Journal of the Operational Research Society

ISSN: 0160-5682 (Print) 1476-9360 (Online) Journal homepage: http://www.tandfonline.com/loi/tjor20

Simulated annealing with improved reheating and learning for the post enrolment course timetabling problem

Say Leng Goh, Graham Kendall & Nasser R. Sabar

To cite this article: Say Leng Goh, Graham Kendall & Nasser R. Sabar (2018): Simulated annealing with improved reheating and learning for the post enrolment course timetabling problem, Journal of the Operational Research Society, DOI: 10.1080/01605682.2018.1468862

To link to this article: https://doi.org/10.1080/01605682.2018.1468862

Published online: 22 Jun 2018.

JOURNAL OF THE OPERATIONAL RESEARCH SOCIETY, 2018
NO. xx, https://doi.org/10.1080/01605682.2018.1468862

Simulated annealing with improved reheating and learning for the post enrolment course timetabling problem

Say Leng Goh[a], Graham Kendall[b,c] and Nasser R. Sabar[d]

[a]Universiti Malaysia Sabah Labuan International Campus, Labuan, Malaysia; [b]The University of Nottingham Malaysia Campus, Selangor Darul Ehsan, Malaysia; [c]University of Nottingham, University Park, Nottingham, UK; [d]Department of Computer Science and Information Technology, La Trobe University, Melbourne, Australia

ABSTRACT
In this paper, we utilise a two-stage approach for addressing the post enrolment course timetabling (PE-CTT) problem. We attempt to find a feasible solution in the first stage. The solution is further improved in terms of soft constraint violations in the second stage. We present an enhanced variant of the Simulated Annealing with Reheating (SAR) algorithm, which we term Simulated Annealing with Improved Reheating and Learning (SAIRL). We propose a reinforcement learning-based methodology to obtain a suitable neighbourhood structure for the search to operate effectively. We incorporate the average cost changes into the reheating temperature function. The proposed enhancements are tested on three widely studied benchmark datasets. Our algorithm eliminates the need for tuning parameters in conventional SA as well as neighbourhood structure composition in SAR. The results are highly competitive with SAR and other state of the art methods. In addition, SAIRL is scalable when the runtime is extended. The algorithm achieves new best results for 6 instances and new mean results for 14 instances.

ARTICLE HISTORY
Received 16 January 2017
Accepted 20 April 2018

KEYWORDS
Timetabling; combinatorial optimisation; reinforcement learning; tabu search with sampling and perturbation (TSSP); simulated annealing with improved reheating and learning (SAIRL)

1. Introduction

Education timetabling can be classified into three main classes, namely school timetabling, course timetabling, and examination timetabling (Schaerf, 1999). They are similar in some ways yet different mainly in terms of stakeholders and the constraints that have to be respected. In this paper, we focus on course timetabling, which is a placement of courses to a finite number of time slots and rooms, satisfying a set of requirements. The universal minimum requirement is that a student should not be required to attend two courses at the same time. This constraint is similar to the graph colouring problems in the sense that two nodes connected by an edge cannot be assigned the same colour. Timetabling construction has been shown to be NP-hard (Cooper & Kingston, 1996; de Werra, 1985). The complexity of a course timetabling problem varies as each institution has their own set of requirements. Due to its importance, various approaches have been proposed to address the problem. Introductions and surveys on timetabling can be found in de Werra (1985), Burke, Jackson, Kingston and Weare (1997), Schaerf (1999), Petrovic and Burke (2004), and Lewis (2008).

In this work, we propose a two-stage method to deal with the course timetabling problem. The first stage focuses on generating feasible solutions whereas second stage tries to further improve the generated solutions. We use Tabu Search with Sampling and Perturbation (TSSP) to find feasible solutions. We then improve the feasible solutions in terms of soft constraint violations by using Simulated Annealing with Improved Reheating and Learning (SAIRL) which is an enhanced version of Simulated Annealing with Reheating (SAR). In SAIRL, we propose a reinforcement learning method to adjust the composition of neighbourhood structures and an improved temperature reheating function. The proposed method is tested on three benchmark datasets for university course timetabling problems. We compare the results of SAIRL with the results of SAR as well as other state of the art methods.

This paper is organised as follows. Section 2 presents the description and formal presentation of the problem. Related work is reviewed in Section 3. The proposed methodology is described in Section 4 and the experimental results are presented in Section 5. The performance and behaviour of the algorithms are discussed in Section 5.5. Concluding remarks are given in Section 6. Finally, suggestions for future work are given in Section 7.

2. Problem description

Solving the problem instances involves assigning a set of E events (with a set of F features and attended by S students) to 45 time slots (5 days of 9 hours each) and a set of R rooms. The objective is to satisfy all hard constraints and minimise soft constraint violations as far as possible. The formal presentation of the problem is as follows:

CONTACT Say Leng Goh gohsayleng@yahoo.com

Given: set of events, $E = \{e_1, \ldots, e_{|E|}\}$
set of time slots, $T = \{1, \ldots, 45\}$
set of rooms, $R = \{r_1, \ldots, r_{|R|}\}$
set of students, $S = \{s_1, \ldots, s_{|S|}\}$
set of features $F = \{f_1, \ldots, f_{|F|}\}$
set of days, $D = \{1, \ldots, 5\}$
set of events that must appear later than e, A_e
set of events that must appear earlier than e, B_e

$$a_{s,e} = \begin{cases} 1 \text{ if student } s \text{ attends event } e \\ \quad e \in E, s \in S \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

$$b_{e,r} = \begin{cases} 1 \text{ if size of event } e \leq \text{capacity of room } r \\ \quad e \in E, r \in R \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

$$c_{e,r} = \begin{cases} 1 \text{ if } \sum_{f\in F} g_{e,f} \cdot h_{r,f} = \sum_{f\in F} g_{e,f} \quad e \in E, r \in R \\ 0 \text{ otherwise} \end{cases} \tag{3}$$

$$g_{e,f} = \begin{cases} 1 \text{ if event } e \text{ requires feature } f \\ \quad e \in E, f \in F \\ 0 \text{ otherwise} \end{cases} \tag{4}$$

$$h_{r,f} = \begin{cases} 1 \text{ if room } r \text{ has feature } f \\ \quad r \in R, f \in F \\ 0 \text{ otherwise} \end{cases} \tag{5}$$

