Computers & Operations Research 65 (2016) 83–92

Contents lists available at ScienceDirect

Computers & Operations Research

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

Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem

Ruggero Bellio [a], Sara Ceschia [b], Luca Di Gaspero [b], Andrea Schaerf [b,*], Tommaso Urli [c]

[a] DIES, University of Udine, via Tomadini 30/A, I-33100 Udine, Italy
[b] DIEGM, University of Udine, via delle Scienze 208, I-33100 Udine, Italy
[c] NICTA, Canberra Research Laboratory, Tower A, 7 London Circuit, Canberra ACT 2601, Australia

ARTICLE INFO

Available online 14 July 2015

Keywords:
Course timetabling
Simulated annealing
Metaheuristics
Feature analysis
Parameter tuning

ABSTRACT

We consider the university course timetabling problem, which is one of the most studied problems in educational timetabling. In particular, we focus our attention on the formulation known as the *curriculum-based course timetabling problem* (CB-CTT), which has been tackled by many researchers and for which there are many available benchmarks.

The contribution of this paper is twofold. First, we propose an effective and robust single-stage simulated annealing method for solving the problem. Second, we design and apply an extensive and statistically-principled methodology for the parameter tuning procedure. The outcome of this analysis is a methodology for modeling the relationship between search method parameters and instance features that allows us to set the parameters for unseen instances on the basis of a simple inspection of the instance itself. Using this methodology, our algorithm, despite its apparent simplicity, has been able to achieve high quality results on a set of popular benchmarks.

A final contribution of the paper is a novel set of real-world instances, which could be used as a benchmark for future comparison.

1. Introduction

The issue of designing the timetable for the courses of the incoming term is a typical problem that universities or other academic institutions face at each semester. There are a large number of variants of this problem, depending on the specific regulations at the involved institutions (see, e.g., [23,26,35]). Among the different variants, two in particular are now considered standard, and featured at the international timetabling competitions ITC-2002 and ITC-2007 [32]. These standard formulations are the *Post-Enrollment Course Timetabling* (PE-CTT) [27] and *Curriculum-Based Course Timetabling* (CB-CTT) [15], which have received an appreciable attention in the research community so that many recent articles deal with either one of them.

The distinguishing difference between the two formulations is the origin of the main constraint of the problem, i.e., the conflicts between courses that prevent one from scheduling them simultaneously. Indeed, in PE-CTT the source of conflicts is the actual student enrollment whereas in CB-CTT the courses in conflict are those that belong to the same predefined group of courses, or curricula. However, this is only one of the differences, which actually include many other distinctive features and cost components. For example in PE-CTT each course is a self-standing event, whereas in CB-CTT a course consists of multiple lectures. Consequently the soft constraints are different: in PE-CTT they are all related to events, penalizing late, consecutive, and isolated ones, while in CB-CTT they mainly involve curricula and courses, ensuring compactness in a curriculum, trying to evenly spread the lectures of a course in the weekdays, and possibly preserving the same room for a course.

In this work we focus on CB-CTT, and we build upon our previous work on this problem [4]. The key ingredients of our method are (i) a fast single-stage Simulated Annealing (SA) method, and (ii) a comprehensive statistical analysis methodology featuring a principled parameter tuning phase. The aim of the analysis is to model the relationship between the most relevant parameters of the solver and the features of the instance under consideration. The proposed approach tackles the parameter selection as a classification problem, and builds a rule for choosing the set of parameters most likely to perform well for a given instance, on the basis of specific features. The effectiveness of this methodology is confirmed by the experimental results on two groups of validation instances. We are able to show that our

* Corresponding author.
E-mail addresses: ruggero.bellio@uniud.it (R. Bellio), sara.ceschia@uniud.it (S. Ceschia), l.digaspero@uniud.it (L. Di Gaspero), schaerf@uniud.it (A. Schaerf), tommaso.urli@nicta.com.au (T. Urli).

http://dx.doi.org/10.1016/j.cor.2015.07.002
0305-0548/

method compares favorably with all the state-of-the-art methods on the available instances.

As an additional contribution of this paper, we extend the set of available problem instances by collecting several new real-world instances that can be added to the set of standard benchmarks and included in future comparison.

The source code of the solver developed for this work is publicly available at https://bitbucket.org/satt/public-cb-ctt.

2. Curriculum-based course timetabling

The problem formulation we consider in this paper is essentially the version proposed for the ITC-2007, which is by far the most popular. The detailed formulation is presented in Di Gaspero et al. [15], however, in order to keep the paper self-contained, we briefly report it also in the following. Alternative formulations of the problem, used in the experimental part of the paper (Section 6.7), are described in Bonutti et al. [8]. Essentially, the problem consists of the following 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*. A *period* is a pair composed of a day and a timeslot.

Courses and teachers: Each course consists of a fixed number of *lectures* to be scheduled in distinct periods, it is attended by a number of *students*, and is taught by a *teacher*. For each course, there is a minimum number of days across which the lectures of the course should be spread. Moreover, there are some periods in which the course cannot be scheduled (e.g., teacher's or students' availabilities).

Rooms: Each *room* has a *capacity*, i.e., a number of available seats.

Curricula: A *curriculum* is a group of courses that share common students. Consequently, courses belonging to the same curriculum are *in conflict* and cannot be scheduled at the same period.

A solution of the problem is an assignment of a period (day and timeslot) and a room to all lectures of each course so as to satisfy a set of *hard* constraints and to minimize the violations of *soft* constraints described in the following.

2.1. Hard constraints

There are three types of hard constraints:

RoomOccupancy: Two lectures cannot take place simultaneously in the same room.

Conflicts: Lectures of courses in the *same curriculum*, or *taught by the same teacher*, must be scheduled in different periods.

Availabilities: A course may not be available for being scheduled in a certain period.

2.2. Soft constraints

For the formulation ITC-2007 there are four types of soft constraints:

RoomCapacity: The capacity of the room assigned to each lecture must be greater than or equal to the number of students attending the corresponding course. The penalty for the violation of this constraint is measured by the number of students in excess.

MinWorkingDays: The lectures of each course cannot be tightly packed, but they must be spread into a given minimum number of days.

IsolatedLectures: Lectures belonging to a curriculum should be adjacent to each other (i.e., be assigned to consecutive periods). We account for 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.

For all the details, including input and output data formats and validation tools, we refer to Di Gaspero et al. [15].

3. Related work

In this section we review the literature on CB-CTT. The presentation is organized as follows: we firstly describe the solution approaches based on metaheuristic techniques; secondly, we report the contributions on exact approaches and on methods for obtaining lower bounds; finally, we discuss papers that investigate additional aspects related to the problem, such as instance generation and multi-objective formulations. A recent survey covering all these topics is provided by Bettinelli et al. [6].

