Operations Research Perspectives 6 (2019) 100092

Contents lists available at ScienceDirect

Operations Research Perspectives

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

Room usage optimization in timetabling: A case study at Universidade de Lisboa

Alexandre Lemos[*], Francisco S. Melo, Pedro T. Monteiro, Inês Lynce

INESC-ID and Instituto Superior Técnico, Universidade de Lisboa, Rua Alves Redol, 9, Lisboa 1000-029, Portugal

ARTICLE INFO

Keywords:
University timetabling
Room usage optimization
Integer linear programming
Greedy algorithms

ABSTRACT

This paper discusses the problem of room usage optimization for university timetables: given a timetable, we want to optimize the room occupation by determining the events allocated to each room, while ensuring that the rooms have enough capacity to “seat” all people participating in those events. This paper contributes with two approaches to the problem of optimizing the timetable scheduled for each room. The first approach consists of a two-stage Integer Linear Programming (ILP) which applies a lexicographic optimization wherein the goal of the first stage is to maximize the number of students seated and that of the second stage is to optimize the room occupation. This is provably optimal, in both optimization criteria. However, it is computationally demanding, requiring significant computation time for large problems. To address this issue, we propose a second approach, consisting of a greedy algorithm. The algorithm assigns lectures to rooms greedily, according to a specific cost function that seeks to maximize the number of students seated. We show that the proposed cost function guarantees that the greedy algorithm performs within 63% of the total number of students. We apply both algorithms in a case study involving real data from Instituto Superior Técnico (IST), the engineering school from Universidade de Lisboa. Our results confirm that the greedy algorithm is two orders of magnitude faster than ILP when considering large data sets. Comparing the performance of the two methods we observe that the performance of the greedy algorithm, when compared to the ILP-based approach, is within 2% for the number of seated students and 34% for the room occupation. The GRASP algorithm is a good extension of the greedy algorithm, which is able to improve in 12% the quality of the solution (in terms of compactness) without adding significant CPU time. Overall, the two proposed approaches provide significant gains for both optimization criteria when compared to the current hand-made solutions.

1. Introduction

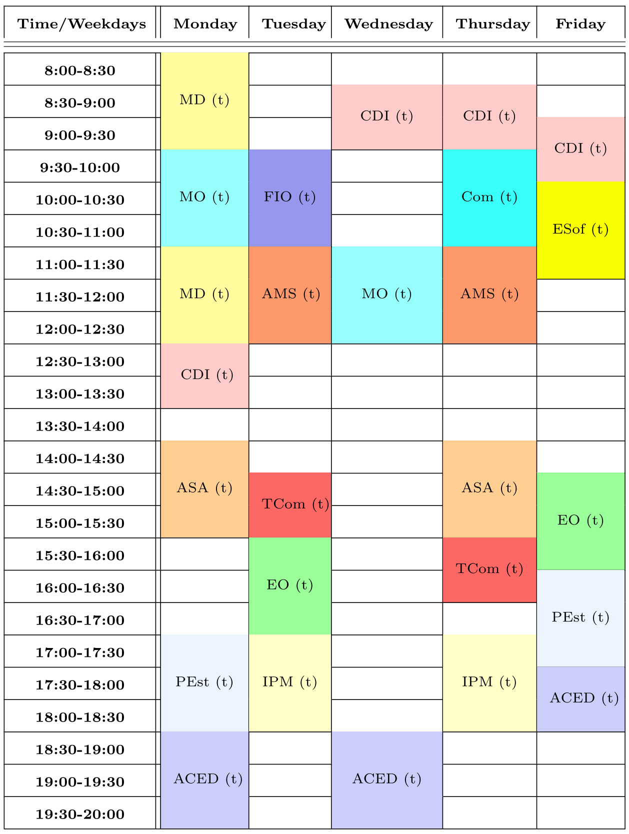
The allocation of events to rooms is an important and complex task. The well-known phrase “space, like time, is money” shows the importance of optimizing the usage of rooms. At Instituto Superior Técnico (IST), the engineering school from Universidade de Lisboa, in Portugal, the allocation of events is still handmade and therefore susceptible to optimization. We expect that the automated optimization of space usage can significantly reduce the number of rooms required, a considerable achievement if we take into account that—at certain time slots—there are no rooms available in the IST campus. Furthermore, having a better optimized distribution of events per rooms makes it easier to add new events without disrupting the existing distribution.

The efficient allocation of events to rooms is only part of the more complex problem of creating timetables. In this work, we do not address the latter—we assume timetables have already been generated—but instead we focus on optimizing the room allocation, given a timetable. The purpose of this work is thus to optimize the room usage while maintaining the original schedule of the events. In practice, the proposed algorithm can only change events from one room to another. Such changes take into account the capacity of the rooms, which should be enough to accommodate the expected number of participants in the events.

As an example of the problem addressed herein, let us consider the timetables for two rooms—referred simply as Room 1 and Room 2—depicted in Fig. 1. For the sake of the example, we assume that both rooms have the same capacity; the events shown in the timetables correspond to lectures of different courses and their names are immaterial for our discussion. The objective is to exchange events between these two rooms to reduce the time gaps between events in the same room without changing the existing timetable (*i.e.*, the time and duration of the existing events). We note that, by maintaining the

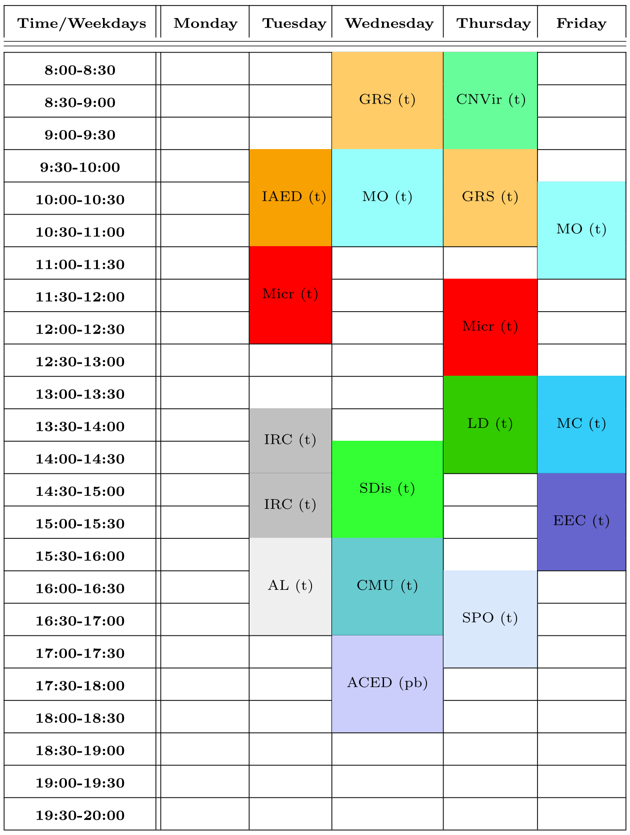
[*] Corresponding author.
E-mail address: alexandre.lemos@tecnico.ulisboa.pt (A. Lemos).

https://doi.org/10.1016/j.orp.2018.100092
Received 18 July 2018; Received in revised form 13 December 2018; Accepted 13 December 2018
Available online 14 December 2018



Time/Weekdays	Monday	Tuesday	Wednesday	Thursday	Friday
8:00-8:30	MD (t)				
8:30-9:00	MD (t)		CDI (t)	CDI (t)	
9:00-9:30	MD (t)		CDI (t)	CDI (t)	CDI (t)
9:30-10:00	MO (t)	FIO (t)		Com (t)	CDI (t)
10:00-10:30	MO (t)	FIO (t)		Com (t)	ESof (t)
10:30-11:00	MO (t)	FIO (t)		Com (t)	ESof (t)
11:00-11:30	MD (t)	AMS (t)	MO (t)	AMS (t)	ESof (t)
11:30-12:00	MD (t)	AMS (t)	MO (t)	AMS (t)	
12:00-12:30	MD (t)	AMS (t)	MO (t)	AMS (t)	
12:30-13:00	CDI (t)				
13:00-13:30	CDI (t)				
13:30-14:00					
14:00-14:30	ASA (t)			ASA (t)	
14:30-15:00	ASA (t)	TCom (t)		ASA (t)	EO (t)
15:00-15:30	ASA (t)	TCom (t)		ASA (t)	EO (t)
15:30-16:00		EO (t)		TCom (t)	EO (t)
16:00-16:30		EO (t)		TCom (t)	PEst (t)
16:30-17:00		EO (t)			PEst (t)
17:00-17:30	PEst (t)	IPM (t)		IPM (t)	PEst (t)
17:30-18:00	PEst (t)	IPM (t)		IPM (t)	ACED (t)
18:00-18:30	PEst (t)	IPM (t)		IPM (t)	ACED (t)
18:30-19:00	ACED (t)		ACED (t)		
19:00-19:30	ACED (t)		ACED (t)		
19:30-20:00	ACED (t)		ACED (t)		

(a) Room 1.



Time/Weekdays	Monday	Tuesday	Wednesday	Thursday	Friday
8:00-8:30			GRS (t)	CNVir (t)	
8:30-9:00			GRS (t)	CNVir (t)	
9:00-9:30			GRS (t)	CNVir (t)	
9:30-10:00		IAED (t)	MO (t)	GRS (t)	
10:00-10:30		IAED (t)	MO (t)	GRS (t)	MO (t)
10:30-11:00		IAED (t)	MO (t)	GRS (t)	MO (t)
11:00-11:30		Micr (t)			MO (t)
11:30-12:00		Micr (t)		Micr (t)	
12:00-12:30		Micr (t)		Micr (t)	
12:30-13:00					
13:00-13:30				LD (t)	MC (t)
13:30-14:00		IRC (t)		LD (t)	MC (t)
14:00-14:30		IRC (t)	SDis (t)	LD (t)	MC (t)
14:30-15:00		IRC (t)	SDis (t)		EEC (t)
15:00-15:30		IRC (t)	SDis (t)		EEC (t)
15:30-16:00		AL (t)	CMU (t)		EEC (t)
16:00-16:30		AL (t)	CMU (t)	SPO (t)	
16:30-17:00		AL (t)	CMU (t)	SPO (t)	
17:00-17:30			ACED (pb)	SPO (t)	
17:30-18:00			ACED (pb)		
18:00-18:30			ACED (pb)		
18:30-19:00					
19:00-19:30					
19:30-20:00					

(b) Room 2.

Fig. 1. Timetables for two rooms.

timetable, any constraints involving the student's curriculum (not having overlaps between courses offered to the same students, for example) are not violated.

