Expert Systems With Applications 122 (2019) 137–151

Contents lists available at ScienceDirect

Expert Systems With Applications

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

A fast simulated annealing algorithm for the examination timetabling problem

Nuno Leite[a,b,*], Fernando Melício[a,b], Agostinho C. Rosa[c,b]

[a] Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa, Rua Conselheiro Emídio Navarro, n.°1, Lisboa, 1959-007, Portugal
[b] LARSyS: Laboratory for Robotics and Systems in Engineering and Science, Universidade de Lisboa, Av. Rovisco Pais, n.°1, Lisboa, 1049-001, Portugal
[c] Department of Bioengineering/Instituto Superior Técnico, Universidade de Lisboa, Av. Rovisco Pais, n.°1, Lisboa, 1049-001, Portugal

ARTICLE INFO

Article history:
Received 13 December 2017
Revised 27 December 2018
Accepted 28 December 2018
Available online 29 December 2018

MSC:
68T20
68R05
05C15
90B99
90C27

Keywords:
Examination timetabling
Hybrid algorithm
ITC 2007 benchmark set
Local search
Simulated annealing
Timetabling

ABSTRACT

The timetabling problem involves the scheduling of a set of entities (e.g., lectures, exams, vehicles, or people) to a set of resources in a limited number of time slots, while satisfying a set of constraints. In this paper, a new variant of the *simulated annealing* (SA) algorithm, named *FastSA*, is proposed for solving the examination timetabling problem. In the FastSA's acceptance criterion, each exam selected for scheduling is only moved (and the associated move is evaluated) if that exam had any accepted moves in the immediately preceding *temperature bin*. Ten temperature bins were formed, ensuring that an equal number of evaluations is performed by the FastSA, in each bin. It was observed empirically that if an exam had zero accepted movements in the preceding temperature bin, it is likely to have few or zero accepted movements in the future, as it is becoming crystallised. Hence, the moves of all exams that are fixed along the way are not evaluated no more, yielding a lower number of evaluations compared to the reference algorithm, the standard SA. A saturation degree-based heuristic, coupled with Conflict-Based Statistics in order to avoid any exam assignment looping effect, is used to construct the initial solution. The proposed FastSA and the standard SA approaches were tested on the 2nd International Timetabling Competition (ITC 2007) benchmark set. Compared to the SA, the FastSA uses 17% less evaluations, on average, and a maximum of 41% less evaluations on one instance. In terms of solution cost, the FastSA is competitive with the SA algorithm attaining the best average fitness value in four out of twelve instances, while requiring less time to execute. In terms of average comparison with the state-of-the-art approaches, the FastSA improves on one out of twelve instances, and ranks third among the five best algorithms.

The article's main impact comprises the points: (i) proposal of a new algorithm (FastSA) which is able to attain a reduced computation time (and number of evaluations computed) compared to the standard SA, (ii) demonstration of the FastSA capabilities on a NP-Complete timetabling problem, (iii) comparison with state-of-the-art approaches where the FastSA is able to improve the best known result on a benchmark instance. Due to the variety of problems solved by expert and intelligent systems using SA-based algorithms, we believe that the proposed approach will open new research paths with the creation of new algorithms that explore the space in a more efficient way.

1. Introduction

Timetabling problems are hard problems that have to be solved effectively by transportation companies, universities, hospitals, sport institutions, among others. Solving these problems for relatively large size institutions is a complex and time-consuming task, due, mainly, to the problem's combinatorial nature and to the absence, in many cases, of automatic tools that are able to solve them effectively. Hence, efficient tools for constructing timetables automatically are being demanded by decision makers and planners.

The timetabling problem involves the scheduling of a set of events (lectures, exams, surgeries, sport events, trips) to a set of resources (teachers, nurses and medical doctors, referees, vehicles) over space (classrooms, examination rooms, operating rooms, sport fields), in a prefixed period of time. In this paper, the *examination timetabling problem* (ETP) (Qu, Burke, McCollum, Merlot, & Lee, 2009) is studied. The goal of the ETP is to allocate exams and corresponding enrolled students to examination rooms over time periods, and trying to fulfil a predefined set of *hard* and *soft* constraints. The hard constraints have to be satisfied in order to have

* Corresponding author at: Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa - ADEETC, Gabinete 16, Rua Conselheiro Emídio Navarro, n.°1, Lisboa, 1959-007, Portugal.
E-mail addresses: nleite@cc.isel.ipl.pt (N. Leite), fmelicio@laseeb.org (F. Melício), acrosa@laseeb.org (A.C. Rosa).

https://doi.org/10.1016/j.eswa.2018.12.048
0957-4174/

a feasible timetable; the soft constraints, on the other hand, may not be observed completely and violations of these may exist. In standard benchmarks, such as the Toronto (Carter, Laporte, & Lee, 1996) and the Second International Timetabling Competition (ITC 2007) sets (McCollum, McMullan, Parkes, Burke, & Qu, 2012), there are essentially two goals, achieving feasibility, i.e. obtain a hard constraint cost of zero, and minimising the soft constraint cost. The ETP belongs to the NP-complete class of problems (de Werra, 1985; 1997). It has been approached mainly by mathematical programming methods (for relatively small size instances), and by approximate methods such as Artificial Intelligence (Schaerf, 1999) methods and metaheuristics (Qu et al., 2009) (e.g., genetic algorithms and simulated annealing).

1.1. Related work

The examination timetabling problem is studied by the scientific community since the 1960s. The surveys from Carter and Laporte (1996) and Schaerf (1999) provide a review of the early approaches to solve the ETP. Welsh and Powell (1967) establish a relation between graph colouring and timetabling. Several graph colouring heuristics were proposed in the literature and applied to the ETP (Carter, 1986). The application of the *saturation degree* heuristic (Brélaz, 1979) to the ETP was studied by Cheong, Tan, and Veeravalli (2009), among other graph colouring heuristics. The authors conclude that the saturation degree heuristic was among the two best out of five tested heuristics (*largest degree, colour degree, saturation degree, extended saturation degree*, and *random*).

The use of *simulated annealing* (SA) (Kirkpatrick, Gelatt, & Vecchi, 1983) for timetabling problems dates back to the 1990s, with the pioneer proposals of Dowsland (1990) and Abramson (1991). In a later investigation, Thompson and Dowsland (1996) use SA to solve a variant of the ETP (a multi-objective formulation of the ETP). The same authors propose in Thompson and Dowsland (1998) a SA approach to solve the ETP, comparing three neighbourhood operators (standard, Kempe chains, and S-Chains). They conclude that the operator based on the graph-theoretic concept of Kempe chains is the most effective one. The algorithm was tested on eight ETP instances from different universities. The SA metaheuristic was applied to other educational timetabling problems (school and course timetabling) by Bellio, Ceschia, Gaspero, Schaerf, and Urli (2016), Melício, Caldeira, and Rosa (2004, 2000) and Zhang, Liu, M'Hallah, and Leung (2010). In a recent work, Battistutta, Schaerf, and Urli (2017) propose a single-stage procedure based on the SA approach for the ITC2007's ETP. In the proposed method, non-feasible solutions are included in the search space and are dealt with appropriate penalties. A statistically-principled experimental analysis is performed in order to understand the effect of parameter selection. Then, a feature-based parameter tuning is carried out. The authors conclude that their properly tuned SA approach is able to compete with state-of-the-art solvers.

In this work we propose an *adaptive* simulating annealing for the examination timetabling problem. The adaptive part comes from the fact that the cooling rate computation is done in an automatic way. Other authors have proposed adaptive SA algorithms, for instance Ingber (2000).

Mühlenthaler (2015) investigates the structure of the course timetabling problem search space and establishes sufficient conditions for the connectedness of clash-free timetables under the Kempe-exchange operation.

The ITC 2007 participants have proposed methods that range from hybrid local search (Müller, 2009), *greedy randomized adaptive search procedure* (GRASP) metaheuristic (Gogos, Alefragis, & Housos, 2008), constraint satisfaction problem solver adopting a hybridization of *tabu search* and *iterated local search* (approach of Atsuta et al. (McCollum et al., 2010)), *tabu search* metaheuristic (De Smet's approach (McCollum et al., 2010)), and an approach based on cell biology (Pillay's approach (McCollum et al., 2010)).

Recent work on the ITC 2007 examination timetabling track have been published. The proposed approaches range from *bee colony optimisation* (Alzaqebah & Abdullah, 2014; 2015), *hill climbing* variants (Bykov & Petrovic, 2013), *great deluge* local search (Hamilton-Bryce, McMullan, & McCollum, 2013; McCollum, McMullan, Parkes, Burke, & Abdullah, 2009), GRASP (Gogos, Alefragis, & Housos, 2012), to *hyper-heuristics* (Burke, Qu, & Soghier, 2014; Demeester, Bilgin, Causmaecker, & Berghe, 2012; Özcan, Elhag, & Shah, 2012; Sabar, Ayob, Kendall, & Qu, 2015).

In relation to other educational timetabling problems, hyper-heuristics are also applied to the school timetabling problem by Ahmed, Özcan, and Kheiri (2015). In Shiau (2011), a hybrid particle swarm optimization approach is used to solve a university course scheduling problem.

Some surveys on the field of educational timetabling were published recently. Qu et al. (2009) present a detailed survey of algorithmic strategies applied to the ETP. Kristiansen and Stidsen (2013), Johnes (2015), and Teoh, Wibowo, and Ngadiman (2015) survey the application of operations research and metaheuristic approaches to academic scheduling problems. Pillay (2016) presents a review of hyper-heuristics for educational timetabling.

1.2. Our contribution

In this work, we propose an approach based on the SA for solving the ETP. The proposed method comprises two phases, a construction phase and an optimisation phase. Based on the findings of Cheong et al. (2009), a construction algorithm is devised that uses the saturation degree heuristic. In order to prevent for the algorithm to cycle between successive exam assignments/removals from the timetable, we have used the Conflict-Based Statistics data structure proposed in Müller (2009). Two SA-based algorithms are proposed. The first is the standard SA. The second one, named *FastSA*, is a new algorithm that executes a lower number of evaluations compared to the standard SA. The FastSA uses a modified acceptance criterion in which the temperature values are divided into temperature intervals, or *temperature bins*, where an equal number of evaluations is computed in each bin. Each exam selected for scheduling is only moved (and the associated move is evaluated) if that exam had any accepted moves in the immediately preceding temperature bin. In preliminary experiences, it was observed that exams having zero accepted movements in the preceding temperature bin, were likely to have few or zero accepted movements in the future, as they become crystallised. Hence, in the FastSA, new selections of exams already fixed are not evaluated. With this strategy, a lower number of evaluations is attained compared to the reference algorithm, the standard SA. Two neighbourhood operators, based on the Kempe chain neighbourhood, are proposed, one that changes examination rooms while maintaining the time slot, and other that shifts an examination to a different period and room.

The proposed research method has the following strengths: the FastSA is similar to the well-studied standard SA; the difference relies in the use of a new acceptance criterion as explained before; the second difference regarding the standard SA is the use of a data structure in the FastSA, with the aim of recording all the *element* (an *exam*, in the studied problem) movements performed in the local search operator, per each *temperature bin*. With this approach, the FastSA is able to determine when the selected element (exam) is to be moved, or it should freeze in the same time slot from this point on. Freezing an exam will reduce the number of evaluations performed as a frozen exam is never moved (and evaluated) again.