3.1. Metaheuristic approaches

Müller [34] solves the problem by applying a constraint-based solver that incorporates several local search algorithms operating in three stages: (i) a construction phase that uses an *Iterative Forward Search* algorithm to find a feasible solution, (ii) a first search phase delegated to a *Hill Climbing* algorithm, followed by (iii) a *Great Deluge* or *Simulated Annealing* strategy to escape from local minima. The algorithm was not specifically designed for CB-CTT but it was intended to be employed on all three tracks of ITC-2007 (including, besides CB-CTT and PE-CTT, also Examination Timetabling). The solver was the winner of two out of three competition tracks, and it was among the finalists in the third one.

The *Adaptive Tabu Search* proposed by Lü and Hao [30] follows a three stage scheme: in the initialization phase a feasible timetable is built using a fast heuristic; then the intensification and diversification phases are alternated through an adaptive tabu search in order to reduce the violations of soft constraints.

A novel hybrid metaheuristic technique, obtained by combining *Electro-magnetic-like Mechanisms* and the *Great Deluge* algorithm, was employed by Abdullah et al. [1] who obtained high-quality results on both CB-CTT and PE-CTT testbeds.

Finally, Lü et al. [31] investigated the search performance of different neighborhood relations typically used by local search algorithms to solve this problem. The neighborhoods are compared using different evaluation criteria, and new combinations of neighborhoods are explored and analyzed.

3.2. Exact approaches and methods for computing lower bounds

Several authors tackled the problem by means of exact approaches with both the goal of finding solutions or computing lower bounds.

Among these authors, Burke et al. [12] engineered a hybrid method based on the decomposition of the whole problem into different sub-problems, each of them solved using a mix of different IP formulations. Subsequently, the authors propose a

novel IP model [10] based on the concept of "supernode" which was originally employed to model graph coloring problems. This new encoding has been applied to the CB-CTT benchmarks and was compared with the established two-index IP model. The results showed that the supernodes formulation is able to considerably reduce the computational time. Lastly, the same authors propose a *Branch-and-Cut* procedure [11] aimed at computing lower bounds for various problem formulations.

Lach and Lübbecke [25] proposed an IP approach that decomposes the problem in two stages: the first aims at assigning courses to periods and mainly focuses on satisfying the hard constraints; the second optimizes for the soft constraints and assigns lectures to rooms by solving a matching problem.

Hao and Benlic [18] developed a partition-based approach to compute lower bounds: the idea behind their method is to divide the original instance into sub-instances by means of an *Iterative Tabu Search* procedure. Afterwards, each subproblem is solved via an IP solver using the model proposed by Lach and Lübbecke [25]; a lower bound of the original instance is obtained by summing up the lower bound values of all these sub-instances.

A somewhat similar approach has been tried recently by Cacchiani et al. [13], who instead exploit soft constraints for the sub-instance partitioning. After the partitioning phase, two separated problems are formulated as IPs and then solved to optimality by a *Column Generation* technique.

Asín Achá and Nieuwenhuis [2] employed several satisfiability (SAT) models for tackling the CB-CTT problem that differ on which constraints are specifically considered soft or hard. Using different encodings they were able to compute lower bounds and obtain new best solutions for the benchmarks.

Finally, Banbara et al. [3] translated the CB-CTT formulation into an Answer Set Programming (ASP) problem and solved it using the ASP solver `clasp`.

A summary of the results of the aforementioned literature is given in Table 1, which reports the lower bounds for the instances of ITC-2007 testbed (named `comp` in the table). The results highlighted in boldface indicate the tightest lower bounds.

Table 1
Lower bounds for the `comp` instances. Proven optimal solutions in the *Best known* column are denoted by an asterisk. *Note:* These bounds were obtained using diverse experimental setups (particularly time budgets) and therefore should not be regarded as a comparison of methods to find lower bounds but rather as a record of the lower bounds available in the literature.

Instance	Lü (2009)	Burke (2010)	Hao (2011)	Lach (2012)	Burke (2012)	Asín Achá (2012)	Cacchiani (2013)	Best known
comp01	4	**5**	4	4	4	0	**5**	5*
comp02	11	6	12	11	0	**16**	**16**	24
comp03	25	43	38	25	2	28	**52**	66
comp04	28	2	**35**	28	0	**35**	**35**	35*
comp05	108	183	183	108	99	48	166	284
comp06	12	6	22	10	0	**27**	11	27*
comp07	**6**	0	**6**	**6**	0	**6**	**6**	6*
comp08	**37**	2	**37**	**37**	0	**37**	**37**	37*
comp09	47	0	72	46	0	35	92	96
comp10	**4**	0	**4**	**4**	0	**4**	2	4*
comp11	0	0	0	0	0	0	0	0*
comp12	57	5	**109**	53	0	99	100	298
comp13	41	0	**59**	41	3	**59**	57	59*
comp14	46	0	**51**		0	**51**	48	51*
comp15			**38**			28	52	66
comp16			16			**18**	13	18*
comp17			48			**56**	48	56*
comp18			24			**27**	52	62
comp19			**56**			46	48	57
comp20			2			**4**	**4**	4*
comp21			**61**			42	42	74

Note that these bounds were obtained using diverse experimental setups (particularly time budgets) and therefore should not be regarded as a fair comparison of methods, but rather as a record of the lower bounds available in the literature. In the table we also report the best results known at the time of writing. Best values marked with an asterisk are guaranteed optimal (i.e., they match the lower bound).

3.3. Additional research issues

We now discuss research activities on CB-CTT that consider other issues besides the solution to the original problem.

The first issue concerns the development of instance generators. To the best of our knowledge, the first attempt to devise an instance generator for CB-CTT is due to Burke et al. [10], who tried to create instances that resemble the structure of the ITC-2007 `comp` instances. That generator has been revised and improved by Lopes and Smith-Miles [29], who based their work on a deeper insight on the instance features and made their generator publicly available. For our experimental analysis we use the generator developed in the latter work.

A further research issue concerns the investigation of the problem as a multi-objective one. To this regard, Geiger [16] considered the CB-CTT problem as a multi-criteria decision problem, and studied the influence of the weighted-sum aggregation methodology on the performance of the technique.

Another issue related to multi-objective studies concerns the concept of *fairness*. Notably, in the standard single objective formulation, some curricula can be heavily penalized in the best result in favor of an overall high quality. Mühlenthaler and Wanka [33] studied this problem and considered various notions of fairness. Moreover, they compared these notions in order to evaluate their effects with respect to the other objectives.

4. Search method

We propose a solution method for the problem that is based on the well-known *Simulated Annealing* metaheuristic [24]. As it is customary for local search, in order to instantiate the abstract solution method to the problem at hand we must define a search space, a neighborhood relation, and a cost function.

4.1. Search space

We consider the search space composed of all the assignments of lectures to rooms and periods for which the hard constraint Availability is satisfied. Hard constraints Conflicts and RoomOccupancy are considered as components of the cost function and their violation is highly penalized.