Suppose that a new event requires a room for the whole Wednesday afternoon; with the current set-up, it is impossible to allocate such a room. However, it is possible to move some events from Room 2 to Room 1 in order to ensure one free (available) room on Wednesday afternoon. Fig. 2 shows one possible *tighter* timetable.

The contributions of this paper are two-fold: (i) we contribute an ILP formulation of the space allocation problem; and (ii) a greedy algorithm to address that same room allocation problem, with provable performance guarantees. The ILP formulation, by construction, provides an optimal solution to the allocation problem. We show, however, that for large problems the computational requirements of this formulation are significant. The greedy algorithm, while requiring significantly less computational effort, is able to deliver a solution that is provably ε-optimal, where we provide a lower bound for ε. We showcase the application of both approaches in a real-world problem involving University timetables and show that both algorithms solve the problem for rooms and events requiring different capacities.

This paper is organized as follows. Section 2 provides a brief review of the relevant state of the art in the area, where the majority of the work is focused on generating complete timetables. Section 3 formally introduces the problem. Section 4 discusses different metrics to analyze the quality of compact timetables. Section 5 analyses the current hand-made solution. Section 6 describes the ILP implementation to optimize the number of seated students while ensuring a compact timetable. Section 7 presents two greedy algorithms and their theoretical implications. Section 8 discusses the results of the different methods for assigning the lectures of the two campi of IST in the academic year of 2016/2017. The results are also compared with the hand-made solution used in that year. Finally, Section 9 concludes the paper and suggests future research directions.

2. Related work

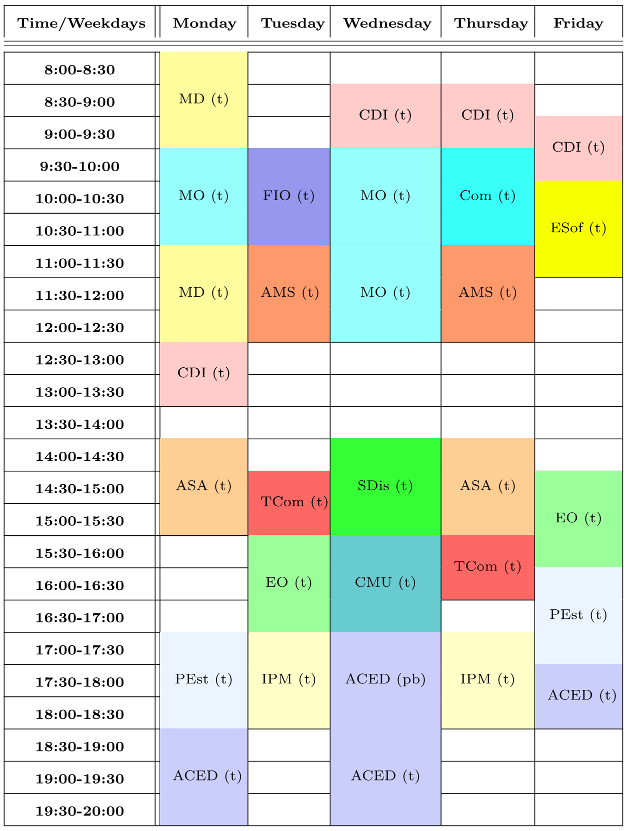
2.1. Timetables

University timetabling problems [1–6] are known to be NP-complete [7] since the search space for possible solutions grows exponentially with the number of events. University timetabling can be classified into two major categories: examination timetabling [8] and course timetabling problems. These categories are characterized along these lines:

- Examination timetabling - focuses on creating the timetables for the examinations. These timetables must ensure that a student can attend examinations for which he is enrolled in.
- Course timetabling:
  - Curriculum-based course timetabling [5] - focuses on creating timetables based on a pre-defined curriculum that the students must follow.
  - Post-enrolments timetabling [4] - deals with creating timetables based on the students enrolments.

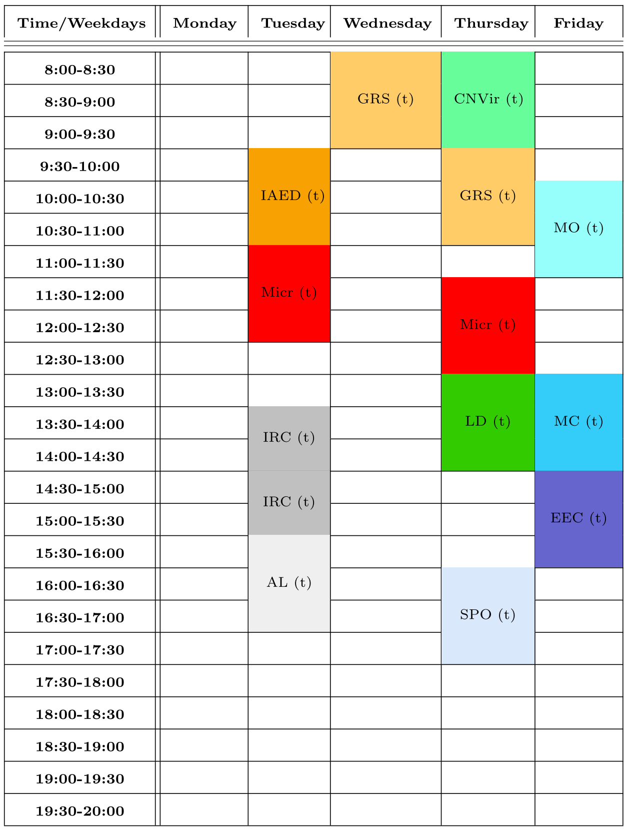
There are many different approaches to solve these problems. For example, Constraint Programming (CP) [6,9], Integer Linear Programming (ILP) [2,3,10,11], multi-agent [12,13] and local search [14–16] approaches have already been successfully applied. The UniTime [17][1] tool is a success case of applying CP techniques to the university timetabling problem. The tool solves both course timetabling and post-

[1] The tool is available on http://unitime.org.



Time/Weekdays	Monday	Tuesday	Wednesday	Thursday	Friday
8:00-8:30	MD (t)				
8:30-9:00	MD (t)		CDI (t)	CDI (t)	
9:00-9:30	MD (t)		CDI (t)	CDI (t)	CDI (t)
9:30-10:00	MO (t)	FIO (t)	MO (t)	Com (t)	CDI (t)
10:00-10:30	MO (t)	FIO (t)	MO (t)	Com (t)	ESof (t)
10:30-11:00	MO (t)	FIO (t)	MO (t)	Com (t)	ESof (t)
11:00-11:30	MD (t)	AMS (t)	MO (t)	AMS (t)	ESof (t)
11:30-12:00	MD (t)	AMS (t)	MO (t)	AMS (t)	
12:00-12:30	MD (t)	AMS (t)	MO (t)	AMS (t)	
12:30-13:00	CDI (t)				
13:00-13:30	CDI (t)				
13:30-14:00					
14:00-14:30	ASA (t)		SDis (t)	ASA (t)	
14:30-15:00	ASA (t)	TCom (t)	SDis (t)	ASA (t)	EO (t)
15:00-15:30	ASA (t)	TCom (t)	SDis (t)	ASA (t)	EO (t)
15:30-16:00		EO (t)	CMU (t)	TCom (t)	EO (t)
16:00-16:30		EO (t)	CMU (t)	TCom (t)	PEst (t)
16:30-17:00		EO (t)	CMU (t)		PEst (t)
17:00-17:30	PEst (t)	IPM (t)	ACED (pb)	IPM (t)	PEst (t)
17:30-18:00	PEst (t)	IPM (t)	ACED (pb)	IPM (t)	ACED (t)
18:00-18:30	PEst (t)	IPM (t)	ACED (pb)	IPM (t)	ACED (t)
18:30-19:00	ACED (t)		ACED (t)		
19:00-19:30	ACED (t)		ACED (t)		
19:30-20:00	ACED (t)		ACED (t)		

(a) Room 1.



Time/Weekdays	Monday	Tuesday	Wednesday	Thursday	Friday
8:00-8:30			GRS (t)	CNVir (t)	
8:30-9:00			GRS (t)	CNVir (t)	
9:00-9:30			GRS (t)	CNVir (t)	
9:30-10:00		IAED (t)		GRS (t)	
10:00-10:30		IAED (t)		GRS (t)	MO (t)
10:30-11:00		IAED (t)		GRS (t)	MO (t)
11:00-11:30		Micr (t)			MO (t)
11:30-12:00		Micr (t)		Micr (t)	
12:00-12:30		Micr (t)		Micr (t)	
12:30-13:00				Micr (t)	
13:00-13:30				LD (t)	MC (t)
13:30-14:00		IRC (t)		LD (t)	MC (t)
14:00-14:30		IRC (t)		LD (t)	MC (t)
14:30-15:00		IRC (t)			EEC (t)
15:00-15:30		IRC (t)			EEC (t)
15:30-16:00		AL (t)			EEC (t)
16:00-16:30		AL (t)		SPO (t)	
16:30-17:00		AL (t)		SPO (t)	
17:00-17:30				SPO (t)	
17:30-18:00					
18:00-18:30					
18:30-19:00					
19:00-19:30					
19:30-20:00					

(b) Room 2.

Fig. 2. Reorganized timetables for the same two rooms, which now include a free room during Wednesday afternoon.

enrolments timetabling. The assignment of lectures to room can be influenced by two constraints: requirements (*e.g.* need of projection support) and capacity. Both constraints are part of the optimization criteria. However, the criteria do not consider room optimization.

The multi-agent [12,13] approach focuses on negotiating the assignment of lectures to rooms and time slots in order to reach a feasible timetable while optimizing the preferences of the teachers. The agents representing the teacher have different ranks and seek to assign their lectures according to their preference. The higher ranking teachers have priority in terms of having their preferences satisfied.

In the literature, there are different views regarding compact timetables: from students timetables [11] to teachers timetables [18,19]. The latter is usually associated to the high school timetabling problem, where the goal is both to avoid idle periods in teachers timetables and to minimize the number of working days.

Burke et al. [11] proposed an ILP based method to solve curriculum-based timetabling. The optimization criteria used considers, among others, the room capacity and curriculum compactness. The objective is to minimize the global number of students not seated and to reduce the number of time gaps between lectures of the same curriculum. Therefore, the focus of the compaction process lies only on the student's timetable. The minimization does not ensure an uniform distribution of the unseated students. The ILP method decomposes the problem into multiple sub-problems where only a part of the optimization criteria are used. These sub-problems can be used to compute bounds in their respective optimization criterion. In the end, the solution to the problem is computed based on the solution of each of the sub-problems.