$$i_{e,t} = \begin{cases} 1 \text{ if event } e \text{ can be assigned to time slot } t \\ \quad e \in E, t \in T \\ 0 \text{ otherwise} \end{cases} \tag{6}$$

$$j_{e,t_m} = \begin{cases} 1 \text{ if } \sum_{e_v\in A_e} x_{e_v t_n r} \cdot p_{t_m,t_n} \\ \quad = \sum_{e_v\in A_e} x_{e_v t_n r} \quad e \in E, r \in R, t \in T \\ 0 \text{ otherwise} \end{cases} \tag{7}$$

$$k_{e,t_m} = \begin{cases} 1 \text{ if } \sum_{e_v\in B_e} x_{e_v t_n r} \cdot q_{t_m,t_n} \\ \quad = \sum_{e_v\in B_e} x_{e_v t_n r} \quad e \in E, r \in R, t \in T \\ 0 \text{ otherwise} \end{cases} \tag{8}$$

$$p_{t_m,t_n} = \begin{cases} 1 \text{ if } t_m < t_n \quad t \in T \\ 0 \text{ otherwise} \end{cases} \tag{9}$$

$$q_{t_m,t_n} = \begin{cases} 1 \text{ if } t_m > t_n \quad t \in T \\ 0 \text{ otherwise} \end{cases} \tag{10}$$

$$x_{e,t,r} = \begin{cases} 1 \text{ if event } e \text{ is assigned to time slot } t \\ \quad \text{and room } r \quad e \in E, r \in R, t \in T \\ 0 \text{ otherwise} \end{cases} \tag{11}$$

$$y_{s,t} = \begin{cases} 1 \text{ if} x_{etr} \cdot a_{se} = 1 \quad e \in E, r \in R, s \in S, t \in T \\ 0 \text{ otherwise} \end{cases} \tag{12}$$

$$z_{s,d} = \begin{cases} 1 \text{ if } \sum_{t=(d-1)\times 9+1}^{d\times 9} y_{s,t} = 1 \quad d \in D, s \in S, t \in T \\ 0 \text{ otherwise} \end{cases} \tag{13}$$

Minimize:

$$\sum_{i=1}^{3} SC_i \tag{14}$$

SC_1: Penalty for students with one event on a day

$$\sum_{s\in S} \sum_{d=1}^{5} z_{s,d} \tag{15}$$

SC_2: Penalty for students with three or more events consecutively.

$$\sum_{s\in S} \sum_{d=1}^{5} \sum_{t=(d-1)\times 9+1}^{d\times 9-2} y_{s,t} \cdot y_{s,(t+1)} \cdot y_{s,(t+2)} \tag{16}$$

SC_3: Penalty for students with one event in the last time slot of the day

$$\sum_{s\in S} \sum_{t\in\{9,18,\ldots 45\}} y_{s,t} \tag{17}$$

Subject to:

HC1: No student must attend more than one event at the same time.

$$\sum_{r\in R} x_{etr} \cdot a_{se} \leq 1 \quad e \in E, s \in S, t \in T \tag{18}$$

HC2: Each event is assigned a room with enough seats for all attending students and all features required.

$$b_{er} \cdot c_{er} \cdot x_{etr} = x_{etr} \quad e \in E, r \in R, t \in T \tag{19}$$

HC3: Only one event per room in any time slot.

$$\sum_{e\in E} x_{etr} \leq 1 \quad r \in R, t \in T \tag{20}$$

HC4: Events are assigned to designated time slots.

$$i_{et} \cdot x_{etr} = x_{etr} \quad e \in E, r \in R, t \in T \tag{21}$$

HC5: Where specified, events should be scheduled in the correct order.

$$j_{e,t_m} \cdot k_{e,t_m} \cdot x_{et_m r} = x_{et_m r} \quad e \in E, r \in R, t \in T \tag{22}$$

Table 1. Statistics for the Socha data-set.

Instance	S	M	L
Event	100	400	400
Room	5	10	10
Feature	5	5	10
Student	80	200	400

The data-sets utilised in this research are publicly available and regarded as the standard benchmarks. Optimal solutions (zero hard and soft constraint violations) are known to exist for many instances of each data-sets.

- **Socha with 11 instances (http://iridia.ulb.ac.be/supp/IridiaSupp2002-001/index.html).** The instances (5 small, 5 medium, and 1 large) are generated using an algorithm developed by Ben Paechter (http://www.soc.napier.ac.uk/$\sim$benp). The time limit for the small, medium, and large instances is set to 90, 900, and 9000 seconds, respectively (Socha, Knowles, & Sampels, 2002). However, such a restriction does not promote a fair comparison as different machines may have different specifications. Therefore, we use a benchmark time limit set by International Timetabling Competition 2002 (ITC02). Refer to Table 1 for the benchmark statistics.
- **International Timetabling competition 2002 (ITC02) with 20 instances (http://www.idsia.ch/Files/ttcomp2002).** This competition was organised by the Metaheuristic Network and the instances were also generated by Ben Paechter. The time limit is benchmarked by running a programme on the host machine, which enables a fair comparison. A machine with a higher specification will be allowed to run longer and vice versa. Our machine is entitled to 190s. Refer to Table 2 for the benchmark statistics.
- **International Timetabling Competition 2007 (ITC07) with 24 instances (http://www.cs.qub.ac.uk/itc2007).** The time limit is benchmarked in the same way as ITC02. Refer to Table 3 for the benchmark statistics.

3. Related work

The Socha instances have been widely used as course timetabling benchmarks for algorithmic comparison. Various approaches have been tested on the instances since their inception. Burke, Kendall and Soubeiga (2003) proposed a method called Tabu Search Hyper-Heuristic to overcome the weaknesses of optimisation methods specifically meta-heuristics, which often require intensive parameter tuning for individual instances. They aimed to develop a general approach which can be easily applied to different problems, yet remains competitive with state of the art approaches. The method selects heuristics at each decision point instead of tackling the problem directly. Heuristics were ranked according to their performance inspired by the principles of reinforcement learning. The value of the selected heuristic is increased by one when applied and which resulted in an improvement to the current cost function. Otherwise, it is decreased by one. A tabu list was also implemented to restrict the use of heuristics which did not perform well recently based on First-In, First-Out (FIFO). A heuristic placed in the list is made tabu even if it has the highest rank. The approach was competitive with ant systems and random restart local search.