4.2. Neighborhood relations

We employ two different basic neighborhood relations, defined by the set of solutions that can be reached by applying any of the following moves to a solution:

MoveLecture (ML): Change the period and the room of one lecture.
SwapLectures (SL): Swap the period and the room of two lectures of distinct courses.

The two basic neighborhoods are merged in a composite neighborhood that considers the set union of ML and SL. However, according to the results of our previous study [4], we restrict the ML neighborhood so that only moves which place the lecture in an

empty room are considered (whenever possible, i.e., when the room occupancy factor is less than 100%, as there would be no empty rooms in such a case). Moreover, our previous findings suggest employing a non-uniform probability for selecting the moves in the composed neighborhood. In detail, the move selection strategy consists of two stages: first the neighborhood is randomly selected with a non-uniform probability controlled by a parameter *sr* (*swap rate*), then a random move in the selected neighborhood is uniformly drawn.

4.3. Cost function

The cost of a solution s is a weighted sum of the violations of the hard constraints Conflicts and RoomOccupancy and of the objectives (i.e., the measure of soft constraints violations)

$$\begin{aligned} Cost(s) = {} & \mathsf{Conflicts}(s) \times w_{hard} \\ & + \mathsf{RoomOccupancy}(s) \times w_{hard} \\ & + \mathsf{MinWorkingDays}(s) \times w_{wd} \\ & + \mathsf{RoomStability}(s) \times w_{rs} \\ & + \mathsf{RoomCapacity}(s) \times w_{rc} \\ & + \mathsf{IsolatedLectures}(s) \times w_{il} \end{aligned}$$

where the weights $w_{wd} = 5$, $w_{rs} = w_{rc} = 1$ and $w_{il} = 2$ are defined by the problem formulation, while w_{hard} is a parameter of the algorithm (see Section 4.5). This value should be high enough to give precedence to feasibility over objectives, but it should not be so high as to allow the SA metaheuristic (whose move acceptance criterion is based on the difference in the cost function between the current solution and the neighboring one) to select also moves that could increase the number of violations in early stages of the search.

4.4. The SA metaheuristic

In a departure from our previous work [4], in which the metaheuristic that guides the search is a combination (token-ring) of Tabu Search and a "standard" version of SA, in this work we employ a single-stage enhanced version of the latter method. We show that this rather simple SA variant, once properly tuned, outperforms such a combination.

The SA metaheuristic [24] is a local search method based on a probabilistic acceptance criterion for non-improving moves. Specifically, at each iteration a neighbor of the current solution is randomly selected and it is accepted if either (i) it improves the cost function value or, (ii) according to an exponential probability distribution that is biased by the amount of worsening and by a parameter T called temperature. Besides the initialization functions (setting the initial solution and fixing temperature T_0), the main hot-spots of the method are the function for updating (i.e., decreasing) the temperature and the stopping condition. In the standard variant of SA, the temperature update is performed at regular intervals (i.e., every n_s iterations) and the cooling scheme employed is a geometric one. That is, the temperature is decreased by multiplying it for a cooling factor α ($0 < \alpha < 1$), so that $T' = \alpha \cdot T$. The search is stopped when the temperature reaches a minimum value T_{min} that will prevent accepting worsening solutions. The main differences between the SA approach implemented in this work and the one proposed in Bellio et al. [4] reside in a different specification of these two components. In this implementation we replace the standard functions with the following strategies:

1. a *cutoff-based* temperature cooling scheme [22];
2. a different *stopping condition* for the solver, based on the maximum number of allowed iterations.

In the following, we detail these two aspects of the SA method employed in this work.

Cutoff-based cooling scheme: In order to better exploit the time at its disposal, our algorithm employs a cutoff-based cooling scheme. In practice, instead of sampling a fixed number n_s of solutions at each temperature level (as it is customary in SA implementations), the algorithm is allowed to decrease the temperature prematurely by multiplying it by a cooling rate *cr*, provided that a portion $n_a \leq n_s$ of the *sampled* solutions has been *accepted* already. This allows us to speed-up the initial part of the search, thus saving iterations that can be used in the final part, where intensification takes place.

Stopping condition: To allow for a fair comparison with the existing literature, instead of stopping the search when a specific (minimum) temperature T_{min} is reached, our algorithm stops on the basis of an iteration budget, which is roughly equivalent to fixing a time budget, given that the cost of one iteration is approximately constant.

A possible limitation of this choice is that the temperature might still be too high when the budget runs out. In order to overcome this problem, we fix an *expected* minimum temperature T_{min} to a reasonable value and we compute the number n_s (see Eq. (1)) of solutions sampled at each temperature so that the minimum temperature is reached exactly when the maximum number of iterations is met. That is

$$n_s = iter_{max} \Big/ \left(\frac{-\log\left(T_0/T_{min}\right)}{\log\ cr} \right). \tag{1}$$

Because of the cutoff-based cooling scheme, at the beginning of the search the temperature might decrease before all n_s solutions have been sampled. Thus T_{min} is reached in k iterations in advance, where k depends on the cost landscape and on the ratio n_a/n_s. These spared iterations are exploited at the end of the search, i.e., after T_{min} has been reached, to carry out further intensification moves.

In order to simplify the parameters of the algorithm, given the dependence of the cutoff-based scheme on the ratio n_a/n_s and the relation $n_a \leq n_s$, we decided to indirectly specify the value of the parameter n_a by including a real-valued parameter $\rho \in [0, 1]$ defined as $\rho = n_a/n_s$.

4.5. SA parameters

Our SA metaheuristic involves many parameters. In order to refrain from making any "premature commitment" (see [21]), we consider all of them in the experimental analysis. They are summarized in Table 2, along with the ranges involved in the experimental analysis which have been fixed based on preliminary experiments.

The iterations budget has been fixed, for each instance, to $iter_{max} = 3 \cdot 10^8$, a value that provides the algorithm with a running time which is approximately equivalent to the one allowed by ITC-2007 computation rules (namely 408s on our test machine).

Table 2
Parameters of the search method.

Parameter	Symbol	Range
Starting temperature	T_0	[1, 100]
Neighbors accepted ratio (n_a/n_s)	ρ	[0.01, 1]
Cooling rate	cr	[0.99, 0.999]
Hard constraints weight	w_{hard}	[10, 1000]
Neighborhood swap rate	sr	[0.1, 0.9]
Expected minimum temperature	T_{min}	[0.01, 1]

Table 3
Minimum and maximum values of the features for the families of instances (#I: number of instances): courses (C), total lectures (Le), rooms (R), periods (Pe), curricula (Cu), room occupation (RO), average number of conflicts (Co), average teachers availability (Av), room suitability (RS), average daily lectures per curriculum (DL).