Vermuyten et al. [3] proposed a two-stage approach to optimize student flows using ILP. The first-stage focuses on assigning lectures to time slots and rooms, and the second focuses on re-assigning rooms to lectures with pre-defined time slots from the first stage. The main optimization criterion of the second-stage is to assign rooms in such a way that congestions are avoided. The ILP implementation also adds some constraints regarding the compaction of the timetables in the perspective of a student. The ILP implementation tries to avoid timetables with two or more hours without lectures for the students. This type of constraint is particularly important for commuting students. However, the proposed method does not reduce the number of isolated lectures (*i.e.* days where a student timetable has only one lecture assigned). This type of decomposition is common since it reduces the problem complexity without loosing any solutions [2]. Another approach [20] follows the same decomposition for examination timetabling: it applies a greedy heuristic to the first task (exams assigned to timeslots) and ILP to the second (exams assigned to rooms).

Beyrouthy et al. [21] studied the utilization of teaching space in order to improve room-size profiles when planning building a campus. The study shows that most rooms have overcapacity (the number of seats of the rooms is larger than the number of students). Furthermore, it was shown that the location of the room has a direct effect on room utilization since both students and teachers prefer certain locations. Beyrouthy et al. [22] proposed methods to split lectures in order to improve the room utilization.

Lindahl et al. [23] studied the impact of the number of time slots allowed, in the quality of the timetable. To this end, linear programming models were developed to solve the timetabling problem with three optimization objectives: number of time slots available; room usage (minimize the number of rooms used); and overall quality. The study of the number of time slots available is particularly important since it is easier to increment the number of time slots when adding new courses than to build more rooms.

Song et al. [14] proposed an iterative algorithm with three stages: initialization, intensification, and diversification to solve the course timetabling problem. The first stage finds a partial feasible solution using a greedy algorithm to allocate the maximum possible number of events. The intensification stage uses a simulated annealing method to find a local optimum. The final stage uses random perturbations (swap of lectures) to improve the solution. The solution found is used as the new starting point of the next iterations.

2.2. Greedy algorithms

Assigning lectures to rooms is a problem that can be solved by a greedy algorithm. Cormen et al. [24] used a simplified version of this problem to illustrate the use of greedy algorithms. The example focuses on allocating the largest amount of activities to a room. For this problem the authors choose the first activity to be assigned based on the ending time, *i.e.* the activity that ends first. This ensures that the room has the largest amount of empty time slots (which is actually the goal). So all activities are sorted by order of ending time. The list of activities are placed in a monotonically increasing order and so the algorithm chooses the first activity to assign. In order to ensure that a solution is found, all activities must start after the last one ends.

Greedy heuristics, such as Greedy Randomized Adaptive Search Procedure (GRASP), have successfully been applied to course timetabling [15,16,25] and examination timetabling [20,26]. The greedy algorithm is guided through a cost function that sums all constraints violated. This method does not ensure any type of confidence in terms of finding an optimal solution. Furthermore, the optimization criterion of this algorithm never considers room optimization.

Greedy algorithm performance

It is possible to compute the approximated value of the optimal solution for a greedy algorithm if the benefit function is sub-modular, positive and monotone. The benefit function describes the advantages of performing an action.

Let us consider C as a finite set. A function f is considered to be monotone if and only if it has the following property:

$$\forall_{A\subseteq C}\, f(A) \leqslant f(C) \tag{1}$$

In order to be considered sub-modular [27], the function f must have the following property:

$$\forall_{A,B\subset C, X\in C\setminus B}\, f(A\cup\{x\}) - f(A) \geqslant f(B\cup\{x\}) - f(B) \tag{2}$$

When one considers a greedy algorithm with a monotone, positive, sub-modular function, the resulting solution is approximated within $1 - e^{-1}$ to the optimal solution [28].

3. Problem description

In this section we formally introduce the problem and the notation used throughout the text.

3.1. Preliminaries

Let us consider a set $R = \{r_1, ..., r_n\}$ of n rooms for which a set $L = \{l_1, ..., l_m\}$ of m lectures has to be scheduled. The lectures can be scheduled in available time slots of a working day $D \in \{1, ..., 5\}$ (1 corresponding to Monday, 2 to Tuesday, and so on). Each day has a set of consecutive working time slots of half an hour, $T \in \{1, ..., 24\}$.[2] Each room r, where $r \in R$, has an ideal capacity, $cap_r > 0$. Each lecture l where $l \in L$ is characterized by:

- Number of enrolled students ($students_l$);
- Working day d_l ($d_l \in D$);
- Subset of sequential time slots $T_l = \{sTime_l, ..., eTime_l\} \subset T$, of size len_l, where the $sTime_l$ is the first time slot of the lecture and $eTime_l$ is the last time slot of the lecture ($sTime_l < eTime_l$).

[2] We consider that the rooms are available for use only between 8am and 8pm, corresponding to a total of 12 h.

As all weeks have basically the same lectures, one can generate the timetables for one week and generalize for the whole semester.

In this problem, we consider that lectures have pre-defined time slots and therefore there is no need to consider curriculum based constraints. The goal is to assign all lectures to rooms.

The pre-defined schedule of a lecture (day and time slots) is represented with an incidence matrix A. $A^l_{d_l,t}$ equals 1 if lecture l is scheduled in the time slot $t \in T_l$ of day d_l and 0 otherwise.

The Boolean variable $x_{l,\,r} \in \{0, 1\}$ is equal to 1 if and only if the lecture $l \in L$ is assigned to the room $r \in R$, and 0 otherwise. A room is considered to be *occupied* in a time slot if and only if a lecture is assigned to that room in that time slot.

The $seated_{l,\,r}$ value corresponds to the number of students seated in case lecture l is assigned to room r. Note that $seated_{l,\,r}$ is given a value even if this lecture l is not assigned to room r. Formally:

$$seated_{l,r} = \begin{cases} students_l & \text{if } cap_r \geq students_l \\ cap_r & otherwise \end{cases} \tag{3}$$

3.2. Constraints

When assigning the lectures to rooms, the constraints considered are as follows:

- *Conflicts*: A room can have at most one lecture scheduled per time slot per day. Formally, for all $r \in R$, $d \in D$ and $t \in T$,

$$\sum_{l\in L} x_{l,r} \times A^l_{d,t} \leq 1. \tag{4}$$

- *Capacity*: The number of attending students must be less or equal to the ideal capacity of the room where the lecture is scheduled. Formally, for all $r \in R$,

$$students_l \times x_{l,r} \leq cap_r \;\forall_{l\in L, r\in R}\,. \tag{5}$$

Besides the constraints above, since the goal of this work is to optimize room usage, it is important to define metrics to correctly evaluate the usage of a room, as detailed in the next section.

4. Optimization: generate compact timetables

4.1. Metrics definitions

Different metrics can be applied to evaluate the usage of a room, for example in terms of the "compactness" of the corresponding occupation timetable. The simplest concept of compactness is, perhaps, the percentage of free space in a schedule. The percentage of free space can be computed as

$$\text{free-space} = \frac{1}{|R||D||T|}\sum_{l\in L}\sum_{t\in T}\sum_{d\in D}\sum_{r\in R} x_{l,r} \times A^l_{d,t},$$

where $|X|$ denotes the cardinality of set X. However, since the average of all free space is constant, as the number of occupied slots is also constant, this metric cannot be used.

One alternative is to calculate the variance in the amount of occupation across rooms. The variance can be computed by:

$$\text{Var} = \frac{\sum_{r\in R}(x_r - \mu)^2}{|R|},$$

where, x_r is the number of free time slots in room r and μ is the average number of free time slots.

The objective would then be to have high variance, as this would mean that some rooms are significantly more occupied than others, instead of uniform occupation of space. However, this metric fails to grasp the difference between a very sparse timetable and a very compact one. A sparse timetable has many more transitions between vacancies and occupations than a compact one.

Such observation suggests that number of transitions — *i.e.*, the number of times a room changes status from vacant to occupied and vice-versa — may be a useful metric. To this end, one can define the auxiliary Boolean variable $c_{l,\ d,\ t,\ r} \in \{0,\ 1\}$ which is equal to 1 if and only if the occupation of the room changes from `occupied` to `free` or vice-verse, and 0 otherwise. And the goal would be to minimize the value of this auxiliary value, *i.e.*:

$$minimize \sum_{r\in R} \sum_{t\in T} \sum_{d\in D} \sum_{l\in L} c_{l,d,t,r} \tag{6}$$

However, this metric considers all holes in a timetable equally bad, which may not be true. For example, a hole corresponding to exactly one time slot (half an hour) is actually worse than a hole of four time slots (two hours) since no event can be scheduled in the former. Such issue can be addressed by applying a weighted metric: the weight is inversely proportional to the size of vacancies, as it is easier to schedule new events in longer vacancies.

4.2. Comparison

While the latter metric may be, perhaps, the one that best captures our desired notion of "compactness", none of the metrics is actually perfect, since none is better across all problems.

For example, consider three rooms, r_1, r_2, r_3, with the same capacity and four lectures with different durations. Further consider that we want to minimize the number of transitions as shown in statement (6). According to this metric, the two timetables shown in Fig. 3a and b are both optimal (with three transitions each). Apart from these there are other optimal timetables that can be obtained.

Consider now that a problem arises and the events in r_1 (*i.e.*, the event l_1) need to be assigned to a different room. One solution for this problem is shown in Fig. 3c. If the timetable in Fig. 3b is the "original" one, it requires less modifications to attain the solution. However, we can equally imagine other cases in which the timetable in Fig. 3a is closer to the final configuration than Fig. 3b.

As another example, consider two new aperiodic events l_5 and l_6, with $sTime_{l_5} = t_2$, $eTime_{l_5} = t_5$, and $sTime_{l_6} = t_1$, $eTime_{l_6} = t_2$, as the respective starting and finishing times. If the original timetable is the one shown in Fig. 3a, then no change to the previously allocated lectures is required. However, if the original timetable is the one shown in Fig. 3b, then lecture l_2 must change from room r_3 to r_2 before adding the new events. As it is impossible to predict future perturbations, it is not guaranteed that, for example, obtaining an empty day is always the best solution. Hence, it is not possible to elect the overall best approach.

5. Instituto superior Técnico case study

In this section, we present the case study of timetables in Instituto Superior Técnico. The current hand-made timetables and their problems are the main motivation for this work. We want to study possible approaches to optimize the current timetables in order to mitigate the problems related to space usage.