Table 1
Specifications of the 12 instances of the ITC 2007 benchmark set. The conflict matrix density is the ratio between the number of non-zero elements in the conflict matrix and the total number of elements.

	# Students	# Exams	# Rooms	Conflict matrix density	# Time slots
Instance 1	7891	607	7	0.05	54
Instance 2	12,743	870	49	0.01	40
Instance 3	16,439	934	48	0.03	36
Instance 4	5045	273	1	0.15	21
Instance 5	9253	1018	3	0.009	42
Instance 6	7909	242	8	0.06	16
Instance 7	14,676	1096	15	0.02	80
Instance 8	7718	598	8	0.05	80
Instance 9	655	169	3	0.08	25
Instance 10	1577	214	48	0.05	32
Instance 11	16,439	934	40	0.03	26
Instance 12	1653	78	50	0.18	12

A second contribution of the proposed research method is the automatic computation of the FastSA's cooling rate, i.e., the cooling rate is computed according to the instance to be solved and the available computation time. In this way, there is no need to tune the FastSA's cooling rate, reducing the number of parameters that have to be tuned.

The research method presented in this work has the following weaknesses: (i) the FastSA is not a general-purpose solver like SA in the sense that the element to be moved and the corresponding bin data structure have to be adapted to each new problem; for some problems, this could be a difficult or even impossible task; (ii) the FastSA can reduce the number of evaluations at the cost of a relatively small degradation in the solution cost, as the frozen exams are not moved any more. In the standard SA, any selected exam could be moved depending only of the temperature: if it is too low, the exam move will often be not accepted.

The remaining sections of the paper are organised as follows. In Section 2, the ITC 2007 examination timetabling problem is described. Section 3 presents the simulated annealing variants developed for approaching the examination timetabling problem. Section 4 reports the experimental results on the ITC 2007 benchmark set and compare the FastSA's results with the ones in the literature. Section 5 presents concluding remarks and future work.

2. Problem description

In this section, an informal formulation of the examination timetabling problem solved in this work (ITC, 2007) Track 1 – examination timetabling, is given. A complete formulation, including the analytical definition of the underlying optimisation problem and the fitness function used, is available in McCollum et al. (2012).

With the introduction of the ITC 2007, two other tracks were also proposed, namely, Track 2 – post enrolment based course timetabling, and Track 3 – curriculum based course timetabling. The examination timetabling track comprises 12 instances with different characteristics, and containing various constraint types, similar to those encountered in practice. The characteristics of these instances are described in Table 1.

The examination timetabling problem specified in the ITC 2007 extends the previous model, formulated in the Toronto specification (Carter et al., 1996), in that new hard and soft constraints are introduced. The extended set of hard constraints are as follows (McCollum et al., 2012):

- *No conflicts* – A *conflict* exists between two exams, attended by a given student, if the exams are scheduled in the same period or time slot. This hard constraint requires that cannot exist such conflicts in the students' scheduled exams.
- *Room occupancy* – For each room and time slot, the number of allocated seats must be less or equal than the number of available seats in that room.
- *Period utilisation* – For each exam, its duration must be less or equal than the period's duration.
- *Period related* – A set of time-ordering requirements between pairs of exams must be observed. In particular, for any pair (e_1, e_2) of exams, the following constraints were specified:
  - *After constraint* – e_1 must take place strictly after e_2,
  - *Exam coincidence* – e_1 must take place at the same time as e_2,
  - *Period exclusion* – e_1 must not take place at the same time as e_2.
- *Room related* – The following room requirement was specified:
  - *Room exclusive* – Exam e_1 must take place on an exclusive room.

In addition, all exams must be scheduled and the exams cannot be split between periods or rooms. A feasible solution is one that satisfies all the hard constraints. The problem soft constraints are as follows:

- *Two exams in a row* – The number of times where examinations are scheduled consecutively for a student is minimised.
- *Two exams in a day* – For the case where there exist three or more periods in a day, the number of occurrences of students having two exams in a day which are not directly adjacent, i.e. they have at least a free period between them, is minimised.
- *Period spread* – This soft constraint requires the spreading of the set of examinations taken by each student, over a fixed specified number of time slots.
- *Mixed durations* – A penalty is incurred whenever there exist exams in the same room and period with mixed durations.
- *Front load* – This soft constraint forces that the distribution of exams, with the largest number of students, should be carried out at the beginning of the examination session.
- *Room and period penalties* – An utilisation penalty for rooms and periods was specified, in order to reduce the utilisation of some rooms or periods to a minimum.

3. Proposed approach

This section describes the two proposed approaches based on the SA metaheuristic, namely the standard SA approach and the accelerated variant of SA, named FastSA. Section 3.1 describes the solution representation used for the ITC 2007 benchmark set. In Section 3.2, the process of constructing an initial, feasible, solution is described. Section 3.3 describes the SA-based search method and the neighbourhood structure used. Finally, in Section 3.4, the FastSA approach is explained.

3.1. Solution representation

Each solution manipulated by the search method encodes a complete and feasible timetable. The adopted solution representation is illustrated in Fig. 1. In the figure, exam e_6, for example, is allocated in time slot t_2 and room r_1, and exams e_2 and e_5 are allocated in time slot t_2 and room r_2. Time slots may have different lengths, in order to be possible to schedule examinations with different durations.

In terms of software implementation, the solution was implemented differently from that shown in Fig. 1. The solution is encoded as a matrix, say M, having in the row index the exam index, and in the column index the period number. Each position M_{ij} refers to exam i and period j and gives the room index if it

	t_1	t_2	...	t_T
r_1	e_3	e_6		e_9, e_{22}
r_2		e_2, e_5		e_{19}, e_{10}
.	.	.		.
.	.	.	...	.
.	.	.		.
r_{R-1}	.	.		.
r_R		e_4		e_7

Fig. 1. Solution representation for the ITC 2007 data set.

is assigned to that time, else is −1. This implementation provides for nuclear operations (exam allocation to period and room, room change, period change) to be performed in constant time.

3.2. Initial solution construction

In this section, the process of constructing an initial, feasible, solution is described. The proposed solution method only works with feasible solutions and thus it does not permit violations of hard constraints.

The initial solution is constructed using a variant of the *saturation degree* (SD) graph colouring heuristic (Brélaz, 1979; Cheong et al., 2009). In the SD heuristic, exams with the fewest valid periods, in terms of satisfying the hard constraints, remaining in the timetable are reinserted first (Cheong et al., 2009). The SD heuristic is implemented using a priority queue of the exams to be placed in the timetable. Each exam's priority is set to the exam's number of available periods. In the proposed approach, for performance reasons, the number of available periods of each exam is computed by considering only the *After* hard constraint (in the initial phase), and the *No Conflicts* (or clash) hard constraint, instead of considering all hard constraints. The remaining hard constraints are verified when the exam is scheduled.

In order to schedule the more constrained exams first, the exams in the queue are initially sorted by decreasing order of their number of *After* hard constraints. The other types of period related hard constraints (*Exam Coincidence* and *Period Exclusion*) are not considered in the initial exam sorting. The exams that must be placed before others (involved in an *After* constraint) have fewer available periods, and are thus more difficult to schedule. The initial sorting is done as follows. The initial priority of each exam is set to the *number of total periods, T*. Then, all the exams that must be scheduled before another exam have their number of feasible periods decremented by one, for each *After* hard constraint in which are involved. For instance, if an exam e_i must be scheduled *before* exam e_j (an *After* constraint exists between e_j and e_i), then the available periods for e_i include all periods except the last period, in order to make room for e_j. Hence, the number of available periods for e_i is equal to $T-1$, and the number of available periods for e_j is equal to T. In this way, e_i has more priority than e_j and is scheduled first.

The construction algorithm proceeds in two steps:

Step 1 Extract an examination from the exam priority queue and, if all hard constraints are met, schedule the exam in the selected period and room, and update the priorities of exams placed in the queue, considering only the *No Conflicts* hard constraint. Then, repeat the same process for the next maximum priority exam located in the queue. If a given exam cannot be scheduled, then go to Step 2. The construction phase ends when all exams are scheduled.

Step 2 If a selected exam cannot be assigned due to violations of hard constraints with existing assignments, the conflicting exams are unscheduled and the selected exam is scheduled. The conflicting exams are again inserted in the exam priority queue. The priority of each exam in the queue is recomputed according to the SD heuristic. Then, go to Step 1.
In order to prevent repetitive assignments of variables (exams) to the same values (period and room), the *conflict-based statistics* (CBS) (Müller, 2009) data structure is used. CBS is a data structure that stores *hard conflicts* (violations of the hard constraints) which have occurred during the search, together with their frequency, and the assignments that caused them. In the (period, room) value selection, the value that has the lowest number of weighted hard conflicts is chosen. For a description of the CBS data structure and its use in the construction of solutions please refer to Müller (2016).

3.3. Simulated annealing-based search method

In the optimisation phase, a SA-based approach is applied. Two variants were implemented: the standard SA and an accelerated one, which was named FastSA. The SA metaheuristic and neighbourhood relation are explained in this section. The FastSA is presented in Section 3.4.

The SA metaheuristic uses a probabilistic acceptance criterion. Let f be the objective function to be minimised on a set X of feasible solutions, and denote by $N(s)$ the neighbourhood for each solution s in X. Let L be the state-space graph induced by X and by the definition of $N(s)$. The SA is an iterative algorithm that starts from an initial solution and attempts to reach an optimal solution by moving step by step in L. In each step, a neighbour solution s' of the current solution s is generated; if the move improves the cost function then the algorithm moves s to the neighbour solution s'; otherwise, the solution s' is selected with a probability that depends on the current temperature and the amount of degradation of the objective function (Talbi, 2009). The temperature is reduced gradually according to a defined cooling schedule. Algorithm 1 describes the template of the SA algorithm (Kirkpatrick et al., 1983).

Algorithm 1 Template of the *simulated annealing* algorithm.

Input: Cooling schedule.

$s \leftarrow s_0$ ▷ Generation of the initial solution
$T \leftarrow T_{\max}$ ▷ Starting temperature
repeat
  repeat ▷ At a fixed temperature
    Generate a random neighbour $s' \in N(s)$
    $\Delta f \leftarrow \frac{f(s')-f(s)}{f(s)}$ ▷ Relative change in cost between $f(s')$ and $f(s)$
    if $\Delta f \leq 0$ **then** $s \leftarrow s'$ ▷ Accept the neighbour solution
    else Accept s' with a probability $e^{\frac{-\Delta f}{T}}$
    end if
  until Equilibrium condition ▷ e.g., a given number of iterations executed at each temperature T
  $T \leftarrow g(T)$ ▷ temperature update
until Stopping criteria satisfied ▷ e.g., $T < T_{\min}$
Output: Best solution found.