Obit et al. proposed a Non-Linear Great Deluge with reinforcement learning (Obit et al., 2009) for the course timetabling problem. Heuristics or neighbourhood structures, were selected probabilistically based on their weights instead of randomly. The weights were increased or decreased based on their performance. Two types of Modified Choice Function (MCF) are investigated, namely MCF with static memory and MCF with a random learning rate. For MCF with static memory, a reward of one point is awarded to the chosen heuristic if the current solution is improved, otherwise no point is awarded. The weights are updated at predefined periods. For MCF with random learning rate, a different set of rewards was used according to the difference between the best cost and the current cost. In addition, the reward was weighted by a random value in the range (0.5, 1.0]. The method involved the acceptance and rejection of solutions using Non-Linear Great Deluge acceptance criterion.

Ceschia et al. applied a highly tuned simulated annealing method on the instances and reported superior results (Ceschia, Di Gaspero, & Schaerf, 2012). Goh et al. recently utilized TSSP and SAR algorithms on these instances (Goh, Kendall, & Sabar, 2017). Their method reported good results and is the current state of the art methodology. Other approaches applied to Socha instances include Ant Systems (Ejaz & Javed, 2007; Jaradat & Ayob, 2010; Socha et al., 2002) Variable Neighborhood Search (Abdullah, Burke, & McCollum, 2005), Memetic Algorithm (Abdullah, Burke, & McCollum, 2007) Non Linear Great Deluge (Landa-Silva & Obit, 2008), Great Deluge (Abdullah, Shaker, McCollum, & McMullan, 2009) Fish Swarm Intelligent Algorithm (Turabieh, Abdullah, McCollum, & McMullan, 2010) and Honey Bee Mating (Sabar, Ayob, Kendall, & Qu, 2012).

Kostuch won the International Timetabling Competition 2002 (ITC02) using simulated annealing algorithm (Kostuch, 2003). The other entries with good performance were Great Deluge (Burke, Bykov, Newall, & Petrovic, 2003) and Tabu Search (Cordeau, Jaumard, & Morales, 2003; Arntzen & Lokketangen, 2003). Differ-

Table 2. Statistics for the ITC02 data-set.

Instance	1	2	3	4	5	6	7	8	9	10
Event	400	400	400	400	350	350	350	400	440	400
Room	10	10	10	10	10	10	10	10	11	10
Feature	10	10	10	5	10	5	5	5	6	5
Student	200	200	200	300	300	300	350	250	220	200
Instance	11	12	13	14	15	16	17	18	19	20
Event	400	400	400	350	350	440	350	400	400	350
Room	10	10	10	10	10	11	10	10	10	10
Feature	6	5	6	5	10	6	10	10	5	5
Student	220	200	250	350	300	220	300	200	300	300

ent approaches have been attempted to solve the problem after competition. Chiarandini proposed a simulated annealing method with better or comparable results (Chiarandini, Birattari, Socha, & Rossi-Doria, 2006). Around the same time, Kostuch published the results of his improved method which achieved the best known results for all the instances (Kostuch, 2005). Ceschia et al. utilised a highly tuned simulated annealing method for ITC02 instances, however their results were inferior to Kostuch's (Ceschia et al., 2012). Goh et al. tested TSSP and SAR algorithms on these instances (Goh et al., 2017). Their method is comparable or better than that of Kostuch.

Cambazard was the winner for the International Timetabling Competition 2007 (ITC07) (Cambazard, Hebrard, OSullivan, & Papadopoulos, 2012). Most of the competitive entries are simulated annealing algorithms (Chiarandini, Fawcett, & Hoos, 2008; Lewis, 2012). Nothegger et al. employed Ant Colony Optimisation (ACO) (Nothegger, Mayer, Chwatal, & Raidl, 2012). The authors remarked that a simulated annealing as a local search component played a critical role in the performance of their method. Mueller utilised a hybrid approach comprising of a hill climbing phase, great deluge, and simulated annealing (Muller, 2009). Several competitive methods have been published after the competition such as Ceschia et al. (2012), Lewis and Thompson (2015) and Goh et al. (2017).

It is noticeable that most of the competitive methods were based on simulated annealing. The majority of them are either highly tuned (Ceschia et al., 2012) or focused on certain instances (Lewis & Thompson, 2015), suggesting a disadvantage of simulated annealing which may require intensive parameter tuning, even for specific instances of the same problem to obtain high-quality solutions.

4. Proposed methodology

The proposed methodology consists of two stages: construction state and improvement stage. In the first stage, we attempt to find a feasible solution which satisfies all the hard constraints. Once a feasible solution is found, it is improved in term of the soft constraint violations in the second stage. The pseudocode of the proposed two stages algorithm is shown in Algorithm 1, with further details below.

Algorithm 1

```
1: procedure TIMETABLECONSTRUCTIONANDIMPROVEMENT
2:     $best \leftarrow$ empty
3:     $E \leftarrow$ list of events
4:     $unassignedE \leftarrow E$
5:
6:     TSSP($best, unassignedE$)        ▷ Stage 1: Finding a feasible solution
7:     if $unassignedE$ is empty then
8:         SAIRL($best, E$)        ▷ Stage 2: Improving soft constraint violations
9:     end if
10: end procedure
```

4.1. Stage 1: Finding a feasible solution

In this stage, a feasible solution is built constructively by using Tabu Search with Sampling and Perturbation (TSSP) (Goh et al., 2017). Only if a feasible solution is found (*unassignedE* is empty), is it passed to Stage 2 for soft constraint improvement. The TSSP procedure is not new and has been previously applied to timetabling problems (Goh et al., 2017). The TSSP pseudocode is shown in Algorithm 2. In TSSP, a neighbour move involves moving an event from the list of unplaced events *unplacedE* to a time slot in the current solution *current*. At the start of each iteration, S number of events are selected randomly from *unplacedE* and added to *sampleE* list. The event sampling size S is set as 0.25% of the number of events e.g., S is 1, 2 and 3 for the number of events between 1–400, 401–800 and 801–1200, respectively. A sample of neighbour moves are evaluated by considering all non-tabu suitable time slots for the events in *sampleE* (lines 10–24). The event e is temporarily removed from *unplacedE*. To feasibly move an event into a particular time slot, minimal conflicting events (violated clash or precedence constraint) are moved from *current* to *unplacedE* list. Matching is used for room assignment only when necessary. If matching could not find a room for the event considered, a room is chosen randomly among the rooms suitable and the related event is moved from *current* to *unplacedE*.