Instituto Superior Técnico (IST)[3] is part of Universidade de Lisboa which is the major university in Portugal and one of the largest in Europe. IST is a higher education institute focused on the fields of Architecture, Engineering, Science and Technology and promoting

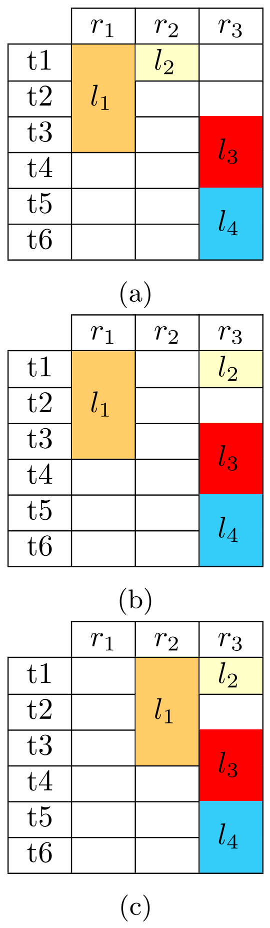


Fig. 3. An optimal timetable (a), an optimal timetable with a free day (b) and an optimal timetable after closing down r_1 (c).

excellence in teaching, research, development and innovation activities.

IST offers 86 different higher education degrees with approximately 11,412 students enrolled (including graduation and post-graduation students) and 1200 teachers. The lectures are scheduled from 8:00 to 20:00, five days a week without mandatory lunch breaks. These lectures can have different durations, from one hour to three hours. IST has two main student campi: Alameda and Taguspark with 94 and 21 rooms, respectively. Each room has a specific capacity. The median capacity of the rooms in Alameda is 58 and in Taguspark is 48. The standard deviation for the room's capacities in Alameda and Taguspark are 31.25 and 64.28, respectively.

Besides offering degrees, IST hosts multiple aperiodic events (*e.g.* student-organized events, workshops, and conferences) on campus. Many times, these events are taking place in the same days as lectures from the different courses. All these new events must be scheduled, when required, in suitable rooms.

Nowadays, the generation of timetables is handmade. The timetables for the current year (2017/2018) were still generated by hand, based on the timetable used in the previous year. This approach reduces the scope of the problem to solve each year since only a small set of events must be changed.

Solving university timetabling manually makes it particularly difficult to optimize space usage. The large variety of rooms available makes choosing the most suitable room for a lecture even a more important decision.

The main problems at IST, in terms of space usage, are (i) the difficult task of scheduling aperiodic events and (ii) closing down a room due to unpredictable problems.

[3] https://tecnico.ulisboa.pt.

Table 1
Data sets characteristics.

	Taguspark		Alameda	
	1st Semester	2nd Semester	1st Semester	2nd Semester
Courses	108	101	1022	1015
Degrees	8		78	
Lectures	290	246	2111	1611
Students	51,523	42,578	283,745	219,217
Hall rooms	5		36	
Rooms	21		94	

When considering these problems, it is important to establish the most compact timetable for each room beforehand, at the beginning of each semester. Typically these problems arise during the semester, and thus it is important to be able to re-optimize the timetables without changing the original timeslots.

Current hand-made timetables

The data used in this work is divided into four data sets corresponding to the classes at Alameda campus and Taguspark campus for both semesters of 2016/2017. The differences between the data sets are summarised in Table 1. Note that, the total number of students is the summation of each enrolment in a lecture. Therefore, this number includes double counting of students. The data sets corresponding to the Alameda campus are larger, in particular for the first semester. The rooms are grouped by capacity and usage. Rooms are usually divided into three types: hall rooms, rooms (R_r) and laboratories. Here, only the first two are considered. Hall rooms can be split in two: large hall rooms (R_{lh}) and hall rooms (R_h). Table 2 shows the summary of the rooms and their average capacities for each campus. R_r is the most heterogeneous set of rooms.

The timetables at IST are mixed between curriculum-based timetabling and post-enrolment timetabling. The undergrads normally have to follow a strict curriculum plan and therefore undergrads timetables are curriculum-based. On the other hand, as students progress in their studies they have more flexibility to choose what course to enroll. In this light, the generation of timetables for these courses can be considered a post-enrolment timetabling problem.

The hand-made timetables for the academic year of 2016/2017 do not seat all the students enrolled in the lectures. The number of students seated and the total number of enrolled students are shown in Table 3. The total number of enrolled students seated may be impossible to achieve due to space restrictions.

6. ILP formulation

An encoding using Integer Linear Programming (ILP) was implemented to solve the problem of assigning lectures to rooms. The ILP implementation is complete which ensures the eventual discovery of the optimal solution to the problem. To solve this problem a two-stage method is used where the goal is to: (i) maximize the number of seated students and (ii) minimize the number of transitions.

Table 2
Characteristics of the rooms: large hall R_{lh}, hall R_h and other rooms R_r.

	Alameda			Taguspark		
Type	R_{lh}	R_h	R_r	R_{lh}	R_h	R_r
Average capacity	148	130	55.77	252	130	36.41
Standard deviation	13.85	0	25.18	0	0	11.43
# of rooms	3	4	113	2	3	48

Table 3
Number of students versus number of seated students in the hand-made solution.

Campus	Alameda		Taguspark	
Semester	1st	2nd	1st	2nd
Total number of seated students	268,668	210,958	51,006	42,286
Total number of students	283,745	219,217	51,523	42,578

6.1. First stage: maximizing the number of students seated

$$\text{maximize:} \sum_{l\in L}\sum_{t_l\in T_l}\sum_{r\in R} seated_{l,r}\times x_{l,r}\times A^{l}_{d_l,t_l} \tag{7}$$

$$\text{subject to:} \sum_{r\in R} x_{l,r}=1 \quad \forall_{l\in L} \tag{8}$$

$$\sum_{l\in L} x_{l,r}\times A^{l}_{d,t}\leq 1 \quad \forall_{r\in R,d\in D,t\in T} \tag{9}$$

$$cap_r \geq x_{l,r}\times(students_l - students_l\times\alpha) \quad \forall_{r\in R,l\in L} \tag{10}$$

In this section we describe our first approach to solve stage (i) of the method.

As previously mentioned, objective function (7) maximizes the number of students seated, where the value of $seated_{l,\ r}$ is defined in (3).

The problem needs two constraints to ensure the correct allocation of the lectures. First, it is necessary to ensure that a lecture is allocated into exactly one room in the predefined time and day (constraint (8)).

Constraint (9) is required to ensure that there is no more than one lecture in each room in a specific slot. Finally, constraint (10) ensures that, in the worse case, only a percentage α of the students enrolled cannot be seated. The reason for this constraint is explained further on.

6.2. Second stage: compactness

The second stage (ii) of the ILP implementation consists on a re-execution of the program with a different objective and adding a new constraint. Now, the goal is to compact the found timetable.

The old maximization statement is replaced by constraint (11) where BEST is the value obtained in the first stage. The new optimization statement (12) minimizes the number of time gaps in a room timetable (independently of the size of the time gap) by counting the number of transitions. The optimization statement (12) is not linear. However, it is easy to convert it to a linear formulation through the use of auxiliary variables and constraints as shown in [29]. The ILOG CPLEX solver [30] automatically transforms this type of statements.

$$\text{subject to:} \sum_{l\in L}\sum_{t_l\in T_l}\sum_{r\in R} seated_{l,r}\times x_{l,r}\times A^{l}_{d_l,t_l}=BEST \tag{11}$$

$$\text{minimize:} \sum_{r\in R}\sum_{d\in D}\sum_{t\in T} c_{l,d,t,r} \tag{12}$$

$$c_{l,d,t,r}=\begin{cases}0 & \text{if} \sum_{l\in L} x_{l,r}\times A^{l}_{d,t-1}=\sum_{l\in L} x_{l,r}\times A^{l}_{d,t}\\ 1 & \text{otherwise}\end{cases} \tag{13}$$

7. Greedy approaches

In this section, the proposed two greedy algorithm are described.

7.1. Greedy algorithm

The algorithm is guided through a cost function. Algorithm 1 shows

```
INPUT: A set of rooms $R$, a set of lectures $L$
OUTPUT: A list of assignments to rooms $Q$
    ▷ $Q$ is composed by pairs $(l, r)$, $l \in L$ and $r \in R$
 1: $alloc \leftarrow 0$ ▷ number of allocated lectures
 2: $allocated \leftarrow \emptyset$ ▷ set of allocated lectures
 3: while $|L| > alloc$ do
 4:     $cost \leftarrow$ MAX_VALUE;
 5:     $lecture \leftarrow -1$; ▷ Lecture ID
 6:     $room \leftarrow -1$; ▷ Room ID
 7:     for each $l \in L$ do
 8:         if $l \notin allocated$ then
 9:             for each $r \in R$ do
10:                 $costT \leftarrow cost(Q_r, l)$;
11:                 if $costT < cost$ then
12:                     $cost \leftarrow costT$;
13:                     $lecture \leftarrow l$;
14:                     $room \leftarrow r$;
15:     if $lecture \neq -1$ then
16:         $Q \leftarrow (l, r)$;
17:         $allocated \leftarrow allocated \cup \{l\}$;
18:     else
19:         break ▷ unable to allocate all lectures
20: end
```

Algorithm 1. Greedy algorithm based on a cost function.

the general idea behind the greedy search implemented. The algorithm ends when all lectures have been allocated. The algorithm chooses the lecture with the smallest cost to allocate at each iteration (lines 7–14). At the end of each iteration, a lecture is assigned and that information is stored in the set *allocated* (line 17) to ensure that the same lecture is not going to be re-assigned in the next iteration (line 8). The cost function originally considered to evaluate a timetable Q for room r was:

$$conflict(Q_r) \times w + transition(Q_r) + idealCAP(Q_r) \tag{14}$$

where the function *conflict* returns the number of overlapping lectures (constraint (4)); the function *transition* returns the number of transitions (computed as shown in constraint (13)) and the function *idealCAP* returns the number of students above the ideal capacity in the allocated lecture. The value of the weight w has to satisfy the following constraint:

$$w \geq 24 \times 5 + \sum_{l \in L} len_l \times students_l. \tag{15}$$

This constraint ensures leaving a lecture without a room is worse than the level of compactness (24*5 is the worse case in terms of transitions) and having students exceeding the ideal capacity. The idea was to guide the search to compact timetables. However, this cost function does not have any type of optimality guarantees. Furthermore, the complexity causes the solution to be worse in the most important aspects: number of conflicts and number of students exceeding the capacity. Therefore, the criterion of compactness was only used in cases where the number of students seated in two different lectures was the same. This change requires modifying Algorithm 1, by the addition of a new *if* for the case where $costT = cost$, this way ensuring that the compaction process only happens when there are two lectures with the same cost. The objective is to have a timetable where all students can have a seat in the lectures. Moreover, it is desirable to guarantee the quality of the solution as discussed further on.

