a special value is the cluster coefficient of a graph being 1. This value is special since it represents the theoretical maximum. For each special value, we added both the number of observations of that value, as well as the number of special values divided by the number of all (special and non-special) values.
The base features we considered are listed in Table 1.
Graph Features In addition to statistics of plain numerical values, we generated several graphs G from the problem instances, and extracted several base features $\mathbf{X}(G)$, of which we computed the statistical features $\mathbf{stat}(\mathbf{X}(G))$ as well. We computed regular, and bipartite graphs, with both weighted, and unweighted edges. Table 2 lists all base features we used for each class of graphs. The local clustering coefficient for each vertex v was computed as the global clustering

Table 2 Base features $\mathbf{X}(G)$ on which the statistical graph features $\mathbf{stat}(\mathbf{X}(G))$ are based

Regular	Base feature	Special values
Unweighted	Sizes of connected comp.	1
	Degrees of vertices	0, 1, $\|V\|$, $\|V\|$ - 1
	Local clustering coefficient	0, 1
	Weighted local clust. coeff.	
Weighted	Edge weights	
	weighted degrees	Depend on specific graph
Bipartite	Base feature	Special values
Unweighted	Left degree	0, 1, $\|R\|$, $\|R\|$ - 1
	Right degree	0, 1, $\|L\|$, $\|L\|$ - 1
Weighted	Weighted left degree	
	Weighted right degree	
	Edge weights	Depend on spec. graph

Note that we do not consider bipartite graphs as a special case of a regular graph, but as a disjoint concept, while the weighted counterparts are considered as a specialization of the unweighted ones

coefficient $\frac{\#\text{ triangles}}{\#\text{possible triangles}}$ of the sub-graph induced by the neighborhood of v. The weighted local clustering coefficient is the local clustering coefficient multiplied by the vertex' degree.

In addition to the statistical measures, we added edge count, edge density $\frac{|E|}{\frac{1}{2}|V|(|V|-1)}$, and global clustering coefficient $\frac{\#\text{ triangles}}{\#\text{possible triangles}}$ as features for regular (weighted) graphs. We also used edge density $\frac{|E|}{|L|\cdot|R|}$ as a feature for bipartite graphs.

For each feature f_R on the right-hand side of a bipartite graph, there is a corresponding feature f_L on the left hand side. Therefore, we added for each such pair of features two new features $\frac{f_L}{f_R}$ and $\frac{f_R}{f_L}$. An example for this is

$$\frac{count(deg(Right(\textbf{BRoomCourse})))}{count(deg(Left(\textbf{BRoomCourse})))}$$

which can be interpreted as "number of courses per room". Note that these features do not only include fractions for simple measures like the number of nodes, but also for other statistical measures like the mean degrees on both sides of the graph.

A list of the graphs we built from the problem instances are listed in Table 3.

The graphs **GCourseTeacher**, **GCourseCurr**, and **GCourseConflict** are the same as used in Smith-Miles and Lopes (2011) and similar to those used in Kostuch and Socha (2004), but with **GCourseCurr** having weighted edges. We generalize the features describing graphs' degrees used in Smith-Miles and Lopes (2011) by not only using mean and standard deviation as statistical measures, but also many other statistical measures as described in Table 2.

In Bellio et al. (2016), it was observed that one of the most important features for parameter tuning is the average number of conflicts per course. This feature is included in our feature set as $mean(degree(\textbf{GCourseConflict})$. Note that due to our general method we obtain not only the average number, but many more statistical features describing the number of conflicts per course.

BRoomCourse is also inspired by a feature in Kostuch and Socha (2004), and Smith-Miles and Lopes (2011): The papers mention that **BRoomCourse**, the average number of room options per event, is a useful feature. A very similar feature is given by $mean(degrees(Right(\textbf{BRoomCourse})))$. It can be interpreted as the mean number of rooms on which a course cannot be scheduled. Therefore, it is an inverse of the prior feature. Nevertheless, this feature is less important in our case since room capacity violations are not hard constraints in our timetabling problem. A feature that is similar but probably more accurate in our case would be $mean(degrees(Left(\textbf{BCourseSlot})))$ which can be interpreted as the average number of scheduling options over all courses. In Smith-Miles and Lopes (2011) and Kostuch and Socha (2004) also the number of one-room events is used as a feature. This feature is also included in this work as $count(specialValue(|L| - 1, degrees(Right(\textbf{BRoomCourse}))))$.

Probing features In addition to the static features already described, we also investigate the usefulness of features describing dynamic aspects of the problem instance. Therefore we have computed so-called probing features by running **SA** 4 times, with timeouts of 0.5, 1, 1.5, and 2 seconds. From each run, we used the cost, number of violations, and the algorithm's internal objective function as a feature, which is a linear combination of the cost and the violations.

Since we used an ASP-encoding for one of the algorithms, we used the tool **`claspre2`** Hoos et al. (2014) to extract features from the ASP-program. The features generated by **`claspre2`** include static as well as probing features. A detailed description of those features can be found in Hoos et al. (2014).

As we will see in Sect. 6, the probing features are useful for algorithm selection, since they are predictive of algorithm performance. Nevertheless, they were not used for the instance space analysis, to get more insight into the importance of the static features of the instance, which are more interpretable than the probing features.

Timetabling features As pointed out in Smith-Miles and Lopes (2011); Kostuch and Socha (2004); Smith-Miles and Lopes (2012), the slackness of an instance is a valuable feature. Therefore we used three different slack features, which can be found in Table 4. As described before, many intuitive timetabling features are included implicitly as graph or statistical features. For example, one such feature is the number of

Table 3 Graphs considered for feature computation

(a) Regular graphs		
		GCourseTeacher
Nodes	V	Courses
Edges	E	vEw iff v and w are taught by the same teacher.
		GCourseCurr
Nodes	V	Courses
Edges	E	vEw iff v and w are both in a common curriculum.
Weights	w	$w(v, w)$ is the number of curricula v and w are both in.
		GCourseConflict
Nodes	V	Courses
Edges	E	cEc' iff c and c' are taught by the same teacher or have a curriculum in common.
		GEvent
Nodes	V	Courses $c \times lectures(c)$
Edges	E	$(c, l)E(c', l')$ iff c and c' are taught by the same teacher or have a curriculum in common.
		GCourseUnav
Nodes	V	Courses
Edges	E	vEw iff v and w share an unavailable period.
Weights	w	$w(v, w)$ is the number of unavailable periods v and w share.
		GCurrCourse
Nodes	V	Curricula
Edges	E	vEw iff v and w share at least one course.
Weights	w	$w(v, w)$ is the number of courses v and w share.
(b) Bipartite graphs		
		BCourseCurr
Nodes	$L + R$	Courses + Curricula
Edges	E	lEr iff l is a course of curriculum r.
		BCourseRoom
Nodes	$L + R$	Courses + Rooms
Edges	E	lEr iff r not a suitable room for l.
		BCurrTeacher
Nodes	$L + R$	Curricula + Teachers
Edges	E	lEr iff l contains a course with r as a teacher.
		BPeriodCourse
Nodes	$L + R$	Periods + Courses
Edges	E	lEr iff r is unavailable on period l.
		BDayCourse
Nodes	$L + R$	Days + Courses
Edges	E	lEr iff r is unavailable on more than the half of the time-slots of day l.
		BRoomCourse
Nodes	$L + R$	Rooms + Courses
Edges	E	lEr iff the number of students of r exceed the capacity of l.
Weights	w	$w(l, r)$ is the number of students the capacity of r is exceeded by.
Special weights		None.
		BCourseSlot
Nodes	$L + R$	Courses + (Periods×Rooms)
Edges	E	$lE(p, r)$ iff the course l is allowed to be scheduled on period p.
Weights	w	$w(l, (p, r))$ is the number of students of l - the capacity of r.
Special weights		the capacity is big enough ($w \geq 0$), the room is too small ($w < 0$)

Table 4 Features describing the slackness of an instance

Name	Definition
slack.seat	$\sum_{Rooms} capacity - \sum_{Courses} students$
slack.seat_period	$\lvert Periods\rvert * \sum_{Rooms} capacity - \sum_{Courses} students * lectures$
slack.events	$\lvert Periods\rvert * \lvert Rooms\rvert - \sum_{Courses} lectures$

courses that occur in only one curriculum, which is given by $count(specialValue(1, degrees(Left(\textbf{BCourseCurr}))))$.

Based on the above descriptions, it can be seen that our feature set is comprehensive and includes features proposed in earlier studies as well as many extended instance properties.

4.4 Performance space $\mathcal{Y}$

As described in Sect. 2, the problem yields a two-dimensional cost measure, consisting of the cost c and the distance to feasibility v (i.e., the number of hard constraint violations).

Regarding the running time, since most algorithms for CB-CTT are compared via their performance on the benchmark instances from ITC-2007 we decided on using the competition's setup for our experiments. This means that each algorithm was run on each instance with a timeout of $t_0 = 288$ seconds (this timeout was set according to the tool which was provided for ITC-2007), as its own single-threaded process. To make up for algorithms estimating their runtime inaccurately, we included a tolerance factor of 1.2 for the timeout, so that we fix $t_{max} = t_0 \times 1.2$.

Note however that, while the metaheuristic approaches fully utilize the time granted, the exact ones may reach the optimal solution in a shorter time. We want to "reward" the exact solver in case of this achievement. Everything else being equal, the faster algorithm will be preferred in practical applications. For this reason, we add to the above two measures, a third one, which is the runtime t.

To be able to perform algorithm selection and to visualize the instance space we decide to normalize the three-dimensional cost measure $perf(a, i) = \langle v, c, t\rangle$ $(a \in \mathcal{A}, i \in \boldsymbol{I})$, consisting of violations, cost, and time into a single real number between 0 and 1, where 1 is the best result and 0 the worst one.

In order to simplify the mapping we decide to give all solutions with violations, i.e., with $v > 0$, the value 0, independent of the actual distance to feasibility, so that we can ignore v. Similarly, in the cases in which one solver, due to some malfunctioning, runs for t longer than t_{max} (or loops indefinitely), then it is interrupted and given score 0.

Regarding the other two measures, we decide to give a higher priority to the cost c and use the time t only as a secondary criterion. To this aim, we define $costTime(c, t) = c + t/t_{max}$, in which c is the integer-valued CB-CTTcost and t/t_{max}, by construction, is always smaller that 1.