The relative change in cost Δf between $f(s')$ and $f(s)$ is expressed as $\Delta f \leftarrow \frac{f(s')-f(s)}{f(s)}$. The temperature T is updated by simulating the temperature exponentially-decreasing function of time, used in metal annealing processes. This is achieved by the function $T(t)$:

$$T(t) = T_{max} \cdot \exp(-r \cdot t), \quad (1)$$

where r is the temperature decreasing rate and $T_{\max}$ is the initial temperature ($T_{\max}$ should have a large value as compared to r).

3.3.1. Kempe chain neighbourhood

In the proposed approach, the *Kempe chain* neighbourhood (Thompson & Dowsland, 1998) is used. In this neighbourhood, a solution exam, included in a *Kempe chain*, is perturbed in a feasible fashion. The pseudo-code of the Kempe chain move is given in Algorithm 2.

Algorithm 2 Pseudo-code of the Kempe chain based heuristic.

1: **function** KEMPECHAIN
2: **Inputs:** Two time slots t_i and t_j (randomly selected); A Kempe chain $K = \{e_1, e_2, \ldots, e_n, e_{n+1}, e_{n+2}, \ldots, e_{n+m}\}$ where exams $e_1, e_2, \ldots, e_n$ belong to t_i and exams $e_{n+1}, e_{n+2}, \ldots, e_{n+m}$ belong to t_j. Exams of K in t_i have hard conflicts with exams of K belonging to t_j.
3: Replace t_i with $(t_i \setminus K) \cup (t_j \cap K)$ and t_j with $(t_j \setminus K) \cup (t_i \cap K)$.
4: **Output:** Updated time slots t_i and t_j
5: **end function**

In the next subsection the application of the Kempe chain heuristic to an example ITC 2007 solution is shown.

3.3.2. ITC 2007 neighbourhood

In solving the ITC 2007 timetabling problem instance, two neighbourhood operators, based on the Kempe chain neighbourhood, were implemented:

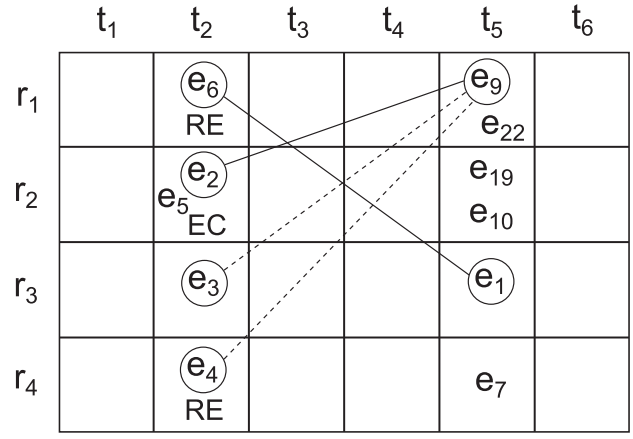
Slot-Room move – A random exam is scheduled in a different period and room, both chosen randomly.
Room move – An exam, chosen randomly, is scheduled into a different room in the same period. The destination room is chosen in a random fashion.

The *Room move* operator was included for two reasons. The first, for completeness, i.e., it is desirable to have a set of operators that could move an exam to any place (period and room). The second was justified by the need to have an operator that could be used to minimise the room specific (*Mixed Durations* and *Room Penalty*) soft constraints violations.

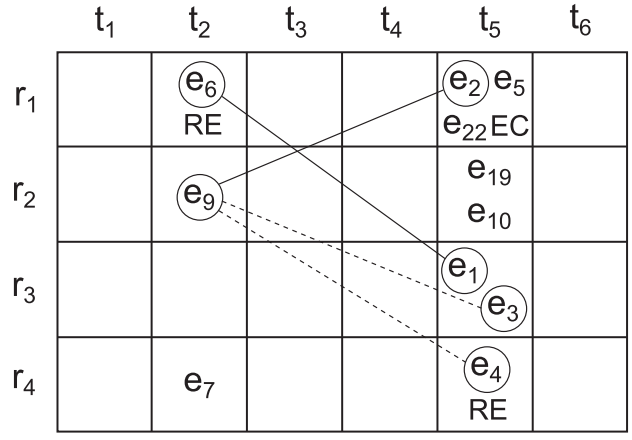
Fig. 2 illustrates the use of the *Slot-Room move* operator on an example ITC 2007 solution. In this example, the Kempe chain involving exam e_2, to be moved from time slot t_2 and room r_2 to time slot t_5 and room r_1, is given by $K_1 = \{e_2, e_3, e_4, e_5, e_7, e_9\}$ (Algorithm 2). Time slots t_i and t_j in Algorithm 2 are set to t_2 and t_5, respectively, with initial contents given by $t_2 = \{e_2, e_3, e_4, e_5, e_6\}$ and $t_5 = \{e_1, e_7, e_9, e_{10}, e_{19}, e_{22}\}$. Then, t_2 is updated with $(t_2 \setminus K_1) \cup (t_5 \cap K_1) = \{e_6, e_7, e_9\}$ and t_5 is updated with $(t_5 \setminus K_1) \cup (t_2 \cap K_1) = \{e_1, e_2, e_3, e_4, e_5, e_{10}, e_{19}, e_{22}\}$.

Both operators try to move exams in a feasible way using the Kempe Chain procedure. If it is not possible to apply the move, then this is set to be infeasible and the move is not applied.

The presented Kempe chain operator was implemented in an incremental fashion, which allows for solutions to be evaluated incrementally. In incremental (also called *delta*) evaluation (Corne, Ross, & Fang, 1994), only the exam edges that are updated by the operator are evaluated, allowing for a substantial increase in the operator's performance.



(a) before moving exam e_2 from time slot t_2 and room r_2 to time slot t_5 and room r_1



(b) after moving exam e_2

Fig. 2. An example of the Kempe chain heuristic. There exist two Kempe chains in the figure. In the first Kempe chain, a move of exam e_2 from time slot t_2 and room r_2 to time slot t_5 and room r_1 requires repair moves to maintain feasibility. Exam e_9 has to be moved to time slot t_2 and room r_2, and exams $\langle e_2, e_5 \rangle$ (because of the presence of the *Exam Coincidence* (EC) constraint), e_3, and e_4, have to be moved to time slot t_5 and rooms r_1, r_3, and r_4, respectively. Consequently, exam e_7 has to be moved from time slot t_5 and room r_4 to time slot t_2 and room r_4 because a *Room Exclusion* (RE) constraint exists.

3.3.3. SA parameter configurations and cooling rate computation

In this section some details about the specification of the SA parameters are given. According to the ITC2007's rules, participants have to benchmark their machine with the programme provided by the organisation. This is done in order to know how much time the participants have available to run their programme on their machines (ITC, 2007). In our case, the available algorithm computation time, determined by the benchmarking tool provided by the ITC 2007, is 276 s. In the ITC2007's rules it is also pointed out that the programmer should not set different parameters for different instances. However, if the computer programme is doing this automatically then this is acceptable.

In this work, two SA parameter configurations are specified, which are used in two different set of experiments. In the first SA parameter configuration, a distinct set of parameters is used for each ITC 2007 instance. To note that this parameter setting does

not comply with the ITC 2007 requirements. However, this parameter configuration is only used in experiments involving proposed variants of the SA algorithm, and not in state-of-the-art algorithm comparison.

The second set of parameters used is in line with the ITC 2007 rules and comprises a fixed set of SA parameters for all instances, excepting the cooling rate value which is computed automatically by the proposed algorithm for each ITC 2007 instance. In previously published work (Nunes, 2015; Nunes, Ferreira, & Leite, 2016) this same issue is addressed; the authors propose a SA metaheuristic in which the cooling rate parameter is computed automatically for each instance. The other SA parameters (initial and final temperatures, and number of iterations at a fixed temperature) remain constant for all benchmark instances. In the present work, a SA is proposed which follows the same ideas introduced in Nunes (2015) and Nunes et al. (2016).

Hence, a SA with a variable rate was implemented to make this metaheuristic run closer to the given ITC 2007 time limit for all ITC 2007 instances. In order to determine the proper cooling rate, the algorithm starts by simulating the execution of the SA metaheuristic, by running the local search neighbourhood operator *AverageReps* times. For simplicity, instead of executing each complete step of the SA metaheuristic (generation and evaluation of the neighbour followed by the application of the acceptance criterion) only the generation and evaluation of the neighbour are performed. The *AverageReps* parameter was set to 10000. This value was found empirically to be adequate, allowing a reasonable number of runs to be performed while not taking too much computation time. Using the elapsed time and the given ITC 2007 time limit, the number of neighbours that would be generated if this heuristic was to be run within the time limit is estimated. The number of neighbours is computed using the expression

$$CompNeighbs = TotalTime/CompTime \cdot AverageReps, \quad (2)$$

where parameter *TotalTime* denotes the ITC 2007 time limit (276000 ms), *CompTime* denotes the elapsed time used in the simulation, and *AverageReps* denotes the number of loops the local search neighbourhood operator was run (10000 runs). After that, the exact number of neighbours (*TotalNeighbs*) that will be generated for the given SA parameters, using an initial default rate, is computed. This initial rate is subsequently adjusted by the algorithm until *TotalNeighbs* is sufficiently close to the *CompNeighbs* value. Algorithm 3 describes the pseudocode of the algorithm used to compute the SA cooling rate in an automatic fashion.

The calculation represented in (2) is done in line 2 of Algorithm 3 by the function *estimateNumberNeighbours*. In lines 3–4, an *offset* term is used in the computation of *comp_neighbours*. This offset represents the percentage of *comp_neighbours* to be considered in the rate computation. The use of a percentage is justified by the process used in the computation of the *comp_neighbours* parameter value, which does not take into account all the steps of the SA metaheuristic, and also by the stochastic nature of the SA. It was verified experimentally that for some instances the given time limit was exceeded, and for other instances the computation time was below the time limit. In the proposed FastSA approach (Section 3.4), the computation times differences to the ITC 2007 time limit were even more noticeable, as documented in Section 4. The offset used is 80%. In lines 7–20 of Algorithm 3, the *rate* parameter is iteratively adjusted until the number of computed neighbours for the chosen SA parameters, i.e. *total_neighbours*, is close and less than the theoretical value of *comp_neighbours*. Function *getSANumberNeighbours* returns the SA number of neighbours, or number of evaluations done, for the given parameters.

Some experiments were made in order to check Algorithm 3's behaviour, for ITC2007's Instance 4, using the following parameters: $T_{Max} = 0.01$, $T_{Min} = 1 \times 10^{-6}$, $reps = 5$, $exec\_time =$ 276000 ms, and computed $rate = 6.8 \times 10^{-6}$ (rate computed automatically using Algorithm 3). For the given parameters, the computed neighbours $comp\_neighbours \cdot offset$ is 6773005, whereas the number of evaluations (*total_neighbours*) is 6772311. The elapsed computation time was 219000 ms. Fig. 3 illustrates the solution fitness evolution for ITC2007's Instance 4 when applying SA with the above parameters. The *x*-axis represents the number of generated feasible neighbours, and the *y*-axis represents the solution fitness. As can be seen in Fig. 3, SA starts by accepting some worse and better neighbour solutions. In the end, the temperature is so low that it becomes harder to accept worse solutions, ending up acting similarly to the *hill-climbing* procedure.