Family	#I	C	Le	R	Pe	Cu
comp	21	30–131	138–434	5–20	25–36	13–150
DDS	7	50–201	146–972	8–31	25–75	9–105
Udine	9	62–152	201–400	16–25	25–25	54–101
EasyAcademy	12	50–159	139–688	12–65	25–72	12–65
Erlangen	6	705–850	788–930	110–176	30–30	1949–3691
Family	#I	RO	Co	Av	RS	DL
comp	21	42.6–88.9	4.7–22.1	57.0–94.2	50.2–72.4	1.5–3.9
DDS	7	20.1–76.2	2.6–23.9	21.3–91.4	53.6–100.0	1.9–5.2
Udine	9	50.2–76.2	4.0–6.6	70.1–95.5	57.5–71.3	1.7–2.7
EasyAcademy	12	17.6–52.0	4.8–22.2	55.1–100.0	41.8–70.0	2.7–7.7
Erlangen	6	15.7–25.1	3.0–3.8	66.7–71.4	45.5–56.0	0.0–0.9

5. Problem instances

We now describe the instances considered in this work and how we use them in our experimental analysis. In particular, according to the customary *cross-validation* guidelines (e.g., [19]), we have split the instance set into three groups: a set of *training instances* used to tune the algorithm, a set of *validation instances* used to evaluate the tuning, and finally a set of *novel instances* to verify the quality of the proposed method on previously unseen instances.

5.1. Training instances

The first group of instances is a large set of artificial instances created using the generator by Leo Lopes (see [29]), which has been specifically designed to reproduce the features of real-world instances.

The generator is parametrized on two instance features: the total number of lectures and the percentage of room occupation.

In order to avoid *overtuning*, this is the only group of instances that has been used for the tuning phase. Accordingly, reporting individual results on these instances it would be of little significance and are therefore omitted. Instead, we will describe in detail the methodology for creating and testing those instances.

Although Lopes and Smith-Miles [29] made available a set of 4200 generated instances, in order to have a specific range of values necessary for our objectives we created our own instances by running their generator.

Specifically, we have created 5 instances for each combination of values of the two control parameters of the instance generator. Namely, the number of lectures ranges in $\{i \cdot 50 | i = 1, \ldots, 24\}$, so that they will be comprised between 50 and 1200, and the percentage of room occupation will be one of the values $\{50\%, 70\%, 80\%, 90\%\}$. On overall, the full testbed consists of 480 instances ($5 \times 24 \times 4$).

After screening them in detail, it appears that not all instances generated are useful for our parameter tuning purposes. In particular, we have identified the following instance classes:

Provably infeasible: Infeasibility is easily proven (e.g., some courses have more lectures than available periods).
Unrealistic room endowment: Only high cost solutions exist, due to the presence of courses with more students than the available rooms.
Too hard: The solver has never been able to find a feasible solution (given a reasonably long timeout).
Too easy: Easily solved to cost 0 by all the solver variants.

Instance belonging to these classes are discarded, since these are features that generally do not appear in real cases. Furthermore, except for the *too hard* class, an instance can be easily classified into one of the remaining classes. In order to have the expected number of instances for any combination of parameter values, we replace the discarded instances with new ones by repeating the instance generation procedure.

5.2. Validation instances

The second group comprises those instances that have been used in the literature. This group consists of the usual set of instances, the so-called comp, which is composed of 21 real-world instances that have been used for ITC-2007.

Over this set we illustrate the solver emerging from the tuning on the first group, and we compare it with the state-of-the-art methods.

5.3. Novel instances

The final set is composed by four families of *novel* instances, which have been proposed recently so that no (or very few) results are available in the literature. This set is a candidate to become a new benchmark for future comparisons.

The first family in this group, contributed by Moritz Mühlenthaler, is called Erlangen and comprises four instances of the course timetabling problem arising at the University of Erlangen. These instances are considerably larger than the comp ones and they exhibit a very different structure than most of the other real-world instances.

The second and third families in this group consist of a set of recent real-world instances from the University of Udine and of a group of cases from other Italian universities. These instances have been kindly provided by EasyStaff S.r.l. (http://www.easystaff.it), a company specializing in timetabling solutions. They were collected by the commercial software EasyAcademy. We call these two families Udine and EasyAcademy respectively.

A further set of 7 instances, called DDS, from other Italian universities, is available and has been used occasionally in the literature.

5.4. Summary of features

Table 3 summarizes the families of instances of the latter two groups, highlighting some aggregate indicators (i.e., minimum and maximum values) of the most relevant features. All instances employed in this work, with the exception of the training instances, are available from the CB-CTT website http://satt.diegm.uniud.it/ctt.

Table 4
Revised intervals for investigated parameters.

Parameter	Symbol	Range
Starting temperature	T_0	[**1**, **40**]
Neighbors accepted ratio (n_a/n_s)	ρ	[**0.034**, **0.05**]
Cooling rate	cr	{0.99}
Hard constraints weight	w_{hard}	{100}
Neighborhood swap rate	sr	{0.43}
Expected minimum temperature	T_{min}	[**0.015**, **0.21**]

Table 5
Percentage of "good performances" for the 20 experimental configurations (i.e., the percentage of instances in which the given configuration is equivalent to the best one).

Configuration	T_0	$T_{min} \times 100$	$\rho \times 100$	% Good performance
1	21.72	20.56	4.76	54.8
2	2.22	18.57	4.68	20.7
3	20.50	17.00	3.48	93.0
4	25.38	19.67	3.80	87.6
5	22.94	15.22	4.12	89.5
6	27.81	17.89	4.44	79.3
7	3.44	20.35	4.04	24.5
8	32.69	19.22	4.28	82.5
9	31.47	17.45	4.92	75.5
10	35.13	18.34	3.96	93.3
11	30.25	15.67	3.64	93.9
12	37.56	16.55	4.60	83.4
13	5.88	17.68	3.72	50.3
14	7.09	19.43	5.00	32.2
15	8.31	15.88	4.36	58.3
16	10.75	19.00	3.56	69.7
17	13.19	17.23	4.20	66.2
18	11.97	15.45	4.84	57.0
19	18.06	19.89	4.52	58.6
20	15.63	16.34	3.88	85.0

6. Experimental analysis

We conduct a principled and extensive experimental analysis, aimed at obtaining a robust tuning of the various parameters of our algorithm, so that its performances compare favorably with those obtained by other state-of-the-art methods.

In order to achieve this, we investigate the possible relationships between instance features (reported in Table 3) and ideal configurations of the solver parameters. The ultimate goal of this study is to find, for each parameter, either a *fixed value* that works well on a broad set of instances, or a *procedure* to predict the best value based on measurable features of each instance. Ideally, the results of this study should carry over to unseen instances, thus making the approach more general than classic parameter tuning. This is, in fact, an attempt to alleviate the effect of the *No Free Lunch Theorems for Optimization* [37], which state that, for any algorithm (or parameter configuration), any elevated performance over one class of problems is exactly paid for in performance over another class.

In the following, we first illustrate the experimental setting and the statistical methodology employed in this study. We then summarize our findings and present the experimental results on the *validation* and *novel instances* compared against the best known results in the literature.