In summary, we use the following exponential function as the performance metric y, within the range [0, 1], which has a limit of 0 for an arbitrary large cost c, and is most sensitive at the best performances, helping us to distinguish the best performing algorithms.

$$\textbf{ExpGlobal}(v, c, t) = \begin{cases} 1.01^{-costTime(c,t)} & \text{if } v = 0 \\ & \text{and } t \le t_{max} \\ 0 & \text{otherwise} \end{cases}$$

The base 1.01 for the exponential function was chosen such that the score is scaled down by 1% if the cost increases by 1 unit.

Ranking the results by descending y scores gives the same ranking as the original competition ranking, except that we ignore the distance to feasibility, meaning that all infeasible solutions are ranked equally. This only has a minor effect, since in the large majority of the cases the solvers do find a feasible solution.

The absolute performance measure **ExpGlobal** is used (see Sect. 5) to separate instances which are easy for all algorithms from those hard for all algorithms, but for a finer-grained evaluation we use a second normalization, which is parametrized by the instance $i \in \mathcal{I}$ for which the performance is being measured. The value of that measure depends on the best result $minCT_i$ for instance i over all algorithms. It is defined such that the best algorithm for i will always get a score of 1, in case any algorithm $a \in \mathcal{A}$ finds a feasible solution within the time limit. Such a measure is needed, to not be biased toward low-cost instances when evaluating classification performance using the score measure in Sect. 6.

$$\textbf{ExpLocal}_i(v, c, t) = \begin{cases} 1.01^{-(costTime(c,t)-minCT_i)} & \text{if } v = 0 \\ & \text{and } t \le t_{max} \\ 0 & \text{otherwise} \end{cases}$$

Since some of the algorithms used in our experiments adopt some kind of randomization, we conducted five runs for each pair of algorithm and solver and used the mean of the performance measures (**ExpGlobal** and $\textbf{ExpLocal}_i$) in our analysis.

5 Instance space analysis

In this section, we will visualize the instance space, give some insights gained by the visualization, and discuss why and how we generated new instances. The instance space was analyzed with the tool MATILDA[4].

[4] https://matilda.unimelb.edu.au.

The resources of this paper, including all instances and the corresponding features, algorithm run results, the projection matrix, and the scripts for generating new random instances and for computing the features of instances, are publicly available and can be found here[5]. Additionally, the reader can analyze the instance space for this problem with the latest version of MATILDA directly online[6].

5.1 Feature selection

In total, we had 2124 features. As described in Sect. 4, for the instance space analysis we did not use the probing features, but only the 2044 static ones. Due to the fact that the graphs we used to compute features sometimes encoded the same characteristics of the instances in different ways, some features were duplicated. We removed those from our set of features. After the removal, 1857 distinct features remained. To decrease computation time for the 2D-projection, and—more importantly—to make it possible to interpret plots, we selected features by their correlation with the algorithms' scores. For each algorithm $a \in \mathcal{A}$, we selected the feature that is most correlated with the algorithms' score $\textbf{ExpGlobal}(perf(a,\cdot))$, and for each pair of algorithms a, a' we selected the feature that is most correlated with their difference in score $\textbf{ExpGlobal}(perf(a,\cdot)) - \textbf{ExpGlobal}(perf(a',\cdot))$. As we selected the most important feature for each algorithms' individual performance, and the most important feature for each paired algorithm difference in performance, in total 10 features were selected, of which 7 features were unique, as some features were the most important for predicting the performance, or performance difference, for several algorithms. The selected features, are shown in 5.

5.2 Feature preprocessing

As our goal was to visualize the instance space using a linear transformation, our features all needed to be finite numerical values. Since some of our features had non-finite values (i.e., NaN, $-\infty$, and ∞), we needed to apply a step of preprocessing. There were two reasons for infinite values: either a division by zero when comparing statistical values of the two partitions of a bipartite graph, or a timeout of some feature computation. The only feature computations that could time out are the ASP, and the simulated annealing probing features.

To represent all values in our plot, we firstly normalized the real values to be within [0, 1], and mapped non-finite numbers $NaN \mapsto -0.1$, $-\infty \mapsto -0.2$, and $\infty \mapsto 1.2$, so that they are not within the range of the valid feature values, but can still be visualized.

5.3 Projection equation

Table 5 shows the projection equation, the linear transformation used to compute an instance's position in the instance space based on its feature vector, determined using the methodology of Muñoz et al. (2018), as well as the minimum and maximum of each feature used for normalization. The projection has been computed using the full set of instances, the generation of which is described in Sect. 5.4. The features used in the projection are the ones selected in the manner described in Sect. 5.1. Their distribution in the instance space projection can be seen in Fig. 2. All instance space figures have axes x and y defined by the projection equation.

The selected features can be interpreted as follows: $moment(4, weights(\textbf{BCourseSlot}))$ is the fourth moment of weights of the graph **BCourseSlot**. Each of these weights describes how many seats would stay free if a specific course would be scheduled at a specific point of time in a specific room. This feature is therefore a similar feature to $slack.seat$, but statistically transformed to the fourth moment, and on a per-slot basis. Since the feature $slack.seat$ has been shown to be predictive for solvers' performance in Smith-Miles and Lopes (2011); Kostuch and Socha (2004), it is not surprising that this similar feature corresponds to the performance of solvers as well. The gradient arrow in Fig. 2 shows that, compared to the other features, it has a rather small impact on the final projection. $quantile(\frac{1}{4}, wghtDeg(Left(\textbf{BCourseSlot})))$ gives us more information about this localized slack feature: it describes the distribution of a kind of slackness in the number of seats over all courses. If we have a look the figure, we can see that this feature is distributed almost orthogonal to $slack.seat$. The feature $sum(minWorkingDays(courses))$ is rather easy to make sense of. This feature is high for problems where many lectures need to be spread out over all working days, which as we will see later highly correlates with an intuitive measure of problem size. $count(1 == deg(Left(\textbf{BCourseCurr})))$ measures how many courses there are that are only contained in one curriculum. We can see that this feature is somewhat correlated to instance size, but has been projected very differently. $count(deg(Right(\textbf{BCourseCurr})))$ is a rather intuitive feature, as it is the total number of curricula. $count(1 == localClusterCoeff(\textbf{GCourseCurr}))$ describes the number of courses c, where all courses conflicting with c, conflict with each other. This feature could be seen as the number of courses that cannot be scheduled easily.

[5] https://cdlab-artis.dbai.tuwien.ac.at/papers/cb-ctt/.

[6] https://matilda.unimelb.edu.au/matilda/problems/opt/tt#tt.

Table 5 This figure lists the minimum and maximum values (Fig. 5b) for each feature used for normalizing the feature values, and transforming infinite and NaN values, and the matrix (Fig. 5a) used to project the normalized features to the 2D plane

(a) Projection matrix

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -0.086370 & -0.210422 \\ -0.240798 & 0.247303 \\ 0.295782 & -0.360920 \\ 0.580145 & 0.244040 \\ -0.091966 & -0.485488 \\ 0.548373 & 0.236265 \\ -0.225460 & 0.611454 \end{bmatrix}^T \begin{bmatrix} \text{moment(4,weights(}\mathbf{BCourseSlot}\text{))} \\ \text{quantile(}\frac{1}{4}\text{,wghtDeg(Left(}\mathbf{BCourseSlot}\text{)))} \\ \text{sum(minWorkingDays(courses))} \\ \text{count(1 == deg(Left(}\mathbf{BCourseCurr}\text{)))} \\ \text{count(deg(Right(}\mathbf{BCourseCurr}\text{)))} \\ \text{count(1 == localClusterCoeff(}\mathbf{GCourseConflict}\text{))} \\ \text{slack.seat} \end{bmatrix}$$

(b) Minimum and maximum values

Feature	Min	Max
moment(4,weights(**BCourseSlot**))	4060.939	609175937945.081
quantile($\frac{1}{4}$,wghtDeg(Left(**BCourseSlot**)))	−5882625.000	250.000
sum(minWorkingDays(courses))	5.000	1723.000
count(1 == deg(Left(**BCourseCurr**)))	0.000	321.000
count(deg(Right(**BCourseCurr**)))	1.000	4036.000
count(1 == localClusterCoeff(**GCourseConflict**))	0.000	154.000
slack.seat	−594005.000	7030.000

5.4 Populating the instance space

Our goal was to gain an understanding of the instance space, and therefore we wanted to explore different regions of the space, in order to find out where the algorithms' performances differ. For this purpose, we need a random generator that can produce instances with strongly varying characteristics. The random generator from Lopes and Smith-Miles (2013) was tailored to produce instances that are similar to **ITC**, and offers only two degrees of freedom, which means that we needed another method for generating random instances for our purpose.

Therefore, we extended the random generator from Lopes and Smith-Miles (2013). It contained many hard-coded values sufficient to produce instances similar to **ITC**, which we turned into parameters. In total, our new random generator has 16 parameters:

- the total number of lectures, courses, working days, rooms, teachers, curricula, and unavailability constraints
- the number of periods per day
- the minimum and maximum of the lectures per course, courses per teacher, courses per curriculum, room size

In order to generate instances similar to our broader set of real-world ones, we configured the values of those parameters based on their distribution in the real-world instances. Many of these properties are highly correlated for real world instances (e.g., the number of courses and the number of rooms), which means sampling the distributions of those parameters independently would lead to very unrealistic and often trivially infeasible instances. Therefore we firstly performed a principal component analysis on the real world instances. Then, we estimated the distributions of the transformed parameters using kernel density estimation with Gaussian kernels. We sampled the resulting distributions and finally performed the inverse of the principal component analysis to get parameters in the original parameter space. The procedure is summarized in Fig. 3.

Since not all combinations of parameters always yield feasible instances, we filtered out instances which could be determined to be infeasible by some simple checks. We therefore removed instances where

- the total number of lectures (over all courses) is greater than the total number of scheduling options ($Rooms \times Periods$),
- there is a course c where the number of allowed periods for c is less than the number of lectures of c

In summary, our method of generation performs the following task: it gets a set of instances as input, samples their distribution of parameters using the method illustrated in Fig. 3, and uses these sampled parameters to generate new instances using the random generator. In essence, we generate a set of random instances with similar characteristics to another set of instances.