In this experiment, the number of generated *feasible* neighbours was 3143070 ($\approx$46.41% of *total_neighbours*), whereas the number of accepted neighbours (the maximum number of iterations in Fig. 3) was 48837, which is $\approx$1.55% of the number of accepted neighbours. These numbers show that it is difficult to generate feasible neighbours with the Kempe chain move and also that the SA initial temperature has a relatively low value, resulting in a low percentage of accepted neighbours. Despite this, the SA is able to obtain a relatively good final solution fitness value.

3.4. Accelerating the SA metaheuristic for the ETP

In this section, the FastSA search method for solving the ETP is explained. In Section 3.4.1, the effect of the cooling schedule on the exam move acceptance rate in SA-based algorithms is studied, introducing the basic framework used by the FastSA method. The FastSA approach is described in Section 3.4.2.

3.4.1. Effect of the cooling schedule on the exam move acceptance in SA-based algorithms

In the study carried out in this section, the effect of the cooling schedule on the exam move acceptance in SA-based algorithms is analysed. Some important properties of the local search operator are identified, which are exploited by the FastSA algorithm (described in Section 3.4.2).

In the experiments, two different cooling schedules were used: a *light* cooling schedule, with parameters $T_{\max} = 0.1$ (initial temperature), $r = 1 \times 10^{-4}$ (decreasing rate), $k = 5$ (# iterations at each temperature), and $T_{\min} = 1 \times 10^{-7}$ (final temperature); and an *intensive* cooling schedule, with parameters $T_{\max} = 0.1$, $r = 1 \times 10^{-5}$, $k = 5$, and $T_{\min} = 1 \times 10^{-7}$. With the light cooling schedule, each SA algorithm execution computes 690780 evaluations. With the intensive cooling schedule, 6907760 evaluations are computed by each SA algorithm execution. The justification for the choice of the SA parameters is given at the end of this section.

Figs. 4 and 5 illustrate the evolution of the number of accepted exam movements as the SA temperature varies, for two example ITC 2007 instances. In these figures, the x-axis value range is divided into ten *temperature bins*. A *temperature bin* corresponds to a temperature interval and is represented by two vertical grid lines, interpreted from the x-axis in the figures.

The temperature bins were calculated in order for the SA metaheuristic to compute, in each bin, 1/10 of the total number of evaluations. The colour codes represented in each bin, and for each exam on the *y*-axis, represent the number of accepted movements for the corresponding exam. The total number of exams' accepted moves per bin is generally lower than the number of evaluations performed in that bin, as not all the exams are subject to an accepted move in each local search move. Some exams do not move because the fitness difference violates the SA acceptance criterion, or are not selected by the neighbourhood operator. The exams are ordered decreasingly by the number of conflicts they have with other events (*largest degree* heuristic Cheong et al., 2009).

Algorithm 3 SA cooling schedule rate computation.

Input:

- s ▷ Solution
- T_{Max} ▷ Maximum temperature
- T_{Min} ▷ Minimum temperature
- $reps$ ▷ Number of iterations per temperature
- $exec\_time$ ▷ ITC 2007 time limit

$n \leftarrow 10000$ ▷ Number of neighbourhood operator runs
$comp\_neighbours \leftarrow estimateNumberNeighbours(n, exec\_time, s)$
$offset \leftarrow 0.8$
$comp\_neighbours \leftarrow comp\_neighbours \cdot offset$
$rate \leftarrow 1e-3$ ▷ Initial default rate
$rate\_to\_sub \leftarrow 1e-3$ ▷ Rate subtraction factor
$depth \leftarrow 3$ ▷ Number of iterations until convergence
while $depth > 0$ **do**
  $total\_neighbours \leftarrow getSANumberNeighbours(T_{Max}, rate, reps, T_{Min})$
  if $total\_neighbours < comp\_neighbours$ **then**
    if $rate <= rate\_to\_sub$ **then**
      $rate\_to\_sub \leftarrow rate\_to\_sub/10$
    end if
    $rate \leftarrow rate - rate\_to\_sub$
  else
    $rate \leftarrow rate + rate\_to\_sub$ ▷ To guarantee that total_neighbours < comp_neighbours
    $depth \leftarrow depth - 1$
    $rate\_to\_sub \leftarrow rate\_to\_sub/10$
  end if
end while
Output: $rate$ ▷ Return computed rate

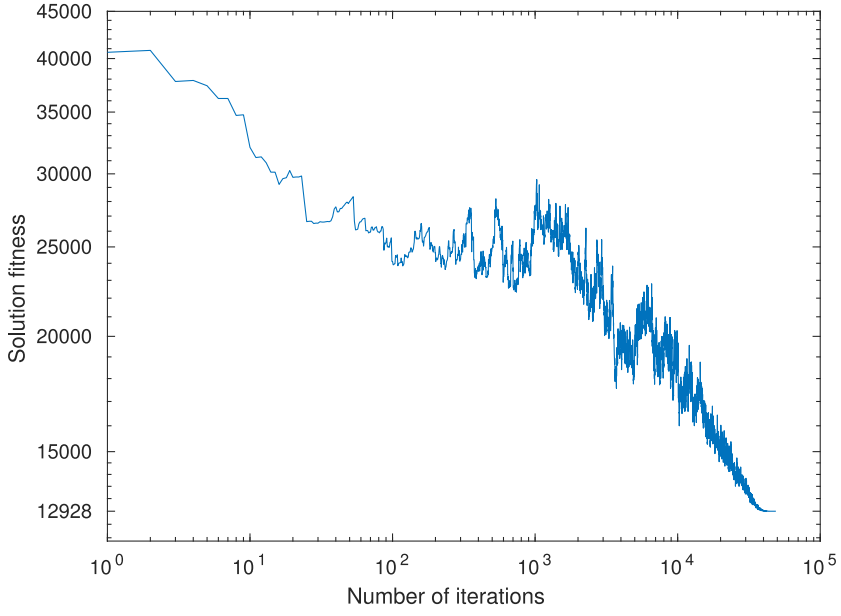


Fig. 3. Solution fitness evolution for ITC2007's Instance 4 when applying simulated annealing.

It can be observed that the more difficult exams (the ones with lower indexes) only move (a move is accepted by the SA) when higher temperatures are applied, being gradually fixed at the same timeslot when lower temperatures are applied (e.g., below temperature 1e-2). The easier exams (those with higher indexes) continue to move, being gradually arranged in their definitive place.

As it can be seen from the difference in the top and bottom plots from Figs. 4 and 5 (and corresponding solutions' fitness), it is essential that the difficult exams be well arranged, in an optimal or near optimal place, for the easier exams also to be placed in an optimal fashion. If a cooling schedule with few evaluations is used, the SA algorithm cannot find the optimal (near optimal) place for the difficult exams and so the easier exams are also placed in a suboptimal way.

For this experiment, the SA parameters were set as follows. We have selected an initial temperature value that allows for the great

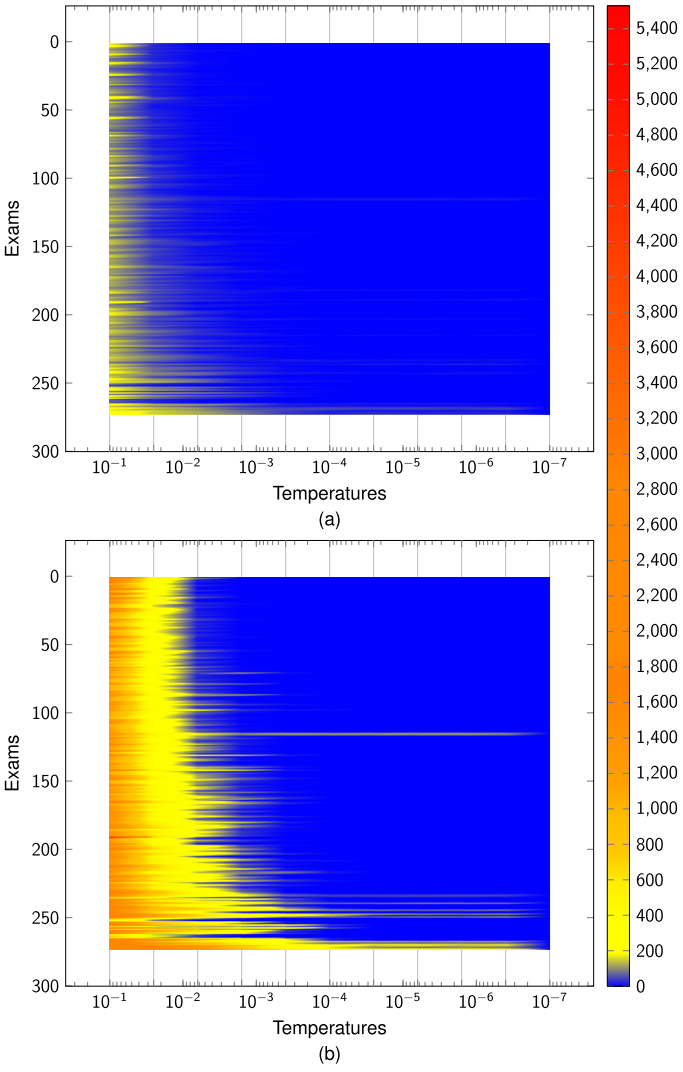


Fig. 4. Study of the effect of applying *light* and *intensive* cooling schedules in the SA algorithm for ITC2007's Instance 4: (a) optimisation using the *light* cooling schedule (obtained cost: 14058); (b) optimisation using the *intensive* cooling schedule (obtained cost: 12528). The figure shows the evolution of the number of accepted exam moves, for each exam in the *y*-axis, per each temperature bin. The exams in the *y*-axis are sorted by decreasing order of number of conflicts with other exams. The marked *x*-axis long ticks represent the temperature intervals. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