The cost function f used to evaluate solutions (*current*, *candidate*, *best*) is based on the number of unplaced events plus the clash ratio:

$$\sum_{e \in unplacedE} 1 + \frac{clash[e]}{clashSum} \tag{23}$$

Algorithm 2 Goh et al. (2017)

```
1: procedure TSSP($best, unassignedE$)
2:     $unplacedE \leftarrow unassignedE$
3:     $current \leftarrow best$
4:     $f(best) \leftarrow f(current)$
5:     $ITER \leftarrow room^3$
6:     $i \leftarrow 0$
7:     while $unplacedE$ is not empty AND $time.elapsed() < T$ do
8:         $sampleE \leftarrow$ select $S$ events randomly from $unplacedE$
9:         $min \leftarrow \infty$
10:        for all $e \in sampleE$ do
11:            $unplacedE \leftarrow unplacedE - e$
12:            for all $s \in S$ | S non-tabu slot suitable for $e$ do
13:                $current \leftarrow current -$ {events conflicting $e$}
14:                $unplacedE \leftarrow unplacedE \cup$ {events conflicting $e$}
15:                if $f(candidate) < min$ then
16:                    $bestEvent \leftarrow e$
17:                    $bestSlot \leftarrow s$
18:                    $min \leftarrow f(candidate)$
19:                end if
20:                $unplacedE \leftarrow unplacedE -$ {events conflicting $e$}
21:                $current \leftarrow current \cup$ {events conflicting $e$}
22:            end for
23:            $unplacedE \leftarrow unplacedE \cup e$
24:        end for
25:        $current \leftarrow current -$ {events conflicting $bestEvent$}
26:        $current \leftarrow current \cup bestEvent$        ▷ $bestSlot$
27:        $f(current) \leftarrow min$
28:        if $f(current) < f(best)$ then
29:            $best \leftarrow current$
30:            $f(best) \leftarrow f(current)$
31:            $unassignedE \leftarrow unplacedE$
32:        end if
33:        set tabu {events conflicting $bestEvent$} from original time slots
34:        $unplacedE \leftarrow unplacedE - bestEvent$
35:        $unplacedE \leftarrow unplacedE \cup$ {events conflicting $bestEvent$}
36:        if $i = ITER$ then
37:            PERTURB($current$)
38:            $i \leftarrow 0$
39:            reset tabu list
40:        end if
41:        $i = i + 1$
42:    end while
43: end procedure
```

where $clash[e]$ is the number of clashes with other events and $clashSum$ is the total number of clashes of all events. Effectively, the candidate solution with the lowest number of unplaced events is preferred and ties are broken using the number of clashes.

The events conflicting with e in $unplacedE$ are moved back to $current$ before evaluating the next non-tabu time slot. After evaluating all the non-tabu time slots, e is placed back to $unplacedE$ before the next event is considered. The best neighbour move is recorded as $bestEvent$ and $bestSlot$ (lines 16–17).

The best neighbour move is applied to $current$ where the $bestEvent$ is moved from $unplacedE$ to the $bestSlot$ (line 26) after extracting the events conflicting with $bestEvent$ from $current$. $best$ and $f(best)$ are updated if $f(current)$ is better than $f(best)$. The extracted events are made tabu from returning to their original time slots for a number of iterations (line 33) according to the tabu tenure:

$$\text{RANDOM}[10] + |unplacedE| \tag{24}$$

where $|unplacedE|$ is the number of unplaced events. The $bestEvent$ is removed from $unplacedE$ and the extracted events are placed into $unplacedE$.

We perturb the current solution at certain iteration intervals $ITER$. If $i = ITER$, $current$ is perturbed (Algorithm 3), i is reset to 0 and tabu list is reset. We try to move each assigned event to each time slot (except the time slot it currently occupies) in $slotList$ (shuffled randomly) by using either a swap or a Kempe operator. The event is moved only if the move is feasible or does not violate any hard constraints. $ITER$ is set as $room^3$ (line 5).

Algorithm 3

```
1: procedure PERTURB($solution$)
2:     for all $e \in solution$ do
3:         SHUFFLE($slotList$)
4:         for all $slot \in slotList$ do
5:             if RANDOM[0, 2) = 1 then
6:                 if SWAP($solution, e, slot$) then
7:                     break;
8:                 end if
9:             else
10:                if KEMPE($solution, e, slot$) then
11:                    break;
12:                end if
13:            end if
14:        end for
15:    end for
16: end procedure
```

The neighbourhood structures used in the PERTURB procedure are:

- Swap: A swap is attempted between e with event in each room (room list shuffled randomly) in $slot$. A swap is carried out if all the hard constraints are satisfied.
- Kempe: Kempe chain interchange (Chiarandini et al., 2006; Lewis & Thompson, 2015; Thompson & Dowsland, 1996) is attempted. A chain is built between events in a time slot occupied by e (time slot A) and events in $slot$ (time slot B). Initially, e is added to the chain. Events in time slot A and B which clash with events currently in the chain are incrementally added to the chain. When the chain is complete, the events in time slot A are moved to time slot B and vice versa if all the hard constraints are satisfied.

4.2. Stage 2: Improving soft constraint violations

In this stage, we improve the feasible solution in term of soft constraint violations by using a method based on simulated annealing (SA) as it has been shown to be very effective in tackling various combinatorial optimisation problems, particularly timetabling problems. In this work, we propose an enhanced variant of SA which we term as Simulated Annealing with Improved

Reheating and Learning (SAIRL). The proposed SAIRL is build upon our recent SA algorithm which integrates SA with Reheating mechanism (SAR) (Goh et al., 2017). SAR was inspired by the idea that when the current cost is high, the search should explore more and when the current cost is low, the search should exploit more. To implement the idea, the current cost was used to determine the initial and reheated temperature, which in turn determines the exploration and exploitation nature of the search. SAR eliminated the need for tuning certain parameters in a conventional SA such as the initial temperature, the final temperature and the Markov chain length. Good results were reported. However, one drawback of SAR is having to pre-set the composition of neighbourhood structures for each data-set in order to obtain good results. Thus, it is difficult to set the right composition in advance as the effectiveness is dependent on the instance. Another drawback of SAR is the limitation of using the current cost exclusively to determine the level of reheated temperature as different instances may require different level of exploration to search effectively. To address these issues, we propose several enhancements based on these shortcomings. We term the improved algorithm Simulated Annealing with Improved Reheating and Learning (SAIRL). The details of SAIRL are shown in Algorithm 4.