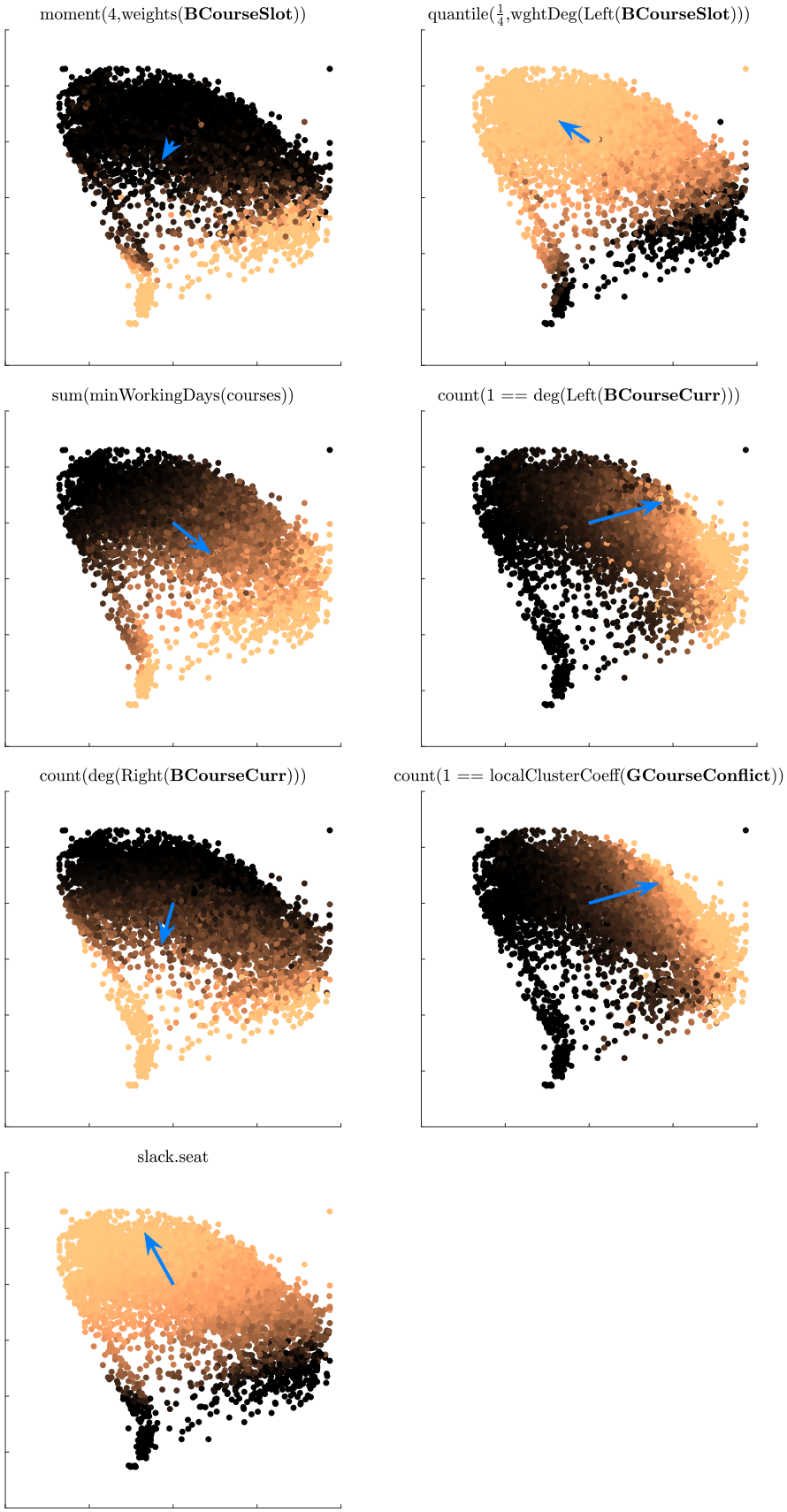


Fig. 2 The distribution of all features that were pre-selected for the visualization of the instance space. The arrow in each of the subplots points in the direction where high values of the feature are shifted. The length of the arrow corresponds to how strong they are shifted. The colors represent the normalized feature values, where 0 is mapped to black, and 1 to cream. Axes are defined by the projection equation in Table 5a

Input
a set of problem instances
Output
a vector of parameters, sampled from the estimated distribution of the parameters in the input set of instances

procedure GENERATESIMILARPARAMS(instances)
par = extractParams(instances) ▷ analyzes the values of the input set
pc = PCA(p) ▷ compute principal components
par' = tansform(pc, par) ▷ transform parameters using principal components
dstr = estimateDistr(par') ▷ estimates the distribution of transformed parameters
smpl = sample(dstr) ▷ samples estimated distribution
return transform^{-1}(pc, smpl) ▷ transforms sample back

Fig. 3 PCA parameter sampling

Our initial projection of the instance space contained holes, and sparse regions. Therefore, we generated new instances, based on those close to the holes, and sparse regions of the instance space, visualized the instance space again and repeated this process until we had an instance space with a rather even distribution of instances.

The projection of the final set of instances $\mathbf{Rand_{PCA}} \cup \mathbf{Rand_0} \cup \mathbf{Real}$, and the location of those groups of instances within the space is given in Fig. 5a. Furthermore, we visualized these groups of instances in Fig. 4, by creating a scatterplot of some of the intuitively most important features. From this figure, we can see that we cover the set of real world instances more accurately than the old random generator. Due to the previously mentioned hard-coded values in this random generator, the instances of the group $\mathbf{Rand_0}$ are hardly scattered around in the plots, but rather just occupy small regions, that are close to most of **ITC**, but far off the majority of the region occupied by **Real**. Our newly generated instances $\mathbf{Rand_{PCA}}$, cover the space occupied by **Real** better, and many of them show similar characteristics to the real world instances, with respect to the analyzed features.

5.5 Discussion

Figure 6 shows the best solution of any algorithm per instance in the instance space projection. Instances where no feasible solution was found are shown in gray. The color encodes the cost of the solution, if a feasible one was found. The cost is mapped to the color by the function **ExpGlobal**.

Comparing the feature distributions (Fig. 2) to the best performance among all algorithms (Fig. 5b), we can see that the number of curricula (count of the number of degrees of the right hand side of the graph **BCourseCurr**) is a good measure of hardness of an instance. Investigating this further showed that this is not a measure of instance size. An intuitive measure of size would be the number of events (sum of the lectures over all courses) that need to be scheduled. As is shown in Fig. 7, the size of the instance does not follow the same gradient as the number of curricula feature. If we compare this intuitive measure problem size to the selected features, we see that the feature $sum(minWorkingDays(courses))$, is distributed rather similarly, hence instance size is still an important feature for comparing different solvers. Interestingly, a large number of difficult instances are to be found in the upper left area of the instance space, where the instance size is small. As this is also the area where the number of curricula is high, and hence the complexity of the instance increases, this points to two different factors being relevant to the hardness of an instance. On the one hand, the size of an instance increases the cost, and lowers the likelihood of finding a feasible solution within the allotted time; on the other hand, a large number of curricula increases the likelihood of violating soft constraints, and of not finding a feasible solution.

If we have a look at which groups of instances occupy which parts of the instance space (Fig. 5a), we see that $\mathbf{Rand_0}$ are mostly located in the region where we have small instances, with high slack. Comparing Fig. 5a to the performance of the individual algorithms (Fig. 6) shows that this is also the region that is easy to solve well for all algorithms. Visualizing ties between algorithms, without taking runtime into account (Fig. 8a), we see that they cover almost the same region as $\mathbf{Rand_0}$. As shown in Fig. 8b this is also the region where mainly the exact methods perform best. The reason for that is that all solvers can find optimal solutions, but the exact methods can prove optimality and terminate earlier than the incomplete solvers.

Figure 5a also reveals that the parts of the instance space occupied by $\mathbf{Rand_0}$ are quite different from those occupied by **Real**. The regions partly overlap but most of the instances from **Real** are not in the range of the randomly generated instances.

At first sight this seems to contradict the results found in Smith-Miles and Lopes (2011); Lopes and Smith-Miles