To produce some guarantees in terms of optimality another cost function was used.

$$\sum_{l \in L} \sum_{t \in T_l} \sum_{r \in R} (students_l - seated_{l,r}) \times x_{l,r} \times A^l_{dl,t} \tag{16}$$

This cost expression is simply based on the number of students that could not be seated in this assignment. Algorithm 1 does not allow conflicting assignments and therefore they are not described in the cost function. When considering the slack α the algorithm also does not allow the allocation of lectures to rooms for which constraint (10) is violated.

Theoretical implications. It is possible to define a benefit function which is sub-modular and monotone. When using cost expression (16), the implicit benefit function f is basically the number of students seated per assignment. Formally, the benefit of assigning lecture l is $f(l) = seated_{l,r_l} \times x_{l,r_l} \times A^l_{dl,t}$, where r_l is the room where the lecture was assigned. Assigning a lecture to a room is always positive, as the total number of students with seats increases. Differently, when using function (14), the benefit can be negative since the number of transitions and conflicts are considered. Not considering conflicts, *i.e.* only assigning the lecture to a room if no conflict exists makes function (16) monotone since it has property (1).

Function (16) is sub-modular [27] since in the best case scenario the benefit will always be the same (all students of the lecture can be seated in the room). In the worst case, there is no room large enough and so the benefit will be smaller. Recall property (2), where C is the set of unassigned lectures and $A, B \subset C$. The way the search is performed, the smaller set A will always have a larger number of rooms available than B. When adding x to A, the algorithm will choose the room that maximizes the number of students seated which will be at least as good as the assignment to B (as B has a smaller number of rooms available but all rooms available in B are also available in A). This way we can see that property (2) is ensured.

To sum up, function (16) was constructed to be a positive, monotone and sub-modular function. Therefore, the solution will be within 63% of the optimal, in terms of students seated [28].

7.2. Greedy randomized adaptive search procedure

The Greedy Randomized Adaptive Search Procedure (GRASP) shown in Algorithm 2 has two main stages: (i) a random greedy algorithm (line 3) and (ii) a local search (line 4). As suggested in [26], we used two stages to optimize different criteria similar to the process used in our ILP approach. The ending condition of the algorithm is either the number of iterations (line 2) or the time limit (which ever comes first). In the end of each iteration the algorithm stores the new solution if and only if it is better than the one found before (line 5).

7.2.1. The random greedy procedure

The goal of the greedy algorithm is to maximize the number of students seated and it is guided by the cost function expressed in (16). The degree of randomness can be customized by changing the size of the Restricted Candidate List (RCL). The RCL is filled with lectures which have the best cost at the time. The allocations are chosen randomly from the RCL. The *size* 0 represents a greedy algorithm in its pure

```
INPUT: Maximum number of iterations ($max\_it$), size of RCL ($size$), a set of
    lectures $L$ and a set of rooms $R$.
OUTPUT: Best solution ($best$).
1: $best \leftarrow \emptyset$ ▷ current best solution
2: for $i = 0, \ldots, max\_it$ do
3:     $solution \leftarrow greedyRandom(size, L, R)$;
4:     $solution \leftarrow Local(solution)$;
5:     $best \leftarrow UpdateSolution(solution, best)$;
6: end
```

Algorithm 2. GRASP algorithm.

form (without randomness) and 1 represents a completely random algorithm. The process ends when a complete solution is found which does not violate any hard constraints.

7.2.2. Local search

The goal of the local search is to minimize the number of transitions without deteriorating the quality of the solution in terms of seated students. This is achieved by swapping lectures between rooms where the number of students seated increases or stays constant. Swapping can never deteriorate the quality of the current solution. This stage ends when no more possible swaps can take place without deteriorating the quality of the solution.

8. Experimental evaluation

In this section, the results for the different methods are examined. The advantages and shortcomings of the methods are also discussed.

8.1. Experimental setup

The algorithms described above are implemented in Java and are available from https://github.com/ADDALemos/Compacter. The Pentaho Data Integration (PDI)[4] software, which was used to transform and clean the data is also available from the same link. The data and the script to generate the plots shown in this paper are also available.

The data used to test the system was obtained through the FenixEdu™public API[5] system which is in use at IST. The data was profiled, transformed and cleaned using PDI which is an extract-transform-load graphical tool. The data had two main data quality problems: duplicates and incorrect values of capacities on the rooms (*e.g.* room with capacity 1). The rooms identified with problems were analyzed and corrected. In most cases, these rooms are offices and thus not useful when solving the timetabling problem.

The tests were executed using the *runsolver* tool [31] with a time out of 6000 seconds and a limit of 3 Gb of memory. The GRASP algorithm was executed with maximum number of iterations of 1000 (the same as in [25]). The GRASP algorithm was tested with different sizes of RCL. The solution with best results in terms of quality is the one with a RCL size of 60% of the total number of lectures. This coincides with the value used in [25] for the complete university timetabling problem. The implementation was executed on a computer running *Ubuntu 14*, with 24 CPUs at 2.6 GHz and 64 Gb of RAM. The ILP model was implemented using the library of ILOG CPLEX (version 12.7.1.0) [30].

8.2. Results & discussion

In this section, experimental results are presented and discussed.

8.2.1. Problem decomposition

To reduce the size of the problem, one can separate the problem into sub-problems. In this case the problem can be decomposed into five sub-problems (one per each day of the week). Decomposing the problem in week days does not remove any possible solutions, since all lectures are already scheduled. This separation is particularly interesting when the data set is large and it is difficult to solve the data set as a whole. In this work, this technique was applied to solve the Alameda data sets when solving with the ILP approach. In the future, the solution can also be used to benefit from parallelization.

8.2.2. Number of attending students

Another issue to discuss is whether the constraint "the number of attending students must be smaller or equal to the capacity of the room" should be considered *hard* or *soft*.

Ideally, all students should be allowed to have a seat in a lecture for which they are enrolled. In practice not all students will actually attend all the lectures. Furthermore, it may be impossible (due to space restrictions) to find rooms with sufficient capacity for all lectures. In the literature, the constraint is usually considered *hard* [4,12,13] in particular when solving post-enrolment timetabling [4,14] since the students choose the course they really want to attend. However, when solving curriculum-based timetabling [5,9] it is common to consider the constraint as *soft*. At IST, as previously mentioned, both types of timetables exist. Undergrads usually have a specified curriculum to follow but, as they progress in the study, the more flexibility they have in the curriculum. Although ideally this constraint should be considered as *hard*, independently of the type of timetables, in practice this may be impossible. The ILP implementation shows that no feasible solution exists when considering the constraint as *hard*.

Table 4 shows the results for the different methods (greedy, ILP, GRASP and hand-made) when maximizing the number of seated students.

The ILP implementation (4)-(9), maximizing the number of students seated, found the optimal value for all data sets. It is possible to see that it is inevitable to allocate some lectures to rooms which do not have the required capacity. Table 4 shows in the column "Optimal", the optimal value in terms of the number of students seated for all data sets, and in column "Total # students" the total number of students that should be seated. With these results and the total number of students it is possible to see that around 2% of the global number of students cannot be seated. This means that, with the current lectures and rooms, it is impossible to seat all enrolled students for all lectures. Note that to obtain the optimal values it was necessary to use the decomposition discussed above (on weekdays) to avoid exceeding the memory limit, for the Alameda data sets. The hand-made timetables for this year leave more students without a seat. Table 4 shows the global number of students seated for the hand-made solution (column Result) and distance to the optimal value in percentage (column Difference). The distance is computed based on the optimal value obtained by the ILP implementation.

The greedy algorithm using the cost function (16) was the only greedy algorithm that obtained a feasible solution. The CPU time of the algorithm for the complete data set is, on average, only 1 second (even for Alameda) making it unnecessary to apply the decomposition.

The solution obtained by the greedy algorithm has only 10% of students without seats. The greedy algorithm is not able to find the optimal value. However, the solutions found are very similar as they are on average less than 2% from the optimal. The CPU time for the greedy algorithm is significantly smaller. This fact has particular importance when considering larger data sets. The greedy algorithm does not require any decomposition into sub-problems as it can solve the complete problem in less than 1 second. The greedy algorithm is two orders of magnitude faster than the ILP implementation (with the decomposition). This shows that ILP does not scale in terms of memory and time.

The GRASP algorithm lies in middle of the results in terms of quality and CPU time. GRASP does not provide any type of quality assurance unlike the greedy and ILP approaches. However, the quality of the solution does improve. The GRASP algorithm improves the quality of the solution but still falls short of the optimal. The randomness and the local search stage improves the quality in terms of seating students.

When comparing with the simple greedy approach, GRASP improves only half percent in terms of the number of students seated. However, this improvement comes with a cost since the GRASP algorithm is worse in terms of CPU time. Note that, both greedy algorithms optimize both criteria at the same time.

In the GRASP approach, most of the time is spent in the second stage of the algorithm. On average the algorithm spends more than 80% of its CPU time on the local search stage. In all the Alameda instances the time limit occurred before reaching the final number of iterations.

[4] http://community.pentaho.com/.

[5] http://fenixedu.org/dev/api/.

Table 4
Comparison of greedy, ILP, Hand-made approaches in terms of global seated students. Result represents the best solution found by the algorithm for each data set. The optimal value was the one obtained by ILP. The CPU time for the decomposed problems corresponds to the sum of the CPU times of all sub-problems.

ILP							
		Decomposition	Time (s)	Result (# students seated)	Difference (%)	Optimal	Total # students
Alameda	1st	Week days	3904.59	281,080	0	281,080	283745
	2nd	Week days	2681.26	216,600	0	216,600	219217
Taguspark	1st	No	15	51,427	0	51,427	51523
	2nd	No	10	42,512	0	42,512	42578
Greedy							
Alameda	1st	No	1	277,422	1.3	281,080	283745
	2nd	No	1	213,731	1.32	216,600	219217
Taguspark	1st	No	1	50,698	1.41	51,427	51523
	2nd	No	1	42,024	1.14	42,512	42578
GRASP							
Alameda	1st	No	3000	278596	0.8	281080	283745
	2nd	No	3000	211612	2.3	216600	219217
Alameda	1st	No	6000	278596	0.8	281080	51523
	2nd	No	6000	214050	1.1	216600	42578
Taguspark	1st	No	2387	51213	0.6	51427	51523
	2nd	No	1439	42341	0.4	42512	42578
Hand-made							
Alameda	1st	N/A	N/A	268,668	4.41	281,080	283745
	2nd	N/A	N/A	210,958	2.6	216,600	219217
Taguspark	1st	N/A	N/A	51,006	0.81	51,427	51523
	2nd	N/A	N/A	42,286	0.53	42,512	42578

However, in our case the quality of the solution converges before the limit. Therefore, one can reduce the number of iterations without losing quality.