6.1. Design of experiments and experimental setting

Our analysis is based on the 480 training instances described in Section 5.1. The parameter configurations used in the analysis are sampled from the *Hammersley point set* [17], for the ranges whose bounds are reported in Table 2. This choice has been driven by two properties that make this point generation strategy particularly suitable for parameter tuning. First, the Hammersley point set is *scalable*, both with respect to the number of sampled parameter configurations, and to the dimensions of the sampled space. Second, the sampled points exhibit *low discrepancy*, i.e., they are space-filling, despite being random in nature. For these reasons, by sampling the sequence, one can generate any number of representative combinations of any number of parameters. Note that the sequence is deterministic, and must be seeded with a list of prime numbers. Moreover, since the sequence generates points $p \in [0,1]^n$, these values must then be re-scaled in their desired intervals.

All the experiments were generated and executed using JSON2RUN [36] on an Ubuntu Linux 13.04 machine with 16 Intel® Xeon® CPU E5-2660 (2.20 GHz) physical cores, hyper-threaded to 32; each experiment was dedicated a single virtual core.

6.2. Exploratory experiments

In preparation to our analysis, we carried out three preliminary steps.

First, we ran an *F-Race* tuning [7] of our algorithm over the training instances with a 95% confidence level, in order to establish a baseline for further comparisons. The race ended with more than one surviving configuration, mainly differing for the values of w_{hard} and cr, but giving a clear indication about good values for the other parameters. This suggested that setting a specific value for w_{hard} and cr, at least within the investigated intervals, was essentially irrelevant to the performance, and allowed us to simplify the analysis by fixing $w_{hard} = 100$ and $cr = 0.99$ (see Table 4). It is worth noticing that removing an irrelevant parameter from the analysis has the double benefit of reducing experimental noise, and allows a finer-grained tuning of the other parameters, at the same computational cost.

Second, we tested all the sampled parameter configurations against the whole set of training instances. This allowed us to further refine our study in two ways. First, we realized that our initial estimates over the parameter intervals were too conservative, encompassing low-performance areas of the parameters space. A notable finding was that, on the whole set of training instances, a golden spot for sr was around 0.43. We thus fixed this parameter as well, along with w_{hard} and cr. Table 4 summarizes the whole parameter space after this preliminary phase (parameters in boldface have not been fixed in this phase, and are the subject of the following analysis). Second, by running a Kruskal–Wallis test (see [20]) with significance level 10% on the dependence of cost distribution on parameter values, we realized that some of the instances were irrelevant to our analysis, and we therefore decided to drop them, and to limit our study to the 314 significant ones.

Finally, we sampled 20 parameter configurations from the remaining 3-dimensional (T_0, ρ, T_{min}) Hammersley point space (see Table 5), repeated the race with fewer parameters, and found a single winning configuration, corresponding to configuration #11 in Table 5. In addition, we performed 10 independent runs of each parameter configuration on every instance. This is the data upon which the rest of our experimental analysis is based.

6.3. Statistical methodology

In order to build a model to *predict* the parameter values for each instance, we proceed in two stages. In the first stage, for each of the selected training instances we identify the effective parameter configurations. The second stage consists in learning a rule to associate the parameter configurations to the features. In

Table 6
Best results and comparison with other approaches over the *validation* instances. Values are averages over multiple runs of the algorithms.

Instance	[34]	[30]	[1]	[4]	us (FBT)	us (F-Race)
comp01	**5.00**	**5.00**	**5.00**	**5.00**	5.23	5.16
comp02	61.30	60.60	53.90	**51.60**	52.94	53.42
comp03	94.80	86.60	84.20	82.70	**79.16**	80.48
comp04	42.80	47.90	51.90	47.90	39.39	**39.29**
comp05	343.50	**328.50**	339.50	333.40	335.13	329.06
comp06	56.80	69.90	64.40	55.90	**51.77**	53.35
comp07	33.90	28.20	**20.20**	31.50	26.39	28.45
comp08	46.50	51.40	47.90	44.90	43.32	**43.06**
comp09	113.10	113.20	113.90	108.30	**106.10**	**106.10**
comp10	**21.30**	38.00	24.10	23.80	21.39	21.71
comp11	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
comp12	351.60	365.00	355.90	346.90	**336.84**	338.39
comp13	73.90	76.20	**72.40**	73.60	73.39	73.65
comp14	61.80	62.90	63.30	60.70	**58.16**	59.71
comp15	94.80	87.80	88.00	89.40	**78.19**	79.10
comp16	41.20	53.70	51.70	43.00	**38.06**	39.19
comp17	86.60	100.50	86.20	83.10	**77.61**	78.84
comp18	91.70	82.60	85.80	84.30	**81.10**	83.29
comp19	68.80	75.00	78.10	71.20	**66.77**	67.13
comp20	**34.30**	58.20	42.90	50.60	46.13	45.94
comp21	108.00	125.30	121.50	106.90	103.32	**102.19**

principle, each of the two stages can be performed using various strategies. Some compelling choices are described in the following.

6.3.1. Identification of the effective parameter configurations

For each instance, we aim at identifying which of the 20 experimental configurations perform well. We start by selecting the *best-performing configuration*, defined here as the configuration with the smallest median cost (as described in Section 4.3). Other robust estimates of location could be employed for the task, such as the Hodges–Lehmann estimator (e.g., [20, Section 3.2]), but the median is the simplest possibility. Then, a series of Wilcoxon rank-sum tests are run to identify the configurations which can be taken as equivalent to the best-performing one. For the latter task, it is crucial to adjust for multiple testing, as a plurality of statistical tests are carried out. Here we control the *false discovery rate* (FDR) [5], thus adopting a more powerful approach than classical alternatives which control the family-wise error rate, such as Bonferroni or Holm methods (see [19, Chapter 18]). In short, we accept to wrongly declare some of the configurations as less performing than the best one, rather than strictly controlling the proportion of such mistakes, ending up declaring nearly all the configurations as equivalent to the best-performing one for several instances. In this study we set a FDR threshold equal to 0.10.

The result of this first stage is a 314×20 binary matrix; the ith row of such a matrix contains the value 1 for those configurations equivalent to the best-performing one on the ith instance, and 0 otherwise. Table 5 summarizes the column averages for such a matrix, thus reporting the percentages of "good performances" for each of the experimental configurations. The striking result is that some of the configurations, such as #3, 4, 5, 10 and 11, perform well in the vast majority of the instances. We will return on this point in Section 6.5.

6.3.2. Prediction of optimal configurations based on measurable features

Once the performances of each parameter configuration have been identified for each instance, we include instance features in the process. We tackle the problem as a classification problem, and use the instance features to predict the outcome of each of the 20 binary variables reporting the performance of the different experimental configuration. In other words, we build 20 different classifiers, each using the instance features to return the probability of good performance of the experimental configuration. The configuration achieving the highest probability is then identified as the outcome of the classification process. The idea is that by making use of the instance features we have a chance to improve over the majority class rule that simply picks up the configuration with the highest success rate in Table 5, namely configuration #11. In the worst case, where no feature is able to provide some useful information for performing the classification, we can revert to the majority class rule, which can be considered as a baseline strategy. In our experimental setting, where the best-performing configuration has a success rate as high as 93.9%, getting a significant improvement is, therefore, not straightforward.