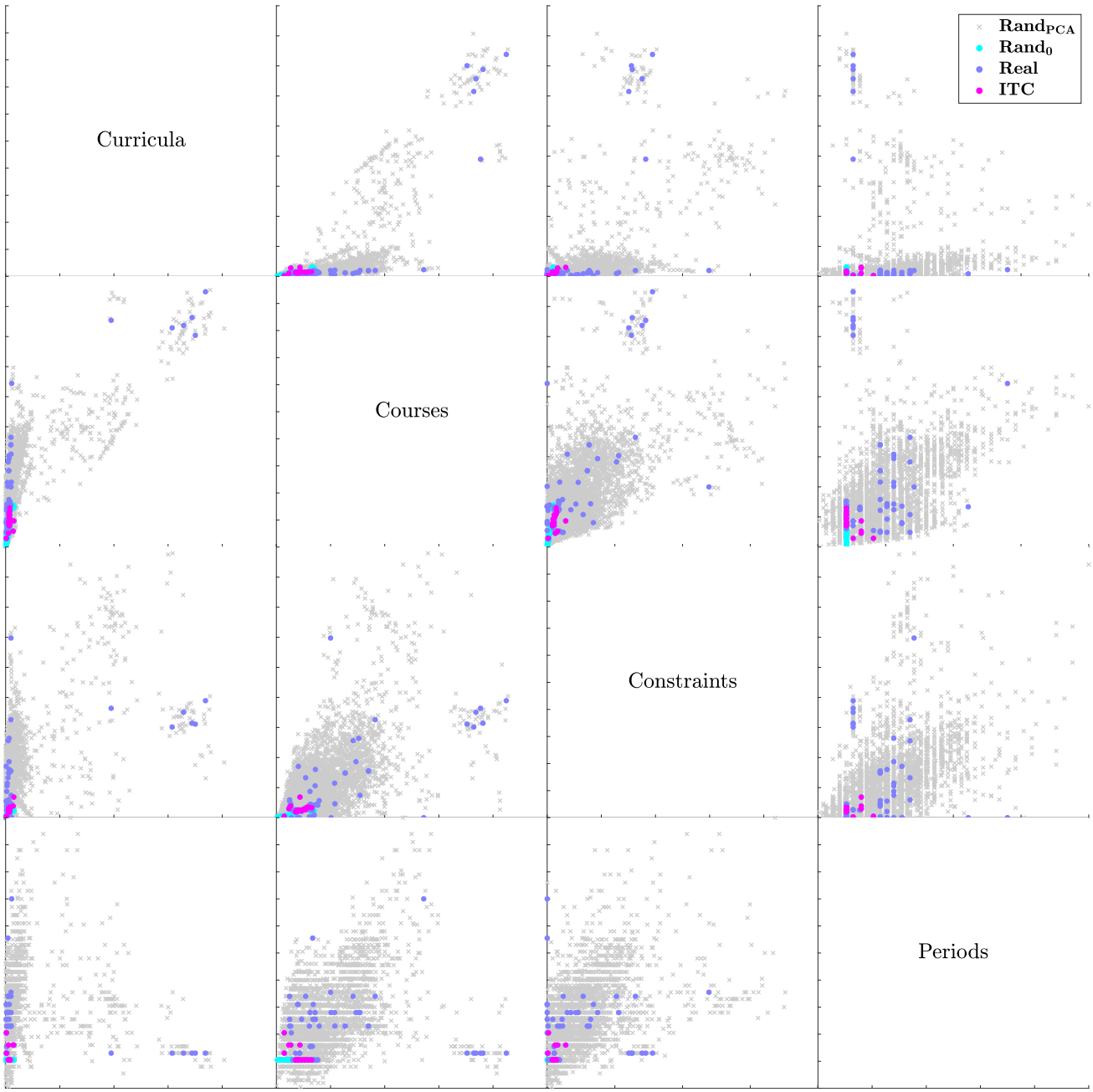
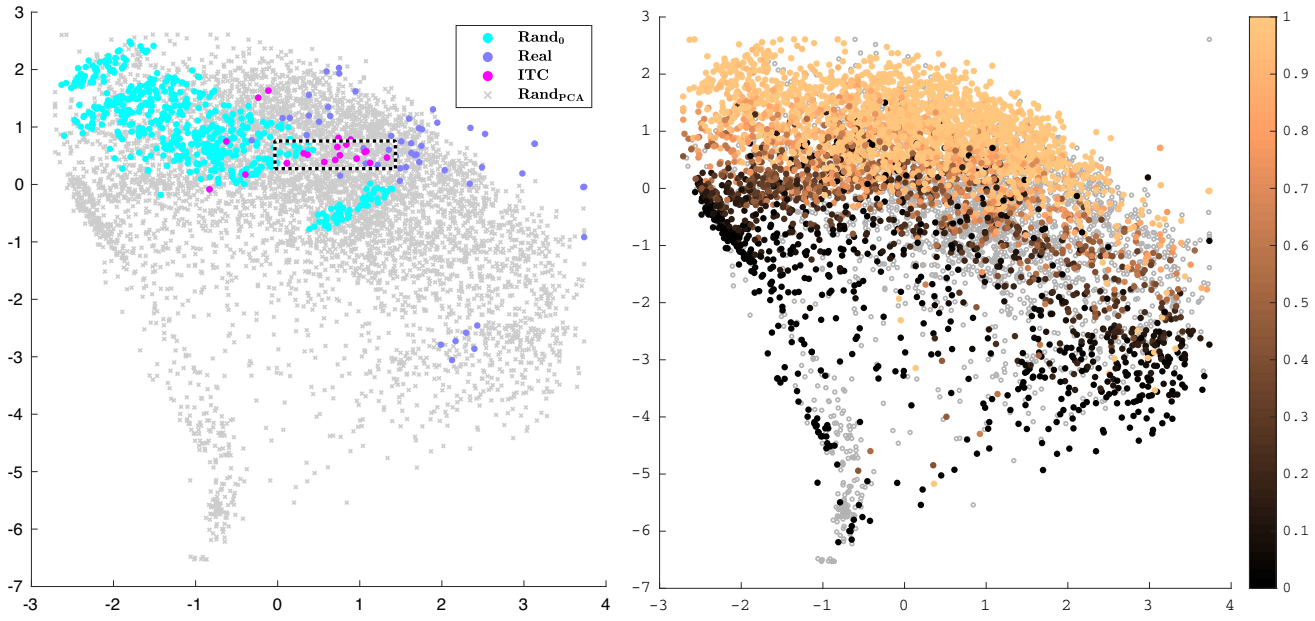


Fig. 4 Scatter plot of intuitive features for different the groups of instances

(2013), namely that the random generator proposed in Lopes and Smith-Miles (2013) yields instances that are similar to real world instances. However, this is due to additional real-life instances being used in our work rather than only **ITC** as in Lopes and Smith-Miles (2013). Moreover, the visualization in this paper does not attempt to cluster instances separating synthetic from real instances as it was done in Smith-Miles and Lopes (2011). In our case, we aim to separate easy from hard instances, and to separate instances that yield different performance among solvers.

We see that almost all instances from **Real** are in regions covered by **Rand$_{PCA}$**, and occupy large parts of the instance space, while **ITC** on which most solvers are compared, only occupy a very small region. We also see in Fig. 8b that the region of **ITC** is the region where there is no clear winner among the algorithms. This could be the case since the algorithms are being compared on those instances and therefore fitted to work best in this region of the instance space. On the other hand, outside of the region of **ITC**, the winning algorithms are easier to separate.



(a) Groups of instances (b) Best found solution

Fig. 5 Projections of all instances, using **ExpGlobal** as a quality measure. In **a** the location of different instance groups is visualized. The rectangle surrounds most of **ITC**. The same region is marked on Fig. 8b in order to be able to compare this regions easily. In **b**, the best solution over all solvers is visualized. Instances where no feasible solution was found are drawn in gray. Axes are defined by the projection equation in Table 5a

In Fig. 5a, we can also see that there is a small cluster of real world instances, in the lower-right region, that are clearly separated from the rest of the instances. This cluster is formed by instances from the University of Erlangen. Their features differ strongly from those of the other real world instances, and most of the solvers could not find feasible solutions for these instances.

This cluster of instances reveals an important fact about the CB-CTTproblem. Namely that we cannot make strong assumptions about what real world instances for this problem will look like, since the Erlangen instances have very different characteristics compared to most other real-world instances, but are also real-world instances. Hence, we cannot tell whether the instances $\mathbf{Rand_{PCA}}$, are realistic. Nevertheless, our results show that $\mathbf{Rand_{PCA}}$covers instances that are similar to real world instances with respect to those features that turned out to be most predictive for the solvers' performances.

Having a closer look at Figs. 6 and 8b reveals that both meta-heuristicsachieve similar performance for many instances. The difference between their performances can be observed in the high-cost regions of the feature space. **SA** is better in the region that is closer to easier real-world instances, while **HYBR** is better for the instances that are not as similar to the real-world ones. We also observe a quite large (but sparse) cluster in the high cost region in the lower-right corner, where **SA** outperforms **HYBR**.

6 Algorithm selection

For the algorithm selection problem, the same four algorithms as discussed in Sect. 4.2, that is, **HYBR**, **SA**, **ASP** and **SAT**, were used.

The instances considered were the same as in Sect. 4.1. For algorithm selection, the full feature set was used, including the probing features, as described in Sect. 4.3. In order to evaluate the performance of the classifiers, the usual split in training, validation, and test set data was used, where the training and validation set consist only of the generated instances, from now on referred to as **Rand**, in a random 80/20 split, and the test set data consists solely of **Real**. In this way, we can evaluate how well algorithm selection generalizes to real-life instances, based on synthetic training data only. We note that the advantage of the previously used instance space analysis to visualize the synthetic and real-life instances is that it enabled us to ascertain that the synthetic instances are sufficiently similar to the real-life instances. As mentioned in Sect. 5, several iterations of instance generation were performed in order to cover the real-life instances, with the initial results of applying algorithm selection train-

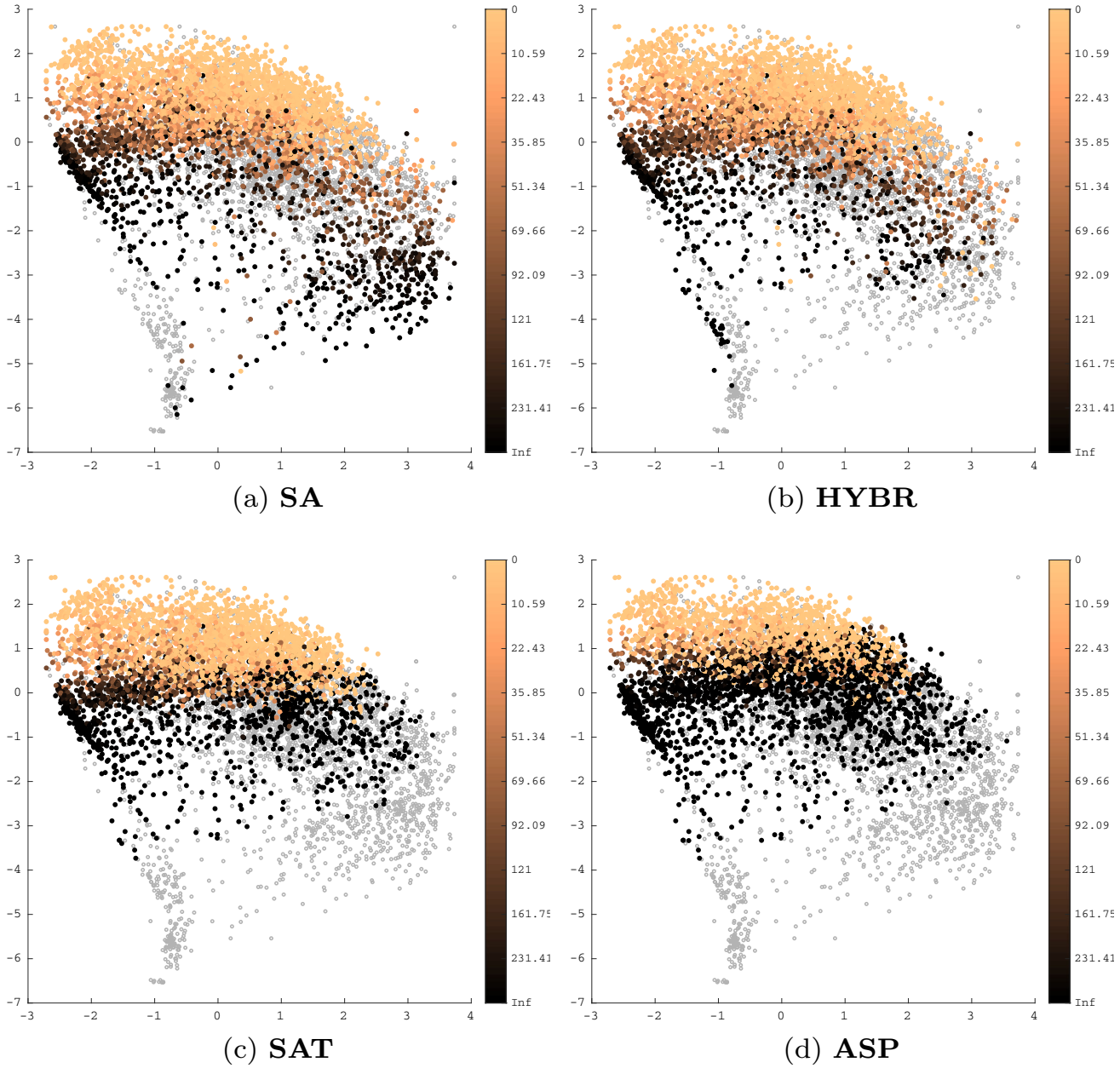


Fig. 6 Algorithms' performances. Color encodes cost using **ExpGlobal**. Grey dots represent instances where no feasible solution was found. Axes are defined by the projection equation in Table 5a