Algorithm 4

```
procedure SAIRL($current, E$)
    $temp \leftarrow f(current) \times C$
    $heat \leftarrow 0$
    $best \leftarrow current$
    $previousCost \leftarrow f(current)$
    $currentStagnantCount \leftarrow 0$
    $stuckBestCost \leftarrow f(current)$
    $stuckCurrentCost \leftarrow f(current)$
    $NS \leftarrow \{ns_1, \ldots, ns_{|NS|}\}$

    while $terminationCondition$ = false do
        for all $e \in E$ do
            $moved \leftarrow$ false
            for $slot = 1$ to 45 do
                $ns_k \leftarrow$ SELECTNEIGHBOURHOODSTRUCTURE($NS$)
                $ns_k.visit++$
                $candidate \leftarrow$ GETCANDIDATE($current, e, slot, n$)
                if $candidate$ exists then
                    if RANDOM[0,1) $\leq \exp\left(-\frac{f(candidate)-f(current)}{temp}\right)$ then
                        $moved \leftarrow$ true
                        $current \leftarrow candidate$
                        if $f(current) < f(best)$ then
                            $best \leftarrow current$
                        end if
                        update $ns_k.value$
                    else
                        update $ns_k.value$
                    end if
                else
                    update $ns_k.value$
                end if
                if $moved$ then
                    break
                end if
            end for
        end for
        if STUCK($f(current), previousCost, currentStagnantCount$) then
            if $f(best) = stuckBestCost$ then
                if $f(current) - stuckCurrentCost < 2\%$ then
                    $heat = heat + 1$
                else
                    $heat \leftarrow 0$
                end if
            else
                $heat \leftarrow 0$
            end if
            $temp \leftarrow [heat \times 0.2 \times f(current) + f(current)] \times \overline{\Delta f} \times D$
            $stuckBestCost \leftarrow f(best)$
            $stuckCurrentCost \leftarrow f(current)$
        else
            $temp \leftarrow temp \times \beta$
        end if
        $previousCost \leftarrow f(current)$
    end while
end procedure
```

Algorithm 5

```
procedure STUCK($f(current), previousCost, currentStagnantCount$)
    if $f(current) - previousCost < 1\%$ then
        $currentStagnantCount = currentStagnantCount + 1$
    else
        $currentStagnantCount \leftarrow 0$
    end if
    if $currentStagnantCount > 5$ then
        return true
    else
        return false
    end if
end procedure
```

Like SAR, an initial temperature is cooled statically according to an update rule $T_{i+1} = T_i \times \beta$. At each temperature, a Markov chain is generated by trying to move each event $e \in E$ into a time slot using a neighbourhood structure selected probabilistically from the given set of neighbourhood structures (transfer, swap and Kempe (Thompson & Dowsland, 1996)). Probabilistic selection ensures that the less favorable neighbourhood structures can still be selected but the better neighbourhood structures are more likely to be selected. Maximal matching is used for room assignment.

Instead of using a pre-set composition of neighbourhood structures as is the case in SAR, we propose a method based on reinforcement learning (RL) to obtain a balanced composition of neighbourhood structures. The method is inspired by the work in Lewis and Thompson (2015) that suggested the use of a feasibility ratio to estimate the solution space connectivity. They presented the relationship between feasibility ratio and performance of various neighbourhood structures where the neighbourhood structure with a higher feasibility ratio is preferred. However, we feel that solutions may still be disconnected by the acceptance criterion, restricting the movements within the solution space. Therefore, we believe that acceptance ratio is a more suitable indicator for the solution space connectivity. In other words, neighbourhood structures with higher acceptance rates should be favored. Meanwhile, through observation, we find that different neighbourhood structures have different acceptance rates and computational costs for different instances. Some neighbourhood structures may have lower acceptance rates but are less computationally expensive which allows more transitions per time unit. Therefore, the objective is to maximise the number of accepted moves per time unit, with the hope that solution space connectivity can be improved.

In our implementation, a *visit* and a *value* are maintained for each neighbourhood structure ns_k. Note that $ns_k.visit$ is incremented by one each time the neighbourhood structure ns_k is selected (line 16). Meanwhile, $ns_k.value$ is updated (lines 25, 27 and 30) as a cumulative mean of rewards:

$$ns_k.value \leftarrow ns_k.value + \frac{reward - ns_k.value}{ns_k.visit} \tag{25}$$

The reward is defined as:

$$reward = \begin{cases} 0, & \text{if candidate is accepted} \\ \text{CPU time}, & \text{otherwise} \end{cases} \tag{26}$$

A zero reward is awarded to the neighbourhood structure ns_k if the candidate solution is accepted (line 25). Otherwise, the neighbourhood structure ns_k is penalised with CPU time (elapsed time since selection) if the candidate is rejected (line 27) or the candidate does not exist because a move is not feasible (line 30). Initially, all neighbourhood structures have an equal probability of being selected. Over time, the probability varies according to:

$$P_{ns_k} = \frac{\frac{1}{ns_k.value}}{\sum_{k=1}^{|NS|} \frac{1}{ns_k.value}} \tag{27}$$

The candidate solution is evaluated using the acceptance criterion where the improving and equal cost solution is accepted while the worsening solution is accepted with a certain probability. If accepted, the candidate solution will be set as the current solution. If the current solution is better than the best solution, the best solution is updated.

After each Markov chain, the STUCK procedure (Algorithm 5) checks whether the search is stuck in a local optima. If the search is stuck, the temperature is reheated. In SAR, the temperature is reheated according to:

$$temp \leftarrow [heat \times 0.2 \times f(current) + f(current)] \times C \tag{28}$$

where C is a coefficient which determines the exploration level of the reheated temperature and *heat* is an incremental step. In SAIRL, we incorporate the average cost changes of uphill and downhill moves ($\overline{\Delta f}$) into the reheating function:

$$temp \leftarrow [heat \times 0.2 \times f(current) + f(current)] \times \overline{\Delta f} \times D \tag{29}$$

where D is a coefficient. The temperature is cooled again until the search is stuck in another local optima. If the search is found to be stuck in the previous local optima, a higher temperature is applied for the next reheating so that the search can explore more. The procedure (a series of cooling and reheating) is repeated until the *terminationCondition* is true when either the elapsed time exceeds the runtime t or an optimal solution is obtained (note that all instances are known to have a zero-cost solution).