Among the possible classification methods, we choose the random forests method [9]. This classification method is an extension of classification trees that perform rather well compared to alternative methods, and for which efficient software is available. We use the implementation provided by the R package `randomForest` [28], using 5000 trees for growing the forest.

We train the different classifiers by including all the available features reported in Table 3, with the exception of periods (Pe), which was assumed to be constant across the 314 training instances selected, courses (C) and rooms (R), which were essentially duplicating the information content of total lectures (Le). A total number of seven features were therefore used in the classification process.

We estimate the additional gain provided by feature-based classification by means of 10-fold cross validation, which provides an estimate of the classification error [19, Section 7.10]. We obtain a classification accuracy for the method based on random forests equal to 95.5%, which is better than simpler methods such as logistic regression and generalized additive models, providing a classification accuracy equal to 94.3% and 93.9% respectively. Although we use random forests as a black box classifier, one of the good properties of such a methodology is that it also returns some measures of feature importance. Based on the prediction strength [19, Section 15.3.2], we found that for 10 out of the 20 classifiers, total lectures (Le) is the most important variable, and in the remaining 10 classifiers that role is played by the average number of conflicts (Co). These two features can then be deemed as the most important for selecting algorithm parameters.

6.4. Comparison with other approaches

In order to validate the quality of our approach, we compared its results against the best ones in the literature using the ITC-2007 rules and instances. For the sake of fairness, we do not include in the comparison results that are obtained by allotting a higher runtime, for example those of Asín Achá and Nieuwenhuis [2], who use 10,000–100,000s instead of about 300–500s as established by the competition rules.

Table 6 shows the average of 31 repetitions of the algorithm, in which we have highlighted in boldface the lowest average costs. We report the results obtained with both feature-based tuning (FBT) and standard F-Race tuning (further discussed in Section 6.5). The figures show that our approach matches or outperforms the state-of-the-art algorithms in more than half of the instances, also improving on our previous results [4].

This outcome is particularly significant because unlike previous approaches we have not involved the validation instances in the tuning process, thus avoiding the classical overtuning phenomenon.

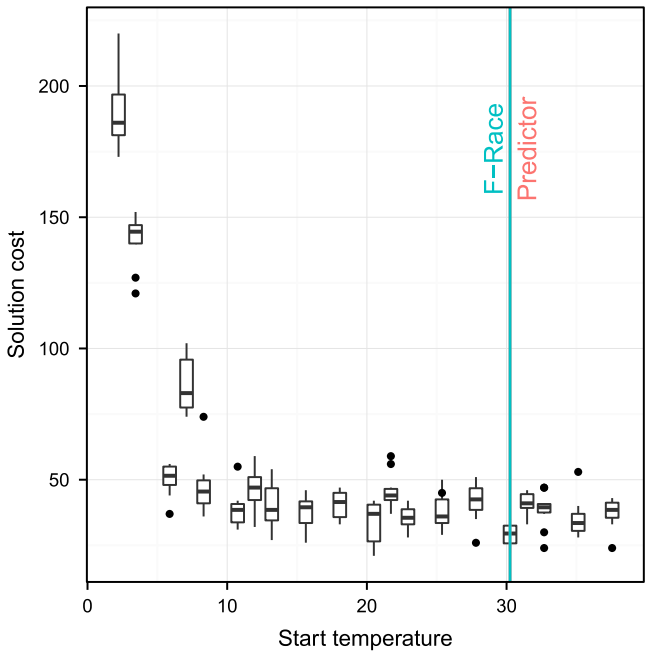

Fig. 1. Relation between T_0 and cost distribution on a single training instance with $Le=1200$. We show the ideal T_0 found by F-Race and by our feature-based predictor.

Table 7
Our results on the *novel* instances.

Instance	us (FBT)		us (F-Race)	
	best	*avg*	*best*	*avg*
Udine1	5	13.29	7	12.45
Udine2	14	19.26	11	21.42
Udine3	3	8.52	5	10.23
Udine4	64	66.16	64	66.32
Udine5	0	3.13	0	2.97
Udine6	0	0.35	0	0.19
Udine7	0	2.03	0	2.32
Udine8	34	39.26	33	39.10
Udine9	23	29.84	26	29.90
EasyAcademy01	65	65.26	65	65.03
EasyAcademy02	0	0.06	0	0.03
EasyAcademy03	2	2.06	2	2.10
EasyAcademy04	0	0.35	0	0.10
EasyAcademy05	0	0.00	0	0.00
EasyAcademy06	5	5.13	5	5.06
EasyAcademy07	0	0.48	0	0.32
EasyAcademy08	0	0.00	0	0.00
EasyAcademy09	4	5.16	4	4.74
EasyAcademy10	0	0.03	0	0.06
EasyAcademy11	0	2.90	0	3.23
EasyAcademy12	4	4.03	4	4.00
DDS1	85	110.26	91	106.55
DDS2	0	0.00	0	0.00
DDS3	0	0.00	0	0.00
DDS4	18	21.13	17	20.55
DDS5	0	0.00	0	0.00
DDS6	5	9.87	4	10.35
DDS7	0	0.00	0	0.00

6.5. Comparison with the F-Race baseline

The figures in Table 6 reveal that the feature-based tuning outperforms the F-Race approach on most instances in terms of average results. A more precise assessment reveals that the two methods are broadly comparable, as the Wilcoxon rank-sum test is significant at the 5% level (in favor of FBT) only on one instance (comp14): consequently, any sensible global test would not flag any significant difference between the two methods. At any rate, we can safely say that the FBT method is no worse than F-Race, and occasionally can be better.

It is worth noting that the above conclusion is something to be expected. Indeed, the parameter configuration selected by F-Race is #11, which corresponds to the best-performing configuration over the training instances (see Table 5). Strictly speaking, the equivalence between F-Race and the proposed FBT when the features are not informative is not perfect, as F-Race uses a different adjustment for multiple testing compared to the FDR method adopted for FBT, but nevertheless it is hardly surprising that the configuration surviving the race is the one with the highest percentage of good performances.

At this point, it makes sense to look for an explanation of why some configurations could achieve quite high success rates in the training phase. By looking at Fig. 1, which is representative of a large portion of the training instances, it is possible to glean some information. The parameter space (in this case for T_0) is split into two well-separated parts. One part (the leftmost in Fig. 1) yields poor results, while any choice of values inside the other part is reasonably safe. In our scenario, the portion of the parameter space leading to poor results is narrow, while the portion leading to better results is broader. This suggests that the chosen algorithm is robust with respect to parameter choice, at least for this problem domain. As a consequence, it is possible to find some parameter configurations that work consistently well across a large set of instances.