ing only on the $\mathbf{Rand_0}$ being disappointing. Having reached a sufficient coverage of the instance space, we proceeded with confidence that applying machine learning to the algorithm selection problem could be successful. Finally, good results in algorithm selection will confirm the quality of the synthetic instances, affirming the conclusions of the instance space analysis.

6.1 Performance evaluation

In the classification setting, the goal is to predict the best-performing algorithm(s) from the algorithm-portfolio for a given instance. Two different performance measures were considered, where a performance measure is given by a function $f : \mathcal{I} \times \mathcal{A} \rightarrow [0, 1]$:

1. **Cost-only**: For the cost-only performance measure, the runtime (up until timeout) of the algorithm is ignored.

The performance is then computed using the **ExpLocal**$_i$ measure, applied to the cost, not including runtime.

2. **Cost-time**: For the cost-time performance measure, the **ExpLocal**$_i$ measure was used as originally defined in Sect. 5.

Especially when the cost-only measure is used, but to a lesser degree also using the cost-time measure, several algorithms can be best-performing for a given instance, as can be seen in Fig. 8. This implies that we have several positive labels for a given instance. Contrary to the usual setting of multi-label classification, we are not interested in predicting every best-performing algorithm, but are satisfied with predicting only one of this set. This is because the final aim of algorithm selection is to get the best performance, and if several algorithms perform equally, the choice between them is immaterial. Hence, two evaluation measures were considered:

1. **Adapted accuracy**: Since the usual accuracy measure for classification problems requires only one positive label for each instance, we used an adapted accuracy measure: $acc = \frac{|\{i \in I | c(i) \in B_i\}|}{|I|}$, where i is a given instance, B_i is the set of best-performing algorithms for the instance i, and $c(i)$ is the prediction for the instance i.
2. **Score**: Although we are primarily interested in accurate classification of the best-performing algorithm, there is still value in a classifier which selects near-optimal algorithms over sub-optimal algorithms. To tease out this aspect of the classifier, we define the score evaluation measure as: $\frac{\sum_{i \in \mathcal{I}} f(i, c(i))}{|I|}$, where f is the performance measure under consideration, i.e. either the aforementioned cost-only or cost-time performance measure.

6.2 Classification algorithms

Four different popular classification algorithms were applied in the algorithm selection problem. They are k-nearest neighbors classification (KNN), random forest classification (RF), gradient-boosted trees (GB), and support vector machines (SVM). For all these algorithms the scikit-learn implementation was used (Pedregosa et al. 2011).

6.3 Parameter configuration and feature selection

In order to select the optimal parameters for each classification algorithm, fivefold cross-validation was performed on the training data. The optimal parameters for each algorithm are summarized in Table 6.

Since often classification algorithms generalize better to unseen data by restricting the number of features, recursive feature selection was performed using a measure of feature importance. Concretely, the recursive feature selection as implemented in scikit-learn was used for the random forest and gradient-boosted tree classifiers. As KNN and SVM using a radial basis function kernel do not expose feature importance, this type of feature selection could not be performed for these algorithms. The results of the feature selection are shown in Fig. 9a, b. As we can see in Fig. 9a, reducing the number of features improves the performance of the classifiers when the cost-only performance measure is used, but with different points at which the performance starts to suffer. In case of the cost-time measure, we can see in Fig. 9b that the performance remains relatively stable with number of features for the RF classifier, and decreases slightly for the GB classifier. In order to keep the computation costs of the remainder of the experiments reasonable, and improve the generalization performance, a final selection of 256 features was made, based on the best run for the cost-time performance measure. These same 256 features were used for the four algorithm selection classifiers.

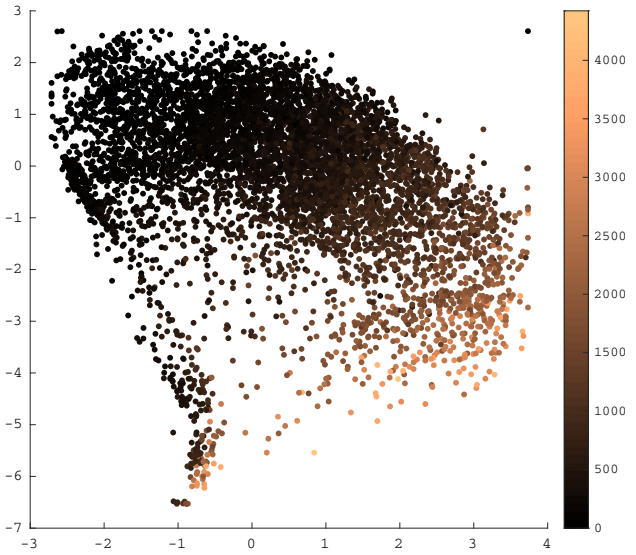


Fig. 7 The distribution of problem size (as defined by the number of lectures that need to be scheduled) in the instance space. Axes are defined by the projection equation in Table 5a

Aside from studying the effect of restricting the number of features, we also looked at the relation between number of training instances and classifier accuracy. Using the same 20% validation set, we performed 10 runs of algorithm selection for differently sized groups of randomly selected training instances. The results are shown in Fig. 10a, b. As one would expect, the accuracy increases with increasing number of training instances for each classifier. Interestingly, the increases with larger number of instances level off, suggesting that more training instances would not have improved accuracy further.