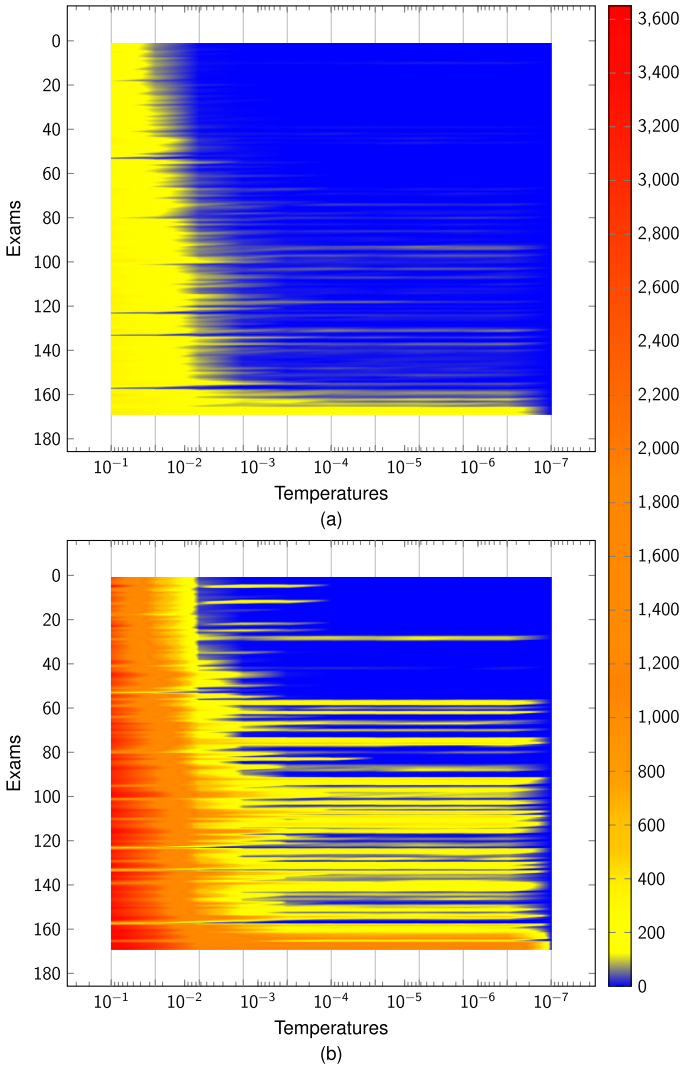


Fig. 5. Study of the effect of applying *light* and *intensive* cooling schedules in the SA algorithm for ITC2007's Instance 9: (a) optimisation using the *light* cooling schedule (obtained cost: 1243); (b) optimisation using the *intensive* cooling schedule (obtained cost: 1024). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

majority of exams to be moved several times, in order to better explore the search space. Graphically, we know that this effect is achieved by having the first temperature bin practically filled in Figs. 4 and 5. After several experiments we have determined that the value $T_{max} = 0.1$ was a reasonable value. The remaining parameters of the cooling schedule (r, k, and T_{min}) were set in order to have two distinct rates (parameter r), a *light* one and an *intensive* one, a low number of iterations at a fixed temperature (parameter k) since the intensification is mainly controlled by the rate parameter, and a sufficiently low temperature (T_{min} parameter), that is, a value from which non-improving moves are practically never accepted.

3.4.2. FastSA search method for the ETP

As it can be observed from Figs. 4 and 5, several of the more difficult exams are going to be fixed (or have few moves) as lower temperatures are reached. In the algorithm, if it were decided not to move (and not evaluate the corresponding movement) an exam having zero or few movements in the previous temperature bin, it would produce a faster SA algorithm. The faster execution comes at the expense of a degradation of the results, if there exist bins with exam movements followed or interleaved with bins with no movements, because the algorithm would stop evaluating moves for a given exam when a bin with zero moves is found. This faster SA variant was implemented in this work, and was named FastSA. The template of the FastSA algorithm is given in Algorithm 4.

The lines with numbers marked in bold are lines that were added to the original SA algorithm (Algorithm 1). The FastSA keeps a record of all successful neighbourhood movements taken in a given bin. When performing a neighbourhood movement, a random exam is selected and the algorithm first checks if there were any movements, for the exam to be moved, in the previous bin. If there was no previous movement for the candidate exam, then the current movement is not performed and not evaluated. With this strategy, the FastSA is able to attain a reduced number of evaluations compared to the standard SA. The cost degradation attained by the FastSA compared to the SA is not significant, as shown in the experimental evaluation carried out in Section 4.

Algorithm 4 Template of the *fast simulated annealing* algorithm.

Input: Cooling schedule.

prevBin ← nil; ▷ Array containing # accepted moves in the previous bin, initially nil
currBin ← createArray(numExams); ▷ Array containing # accepted moves in the current bin, initialised with zeros
$s \leftarrow s_0$; ▷ Generation of the initial solution
$T \leftarrow T_{max}$; ▷ Starting temperature
repeat
  repeat ▷ At a fixed temperature
    Generate a random neighbour $s' \in N(s)$;
    examToMove ← selectExamFromSolution(s')
    if *prevBin* = nil or (*prevBin* <> nil and *prevBin*[*examToMove*] > 0) **then** ▷ Evaluate neighbour solution
      $\Delta f \leftarrow \frac{f(s')-f(s)}{f(s)}$ ▷ Relative change in cost between $f(s')$ and $f(s)$
      if $\Delta f \leq 0$ **then** $s \leftarrow s'$ ▷ Accept the neighbour solution
      else Accept s' with a probability $e^{\frac{-\Delta f}{T}}$;
      end if
      if s' was accepted **then**
        currBin[*examToMove*] ← *currBin*[*examToMove*] + 1
      end if
    end if
  until Equilibrium condition ▷ e.g., a given number of iterations executed at each temperature T
  $T \leftarrow g(T)$; ▷ temperature update
  prevBin ← updateBin(T, *currBin*) ▷ If T is in the next bin, make *prevBin* ← *currBin* and zero *currBin*
until Stopping criteria satisfied ▷ e.g., $T < T_{min}$
Output: Best solution found.

Example 3.1 illustrates a simple example of the bin structure used by the FastSA.

Example 3.1. This example uses synthetic data to show the bin structure used by the FastSA. Six examinations and four bins were considered. Fig. 6(a) illustrates the number of times each exam is selected in each bin. Fig. 6(b) gives a graphical representation of the same data using a colour map.

When the FastSA is run using this data, the following behaviour is registered. Because exam e_1 was never selected in bin b_2, e_1 remains fixed (a move involving this exam is not evaluated) from bin b_3 onwards, even if it had a positive number of selections in bins b_3 and b_4. Using the same reasoning, exams e_2, e_3, and e_4, remain fixed from bin b_4 onwards. Note the case of exam e_4 which is selected in bin b_4 (marked in shaded grey). As the number of movements in the previous bin (b_3) is zero, e_4 will be fixed from bin b_4 onwards, and the three selections in bin b_4 are not evaluated by the FastSA, yielding a faster algorithm.

4. Experimental results and discussion

This section presents the experimental simulations conducted to test the proposed method in addressing the examination timetabling problem. Section 4.1 describes the experiments' settings and algorithm parameter settings. Section 4.2 presents a comparison between three algorithms: two FastSA variants, named FastSA100 and FastSA80, and the standard SA. The FastSA100 variant is further compared with state-of-the-art approaches in Section 4.3.

4.1. Settings

The developed algorithms were programmed in the C++ language using the ParadisEO framework (Talbi, 2009). The hardware and software specifications are: Intel Core i7-2630QM, CPU @ 2.00 GHz × 8, with 8GB RAM; OS: Ubuntu 16.10, 64 bit; Compiler used: GCC v. 4.8.2. The algorithm computation time was set to 276 seconds, as measured by the provided benchmarking tool from the ITC 2007 site (ITC, 2007) (please refer to Section 3.3.3 for more details).

Table 2
Cooling schedules used to solve the different ITC 2007 instances. The value presented in the bottom row, rightmost column, corresponds to the sum of the # evaluations. The cooling schedule values were adapted empirically for each dataset instance using as reference the FastSA100 algorithm.

Inst.	T_{Max}	*rate*	k	T_{Min}	# Evaluations
1	0.01	0.000003	5	1.00×10^{-6}	15,350,571
2	0.01	0.000004	5	1.00×10^{-6}	11,512,931
3	0.01	0.000005	4	1.00×10^{-6}	7,368,277
4	0.1	0.000005	6	1.00×10^{-6}	13,815,517
5	0.01	0.000007	6	1.00×10^{-6}	7,894,579
6	0.002	0.000003	5	1.00×10^{-6}	12,668,176
7	0.001	0.000003	5	1.00×10^{-6}	11,512,931
8	0.001	0.0000015	5	1.00×10^{-6}	23,025,856
9	0.01	0.000001	5	1.00×10^{-6}	46,051,706
10	0.01	0.000001	5	5.00×10^{-6}	38,004,516
11	0.01	0.000009	5	1.00×10^{-6}	5,116,861
12	0.01	0.0000013	5	1.00×10^{-6}	35,424,391
				Σ	227,746,312

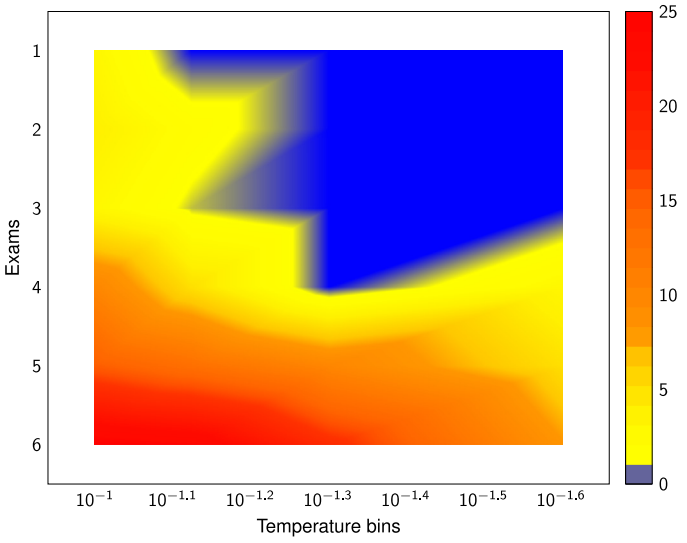
4.1.1. Parameter settings

As mentioned in Section 3.3.3, two parameter configurations were used in the conducted experiments. In the first configuration, the parameters were manually tuned for each ITC 2007 instance. This configuration was only used in the FastSA variants' comparison. In the second configuration, a fixed set of parameters was used for all instances, excepting the cooling rate, which was computed automatically by the optimisation algorithm (refer to Section 3.3.3 for more details). The second parameter configuration was used to compare the FastSA100 with the state-of-the-art approaches.

Table 2 summarises the cooling schedule parameters used in the first configuration. The parameter values were chosen using as reference the FastSA100 algorithm (described in the next section). Specifically, the cooling schedule values were adapted empirically

	b_1	b_2	b_3	b_4
e_1	3	0	0	0
e_2	4	2	0	0
e_3	3	1	0	0
e_4	10	5	0	3
e_5	15	13	10	5
e_6	25	24	18	8

(a)



(b)

Fig. 6. Different representations of the bin structure used by the FastSA. Six examinations and four bins were considered: (a) number of times each exam is selected in each bin, (b) graphical representation of the data in (a) using a colour map. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

for each dataset instance, while guaranteeing that the FastSA100 computation time was within the ITC 2007 time limit constraint.

For the second parameter configuration, the following parameter set was used: $T_{Max} = 0.01$, $T_{Min} = 1 \times 10^{-6}$, $reps = 5$, and *rate* computed automatically using Algorithm 3.

The neighbourhood operators, the *Room* and *Slot-Room* moves, were selected with equal probability. This value was found after conducting some tuning experiments. Three runs of each ITC 2007 instance were executed, varying the probabilities values (P_{sr}, P_r) in the set: {(0, 1), (0.25, 0.75), (0.5, 0.5), (0.75, 0.25), (1, 0)}. P_{sr} and P_r denote, respectively, the probabilities of choosing the *Slot-Room* and *Room* moves. Selecting only the *Room* move lead to the worse results, as only the room is changed. Triggering only the *Slot-Room* move gives good results but slightly worse results in some instances compared to using both operators. No significant distinction was found using both operators with the remaining probabilities combinations.