Fig. 4 shows the evolution of the quality, in terms of students seated, of current best solution for Alameda data sets as a function of the number of iterations. The values shown in Fig. 4 represent the quality of the stored solution in a given iteration. The quality for Alameda 2nd semester does not improve significantly over time. However, the quality of the original solution is closer to the optimal than for Alameda 1st semester. This can be explained by the fact that this instance has a smaller number of courses and lectures.

Furthermore, one can see, in Fig. 4, that after 300 iterations the solution quality is stable. Table 4 shows the quality of the results after 3000 seconds and 6000 seconds (time limit) for the Alameda data sets.

From these results, we can conclude that the constraint “the number of attending students must be smaller or equal to the capacity of the room” must be considered a *soft* constraint.

8.2.3. Slack on the number of attending students

Maximizing the number of students seated may not be the best approach since all students are considered equally independently of the size and type of lecture. For example, it is worse not seating 3 students in a practical lecture with 20 students than not seating 3 students in a

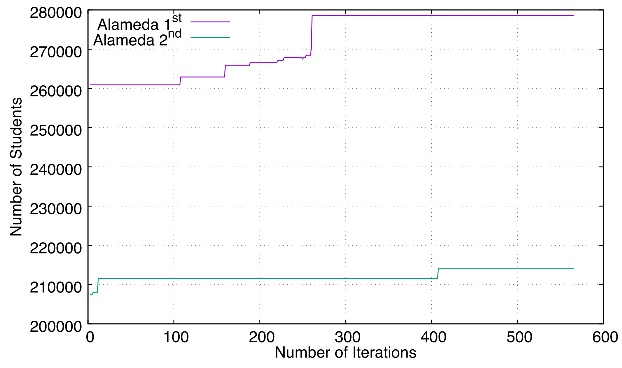


Fig. 4. The evolution of the quality, in terms of students seated, of current best solution found by GRASP, for Alameda data sets.

theoretical lecture with 100 students enrolled. For this reason, we added constraint (10) of the formulation to ensure a more even distribution of students which are above the ideal capacity.

Adding constraint (10) slack on the number of attending students, does not compromise the CPU time of the first stage of the ILP implementation (number of seated students). Naturally, in some cases the global quality of the solution may be worse (*i.e.* more students without seat). However, the distribution should be better since the percentage of students above the ideal capacity per room is smaller. Interestingly, only when solving the sub-problem for Monday on Alameda 1st semester the optimal was below the known optimal (only 1% worse).

α can never be below 27% in order to find feasible solutions in Alameda 1st semester. However, in the 2nd semester the value of α rises for a staggering 35%. For Taguspark, it is possible to impose a α of 10% in the 1st semester. The results are summarized in the Table 5. The lowest values of α were obtained by checking all the possible values until finding a feasible solution (starting with $\alpha = 0$). However, these results only show the lower value of α. The lower value may be caused by a single overbooked lecture.

Therefore, a more detailed analyses of the results was performed. Figs. 5, 6, 7 and 8 show the cumulative distribution of the number of slots with the number of students enrolled above the ideal capacity as a function of the percentage of students above the ideal capacity. In these figures one can extract the number of time slots with at-most α of overbooking. It can be observed that as the percentage rises the number of slots decreases. In other words, it is more common to have rooms with a small percentage of students above the ideal capacity. For example, in Fig. 7 (Taguspark 1st semester) when considering the ILP approach there are only a small number of slots with the percentage of students above 30% of the ideal capacity. In fact, there are only 3 time slots (which represent only one lecture) in this situation. Ignoring this lecture, the α value could be lower than 5%. Interestingly, the number of slots with overbooking is higher in the 1st semester in both campi. This result is expected since in both campi the 1st semester has a larger number of students enrolled.

To sum up, we have observed that high values of slack are caused by a small number of lectures.

Table 5
Minimal value of slack necessary to find a feasible solution for the Alameda and Taguspark data sets.

		Weekday				
		Monday (%)	Tuesday (%)	Wednesday (%)	Thursday (%)	Friday (%)
Alameda	1st	35	27	27	27	27
	2nd	35	35	35	35	12
Taguspark	1st	33				
	2nd	10				

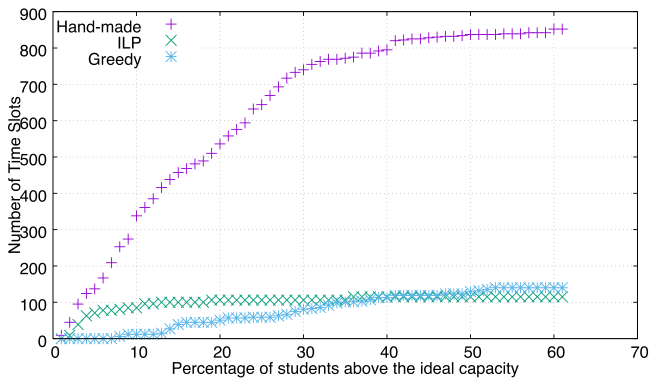


Fig. 5. The cumulative distribution of slots with the number of students enrolled above the ideal capacity as a function of the percentage of students above the ideal capacity for Alameda 1st semester.

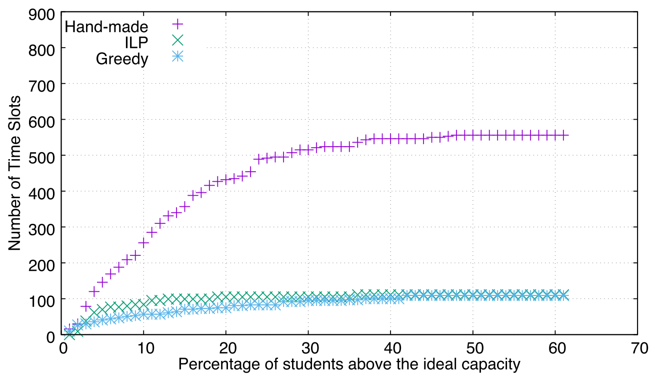


Fig. 6. The cumulative distribution of slots with the number of students enrolled above the ideal capacity as a function of the percentage of students above the ideal capacity for Alameda 2nd semester.

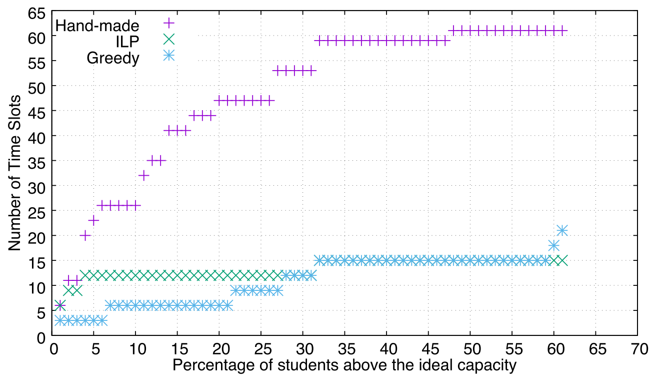


Fig. 7. The cumulative distribution of slots with the number of students enrolled above the ideal capacity as a function of the percentage of students above the ideal capacity for Taguspark 1st semester.

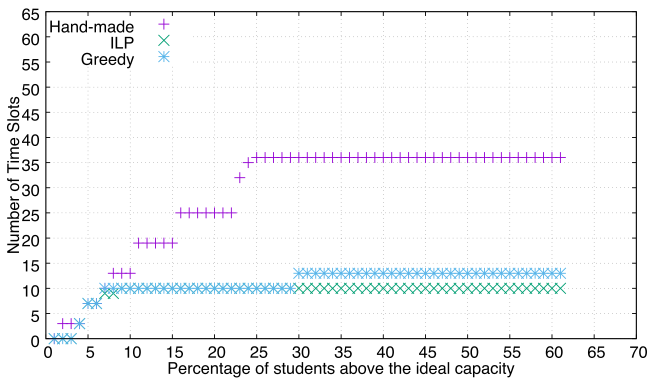


Fig. 8. The cumulative distribution of slots with the number of students enrolled above the ideal capacity as a function of the percentage of students above the ideal capacity for Taguspark 2nd semester.

8.2.4. Overbooking

The results presented above clearly show that overbooking is a reality in these data sets. As such, the level of overbooking can be a metric to evaluate the quality of the solution. The comparison between the ILP and the hand-made approaches are shown in Figs. 5–8 for Alameda and Taguspark data sets. The hand-made solution forces more rooms to be overbooked. For example, in Fig. 7 (Taguspark 1st semester) there are 2 slots with 48% of students above the ideal capacity in the hand-made solution. However, the solution found by the ILP implementation does not require any room to have these percentages. In the Alameda data sets, the ILP implementation found a solution with the highest percentage being 35% which is a clear improvement. The hand-made solution has rooms with more than 50% of students above the ideal capacity. Furthermore, in Alameda 1st semester there are only 112 slots with overbooking with the ILP implementation, versus 852 slots with overbooking in the hand-made solution (Fig. 5).

The greedy algorithm is, once again, a bit worse in terms of the quality of the results since it has rooms with a larger percentage of students above the ideal capacity. It is important to remember that the algorithm deals with this constraint as it deals with conflicts; it does not allow the assignment of lectures to rooms above a certain threshold α. When comparing the results with the ones found by the ILP implementation, one can see that the threshold required to find a feasible solution is higher. However, the number of slots with overbooking is similar. When analyzing Fig. 8, corresponding to Taguspark 2nd semester, one can see that the difference between Greedy and ILP approaches is small. In fact, the Greedy approach has 13 time slots with overbooking versus only 10 in the ILP approach. This difference, in fact, corresponds to a single lecture. However, this lecture has 30% of students above the ideal capacity. On the other hand, in the data set of Alameda 2nd semester (Fig. 6) the ILP implementation has 2 more time slots with overbooking than the greedy algorithm. This fact can be explained by an imposed lower value to α. The value of α for the ILP is 35% which is smaller than the 42% used for the greedy algorithm.