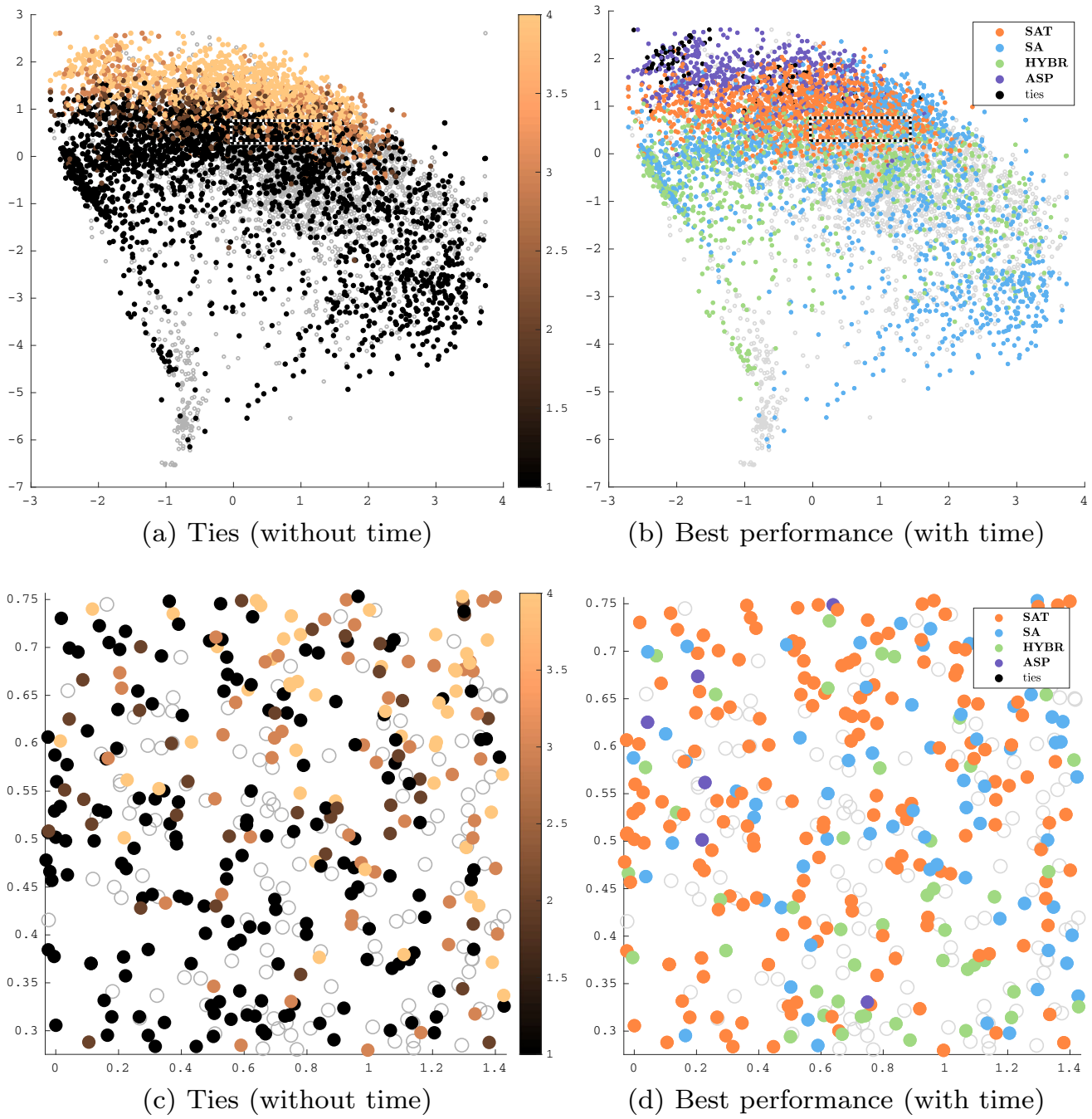


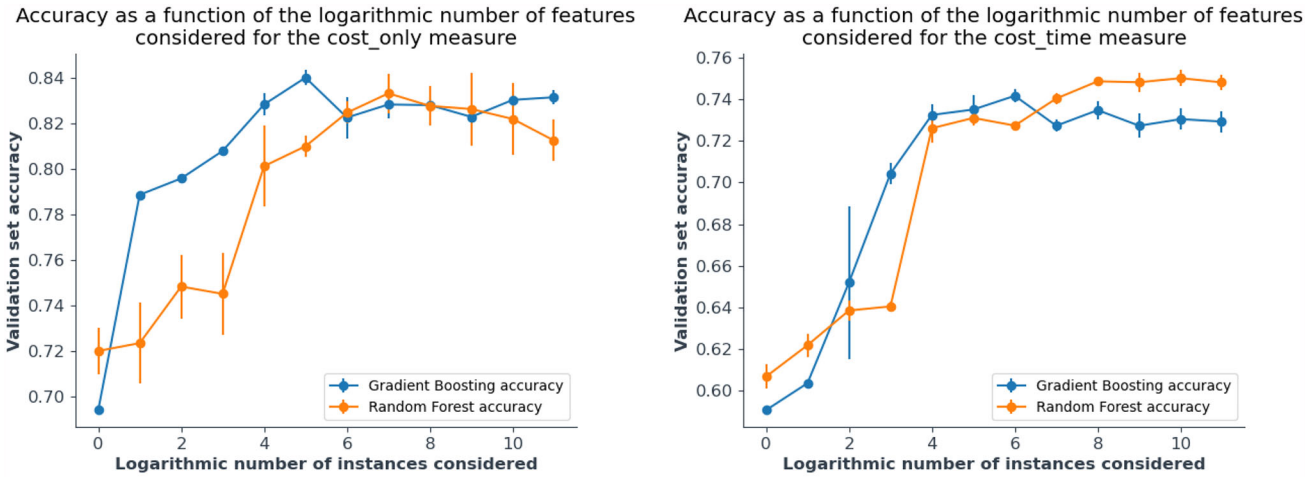
Fig. 8 **a** and **c** show instances where algorithms performed equally well-concerning cost. The colors encode how many algorithms found the lowest cost solution. **b** and **d** visualize the winning algorithm for each instance, with time taken into account. In all subfigures instances where no feasible solution was found are displayed in gray. The rectangle in **a** and **b** marks the same region as in Fig. 5a, in order to be able to locate the region of **ITC** easily. **c** and **d** show a zoom of this rectangle. Axes are defined by the projection equation in Table 5a

6.4 Experimental results

The performance of the 4 different classifiers was gauged by training each of them on a randomly selected training data-set, consisting of 80% of **Rand**, and evaluating the performance on the validation data set consisting of 20% of **Rand** and test-set consisting of **Real**. This process was repeated 20 times, with different random seeds, in order to estimate the spread in performance. In case several algorithms performed equally well on a given instance, the same instance was placed multiple times in the training data, each time with a different winner as label. The results for the

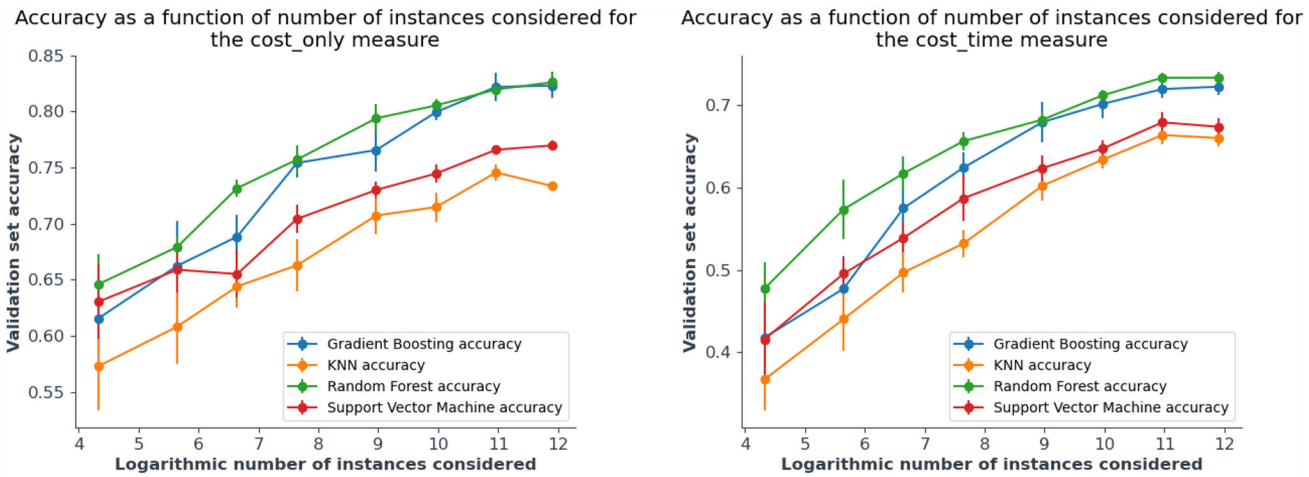
Table 6 Parameter choice for the different classification algorithms

Classifier	Parameter choice	Parameter choice	Parameter choice
splk-NN	n neighbors = 10	Algorithm = 'auto'	
SVM	kernel = 'rbf'	C = 100	$\gamma = 1 \times 10^{-5}$
RF	n estimators = 800	Max depth = 8	
GB	n estimators = 100	Max depth = 4	



(a) Cost-only measure (b) Cost-time measure

Fig. 9 Feature selection for the two different performance measures. The number of features on the x-axis shown in base-2 logarithm



(a) Cost-only measure (b) Cost-time measure

Fig. 10 Dependency of algorithm selection accuracy on number of training instances. The number of training instances on the x-axis are shown in base-2 logarithm

classifiers and individual algorithms, separated by performance measure and instance set of evaluation, are shown in Fig. 11. On the left-hand side of the plots, the performance of always predicting the individual algorithm is shown, whereas on the right-hand side the performance of the classifiers for algorithm selection is shown. Notice that the results on the left-hand side of the plot correspond to the results of the individual algorithms, i.e., no algorithm selection is involved. For example, considering the cost-only measure and the instances in **Real** (test-set) in Fig. 11b, the **HYBR** algorithm performs best, reaching the lowest cost on 68% of the instances.

Several interesting observations can be made. First of all, the performance of the different classifiers is higher on the validation set than on the test set. This should not be surprising given the results of the instance space analysis, especially apparent in Fig. 8b, whereas the best-performing algorithms are relatively well-separated in instance space, many instances in **Real**, **ITC** in particular, are located mostly in the boundary regions where no single algorithm outperforms all others. Hence, predicting for the validation set, which is selected from all over the instance space, is easier than predicting for **Real**.

Secondly, although in general algorithm selection performs better when considering the adapted accuracy evaluation metric, it only outperforms the best-performing algorithm for the score metric on the validation set. For the test set, algorithm selection does not perform better on the score metric than the **HYBR** and **SA** baseline. Since the exact solvers perform much worse on the score measure for the harder instances, and the classifiers are optimized to improve accuracy rather than score, this is not unexpected. After all, the classification accuracy can only be improved by correctly predicting that an exact solver will perform best, but a single misclassification will have a large impact on the average score achieved. In practice, this means that the **HYBR** and **SA** algorithms will on average perform better than algorithm selection, whereas algorithm selection will more often produce the best result for a given instance. One could address this issue by optimizing algorithm selection for the average score, rather than the accuracy measure considered, which is interesting for future work.

Thirdly, the classification accuracy is higher when the cost-only performance measure is used, both for the algorithms themselves, and the classifiers. This is readily explained by the fact that many more ties result using the cost-only measure, and hence, it is easier to predict at least one best-performing algorithm correctly. On the other hand, although the classification accuracy is lower when using the cost-time performance measure, the relative improvement in classification accuracy compared to the baseline of predicting the best individual algorithm is much higher. This is due to the fact that the classifier can exploit the ability of the exact solvers to prove the optimality of their solution, thus finishing before the time-out, in its predictions.

Finally, we can see that algorithm selection performs very well when the cost-time performance measure is considered, with the RF-classifier performing 19% better in accuracy than the **SAT** baseline on the test set, but this advantage is reduced to only 3% for the RF classifier compared to the **HYBR** baseline when the cost-measure is considered. This is at least in part due to the fact that when considering only the cost, the meta-heuristic algorithms perform extremely well, and **HYBR** and **SA** are hard to distinguish in the relevant area of the instance-space for the different real-life instances.