To obtain our simulation results, each algorithm was run ten times on each instance with different random seeds. In the experiments, all the statistical tests were performed with a 95% confidence level. The developed source code, the resulting solution files for each instance/run, and the produced statistics, are publicly available on the following Git repository: https://github.com/nunocsleite/FastSA-ETP-ITC2007.

4.2. Comparison between FastSA and SA approaches

In this section, three algorithms, described as follows, were compared:

1. SA – The original SA metaheuristic as described in Algorithm 1.
2. FastSA100 – The basic FastSA algorithm as described in Algorithm 4. In this approach, the exam movements in each bin are recorded, and if, for a selected exam, the number of movements in the previous bin is zero, that exam is *fixed* until the end of the optimisation phase. This means that the current and future movements of this exam are not considered and thus not evaluated.
3. FastSA80 – In this variant, the behaviour is similar to the FastSA100 but the selected exam is fixed only if two conditions hold: the number of movements in the previous bin is zero *and the exam belongs to the set comprising 80% of the exams with the highest degree*. Observing again Figs. 4 and 5, we can see that the highest degree exams are those having the lower indices (top indices in the *y*-axis); on the other side, the lowest degree exams are the ones having the higher indices (the bottom ones in the *y*-axis). For the mentioned set of largest degree exams, the behaviour is the same as described for the FastSA100; for the remaining 20% of the exams, the method behaves like the original SA, that is, the selected exam is never fixed and its move is always evaluated. The reasoning behind this variant is the following: because the lowest degree exams have a low number of conflicts with other exams, their moves imply a small change in the objective function; this, in turn, imply that they are likely to move more often than the higher degree exams, because small changes in the objective function are often accepted by the SA acceptance criterion, even at low temperatures. Hence, in the scenario that a lowest degree exam has no accepted moves in a previous bin, it is likely that it will have accepted moves in the future. With the FastSA80, the moves of the set comprising 20% of the exams with the smallest degree are always evaluated, despite having some bins with zero counts.

Tables 3 and 4, show, respectively, the obtained costs and the corresponding execution times in seconds for the FastSA100, FastSA80, and SA algorithms, for the twelve instances of the ITC 2007 benchmark set. For each algorithm the minimum and average values, and the standard deviation of ten runs are shown.

4.2.1. Discussion

Analysing the results in terms of solution cost (Table 3), it can be observed that the FastSA100 has results that compete with the SA, obtaining the best average fitness in four of twelve instances, while requiring less evaluations. In terms of the performed number of evaluations (Table 3, column '% less evals'), the FastSA100 and FastSA80 execute, respectively, 17% and 16% less evaluations, on average, compared to the SA algorithm. The sum of the mean values of the number of evaluations obtained for each instance is 184208745.7 and 185236114.1, respectively, for FastSA100 and FastSA80. For the SA, the sum of the number of evaluations obtained for each instance is 227746312. These values show that the FastSA100 is the technique that performs the lowest number of evaluations on average.

For the FastSA100, on more than half of the datasets, the number of neighbours not evaluated varies from 9% to 41% while not degrading the fitness value significantly, which is a significant number. From Table 4 results, we can conclude that the presented FastSA approaches are globally faster than the SA approach.

Table 3
Solution costs obtained on the ITC 2007 benchmark set by the FastSA100, FastSA80, and SA algorithms. For the Fast SA variants, the percentage of evaluations not done (marked in column '% less evals') is shown.

Instance	FastSA100				FastSA80				SA		
	% less evals	f_{min}	f_{avg}	σ	% less evals	f_{min}	f_{avg}	σ	f_{min}	f_{avg}	σ
1	**34**	4872	5054.7	111.3	30	4916	5026.9	73.9	**4855**	**4953.9**	100.5
2	**1**	**395**	408.5	7.1	**1**	400	413.0	10.3	**395**	**407.5**	8.3
3	**5**	**9607**	**9945.6**	258.6	**5**	9749	10,141.7	239.6	9737	10,079.9	277.3
4	**41**	12,076	12,825.8	407.7	38	**12,073**	**12,667.2**	420.0	12,268	12,783.5	407.8
5	**6**	3058	**3378.2**	194.6	**6**	**3052**	3431.6	345.7	3188	3666.1	512.1
6	**14**	**25,515**	**25,960.5**	213.8	12	25,790	26,166.0	292.1	25,870	26,139.5	180.1
7	**7**	4291	4533.9	120.6	**7**	4306	4446.8	104.5	**4182**	**4426.6**	159.3
8	**25**	**7226**	7532.2	168.8	24	7471	7569.9	97.6	7287	**7515.7**	147.4
9	28	983	1025.8	33.1	**29**	**970**	1027.8	33.3	991	**1017.3**	23.8
10	**8**	13,400	13,662.3	214.5	**8**	13,412	13,656.9	145.2	**13,351**	**13,511.7**	104.0
11	**9**	30,090	**31,520.8**	1027.9	**9**	29,860	31,886.8	1375.8	**29,800**	31,930.9	1156.8
12	19	5137	5200.6	46.8	**20**	5143	5191.4	56.2	**5109**	**5179.1**	51.8
Avg:	**17**				16						

Table 4
Execution times (in seconds) on the ITC 2007 benchmark set for the FastSA100, FastSA80, and SA algorithms.

Instance	FastSA100			FastSA80			SA		
	t_{min}	t_{avg}	σ	t_{min}	t_{avg}	σ	t_{min}	t_{avg}	σ
1	**224**	**234.70**	6.80	232	236.30	2.58	369	377.20	6.11
2	227	233.70	4.32	**225**	**229.70**	3.53	249	258.00	8.03
3	193	197.90	4.33	**190**	**197.60**	5.23	206	219.95	9.06
4	**236**	**244.90**	4.36	245	255.70	5.60	428	442.60	8.85
5	**198**	211.00	7.13	204	**209.00**	2.62	236	257.70	14.84
6	**172**	192.41	10.82	174	**188.90**	9.00	217	226.20	5.47
7	228	234.40	3.92	**217**	**229.18**	8.33	260	277.80	15.13
8	227	234.80	4.71	**221**	**232.10**	6.08	305	314.26	10.04
9	**217**	238.90	11.64	226	**236.70**	10.04	327	342.50	11.65
10	**232**	242.70	5.60	234	**240.40**	3.92	253	258.20	2.53
11	208	**209.90**	1.97	**204**	210.40	5.21	231	238.80	5.51
12	**210**	220.70	6.06	214	**219.20**	6.09	269	274.20	4.24

Concerning the execution times, we can observe that a lower average execution time on a given instance time does not necessary mean a lower average number of evaluations. If we compare, for example, Instance 6 solved by FastSA100 and FastSA80 (Tables 3 and 4), we observe that the FastSA100 performs 14% less evaluations, whereas FastSA80 performs 12% less evaluations. Despite executing a lower number of evaluations, the FastSA100 obtains a better cost and spends more time, on average. This behaviour is justified by the several factors influencing the execution of the algorithm, namely: the stochastic nature of the construction algorithm and of the FastSA algorithm itself, and the existence of two move operators (room and slot-room moves), with different execution times. The slot-room move is more complex than the room operator thus taking more time to execute. It should be noted that some of the times of the SA algorithm are not within the time limit imposed by ITC 2007, due to the use of cooling schedules tuned, in this particular experiment, for the reference algorithm, the FastSA100. Doing this, we were able to analyse how slow the standard SA was with relation to the FastSA variants. Using as criteria the average percentage of the number of evaluations not performed, and also the execution time within the ITC 2007 rules, the FastSA100 is the best method. In the next section, the FastSA100 is compared with the state-of-the-art approaches.

4.3. Comparison with state-of-the-art approaches

In this section we perform a comparison between the FastSA100 approach and other state-of-the-art methodologies. The tested FastSA100 variant uses a cooling rate computed automatically as mentioned in Section 4.1.1. In the following tables, the compared approaches are identified by an acronym of the first author's name and publication's year. The compared authors are: Gog08 – (Gogos et al., 2008), Ats08 – Atsuta et al. - 2008 (McCollum et al., 2010), Sme08 – De Smet - 2008 (McCollum et al., 2010), Pil08 – Pillay - 2008 (McCollum et al., 2010), Mul09 – (Müller, 2009), Col09 – (McCollum et al., 2009), Dem12 – (Demeester et al., 2012), Gog12 – (Gogos et al., 2012), Byk13 – (Bykov & Petrovic, 2013), Ham13 – (Hamilton-Bryce et al., 2013), Alz14 – (Alzaqebah & Abdullah, 2014), Alz15 – (Alzaqebah & Abdullah, 2015), and Bat17 – (Battistutta et al., 2017).

The compared authors, in their papers, did not publish their source code, and few had published the solution files of the executed runs, preventing that a study based on distributions could be done. Thus, comparisons are done using the *best* and *average* of fitness values.

Table 5 presents the fitness values and computation times (in seconds) for the FastSA100 algorithm with rate computed automatically, tested on the ITC 2007 benchmark set.

Table 6 presents a comparison between the ITC2007's five finalists and the FastSA, considering the solutions' minimum fitness. In Table 7, the average results obtained by state-of-the-art approaches and by the FastSA are compared. The last column reports the standard deviation of the FastSA over ten runs. The results included in Table 7 are from approaches whose evaluation is compliant with the ITC 2007 rules, as mentioned in the original papers.

4.3.1. Statistical analysis

We now present a statistical analysis of the obtained average results (Table 7) for the complete ITC 2007 benchmark set (12 instances). We assessed the statistical significance of our results us-

Table 5
Fitness values and computation times (in seconds) on the ITC 2007 benchmark set for the FastSA100 algorithm with rate computed automatically. The minimum, median, maximum, average, and standard deviation values are shown for the fitness and computation time. The percentage of evaluations not done (marked in column '% less evals') is shown.

Instance	Fitness value						Computation time				
	% less evals	f_{min}	f_{med}	f_{max}	f_{avg}	σ	t_{min}	t_{med}	t_{max}	t_{avg}	σ
1	34	5050	5239.00	5390	5231.60	106.87	140	159.50	167	155.90	8.84
2	1	395	400.50	425	405.10	9.67	252	264.50	272	263.10	6.49
3	5	9574	9914.00	10,526	9940.10	286.19	228	235.50	242	235.60	4.77
4	57	12,299	13,060.00	13,555	12,992.50	433.25	84	95.50	101	93.70	5.91
5	7	3115	3480.00	4010	3490.40	230.06	167	175.50	193	176.90	7.64
6	9	25,750	25,947.50	26,345	26,008.50	221.45	211	222.50	248	226.40	11.69
7	5	4308	4548.50	4743	4534.70	138.63	227	236.50	241	235.00	4.67
8	17	7506	7603.00	7711	7602.60	68.02	186	192.00	203	192.00	4.76
9	28	977	1014.50	1071	1017.60	25.81	149	175.50	189	172.90	12.45
10	10	13,449	13,628.00	13,882	13,631.70	116.08	223	233.50	241	233.30	5.52
11	9	30,112	31,856.50	33,562	31,792.20	1288.02	191	206.50	215	205.30	7.09
12	21	5148	5193.50	5280	5204.80	42.36	165	181.50	200	182.70	10.02
Avg:	17										

Table 6
Minimum fitness obtained by the ITC 2007 finalists' approaches and by the FastSA. The best solutions are indicated in bold. "–" indicates that the corresponding instance is not tested or a feasible solution was not obtained.