6.6. Results on the novel instances

In Tables 7 and 8 we show the results obtained by 31 repetitions of our algorithm on the instances of the *novel* group of families, for future comparison with these instances. The results are reported both for the F-Race and the feature-based tuning.

Across all the 28 instances of Table 7, only in two cases there was a significant difference at the 5% level (in favor of FBT, namely for the Udine2 and Udine3), so that the two methods are largely equivalent in performance.

Table 7 does not include the instances of the Erlangen family because this family shows a very particular structure, caused by the peculiar way in which the original timetabling problem is represented. In particular, the average number of lectures (Le) is very close to the average number of courses (C) (see Table 3), suggesting that, in almost all cases, courses are composed of a single lecture. Similarly, the number of curricula (Cu), which is typically smaller than the number of courses, is instead one order of magnitude larger, possibly because each curriculum represents the choice of a single student. This kind of mapping is closer in structure to the one used in the PE-CTT formulation, which has been addressed in Ceschia et al. [14].

Therefore, for the Erlangen instances we decided to present also the results of a new F-Race tuning done *ad hoc* for these cases, using different ranges for the parameter values. In fact, it turned out that the winning parameter configuration is $T_0=100$, $T_{min}=0.36$, $\rho=0.23$, $sr=0.43$ and $w_{hard}=500$, which is considerably different from the ones used in the previous analyses.

For comparison, we report in Table 8 also the results obtained by re-running the solver by Müller [34], which is available online. Given that these instances are larger, we used a longer timeout of 600 s.

It is worth mentioning that, although the different semantics of the involved features for this family of instances, our feature-based predictor was the only one able to obtain always feasible solutions, proving its robustness and wide range applicability. Indeed, Müller's solver, and the F-Race and Ad Hoc F-Race-tuned versions

Table 8
Our results on the *Erlangen* instances.

Instance	Müller [34]			us (FBT)			us (F-Race)			us (*ad hoc* F-Race)		
	best	*avg*	*feas* (%)	*best*	*avg*	*feas* (%)	*best*	*avg*	*feas* (%)	*best*	*avg*	*feas* (%)
Erlangen-2011-2	5569	5911.10	32	5444	6568.42	100	–	–	0	4680	4971.80	56
Erlangen-2012-1	8059	8713.48	100	7991	9130.87	100	8174	8604.77	71	7519	7856.85	100
Erlangen-2012-2	11 267	13 367.53	61	13 693	16 183.74	100	–	–	0	9587	10 130.33	7
Erlangen-2013-1	7993	8878.19	94	7445	9299.94	100	7745	8174.33	29	7159	7825.55	95
Erlangen-2013-2	8809	19 953.81	100	8846	11 273.97	100	–	–	0	8329	8844.34	86
Erlangen-2014-1	6728	7978.42	100	6601	8251.13	100	6418	7080.75	26	6144	6525.05	100

of our solver, found feasible solutions for 81%, 21% and 74% of the runs, respectively.

Looking at the average values reported in Table 8, we notice that whenever a feasible solution was found, our F-Race tuned solver outperforms all the other methods on all 6 instances, including [34].

6.7. Results on the comp instances for the other problem formulations

Finally, in order to given a more comprehensive picture of the solver robustness, we report its results on other formulations of the problem with different constraints and objectives. In particular, we consider the formulations UD3, UD4, and UD5, proposed by Bonutti et al. [8], that focus on student load, double lectures presence, and travel costs, respectively.

Tables 9–11 present average and best results for the configurations obtained by both tuning methods, on the three formulations. Those tables also show the results by Banbara et al. [3], which are, to our knowledge, the only published results on these formulations.

The comparison reveals that again FBT and F-Race perform similarly, and that in general, they perform better than the solver by Banbara et al. [3].

Table 9
Our results on the comp instances for the UD3 formulation.

Instance	us (FBT)		us (F-Race)		Banbara et al. [3]
	best	*avg*	*best*	*avg*	*z*
comp01	8	8.00	8	8.00	10
comp02	15	22.13	14	24.06	12
comp03	29	35.71	27	34.00	147
comp04	2	2.06	2	2.19	2
comp05	271	331.55	271	329.29	1232
comp06	8	11.42	8	11.94	8
comp07	0	2.00	0	1.81	0
comp08	2	3.61	2	4.19	2
comp09	8	9.87	8	11.19	8
comp10	0	1.74	0	1.35	0
comp11	0	0.00	0	0.00	0
comp12	54	76.23	57	77.94	1281
comp13	22	24.77	22	25.29	63
comp14	0	0.13	0	0.19	0
comp15	18	24.06	16	24.19	118
comp16	4	5.81	4	5.87	4
comp17	12	14.35	12	14.39	12
comp18	0	0.84	0	0.39	0
comp19	24	30.65	26	33.13	93
comp20	0	5.42	0	4.23	0
comp21	12	17.87	10	17.29	6

7. Conclusions and future work

In this paper we have proposed a simple yet effective SA approach to the CB-CTT problem. Moreover, we performed a comprehensive statistical analysis to identify the relationship between instance features and search method parameters. The outcome of this analysis makes it possible to set the parameters for unseen instances on the basis of a simple inspection of the instance itself.

The results of the feature-based tuned algorithm on a testbed of *validation* instances allow us to improve our previous results [4] and outperform the results in the literature on 10 instances out of 21.

The results of this work support the conclusion that instance features may provide useful information for tuning the parameters of the solver. Therefore, a sensible direction for future investigation may focus on refining this approach by exploring novel features obtained as functions, possibly complex in nature, of the original features.

In the future, we plan to investigate new versions of SA that could be applied to this problem. For example, we plan to consider the version that includes reheating. In addition, we plan to investigate other metaheuristics techniques for the problem through an analogous statistical analysis. Finally, we aim at applying this search and tuning technique to other timetabling problems.

Table 10
Our results on the comp instances for the UD4 formulation.

Instance	us (FBT)		us (F-Race-F2)		Banbara et al. [3]
	best	*avg*	*best*	*avg*	*z*
comp01	6	6.00	6	6.00	9
comp02	26	31.19	28	32.81	107
comp03	501	547.55	512	551.77	474
comp04	13	14.48	13	14.55	13
comp05	247	258.71	248	260.81	584
comp06	14	18.13	15	19.00	39
comp07	6	8.45	5	8.06	3
comp08	15	16.84	15	16.74	15
comp09	38	40.35	38	40.48	122
comp10	6	9.74	6	10.39	3
comp11	0	0.00	0	0.00	0
comp12	98	108.65	100	111.35	479
comp13	41	43.65	41	44.16	109
comp14	16	17.65	16	18.03	18
comp15	30	34.81	29	35.13	129
comp16	12	14.35	11	13.97	7
comp17	25	28.13	25	28.29	58
comp18	25	27.84	24	27.52	93
comp19	33	37.52	33	37.23	123
comp20	11	14.29	10	14.03	168
comp21	36	41.23	36	40.81	121

Table 11
Our results on the comp instances for the UD5 formulation.