We set the decay rate β to 0.9995 and the coefficient D to 0.001. To allow a fair comparison between SAR and SAIRL, the initial temperature is set to the same value used in SAR where the initial cost is multiplied by the coefficient C=0.01 (1% of the initial cost). The same settings are used across all instances in our experiments.

5. Experimental results

We performed the experiments on an Intel Xeon (3.1 GHz) with 4Gb RAM machine. Java is used to code the algorithms. The computation time limit allowed by running the benchmark programme (http://www.idsia.ch/Files/ttcomp2002) is T=190 seconds for each single run. When a feasible solution is found, the focus is switched to minimising soft constraint violations by using the remaining available time. Each run will stop when the time limit is reached. A total of 31 runs were executed for each instance. This section divided into five subsections. The first and second subsections discuss the benefit of proposed enhancements on the performance of SA and SAR. The third subsection compares the results of the proposed algorithm with the state of the art methods. The fourth subsection presents the results of the proposed algorithm using extended run time. The discussion on the performance of the proposed algorithm is presented in last subsection.

5.1. The effect of learning and improved reheating

For SAR, the neighbourhood structure composition is set manually for specific data-sets. The composition is 70:29:1 (Socha/ITC02 instances) and 70:20:10 (ITC07 instances) for Transfer:Swap:Kempe operators. Meanwhile, a reinforcement learning-based method is used in SARL to optimise the composition as the search progresses. In effect, the need for manual setting of the neighbourhood structure composition (as required in SAR) is eliminated. Consequently, the performance of SARL (a transition algorithm) is affected as shown by the total average of soft constraint violations in Table 4.

Table 3. Statistics for the ITC07 data-set.

Instance	1	2	3	4	5	6	7	8
Event	400	400	200	200	400	400	200	200
Room	10	10	20	20	20	20	20	20
Feature	10	10	10	10	20	20	20	20
Student	500	500	1000	1000	300	300	500	500
Instance	9	10	11	12	13	14	15	16
Event	400	400	200	200	400	400	200	200
Room	10	10	10	10	20	20	10	10
Feature	20	20	10	10	10	10	20	20
Student	500	500	1000	1000	300	300	500	500
Instance	17	18	19	20	21	22	23	24
Event	100	200	300	400	500	600	400	400
Room	10	10	10	10	20	20	20	20
Feature	10	10	10	10	20	20	30	30
Student	500	500	1000	1000	300	500	1000	1000

Table 4. Comparing average of soft constraint violations between SAR, SARL, and SAIRL on data-sets.

	Algorithm		
Dataset	SAR	SARL	SAIRL
Socha	20.52	**20.31**	21.67
ITC02	**24.17**	25.28	24.60
ITC07	171.05	204.81	**126.38**
Total	87.53	102.63	**68.43**

Note: $n = 31$ runs for each instance in the data-set.

We further improve SARL by incorporating the average cost changes into the reheated temperature function in SAIRL. As evident in Table 4, each algorithm recorded the lowest average of soft constraint violations for each data-set. The averages produced by these algorithms are comparable for Socha and ITC02 data-sets. Meanwhile, SAIRL clearly outperforms SAR and SARL on the ITC07 data-set. Overall, SAIRL seems to be the most effective algorithm.

5.2. Comparing SAIRL with SAR

Here, we compare the performance of SAR and SAIRL in minimising the soft constraint violations. For Socha instances, SAIRL is comparable to SAR as shown in Table 5. The p values reveal that there is no significant difference between the means of SAR and SAIRL for all the instances except M2 where SAR is better than SAIRL.

Results comparison between SAIRL and SAR for ITC02 instances is shown in Table 6. The t-tests show that SAR performed better than SAIRL for instances 1, 2, 9, 16, 18. Meanwhile, SAIRL is more effective for instances 5 and 17. There is no significant difference between the means of both methods for the rest of the instances.

For ITC07 instances, SAIRL performed significantly better compared to SAR for instances 1, 2, 3, 9, 11, 15, 16, 19, 24 as shown in Table 7. SAR is better than SAIRL for instances 14 and 23. No significant difference is evident between the means of both methods for the rest of the instances.

On the whole, SAR is significantly better than SAIRL on 8 instances. Meanwhile, SAIRL is significantly better than SAR on 11 instances. t-tests do not reveal a statistically significant difference between the mean of the two algorithms for the rest of the instances.

5.3. Comparing SAIRL with state of the art methods

We now compare SAIRL with the best results in the literature. Table 8 summarises the details of the solvers we use for comparison.

SAIRL outperformed all the other solvers for all Socha instances except solver R which we attempt to improve in this work as shown in Table 9. Both SAIRL and solver R found optimal solutions for 9 out of 11 instances. In addition, SAIRL achieved a new best result for instance M3.

Results comparison for ITC02 is given in Table 10. Our results are competitive or better than the other solvers on all the instances. In fact, SAIRL managed to get optimal solutions for 7 out of 20 instances in comparison to the solver J2 (four) and solver R (seven). Furthermore, SAIRL obtained four new best results (instance 5, 12, 14 and 17) and four new means (instance 5, 7, 12 and 17).

Table 11 shows the results comparison for ITC07. Our results are competitive compared to the other solvers. SAIRL found eighteen optimal solutions compared to the solver Q (seventeen) and solver R (fifteen). SAIRL achieved one new best result (instance 22) and ten new means (instance 1, 2, 9, 11, 12, 15, 16, 19, 22, 24). The solver P did not attempt their methods on instances 17-24. Perhaps, these instances are not accessible at that time as they were initially hidden.

5.4. Extended runtime for SAIRL

Lastly, we performed some experiments to see the effects of an extended runtime with regard to soft constraint violations on selected instances. The algorithm was ran for five times the time limit or $5T$ (950s). As

Table 5. Comparison between SAR and SAIRL on Socha instances. Depicted is best(mean) of soft constraint violations.

Inst.	SAR	SAIRL	*t*-test (*p* value)
S1	**0**(**0.0**)	**0**(**0.0**)	–
S2	**0**(**0.0**)	**0**(**0.0**)	–
S3	**0**(**0.0**)	**0**(**0.0**)	–
S4	**0**(**0.0**)	**0**(**0.0**)	–
S5	**0**(**0.0**)	**0**(**0.0**)	–
M1	**0**(**1.5**)	**0**(2.32)	0.057
M2	**0**(**2.2**)	**0**(3.58)	**0.007**
M3	7(**13.4**)	**6**(14.39)	0.443
M4	**0**(**0.7**)	**0**(1.35)	0.073
M5	**0**(**1.2**)	**0**(1.42)	0.600
L	**165**(**206.6**)	181(215.19)	0.127

Note: $n = 31$ runs.