Instance	Müller (2009)	Gogos et al. (2008)	Atsuta et al. - 2008 (McCollum et al., 2010)	De Smet - 2008 (McCollum et al., 2010)	Pillay - 2008 (McCollum et al., 2010)	FastSA
1	**4370**	5905	8006	6670	12,035	5050
2	400	1008	3470	623	2886	**395**
3	10,049	13,771	17,669	–	15,917	**9574**
4	18,141	18,674	22,559	–	23,582	**12,299**
5	**2988**	4139	4638	3847	6860	3115
6	26,585	27,640	29,155	27,815	32,250	**25,750**
7	**4213**	6572	10,473	5420	17,666	4308
8	7742	10,521	14,317	–	15,592	**7506**
9	1030	1159	1737	1288	2055	**977**
10	16,682	–	15,085	14,778	17,724	**13,449**
11	34,129	43,888	–	–	40,535	**30,112**
12	5535	–	5264	–	6310	**5148**

Table 7
Comparison of the average results obtained by the FastSA algorithm with state-of-the-art approaches for the ITC 2007 benchmark set. The best solutions are indicated in bold. "–" indicates that the corresponding instance was not tested or a feasible solution was not obtained.

Instance	Col09	Dem12	Gog12	Byk13	Ham13	Alz14	Alz15	Bat17	FastSA	
1	4799	6330.20	5032	4008	5469	5517.30	5227.81	**3926.96**	5231.60	(106.87)
2	425	612.50	**404**	**404**	450	537.90	457.55	407.72	405.10	(9.67)
3	9251	23,580.00	9484	**8012**	10,444	10,324.90	10,421.64	8849.46	9940.10	(286.19)
4	15,821	–	19,607	13,312	20,241	16,589.10	16,108.27	15,617.82	**12,992.50**	(433.25)
5	3072	5323.00	3158	**2582**	3185	3631.90	3443.72	2849.00	3490.40	(230.06)
6	25,935	28,578.13	26,310	**25,448**	26,150	26,275.00	26,247.27	26,081.35	26,008.50	(221.45)
7	4187	6250.00	4352	3893	4568	4592.40	4415.00	**3661.64**	4534.70	(138.63)
8	7599	9260.90	8098	**6944**	8081	8328.80	8225.81	7729.46	7602.60	(68.02)
9	1071	1255.90	–	**949**	1061	–	–	991.57	1017.60	(25.81)
10	14,552	16,113.33	–	**12,985**	15,294	–	–	13,999.56	13,631.70	(116.08)
11	29,358	–	–	**25,194**	44,820	–	–	27,781.50	31,792.20	(1288.02)
12	5699	5829.14	–	**5181**	5464	–	–	5550.20	5204.80	(42.36)

ing Friedman's test, as suggested in García and Herrera (2008). The results of the statistical tests were produced using the Java tool made available by the same authors.

Table 8 presents the algorithms' rankings based on their average performance. All algorithms are considered, even if they have missing values. The rankings are computed as in Demeester et al. (2012). The algorithm that produces the best solution on a given instance gets the lowest value, while the worst algorithm gets the highest value. The same procedure is applied to the average values obtained after ten runs on each instance. The final overall ranking is based on the average of the rankings over all instances. The algorithm with the overall lowest rank can be considered the best performing algorithm.

Table 8
Ranking of the analysed algorithms according to their average performance on the ITC 2007 benchmark set. The lower the rank, the better the algorithm's performance.

Algorithm	Ranking
Byk13	1.29
Bat17	2.67
Col09	3.50
FastSA	3.75
Ham13	5.67
Gog12	5.83
Alz15	6.63
Alz14	7.54
Dem12	8.13

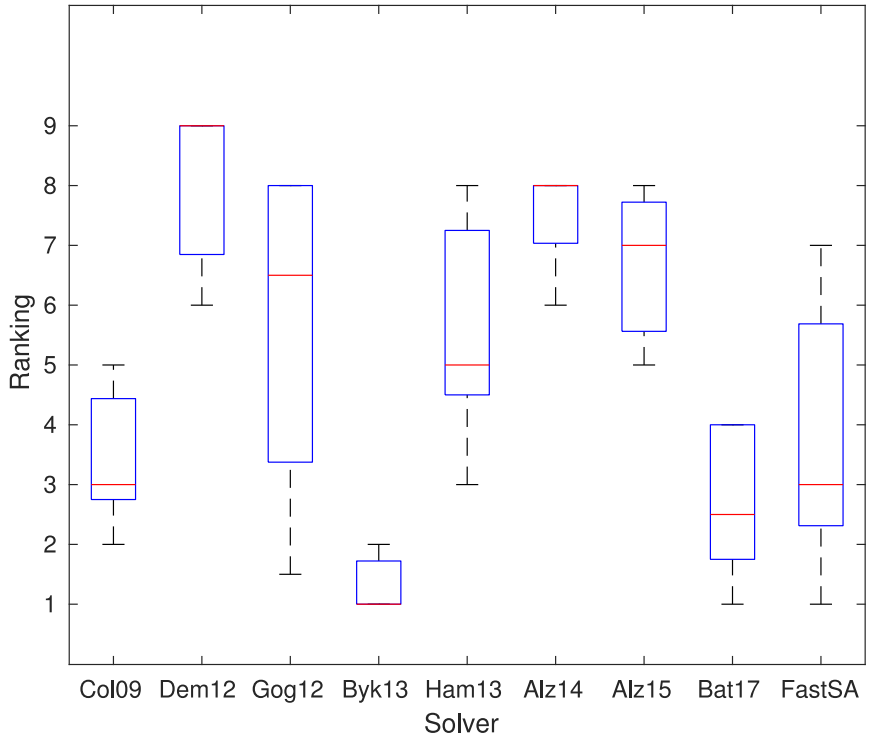


Fig. 7. Ranking distributions for the analysed solvers.

Table 9
Average ranking of Friedman's test for the compared methods, considering the complete data set (12 instances). The average values of the distributions were used in the ranking computation. The p-value computed by the Friedman test is 5.55×10^{-6}.

Algorithm	Ranking
Byk13	1.25
Bat17	2.58
FastSA	3.08
Col09	3.42
Ham13	4.67

Table 10
Adjusted p-values (Friedman) produced by running Holm's test (Byk13 is the control algorithm) for the algorithms compared in Table 9. Values marked in boldface represent statistically significant values.

Byk13 vs.	Unadjusted p	p_{Holm}
Ham13	$1.20\cdot10^{-7}$	$\mathbf{4.81\cdot10^{-7}}$
Col09	$7.89\cdot10^{-4}$	**0.002 40**
FastSA	0.0045	**0.009 00**
Bat17	0.0389	**0.0389**

Fig. 7 shows the ranking distributions (i.e., box plots) for each solver.

Table 9 summarises the rank obtained by the Friedman test (only algorithms that have values for all instances are considered). The p-value computed by the Friedman test is 5.55×10^{-6}, which is below the significance interval of 95% ($\alpha = 0.05$), confirming that there is a significant difference among the observed results. Byk13 is the best performing algorithm.

Table 10 shows the adjusted p-values obtained by the post-hoc Holm's test considering Byk13 as the control algorithm. The Holm's test confirms that Byk13 is better than all algorithms with $\alpha = 0.05$.

Table 11
Adjusted p-values (Friedman) of multiple algorithm comparison obtained by Holm's procedure. Only statistically significant values are shown (marked in boldface).

i	Hypothesis	Unadjusted p	p_{Holm}
1	Byk13 vs Ham13	$1.20\cdot10^{-7}$	$\mathbf{1.20\cdot10^{-6}}$
2	Col09 vs Byk13	$7.89\cdot10^{-4}$	**0.007 10**
3	Ham13 vs Bat17	0.0012	**0.0100**
4	Byk13 vs FastSA	0.0045	**0.0316**

Table 11 presents the adjusted p-values for the same studied scenario, obtained using the Holm's procedure, and showing pairwise comparisons. Only cases having significant differences are shown.

4.3.2. Discussion

Analysing the results, we observe that the FastSA is able to compete with the ITC 2007 finalists (Table 6), being superior on all instances except three. With respect to average results (Table 7), the FastSA is also able to compete with the state-of-the-art approaches obtaining the best result on one instance.

The employed statistical tests on the average results support our observations: the FastSA is ranked in third position among five algorithms (Table 9), where Byk13 is considered the best approach; Table 10, containing the adjusted p-values obtained by Holm's test comparing the control algorithm with the remaining algorithms ($1\times N$ comparison), allows to ascertain that Byk13 is better than all methods. Finally, analysing the results of the multiple algorithm comparison ($N\times N$) from Table 11, we can confirm almost the same conclusions of Table 10, excepting the hypothesis Byk13 vs Bat17 which was not rejected, and also ascertain that Bat17 is better than Ham13. From these tables, we can say that only Byk13 is better than FastSA. The presented conclusions are in accordance with rankings inferred from the ranking distributions illustrated in Fig. 7. These results have statistical significance.

5. Conclusions

In this work, a novel approach for solving examination timetabling problems is described. The proposed metaheuristic algorithm uses a graph colouring heuristic for solution initialisation and the SA metaheuristic for solution optimisation. Two optimisation algorithms were proposed. The first one is the original SA algorithm. The second one, named *FastSA*, is based on the SA algorithm but uses a modified acceptance criterion, which fixes the selected exam as soon as the exam's number of accepted moves in the previous bin is zero.

The developed SA and FastSA approaches were tested on the public ITC 2007 data set. Compared to the SA, the FastSA uses 17% less evaluations, on average. In terms of solution cost, the FastSA is competitive with the SA algorithm attaining the best average value in four out of twelve instances. Compared with the state-of-the-art approaches, the FastSA improves on one out of twelve instances, and ranks third out of five algorithms. An algorithm comparison was carried out confirming that only one algorithm, Byk13, is better than the FastSA. These results have statistical significance.

Future work will focus on four research lines. The first will encompass the use of the FastSA metaheuristic to the remaining ITC 2007 tracks (course timetabling and post-enrolment course timetabling tracks), and also to other problems where the SA can be applied. Application of the FastSA to the mentioned timetabling problems should be done in a similar way as that presented in this paper. Application to other optimisation problems will involve further investigation with the following goals: i) devise proper problem representation and move operator that can perturb a solution *element* (e.g., an exam in the ETP), ii) in each move, a randomly chosen *element* is perturbed if that *element* has a non-zero number of moves in the previous bin; otherwise, the element will freeze from that point on, and other elements are selected to move.

A second line of research includes the study and application of the FastSA on problems (e.g., automated analog Integrated Circuit design) where the evaluation function involves heavy computations or simulations. Because the FastSA is able to reduce the SA's number of evaluations for some instances, it is expected a considerable gain in performance on these instances.