When comparing the results for the greedy algorithm with the hand-made solution, one can see that the improvement is significant in both the number of overbooked slots and percentage of students who cannot be seated in those slots. For Alameda 1st semester, the greedy algorithm only has 140 slots with overbooking versus 852 slots in the hand-made solution. Moreover, the lower value of *alpha* of the hand-made solution is 60% versus 50% obtained by the the greedy algorithm.

Some of the assignments that cause the highest number of students above the ideal capacity in the hand-made solution could eventually be solved by simple reassignments of lectures. However, it is possible that the assignment of certain lectures to specific rooms without the proper capacity may be due to empirical knowledge; some teachers may justifiably prefer specifics rooms, even though not all students can be seated. Furthermore, it could also mean that in practice not all students will actually attend the lecture. Not considering this criterion could cause students that actually attend lectures not to be seated in order to seat students which may never attend. It should also be noted that the set of overbooked lectures obtained by the ILP and greedy algorithms are actually a disjoint set of the set of overbooked lectures found in the hand-made solution. Thus, assuming there is a reason for assigning these rooms, we have forced these lectures to be assigned to the preferred rooms. The addition of this criterion does not change the CPU time. As expected, the overall number of unseated students is higher. When forcing these assignments, also as expected, the set of overbooked lectures is the same for all algorithms. Even though they are now equal in terms of students seated, the ILP and greedy solutions are better in terms of compactness. Table 6 summarizes the number of seated students when allowing the ILP approach to keep the same room for the overbooked lectures as in the hand-made solution. These results are optimal, for each instance, and were obtained using the ILP approach.

8.2.5. *Compactness process*

In terms of compactness, the greedy algorithm only performs the compaction procedure in tie-breaking scenarios. In other words, the compaction is only executed when allocating lectures with the same gain in terms of seated students. Nevertheless, the results are an improvement when comparing them with the hand-made solutions used in IST, even though the values found by the greedy algorithm are not optimal. Table 7 compares the values obtained by the ILP implementation, the greedy algorithm and the hand-made solution. The values represent the number of transitions from free to occupied slots and vice-verse.

The ILP implementation finds the optimal solution for the Taguspark data sets although it requires significantly more CPU time. The number of transitions obtained is higher when using the greedy algorithm. Nevertheless, the results are close to the optimal found by the ILP implementation. The greedy algorithm finds a solution which is 1.2x worse than the optimal.

The ILP implementation was not able to compact all the allocations when considering the decomposed data sets from Alameda within a predefined time-frame. In the worst case, three days of CPU time were spent to find the optimal solution.

Overall, the number of transitions of the greedy implementation is 1.4x and 1.5x higher than the optimal for the Alameda 1st semester and 2nd semester, respectively. The number of days spent for each weekday for the Alameda data sets is shown in Table 8.

Once again the quality of the solution found by the GRASP algorithm lies in the middle of the ILP and greedy approaches. Only the local search stage of GRASP optimizes the quality in terms of transitions. When comparing GRASP with the simple greedy algorithm, the improvement is more significant when considering the number of transitions. In this optimization criterion GRASP is able, on average, to reduce the distance to the optimal value in 12%. The summary of the results are shown in Table 7. Nevertheless, even with fewer iterations the algorithm is considerably worse in terms of CPU time.

Fig. 9 shows the evolution of the quality, in terms of transitions, of the current best solution for Alameda data sets as function of the number of transitions. We conclude that, the algorithm converges in more or less the same number of iterations for both optimization criteria (Fig. 4). Considering this optimization criterion, the first solution for the Alameda 2nd semester has a larger distance to the optimal than when considering the number of students as an optimization criterion. Nevertheless, the Alameda 2nd semester instance has a smaller number of transitions which is normal since it has fewer lectures.

8.2.6. *Time limit for the ILP compactness process*

The greatest weakness of the ILP implementation is the large CPU time required, in particular in the second stage. One approach to reduce the CPU time lies on warm-starting the second stage with the solution from the first stage. However, this process, on average did not improve the CPU time. We believe that the improvement really depends on how close the solution found in the 1st stage is to an optimal solution of the 2nd stage. In this case, the solution, of most instances, for the 1st stage is not close to an optimal solution for the 2nd stage.

In the second stage, most of the time spent by the ILP solver is not improving the quality of the solution. The solver is actually proving optimality. In practice, the trade-off between optimality and CPU time may sometimes lead to a shorter CPU time.

To this end, a study on the evolution of the quality of the solutions over time for the Alameda data sets (which are the most complex) was performed. Figs. 10 and 11 show the evolution for each weekday for the 1st and 2nd semesters, respectively. Both graphs have logarithm scale and shown the time in seconds. The large round grey symbols mark the reaching of the optimal solution (without proving it). The large green triangle symbols mark the time for proving the optimal solution. We conclude that, in most of the cases, the time spent after 6000 seconds produces small changes in the quality of the solution. If we consider the 6000 seconds as the time limit we can see that for the 2nd semester most of the data sets had already their optimal value found, even though it was not yet proven.

As one can see, the evolution shown in Figs. 10 and 11 is not smooth, but rather exhibits some *jumps*. These *jumps* correspond to a restart on the execution of the solver. Every time a *jump* occurs, the solution found is significantly improved. Therefore, one can try to produce more *jumps*. The ILP implementation was rerun with the `RepeatPresolve` flag with value 3 (re-presolve with cuts and allow new root cuts). The results for Alameda data sets are shown in Figs. 12 and 13. The optimal solution is always found faster with this configuration. Nevertheless, the solver still spends most of the CPU time to prove optimality. When solving most of the instances the solver spends 50% or more time proving the optimality. In the worst case, for the data set of Thursday Alameda 2nd semester, the CPU time spent on proving optimality is 92% of the total CPU time.

Table 6
The maximum number of seated students, obtained using the ILP approach, when pre-assigning the overbooked lectures present in the hand-made solution.

Campus	Working day	Semester			
		1st		2nd	
	(Mon to Fri)	Optimal	# Students	Optimal	# Students
Taguspark	All	49,038	49,555	40,231	40,511
Alameda	Monday	52,933	55,649	48,808	50,905
	Tuesday	55,509	59,228	42,609	44,457
	Wednesday	48,465	50,393	31,389	32,293
	Thursday	53,638	57,168	27,528	28,530
	Friday	40,657	42,957	44,978	47,339

9. Conclusion and future work

This paper discusses the problem of room usage optimization for university timetables. In particular, the methods proposed optimize the room occupation by determining where to allocate events with predefined time slots. This optimization process must ensure that the

Table 7
Compaction results in terms of number of transitions for the greedy algorithm, GRASP, ILP before and after the compactness optimization. The ILP finds the optimal solution to the Taguspark data sets within the time limit. The time spent just in compaction routine is also shown. The cells highlighted in grey represent values found through decomposition.

		Hand-made	Greedy	ILP			GRASP
		# Trans.	# Trans.	# Trans. before comp.	# Trans. after comp.	Extra time (s)	# Trans.
Alameda	1st	2148	1242	2068	937	8.4×10^5	1118
	2nd	1945	1030	2001	685	4.2×10^5	944
Taguspark	1st	383	279	369	217	1.2×10^2	238
	2nd	330	241	307	193	3.8×10	210

Table 8
ILP compactness results in terms of number transitions and CPU time (in days), for the Alameda data sets.

Weekday		Monday		Tuesday		Wednesday		Thursday		Friday	
Results		Time (d)	# Trans	Time (d)	# Trans	Time (d)	# Trans	Time (d)	# Trans	Time (d)	# Trans
Alameda	1	0.8	146	1.6	272	3.5	145	3	202	0.7	148
	2	0.5	155	0.7	148	2.5	123	1	122	0.4	137

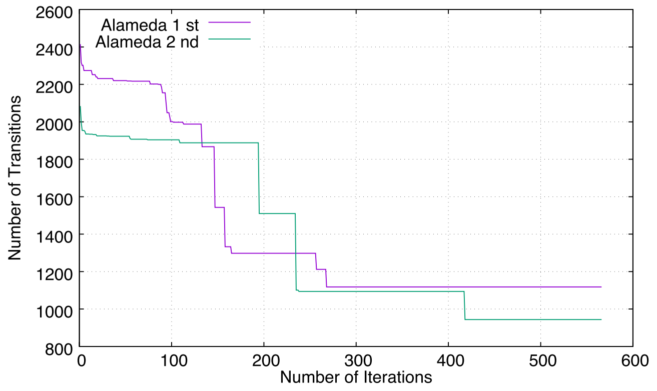


Fig. 9. The evolution of the quality, in terms of transitions seated, of current best solution found by GRASP, for Alameda data sets.

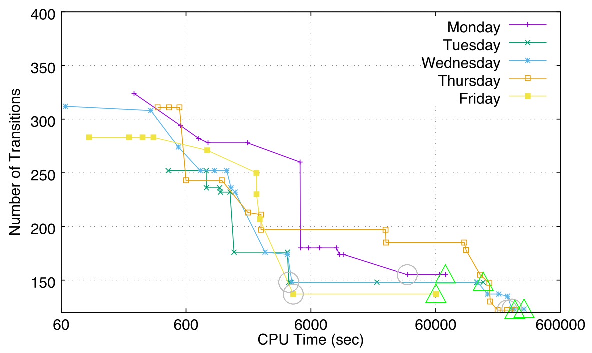


Fig. 10. The evolution of the number of transitions over time in seconds (log scale), for the Alameda 1st semester. The grey circle and the green triangle symbols mark the finding of an optimal value and the proving of optimality, respectively. The results were obtained by the execution of CPLEX with the default configurations. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

rooms have enough capacity to seat all people participating in those events. In this work, we propose two different approaches to solve the above-mentioned problem. Both approaches were successfully tested with data sets from both campi of IST in the academic year of 2016/2017.