Table 6. Comparison between SAR and SAIRL on ITC02 instances. Depicted is best(mean) of soft constraint violations.

Inst.	SAR	SAIRL	*t*-test (*p* value)
1	**23**(**32.6**)	26(37.0)	**0.004**
2	7(**13.7**)	**6**(16.3)	**0.031**
3	**26**(**36.4**)	27(38.2)	0.291
4	50(**63.1**)	**47**(69.0)	0.062
5	38(58.6)	**36**(**51.8**)	**0.005**
6	**0**(**0.8**)	**0**(**0.8**)	0.826
7	**0**(2.6)	**0**(**2.4**)	0.579
8	**0**(**1.4**)	**0**(1.5)	0.782
9	**0**(**4.6**)	**0**(6.4)	**0.025**
10	28(40.9)	**22**(**40.4**)	0.761
11	**10**(**17.7**)	**10**(19.0)	0.318
12	53(64.5)	**47**(**64.1**)	0.881
13	38(53.3)	**33**(**51.0**)	0.297
14	5(**12.9**)	**4**(13.6)	0.587
15	**0**(**4.0**)	**0**(4.8)	0.234
16	**0**(**0.5**)	**0**(2.2)	**0.000**
17	26(41.6)	**25**(**36.8**)	**0.044**
18	**2**(**9.7**)	3(12.5)	**0.005**
19	**11**(**24.7**)	15(25.6)	0.577
20	**0**(**0.0**)	**0**(**0.0**)	–

Note: $n = 31$ runs.

Table 7. Comparison between SAR and SAIRL on ITC07 instances. Depicted is best(mean) of soft constraint violations.

Inst.	SAR	SAIRL	*t*-test (*p* value)
1	**0**(307.6)	**0**(**209.4**)	**0.025**
2	**0**(63.4)	**0**(**10.1**)	**0.048**
3	163(199.4)	**141**(**188.6**)	**0.050**
4	242(328.8)	**192**(**320.9**)	0.456
5	**0**(**2.7**)	**0**(2.9)	0.845
6	**0**(**33.2**)	**0**(54.7)	0.074
7	5(18.0)	**4**(**14.5**)	0.614
8	**0**(**0.0**)	**0**(1.6)	0.156
9	**0**(100.7)	**0**(**15.2**)	**0.009**
10	**0**(65.3)	**0**(**30.5**)	0.160
11	161(244.3)	**136**(**201.6**)	**0.001**
12	**0**(318.2)	**0**(**303.5**)	0.641
13	**0**(99.5)	**0**(**90.4**)	0.605
14	**0**(**0.2**)	**0**(25.6)	**0.001**
15	**0**(192.0)	**0**(**12.5**)	**0.000**
16	10(105.8)	**0**(**45.8**)	**0.000**
17	**0**(0.8)	**0**(**0.5**)	0.590
18	**0**(12.5)	**0**(**7.7**)	0.366
19	**0**(516.7)	**0**(**11.0**)	**0.000**
20	586(**650.7**)	**555**(664.0)	0.280
21	**0**(**12.5**)	**0**(25.7)	0.071
22	1(136.0)	**0**(**5.8**)	0.099
23	**11**(**504.4**)	56(713.6)	**0.005**
24	5(192.6)	**0**(**77.5**)	**0.000**

Note: $n = 31$ runs.

evident in Table 12, the algorithm is scalable as the best and average cost improved significantly when the runtime is extended. Note that the runtime is simply reset without tuning any parameters. The p values ($0.000 < 0.05$) of t-tests reject the null hypotheses $H_0 : \mu_{190s} = \mu_{1900s}$ and revealed a statistically significant

Table 8. Solver details.

Solver	Technique	Reference
A	Ant System	Socha et al. (2002)
B	Tabu Search Hyperheuristic	Burke et al. (2003)
C	Extended Great Deluge	McMullan (2007)
D	Great Deluge + Tabu Search	Abdullah et al. (2009)
E	Non Linear Great Deluge + Learning	Obit et al. (2009)
F	Fish Swarm	Turabieh et al. (2010)
G	Round Robin Multi Algorithms	Shaker & Abdullah (2010)
H	Honey Bee Mating	Sabar et al. (2012)
I	Simulated Annealing	Ceschia et al. (2012)
J1	Simulated Annealing	Kostuch (2003)
J2	Simulated Annealing	Kostuch (2005)
K	Tabu Search	Cordeau et al. (2003)
L	Great Deluge	Burke et al. (2003)
M	Local Search + Tabu Search	Di Gaspero and Schaerf (2003)
N	Hybrid Algorithm	Chiarandini et al. (2006)
O	Simulated Annealing	Cambazard et al. (2012)
P	Ant Colony Optimisation	Nothegger et al. (2012)
Q	Simulated Annealing	Lewis and Thompson (2015)
R	Simulated Annealing with Reheating (SAR)	Goh et al. (2017)

Table 9. Comparing SAIRL with other solvers on Socha instances. Depicted is best(mean) of soft constraint violations.

	Solver										
Inst.	A	B	C	D	E	F	G	H	I	R	SAIRL
S1	1	1	**0**(0.8)	**0**	**0**	**0**	**0**	**0**	**0**(**0.0**)	**0**(**0.0**)	**0**(**0.0**)
S2	3	2	**0**(2.0)	**0**	**0**	**0**	**0**	**0**	**0**(**0.0**)	**0**(**0.0**)	**0**(**0.0**)
S3	1	**0**	**0**(1.3)	**0**	**0**	**0**	**0**	**0**	**0**(**0.0**)	**0**(**0.0**)	**0**(**0.0**)
S4	1	1	**0**(1.0)	**0**	**0**	**0**	**0**	**0**	**0**(0.1)	**0**(**0.0**)	**0**(**0.0**)
S5	**0**	**0**	**0**(0.2)	**0**	**0**	**0**	**0**	**0**	**0**(**0.0**)	**0**(**0.0**)	**0**(**0.0**)
M1	195	146	80(101.4)	78	38	45	117	75	9(26.5)	**0**(**1.5**)	**0**(2.3)
M2	184	173	105(116.9)	92	37	40	108	88	15(25.9)	**0**(**2.2**)	**0**(3.6)
M3	248	267	139(162.1)	135	60	61	135	129	36(49.0)	7(**13.4**)	**6**(14.4)
M4	164.5	169	88(108.8)	75	39	35	75	74	12(23.8)	**0**(**0.7**)	**0**(1.4)
M5	219.5	303	88(119.7)	68	55	49	160	64	3(10.9)	**0**(**1.2**)	**0**(1.4)
L	851.1	1166	730(834.1)	556	638	407	589	523	208(259.8)	**165**(**206.6**)	181(215.2)

Notes: *n* = 31 runs. Note that some authors only reported their best results.