The third research line will involve the study and design of other neighbourhood movements and its combination with the presented ones. For example, simpler neighbourhood operators, such as the ones used in Müller (2009), could be combined with Kempe chain ones. We could also use non-feasible operators and a penalisation scheme as it is done in Battistutta et al. (2017).

The fourth research direction will involve the study and application of the proposed framework to other simulated annealing variants such as the *threshold acceptance* (Dueck & Scheuer, 1990) algorithm. Another point of interest is the inclusion of a cut-off mechanism (Johnson, Aragon, McGeoch, & Schevon, 1989) in the FastSA and analysis of the improvements attained.

Funding

This work was supported by the Fundação para a Ciência e a Tecnologia [grant numbers UID/EEA/50009/2013, PEst-OE/EEI/LA0009/2013]; and the PROTEC Program funds [grant number SFRH/PROTEC/67953/2010].

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.eswa.2018.12.048.

CRediT authorship contribution statement

Nuno Leite: Conceptualization, Formal analysis, Investigation, Methodology, Project administration, Software, Validation, Visualization, Writing - original draft, Writing - review & editing. **Fernando Melício:** Formal analysis, Methodology, Project administration, Resources, Supervision, Validation, Writing - review & editing. **Agostinho C. Rosa:** Funding acquisition, Formal analysis, Methodology, Project administration, Resources, Supervision, Validation, Writing - review & editing.

References

Abramson, D. (1991). Constructing school timetables using simulated annealing: Sequential and parallel algorithms. *Management Science, 37*(1), 98–113. doi:10.1287/mnsc.37.1.98.

Ahmed, L. N., Özcan, E., & Kheiri, A. (2015). Solving high school timetabling problems worldwide using selection hyper-heuristics. *Expert Systems with Applications, 42*(13), 5463–5471. doi:10.1016/j.eswa.2015.02.059.

Alzaqebah, M., & Abdullah, S. (2014). An adaptive artificial bee colony and late-acceptance hill-climbing algorithm for examination timetabling. *Journal of Scheduling, 17*(3), 249–262.

Alzaqebah, M., & Abdullah, S. (2015). Hybrid bee colony optimization for examination timetabling problems. *Computers & Operations Research, 54*(0), 142–154. doi:10.1016/j.cor.2014.09.005.

Battistutta, M., Schaerf, A., & Urli, T. (2017). Feature-based tuning of single-stage simulated annealing for examination timetabling. *Annals of Operations Research, 252*(2), 239–254. doi:10.1007/s10479-015-2061-8.

Bellio, R., Ceschia, S., Gaspero, L. D., Schaerf, A., & Urli, T. (2016). Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. *Computers & Operations Research, 65*(Supplement C), 83–92. doi:10.1016/j.cor.2015.07.002.

Brélaz, D. (1979). New methods to color the vertices of a graph. *Communications of ACM, 22*(4), 251–256. doi:10.1145/359094.359101.

Burke, E. K., Qu, R., & Soghier, A. (2014). Adaptive selection of heuristics for improving exam timetables. *Annals of Operations Research, 218*(1), 129–145. doi:10.1007/s10479-012-1140-3.

Bykov, Y., & Petrovic, S. (2013). An initial study of a novel step counting hill climbing heuristic applied to timetabling problems. In *Proceedings of the 6th multidisciplinary international conference on scheduling: Theory and applications (MISTA 2013)* (pp. 691–693).

Carter, M. (1986). A survey of practical applications of examination timetabling algorithms. *Operations Research, 34*(2), 193–202. doi:10.1287/opre.34.2.193.

Carter, M., & Laporte, G. (1996). Recent developments in practical examination timetabling. In E. Burke, & P. Ross (Eds.), *Practice and theory of automated timetabling*. In *Lecture Notes in Computer Science: Vol. 1153* (pp. 1–21). Springer Berlin / Heidelberg.

Carter, M., Laporte, G., & Lee, S. Y. (1996). Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society, 47*(3), 373–383.

Cheong, C., Tan, K., & Veeravalli, B. (2009). A multi-objective evolutionary algorithm for examination timetabling. *Journal of Scheduling, 12*, 121–146.

Corne, D., Ross, P., & Fang, H.-L. (1994). Fast practical evolutionary timetabling. In T. C. Fogarty (Ed.), *Evolutionary computing*. In *Lecture Notes in Computer Science: Vol. 865* (pp. 250–263). Springer Berlin Heidelberg. doi:10.1007/3-540-58483-8_19.

Demeester, P., Bilgin, B., Causmaecker, P. D., & Berghe, G. V. (2012). A hyperheuristic approach to examination timetabling problems: Benchmarks and a new problem from practice. *Journal of Scheduling, 15*(1), 83–103.

Dowsland, K. A. (1990). A timetabling problem in which clashes are inevitable. *Journal of the Operational Research Society, 41*(10), 907–918. doi:10.1057/jors.1990.143.

Dueck, G., & Scheuer, T. (1990). Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics, 90*(1), 161–175. doi:10.1016/0021-9991(90)90201-B.

García, S., & Herrera, F. (2008). An extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons. *Journal of Machine Learning Research, 9*, 2677–2694.

Gogos, C., Alefragis, P., & Housos, E. (2008). A multi-staged algorithmic process for the solution of the examination timetabling problem. In *Proceedings of the 7th international conference on practice and theory of automated timetabling*. University of Montreal, Canada.

Gogos, C., Alefragis, P., & Housos, E. (2012). An improved multi-staged algorithmic process for the solution of the examination timetabling problem. *Annals OR, 194*(1), 203–221.

Hamilton-Bryce, R., McMullan, P., & McCollum, B. (2013). Directing selection within an extended great deluge optimization algorithm. In *Proceedings of the 6th multidisciplinary international conference on scheduling: Theory and applications (MISTA 2013)* (pp. 499–508).

Ingber, L. (2000). Adaptive simulated annealing (ASA): Lessons learned. CoRR. arXiv:0001018.

Johnes, J. (2015). Operational research in education. *European Journal of Operational Research, 243*(3), 683–696. doi:10.1016/j.ejor.2014.10.043.

Johnson, D. S., Aragon, C. R., McGeoch, L. A., & Schevon, C. (1989). Optimization by simulated annealing: An experimental evaluation; Part I, graph partitioning. *Operations Research, 37*(6), 865–892.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680. doi:10.1126/science.220.4598.671.

Kristiansen, S., & Stidsen, T. R. (2013). A comprehensive study of educational timetabling, a survey. *Technical Report*. Department of Management Engineering, Technical University of Denmark.

McCollum, B., McMullan, P., Parkes, A. J., Burke, E. K., & Abdullah, S. (2009). An extended great deluge approach to the examination timetabling problem. *Proceedings of the 4th Multidisciplinary International Scheduling: Theory and Applications 2009 (MISTA 2009)*, 424–434.

McCollum, B., McMullan, P., Parkes, A. J., Burke, E. K., & Qu, R. (2012). A new model for automated examination timetabling. *Annals of Operations Research, 194*, 291–315. doi:10.1007/s10479-011-0997-x.

McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A. J., et al. (2010). Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing, 22*(1), 120–130. doi:10.1287/ijoc.1090.0320.

Melício, F., Caldeira, J. P., & Rosa, A. (2004). Two neighbourhood approaches to the timetabling problem. In *Proceedings of the practice and theory of automated timetabling (PATAT'04)* (pp. 267–282).

Melício, F., Caldeira, P., & Rosa, A. (2000). Solving the timetabling problem with simulated annealing. In J. Filipe (Ed.), *Enterprise information systems* (pp. 171–178). Dordrecht: Springer Netherlands. doi:10.1007/978-94-015-9518-6_18.

Mühlenthaler, M. (2015). *Fairness in academic course timetabling*: VOL. 678. Springer International Publishing Lecture Notes in Economics and Mathematical Systems. 10.1007/978-3-319-12799-6.

Müller, T. (2009). ITC2007 solver description: A hybrid approach. *Annals of Operations Research, 172*(1), 429–446. doi:10.1007/s10479-009-0644-y.

Müller, T. (2016). Real-life examination timetabling. *Journal of Scheduling, 19*(3), 257–270. doi:10.1007/s10951-014-0391-z.

Nunes, M. (2015). Examination timetabling automation using hybrid meta-heuristics. Master's thesis.Lisbon, Portugal Instituto Superior de Engenharia de Lisboa.

Nunes, M., Ferreira, A., & Leite, N. (2016). Examination timetabling automation using hybrid meta-heuristics. *Third conference on electronics, telecommunications and computers (CETC2016)*. (Extended Abstract).

Özcan, E., Elhag, A., & Shah, V. (2012). A study of hyper-heuristics for examination timetabling. In *Proceedings of the 9th international conference on the practice and theory of automated timetabling (PATAT-2012)* (pp. 410–414).

Pillay, N. (2016). A review of hyper-heuristics for educational timetabling. *Annals OR, 239*(1), 3–38. doi:10.1007/s10479-014-1688-1.

Qu, R., Burke, E., McCollum, B., Merlot, L. T. G., & Lee, S. Y. (2009). A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling, 12*, 55–89.

Sabar, N. R., Ayob, M., Kendall, G., & Qu, R. (2015). A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Cybernetics, 45*(2), 217–228. doi:10.1109/TCYB.2014.2323936.

Schaerf, A. (1999). A survey of automated timetabling. *Artificial Intelligence Review, 13*(2), 87–127.

Shiau, D.-F. (2011). A hybrid particle swarm optimization for a university course scheduling problem with flexible preferences. *Expert Systems with Applications, 38*(1), 235–248. doi:10.1016/j.eswa.2010.06.051.

Talbi, E.-G. (2009). *Metaheuristics - From design to implementation*. Wiley.

Teoh, C. K., Wibowo, A., & Ngadiman, M. S. (2015). Review of state of the art for metaheuristic techniques in academic scheduling problems. *Artificial Intelligence Review, 44*(1), 1–21. doi:10.1007/s10462-013-9399-6.

The Second International Timetabling Competition (ITC 2007) (2007). Home page of ITC 2007. Accessed 29 November 2017 http://www.cs.qub.ac.uk/itc2007/.

Thompson, J. M., & Dowsland, K. A. (1996). Variants of simulated annealing for the examination timetabling problem. *Annals of Operations Research, 63*(1), 105–128. doi:10.1007/BF02601641.

Thompson, J. M., & Dowsland, K. A. (1998). A robust simulated annealing based examination timetabling system. *Computers & OR, 25*(7–8), 637–648.

Welsh, D. J. A., & Powell, M. B. (1967). An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal, 10*(1), 85–86. doi:10.1093/comjnl/10.1.85.

de Werra, D. (1985). An introduction to timetabling. *European Journal of Operational Research, 19*(2), 151–162. doi:10.1016/0377-2217(85)90167-5.

de Werra, D. (1997). The combinatorics of timetabling. *European Journal of Operational Research, 96*(3), 504–513. doi:10.1016/S0377-2217(96)00111-7.

Zhang, D., Liu, Y., M'Hallah, R., & Leung, S. C. (2010). A simulated annealing with a new neighborhood structure based algorithm for high school timetabling problems. *European Journal of Operational Research, 203*(3), 550–558. doi:10.1016/j.ejor.2009.09.014.