Instance	us (FBT)		us (F-Race-F2)		Banbara et al. [3]
	best	*avg*	*best*	*avg*	*z*
comp01	11	11.06	11	11.13	45
comp02	135	160.19	139	161.45	714
comp03	140	165.58	140	167.65	523
comp04	56	67.42	57	67.74	215
comp05	568	619.29	587	633.58	2753
comp06	82	102.45	88	102.39	747
comp07	43	55.77	37	53.10	910
comp08	64	72.87	64	72.68	212
comp09	154	166.97	152	165.90	428
comp10	68	86.42	69	88.29	633
comp11	0	0.00	0	0.00	0
comp12	471	515.32	471	522.03	2180
comp13	148	160.39	147	159.32	488
comp14	97	107.06	96	109.39	541
comp15	173	189.84	170	191.65	656
comp16	95	109.26	93	109.58	914
comp17	154	165.19	152	169.00	818
comp18	133	143.19	136	142.65	509
comp19	123	140.52	121	137.19	619
comp20	124	146.10	123	147.61	2045
comp21	151	168.00	145	170.52	651

References

[1] Abdullah S, Turabieh H, McCollum B, McMullan P. A hybrid metaheuristic approach to the university course timetabling problem. J Heuristics 2012;18(1):1–23.
[2] Asín Achá R, Nieuwenhuis R. Curriculum-based course timetabling with SAT and MaxSAT. Ann Oper Res 2014;218(February):71–91.
[3] Banbara M, Soh T, Tamura N, Inoue K, Schaub T. Answer set programming as a modeling language for course timetabling. Theory Pract Log Program 2013;13(4–5):783–98.
[4] Bellio R, Di Gaspero L, Schaerf A. Design and statistical analysis of a hybrid local search algorithm for course timetabling. J Sched 2012;15(1):49–61.
[5] Benjamini Y, Hochberg Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. J R Stat Soc Ser B (Methodol) 1995;57(1):289–300.
[6] Bettinelli A, Cacchiani V, Roberti R, Toth P. An overview of curriculum-based course timetabling. TOP 2015:1–37.
[7] Birattari M, Yuan Z, Balaprakash P, Stützle T. F-Race and iterated F-race: an overview. Berlin: Springer; 2010.
[8] Bonutti A, De Cesco F, Di Gaspero L, Schaerf A. Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. Ann Oper Res 2012;194(1):59–70.
[9] Breiman L. Random forests. Mach Learn 2001;45(1):5–32.
[10] Burke E, Mareček J, Parkes A, Rudová H. A supernodal formulation of vertex colouring with applications in course timetabling. Ann Oper Res 2010;179(1):105–30.
[11] Burke EK, Mareček J, Parkes AJ, Rudová H. A branch-and-cut procedure for the Udine course timetabling problem. Ann Oper Res 2012;194:71–87.
[12] Burke EK, Mareček J, Parkes AJ, Rudová H. Decomposition, reformulation, and diving in university course timetabling. Comput Oper Res 2010;37(3):582–97.
[13] Cacchiani V, Caprara A, Roberti R, Toth P. A new lower bound for curriculum-based course timetabling. Comput Oper Res 2013;40(February (10)):2466–77.
[14] Ceschia S, Di Gaspero L, Schaerf A. Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. Comput Oper Res 2012;39:1615–24.
[15] Di Gaspero L, McCollum B, Schaerf A. The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3). Technical report. Belfast, UK: Queen's University; August 2007.
[16] Geiger MJ. Multi-criteria curriculum-based course timetabling—a comparison of a weighted sum and a reference point based approach. In: Evolutionary multi-criterion optimization. Berlin, Heidelberg: Springer; 2009. p. 290–304.
[17] Hammersley JM, Handscomb DC, Weiss G. Monte Carlo methods. Phys Today 1965;18:55.
[18] Hao J-K, Benlic U. Lower bounds for the ITC-2007 curriculum-based course timetabling problem. Eur J Oper Res 2011;212(3):464–72.
[19] Hastie T, Tibshirani R, Friedman J. The elements of statistical learning. 2nd ed. New York: Springer; 2009.
[20] Hollander M, Wolfe DA, Chicken E. Nonparametric statistical methods. 3rd ed. New York: Wiley; 2014.
[21] Hoos HH. Programming by optimization. Commun ACM 2012;55(2):70–80.
[22] Johnson DS, Aragon CR, McGeoch LA, Schevon C. Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. Oper Res 1989;37(6):865–92.
[23] Kingston JH. Educational timetabling. In: Uyar AS, Ozcan E, Urquhart N, editors, Automated scheduling and planning. Studies in computational intelligence, vol. 505. Berlin Heidelberg; Springer; 2013. p. 91–108.
[24] Kirkpatrick S, Gelatt Jr CD, Vecchi MP. Optimization by simulated annealing. Science 1983;220:671–80.
[25] Lach G, Lübbecke M. Curriculum based course timetabling: new solutions to Udine benchmark instances. Ann Oper Res 2012;194(1):255–72.
[26] Lewis R. A survey of metaheuristic-based techniques for university timetabling problems. OR Spectr 2008;30(1):167–90.
[27] Lewis R, Paechter B, McCollum B. Post enrolment based course timetabling: a description of the problem model used for track two of the second international timetabling competition. Technical report. Wales, UK: Cardiff University; 2007.
[28] Liaw A, Wiener M. Classification and regression by randomForest. R News 2002;2(3):18–22.
[29] Lopes L, Smith-Miles K. Pitfalls in instance generation for Udine timetabling. In: Learning and intelligent optimization (LION4). Berlin, Heidelberg: Springer; 2010. p. 299–302.
[30] Lü Z, Hao J-K. Adaptive tabu search for course timetabling. Eur J Oper Res 2009;200(1):235–44.
[31] Lü Z, Hao J-K, Glover F. Neighborhood analysis: a case study on curriculum-based course timetabling. J Heuristics 2011;17(2):97–118.
[32] McCollum B, Schaerf A, Paechter B, McMullan P, Lewis R, Parkes AJ, et al. Setting the research agenda in automated timetabling: the second international timetabling competition. INFORMS J Comput 2010;22(1):120–30.
[33] Mühlenthaler M, Wanka R. Fairness in academic course timetabling. CoRR abs/1303.2860; 2013.
[34] Müller T. ITC2007 solver description: a hybrid approach. Ann Oper Res 2009;172(1):429–46.
[35] Schaerf A. A survey of automated timetabling. Artif Intell Rev 1999;13(2):87–127.
[36] Urli T. json2run: a tool for experiment design & analysis. CoRR abs/1305.1112; 2013.
[37] Wolpert DH, Macready WG. No free lunch theorems for optimization. IEEE Trans Evolut Comput 1997;1(1):67–82.