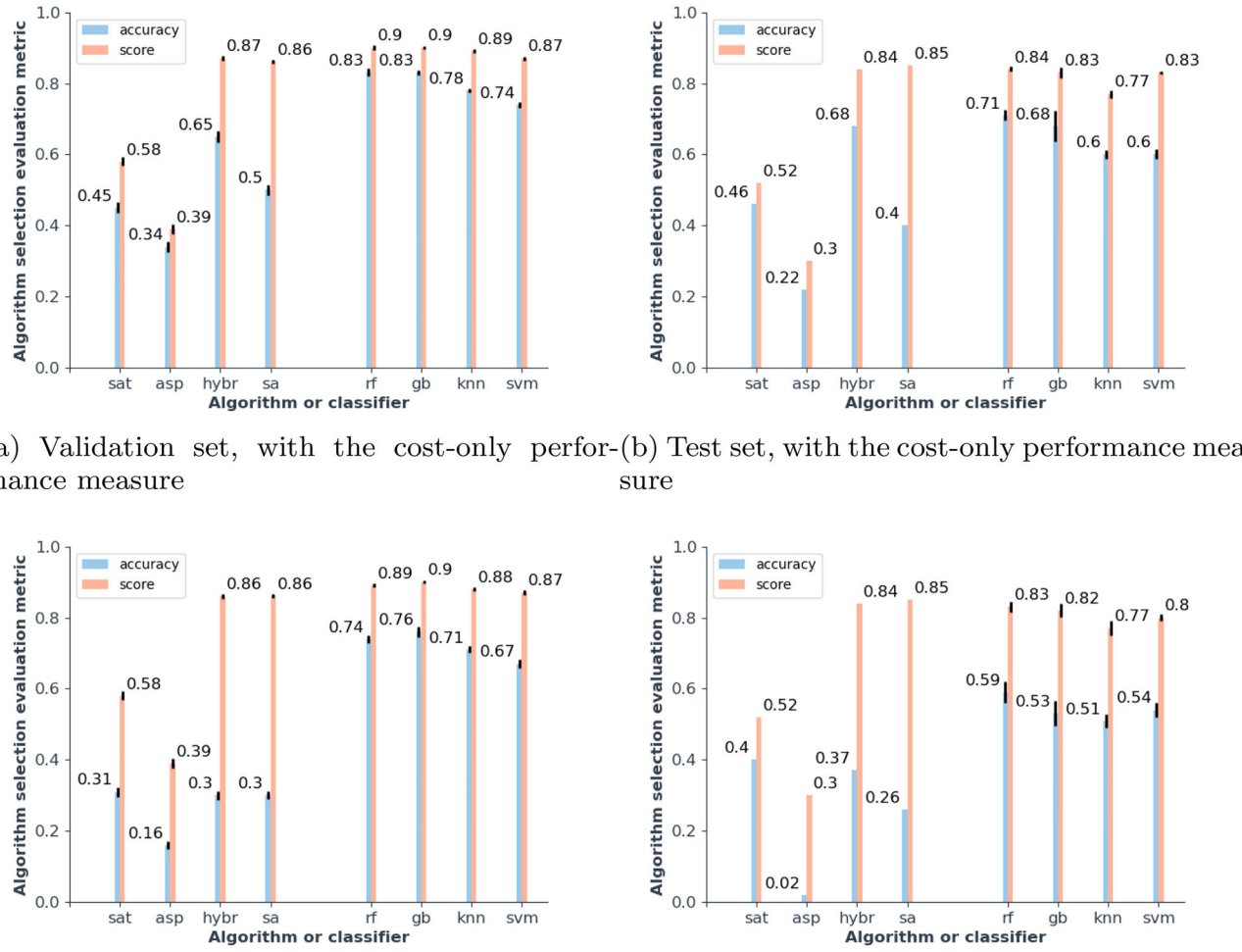
In general, we can conclude that algorithm selection is useful in cases where both the runtime and the cost is taken into consideration. As this is the most common and relevant scenario in practice, we believe that algorithm selection offers substantial benefits to the practice.

In order to gain a deeper understanding of the performance of algorithm selection, the confusion-matrices for RF, separated again by performance metric and instance set, are shown in Fig. 12. The confusion matrices show the predictions of the RF classifier on the horizontal axis, with the actual best performing algorithms on the vertical axis. Since several algorithms can tie in performance for a given instance, the vertical axis contains all subsets of the four algorithms considered. It can be seen in Fig. 12a, b that indeed the likelihood of a tie is much larger when using the cost-only performance metric, confirming the results shown in Fig. 8. It is also apparent that the RF-classifier struggles the most with separating the two meta-heuristic algorithms, since in all four confusion matrices often **HYBR** will be predicted when **SA** is the winner and vice-versa. The exact and meta-heuristic algorithms on the other hand are easier to separate, as **SAT** and **ASP** are rarely predicted when a meta-heuristic is the winner. Given the instance space results shown in Fig. 13, this should not be surprising. Indeed, **SA** and **HYBR** overlap much more in the instance space than the exact algorithms and meta-heuristics do.

Aside from the feature selection already in place, studying the impact of one particular class of features is especially interesting. The dynamically calculated probing features, both **SA** and **ASP** based (see Sect. 5), differ from the static features in that they are not in themselves descriptive of the instance, but rather descriptive of algorithm performance on the instance. As can be seen in Fig. 12b, this is not a small difference. The classification performance on the adapted accuracy metric is reduced by 3% for the best-performing GB-classifier on the validation set, and drops by 4% on the test-set if probing features are excluded. Interestingly, the SVM-classifier seems more robust in the sense that it does not worsen with exclusion of the probing features, accounting for its relatively good performance on the test set when probing features are excluded. The previous analysis underlines the effectiveness of using probing features, although a trade-off must be made in practice to take on the additional runtime and implementation complexity of the probing feature generation.

7 Conclusion

In this paper, we have both applied an instance space analysis to the CB-CTT problem, and shown the viability of automated algorithm selection. For the instance space analysis, we characterized the instances with an extensive set



(a) Validation set, with the cost-only performance measure

(b) Test set, with the cost-only performance measure

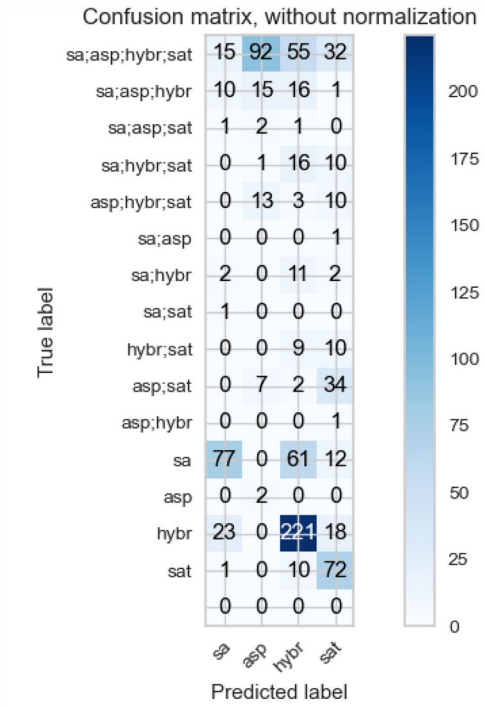
(c) Validation set, with the cost-time performance measure

(d) Test set, with the cost-time performance measure

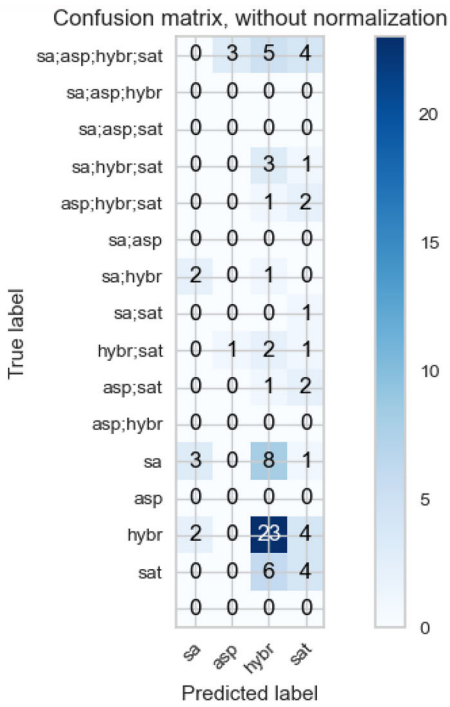
Fig. 11 Evaluation measure results for the different algorithms and classifiers on different instance sets. The algorithm results are shown on the left, the classifier results are shown on the right

of features. Based on our investigation, we concluded that the competition instances and the original random generator for CB-CTT do not sufficiently cover the instance space. In particular, the instances generated with the original random generator are generally easier than the competition instances and many of the other real-life instances. Moreover, the competition instances are shown to occupy a small region in the instance space, compared to the other real-life instances. Based on these results, we improved the random generator to generate instances which cover a larger part of the instance space, helping us distinguish the different areas where each algorithm performs well. Our results show that the exact methods, **SAT** and **ASP**, tend to perform better for easier instances, if the run-time is taken into account, since the solution optimality can be proved. On the other hand, for the harder instances, the meta-heuristics dominate. However, **SA** and **HYBR** are not uniformly better for the harder instances, but perform best in different regions of the instance space.

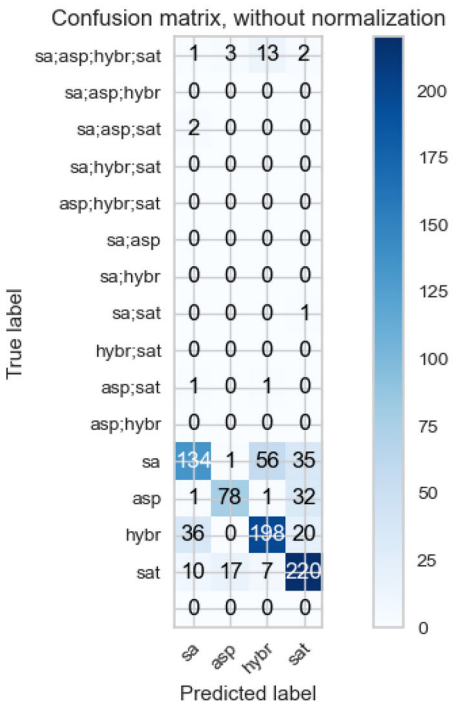
Building on the results of the instance space analysis, we explored automated algorithm selection for the CB-CTT problem. Four different classifiers were explored, of which the RF-classifier performed best, significantly outperforming any of the individual algorithms with regards to accuracy, when both cost and run-time are taken into account. In the case where only the cost is used as the performance measure, the random forest classifier performs best, but the relative advantage to the individual algorithms is smaller. These results confirm the conclusions of the instance space analysis, which show that while **SAT** and **ASP** perform better in the easy regions of the instance space when run-time is included in the performance measure, this advantage evaporates when only the cost is considered. Hence, a meaningful distinction between algorithms cannot be exploited by the classifier in



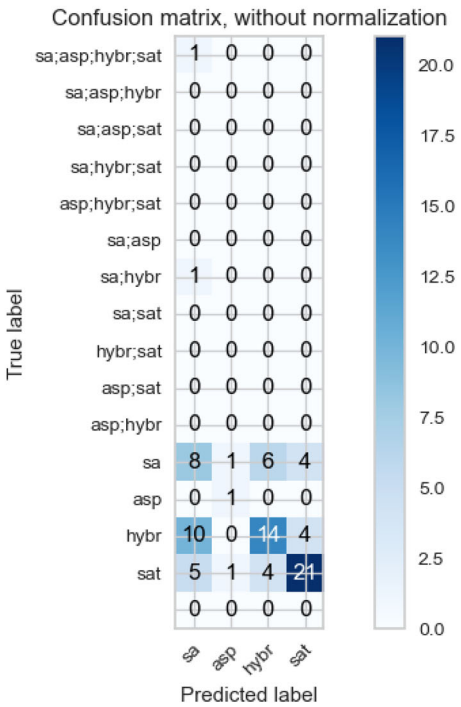
(a) Validation set, with the cost-only performance measure