First, we propose a two-stage ILP implementation to allocate lectures to rooms while optimizing the room usage. The first stage focuses

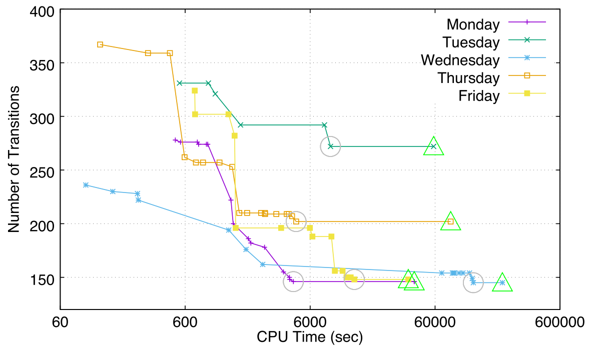


Fig. 11. The evolution of the number of transitions over time in seconds (log scale), for the Alameda 2nd semester. The grey circle and the green triangle symbols mark the finding of an optimal value and the proving of optimality, respectively. The results were obtained by the execution of CPLEX with the default configurations. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

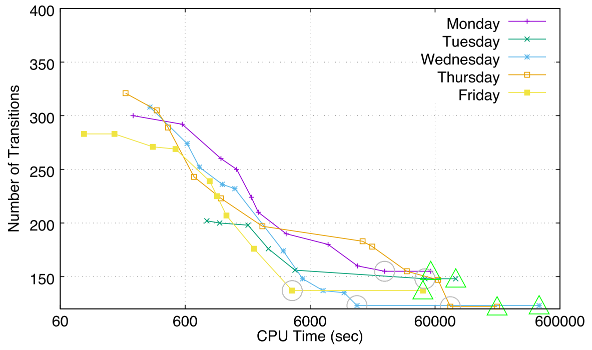


Fig. 12. The evolution of the number of transitions over time in seconds (log scale), for the Alameda 1st semester. The grey circle and the green triangle symbols mark the finding of an optimal value and the proving of optimality, respectively. The results were obtained by the execution of CPLEX configured to re-apply presolve with cuts and allow new root cuts. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

on allocating lectures to rooms in order to maximize the number of students seated. To ensure fairness, we added a slack to the capacity of each room before maximizing the number of students seated. The second stage focuses on the minimization of the number of transitions from free to occupied (and vice-versa) for each room. The ILP

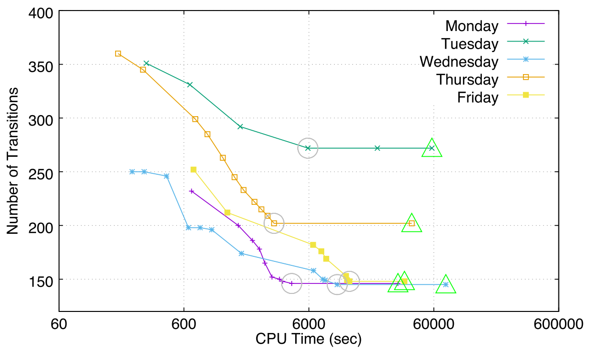


Fig. 13. The evolution of the number of transitions over time in seconds (log scale), for the Alameda 2nd semester. The grey circle and the green triangle symbols mark the finding of an optimal value and the proving of optimality, respectively. The results were obtained by the execution of CPLEX configured to re-apply presolve with cuts and allow new root cuts. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

implementation finds the optimal solution for all data sets tested. However, it requires some decomposition in order to solve all data sets within the time and memory constraints.

Even with the decomposition in working days the ILP implementation has a significantly long CPU time. The execution is particularly large for the second-stage. To reduce this problem, we proposed a greedy algorithm. The greedy algorithm does not find an optimal solution for the data sets but, as expected, is significantly faster. The cost function allows us to ensure that the solution is within 63% of the optimal value (when considering the number of students seated as an optimization criterion). In practice, the results are much closer to the ones found by the ILP implementation. The proposed GRASP algorithm is a good extension of the greedy algorithm which is able to improve the quality of the solution without adding much CPU time. However, this algorithm, does not give any guarantees in terms of the quality of the solution found.

The two approaches proposed in this paper improve the quality of the hand-made solution for either optimization criteria.

As future work, we propose extending the methods in order to solve Minimal Perturbation Problems. In particular, re-assigning lectures to rooms after a perturbation occurs maintaining the timetables stability. Moreover, one can use domain knowledge to improve the CPU time of the ILP implementation (similar to the decomposition proposed in this paper). In relation with the greedy algorithm proposed, it may be possible to improve the quality of the solution by searching neighborhood structures. The application of this local search algorithm does not remove the proven guarantees associated with the chosen cost function.

Acknowledgments

The authors would like to thank the reviewers for their helpful comments and suggestions that contributed to an improved manuscript. This work was supported by Universidade de Lisboa, Instituto Superior Técnico and Departamento de Engenharia Informática (DEI) and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID multi-annual funding).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.orp.2018.100092.

References

[1] McCollum B. University timetabling: bridging the gap between research and practice. Proceedings of the 5th international conference on the practice and theory of automated timetabling (PATAT). Springer; 2006. p. 15–35.
[2] Lach G, Lübbecke ME. Optimal university course timetables and the partial transversal polytope. Experimental algorithms, 7th international workshop. 2008. p. 235–48. https://doi.org/10.1007/978-3-540-68552-4_18.
[3] Vermuyten H, Lemmens S, Marques I, Beliën J. Developing compact course timetables with optimized student flows. Eur J Oper Res 2016;251(2):651–61. https://doi.org/10.1016/j.ejor.2015.11.028.
[4] Lewis R, Paechter B, McCollum B. Post enrolment based course timetabling: adescription of the problem model used for track two of the second international timetabling competition. Cardiff Business School; 2007. Cardiff Working Papers in Accounting and Finance A2007/3
[5] Di Gaspero L, Schaerf A, McCollum B. The second international timetabling competition (itc-2007): curriculum-based course timetabling (track 3). Tech. Rep.. Queen's University; 2007.
[6] Müller T. Constraint-based timetabling. Charles University in Prague Faculty of Mathematics and Physics; 2005. PhD dissertation.
[7] Even S, Itai A, Shamir A. On the complexity of timetable and multicommodity flow problems. Soc Ind Appl MathSIAM J Comput 1976;5(4):691–703.
[8] Müller T. Itc2007 solver description: a hybrid approach. Ann Oper Res 2009;172(1):429.
[9] Banbara M, Inoue K, Kaufmann B, Schaub T, Soh T, Tamura N, et al. teaspoon: solving the curriculum-based course timetabling problems with answer set programming. Proceeding of 11th international conference of the practice and theory of automated timetabling PATAT. 2016. p. 13–32.
[10] Cacchiani V, Caprara A, Roberti R, Toth P. A new lower bound for curriculum-based course timetabling. Comput Oper Res 2013;40(10):2466–77. https://doi.org/10.1016/j.cor.2013.02.010.
[11] Burke EK, Marecek J, Parkes AJ, Rudová H. Decomposition, reformulation, and diving in university course timetabling. Comput Oper Res 2010;37(3):582–97. https://doi.org/10.1016/j.cor.2009.02.023.
[12] Nouri HE, Driss OB. Distributed model for university course timetabling problem. IEEE international conference on computer applications technology (ICCAT). 2013. p. 1–6.
[13] Nouri HE, Driss OB. MATP: a multi-agent model for the university timetabling problem. Software engineering perspectives and application in intelligent systems - proceedings of the 5th computer science on-line conference (CSOC2016), Vol 2. 2016. p. 11–22. https://doi.org/10.1007/978-3-319-33622-0_2https://doi.org/10.1007/978-3-319-33622-0_2.
[14] Song T, Liu S, Tang X, Peng X, Chen M. An iterated local search algorithm for the university course timetabling problem. Appl Softw Comput 2018;68:597–608. https://doi.org/10.1016/j.asoc.2018.04.034.
[15] Kampke EH, de Souza Rocha W, Boeres MCS, Rangel MC. A GRASP algorithm with path relinking for the university courses timetabling problem. Proc Ser Braz Soc Comput Appl Math 2015;3(2):1–7. https://doi.org/10.5540/03.2015.003.02.0108.
[16] de Souza Rocha W, Claudia M, Boeres S, Rangel MC. A grasp algorithm for the university timetabling problem. Proceeding of 9th international conference of the practice and theory of automated timetabling PATAT. 2012. p. 404–6.
[17] Müller T, Rudová H. Real-life curriculum-based timetabling with elective courses and course sections. Ann Oper Res 2016;239(1):153–70. https://doi.org/10.1007/s10479-014-1643-1.
[18] Santos HG, Uchoa E, Ochi LS, Maculan N. Strong bounds with cut and column generation for class-teacher timetabling. Ann Oper Res 2012;194(1):399–412. https://doi.org/10.1007/s10479-010-0709-y.
[19] Dorneles ÁP, de Araújo OCB, Buriol LS. A fix-and-optimize heuristic for the high school timetabling problem. Comput Oper Res 2014;52:29–38. https://doi.org/10.1016/j.cor.2014.06.023.
[20] Gogos C, Alefragis P, Housos E. An improved multi-staged algorithmic process for the solution of the examination timetabling problem. Ann Oper Res 2012;194(1):203–21. https://doi.org/10.1007/s10479-010-0712-3.
[21] Beyrouthy C, Burke EK, Landa-Silva D, McCollum B, McMullan P, Parkes AJ. Towards improving the utilization of university teaching space. J Oper Res Soc 2009;60(1):130–43. https://doi.org/10.1057/palgrave.jors.2602523.
[22] Beyrouthy C, Burke EK, Silva DL, McCollum B, McMullan P, Parkes AJ. The teaching space allocation problem with splitting. Practice and theory of automated timetabling VI, 6th international conference, PATAT revised selected papers. 2006. p. 228–47. https://doi.org/10.1007/978-3-540-77345-0_15https://doi.org/10.1007/978-3-540-77345-0_15.
[23] Lindahl M, Mason AJ, Stidsen TR, Sørensen M. A strategic view of university timetabling. Eur J Oper Res 2018;266(1):35–45. https://doi.org/10.1016/j.ejor.2017.09.022.
[24] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. The MIT Press9780262033848; 2009.
[25] Moura AV, Scaraficci RA. A GRASP strategy for a more constrained school timetabling problem. Int J Oper Res 2010;7(2):152. https://doi.org/10.1504/ijor.2010.030801.
[26] Casey S, Thompson J. Grasping the examination scheduling problem. Proceedings of the 4th international conference on the practice and theory of automated timetabling (PATAT). Springer; 2002. p. 232–44.
[27] Edmonds J. Submodular functions, matroids, and certain polyhedra. Comb Optim 1970;11:11–26.
[28] When greedy algorithms are good enough: Submodularity and the (1 - 1/e) approximation. https://jeremykun.com/2014/07/07/when-greedy-algorithms-are-good-enough-submodularity-and-the-1-1e-approximation/; 2014. Accessed: 2017-07-03.
[29] Gurski F. Efficient binary linear programming formulations for boolean functions. Stat Optim InfComput 2014;2(4). https://doi.org/10.19139/soic.v2i4.83.
[30] IBM ILOG. Optimization studio CPLEX user' s manual, version 12 release 7. 2016.
[31] Roussel O. Controlling a solver execution with the runsolver tool. J Satisfiability Boolean ModellComput 2011;7(4):139–44.