(b) Test set, with the cost-only performance measure



(c) Validation set, with the cost-time performance measure



(d) Test set, with the cost-time performance measure

Fig. 12 Confusion matrix for the random forest classifier on different instance sets, with the different performance measures. Note that the true labels are in this case sets, but only one prediction is possible. A good prediction is a prediction that is an element of the true labels set

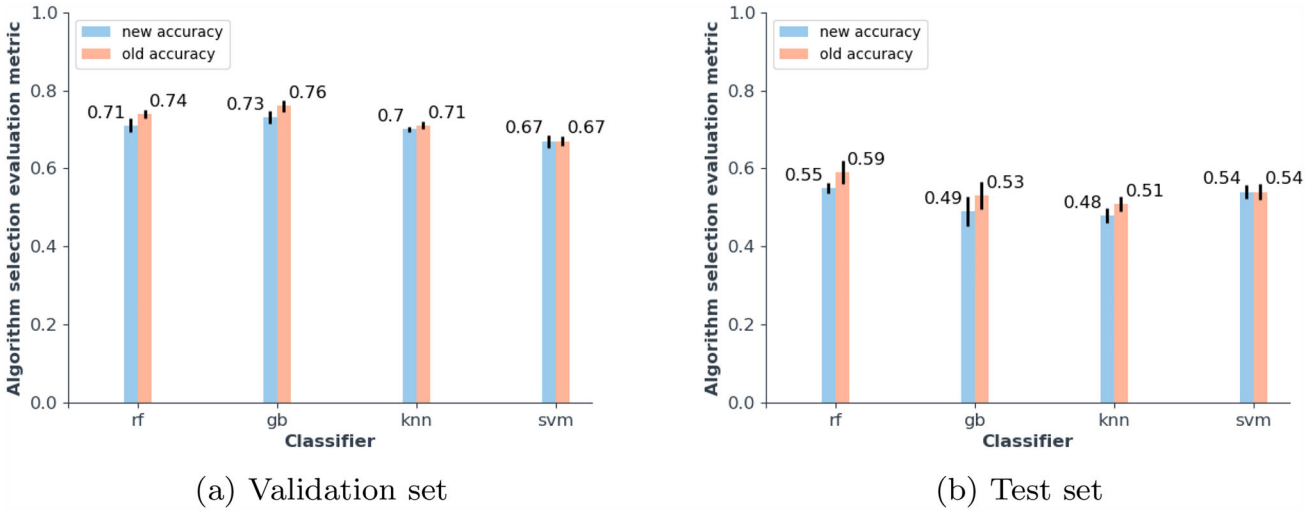


Fig. 13 Algorithm selection using the cost-time performance measure without probing features

this case. However, this scenario is less relevant, and hence we believe that algorithm selection is useful in practice in the case that both runtime and cost are considered. Finally, we investigated the effect of including the dynamically generated probing features and observed that they do contribute significantly to the classification performance, especially for the real-life instances.

One overarching aim of this paper is to show that algorithm selection and an instance space analysis complement each other very well. Particularly, algorithm selection alone would not have been sufficient to recognize that the original synthetic instances did not cover the real-life instances. Due to the instance space analysis, we were able to identify the regions of the instance space with gaps, and generate more meaningful instances with which to learn algorithm selection models more effectively.

In future work, one could also investigate another instance generation process where instances are generated randomly and then optimized heuristically to match the desired features, as proposed in Smith-Miles and Tan (2012). We also aim to consider either more CB-CTT solvers or other real-life timetabling problem variants and optimization problems. Lastly, our focus was on classification and improving the classification accuracy. As our results show, this does not imply that the average score will be optimized. One approach to directly optimize the score would be to perform regression on the performance measure, using these results to optimize the score rather than classifying the instance directly.

Acknowledgements The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged. This work was also supported by the Austrian Development Cooperation: Project HERAS – Higher Education, Research and Applied Science. Support from the Australian Research Council is also acknowledged through the Laureate Fellowship grant FL140100012.

References

Achá, R. A., & Nieuwenhuis, R. (2014). Curriculum-based course timetabling with sat and maxsat. *Annals of Operations Research*, *218*(1), 71–91.

Banbara, M., Inoue, K., Kaufmann, B., Okimoto, T., Schaub, T., Soh, T., et al. (2019). Teaspoon: Solving the curriculum-based course timetabling problems with answer set programming. *Annals of Operations Research*, *275*(1), 3–37.

Bellio, R., Ceschia, S., Di Gaspero, L., Schaerf, A., & Urli, T. (2016). Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. *Computers and Operations Research*, *65*, 83–92.

Berg, J., Demirovic, E., & Stuckey, P.J. (2019). Core-boosted linear search for incomplete maxsat. In *Integration of constraint programming, artificial intelligence, and operations research - 16th international conference, CPAIOR 2019, Thessaloniki, Greece, June 4–7, 2019, Proceedings*, pp. 39–56.

Bonutti, A., De Cesco, F., Di Gaspero, L., & Schaerf, A. (2012). Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. *Annals of Operations Research*, *194*(1), 59–70.

Burke, E. K., Causmaecker, D., & Patrick, S. (Eds.). (2003). *Practice and theory of automated timetabling iv, 4th international conference, PATAT 2002, Gent, Belgium, August 21–23, 2002, selected revised papers*. Lecture Notes in Computer Science (Vol. 2740). Springer.

Burke, E. K., Mareček, J., Parkes, A. J., & Rudová, H. (2008). Penalising patterns in timetables: Novel integer programming formulations. In J. Kalcsics & S. Nickel (Eds.), *Operations Research Proceedings 2007* (pp. 409–414). Heidelberg: Berlin.

Chiarandini, M., & Stützle, T. (2003). *Experimental evaluation of course timetabling algorithms. fachgebiet intellektik at tu darmstadt.,03*,

Coello C., Carlos A., editor. (2011). *Learning and intelligent optimization - 5th international conference, LION 5, Rome, Italy, January*

17–21, 2011. Selected Papers, volume 6683 of *Lecture notes in computer science*. Springer, Berlin

Gebser, M., Kaufmann, B., & Schaub, T. (2012). Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, *187*, 52–89.

Gottlieb, J., & Raidl, G.R., (eds.) (2004). *Evolutionary computation in combinatorial optimization, 4th european conference, EvoCOP 2004, Coimbra, Portugal, April 5–7, 2004, Proceedings*, volume 3004 of *Lecture notes in computer science*. Springer, Berlin

Hoos, H. H., Lindauer, M. T., & Schaub, T. (2014). claspfolio 2: Advances in algorithm selection for answer set programming. *TPLP*, *14*(4–5), 569–585.

Kostuch, P., & Socha, K. (2004). Hardness prediction for the university course timetabling problem. In *Evolutionary computation in combinatorial optimization, 4th European conference, EvoCOP 2004, Coimbra, Portugal, April 5–7, 2004, Proceedings*, pp. 135–144.

Lin, X., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2008). Satzilla: Portfolio-based algorithm selection for SAT. *The Journal of Artificial Intelligence Research*, *32*, 565–606.

Lopes, L., & Smith-Miles, K. (2010). Pitfalls in instance generation for udine timetabling. In C. Blum & R. Battiti (Eds.), *Learning and intelligent optimization* (pp. 299–302). Heidelberg: Berlin.

Lopes, L., & Smith-Miles, K. (2013). Generating applicable synthetic instances for branch problems. *Operations Research*, *61*, 563–577.

McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A. J., et al. (2010). Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, *22*(1), 120–130.

Muñoz, M. A., & Smith-Miles, K. A. (2017). Performance analysis of continuous black-box optimization algorithms via footprints in instance space. *Evolutionary Computation*, *25*(4), 529–554.

Muñoz, M. A., Villanova, L., Baatar, D., & Smith-Miles, K. (2018). Instance spaces for machine learning classification. *Machine Learning*, *107*(1), 109–147.

Musliu, N., Schwengerer, M. (2013). Algorithm selection for the graph coloring problem. In *Learning and intelligent optimization - 7th international conference, LION 7, Catania, Italy, January 7–11, 2013, Revised Selected Papers*, pp. 389–403.

Müller, T. (2009). Itc 2007 solver description: A hybrid approach. *Annals of Operations Research*, *172*(1), 429–446.

Nicosia, G., & Pardalos, P.M. (eds.). (2013). *Learning and intelligent optimization - 7th international conference, LION 7, Catania, Italy, January 7–11, 2013, Revised Selected Papers*, volume 7997 of *Lecture Notes in Computer Science*. Springer, Berlin

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers*, *15*, 65–118.

Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L.M., Knowles, J.D., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L., & Stützle, T. (2002). A comparison of the performance of different metaheuristics on the timetabling problem. In *Practice and theory of automated timetabling iv, 4th international conference, PATAT 2002, Gent, Belgium, August 21–23, 2002, Selected Revised Papers*, pp. 329–354.

Smith-Miles, K., Baatar, D., Wreford, B., & Lewis, R. (2014). Towards objective measures of algorithm performance across instance space. *Computers and Operations Research*, *45*, 12–24.

Smith-Miles, K., & Bowly, S. (2015). Generating new test instances by evolving in instance space. *Computers and Operations Research*, *63*, 102–113.

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

Smith-Miles, K., & Tan, T. T. (2012). Measuring algorithm footprints in instance space. *IEEE CEC*, *12*, 3446–3453.

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

Smith-Miles, K. A. (2009). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Survey*, *41*(1), 6:1–6:25.

Smith-Miles, K., & Lopes, L. (2011). Generalising algorithm performance in instance space: A timetabling case study. In *Learning and intelligent optimization - 5th international conference, LION 5, Rome, Italy, January 17–21, 2011. Selected Papers*, pp. 524–538.

Smith-Miles, K., & Lopes, L. (2012). Measuring instance difficulty for combinatorial optimization problems. *Computers and OR*, *39*(5), 875–889.