Table 10. Comparing SAIRL with other solvers on ITC02 instances. Depicted is best(mean) of soft constraint violations.

	Solver								
Inst.	J1	K	L	M	N	J2	I	R	SAIRL
1	45	61	85	63	45	**16**(**30.2**)	45(57.1)	23(32.6)	26(37.0)
2	25	39	42	46	14	**2**(**11.4**)	20(33.2)	7(13.7)	6(16.3)
3	65	77	84	96	45	**17**(**31.0**)	43(53.2)	26(36.4)	27(38.2)
4	115	160	119	166	71	**34**(**60.8**)	87(109.9)	50(63.1)	47(69.0)
5	102	161	77	203	59	42(72.1)	71(91.7)	38(58.6)	**36**(**51.8**)
6	13	42	6	92	1	**0**(2.4)	2(14.1)	**0**(**0.8**)	**0**(**0.8**)
7	44	52	12	118	3	2(8.9)	2(13.7)	**0**(2.6)	**0**(**2.4**)
8	29	54	32	66	1	**0**(2.0)	9(20.0)	**0**(**1.4**)	**0**(1.5)
9	17	50	184	51	8	1(5.8)	15(21.9)	**0**(**4.6**)	**0**(6.4)
10	61	72	90	81	52	**21**(**35.0**)	41(60.7)	28(40.9)	22(40.4)
11	44	53	73	65	30	**5**(**12.9**)	24(38.2)	10(17.7)	10(19.0)
12	107	110	79	119	75	55(76.3)	62(83.7)	53(64.5)	**47**(**64.1**)
13	78	109	91	160	55	**31**(**47.1**)	59(78.0)	38(53.3)	33(51.0)
14	52	93	36	197	18	11(22.3)	21(34.2)	5(**12.9**)	**4**(13.6)
15	24	62	27	114	8	2(8.4)	6(11.8)	**0**(**4.0**)	**0**(4.8)
16	22	34	300	38	55	**0**(3.4)	6(16.7)	**0**(**0.5**)	**0**(2.2)
17	86	114	79	212	46	37(54.0)	42(56.5)	26(41.6)	**25**(**36.8**)
18	31	38	39	40	24	4(**9.4**)	11(25.9)	**2**(9.7)	3(12.5)
19	44	128	86	185	33	**7**(**16.4**)	56(73.0)	11(24.7)	15(25.6)
20	7	26	**0**	17	**0**	**0**(0.5)	**0**(1.8)	**0**(**0.0**)	**0**(**0.0**)

Notes: *n* = 31 runs. Note that some authors only reported their best results.

difference between the mean of runtime t of T and $5T$. The soft constraint violations for SAIRL with extended runtime on Socha-L, ITC02-1, and ITC07-1 instances are illustrated in Figures 1–3. Meanwhile, the respective descriptive statistics are given in Tables 13–15. Note that the circles and stars in the box plots are mild and extreme outliers.

In addition, we compare the results of extended

Table 11. Comparing SAIRL with other solvers on ITC07 instances. Depicted is best(mean) of soft constraint violations.

	Solver				
Inst.	P	I	Q	R	SAIRL
1	**0**(613.0)	59(399.2)	**0**(377.0)	**0**(307.6)	**0**(**209.4**)
2	**0**(556.0)	**0**(142.2)	**0**(382.2)	**0**(63.4)	**0**(**10.1**)
3	**110**(680.0)	148(209.9)	122(**181.8**)	163(199.4)	141(188.6)
4	53(580.0)	25(349.6)	**18**(**319.4**)	242(328.8)	192(320.9)
5	13(92.0)	**0**(7.7)	**0**(7.5)	**0**(**2.7**)	**0**(2.9)
6	**0**(212.0)	**0**(**8.6**)	**0**(22.8)	**0**(33.2)	**0**(54.7)
7	**0**(4.0)	**0**(4.9)	**0**(5.5)	5(18.0)	4(14.5)
8	**0**(61.0)	**0**(1.5)	**0**(0.6)	**0**(**0.0**)	**0**(1.6)
9	**0**(202.0)	**0**(258.8)	**0**(514.4)	**0**(100.7)	**0**(**15.2**)
10	**0**(**4.0**)	3(186.4)	**0**(1202.4)	**0**(65.3)	**0**(30.5)
11	143(774.0)	142(269.5)	**48**(202.6)	161(244.3)	136(**201.6**)
12	**0**(538.0)	267(400.0)	**0**(340.2)	**0**(318.2)	**0**(**303.5**)
13	5(360.0)	1(120.0)	**0**(79.0)	**0**(99.5)	**0**(90.4)
14	**0**(41.0)	**0**(3.6)	**0**(0.5)	**0**(**0.2**)	**0**(25.6)
15	**0**(29.0)	**0**(48.0)	**0**(139.9)	**0**(192.0)	**0**(**12.5**)
16	**0**(101.0)	**0**(50.1)	**0**(105.2)	10(105.8)	**0**(**45.8**)
17	–	**0**(**0**)	**0**(0.1)	**0**(0.8)	**0**(0.5)
18	–	**0**(41.1)	**0**(**2.2**)	**0**(12.5)	**0**(7.7)
19	–	**0**(951.5)	**0**(346.1)	**0**(516.7)	**0**(**11.0**)
20	–	543(700.2)	557(724.5)	586(**650.7**)	555(664.0)
21	–	5(35.9)	1(32.1)	**0**(**12.5**)	**0**(25.7)
22	–	5(19.9)	4(1790.1)	1(136.0)	**0**(**5.8**)
23	–	1292(1707.7)	**0**(514.1)	11(**504.4**)	56(713.6)
24	–	**0**(105.3)	18(328.2)	5(192.6)	**0**(**77.5**)

Note: $n = 31$ runs.

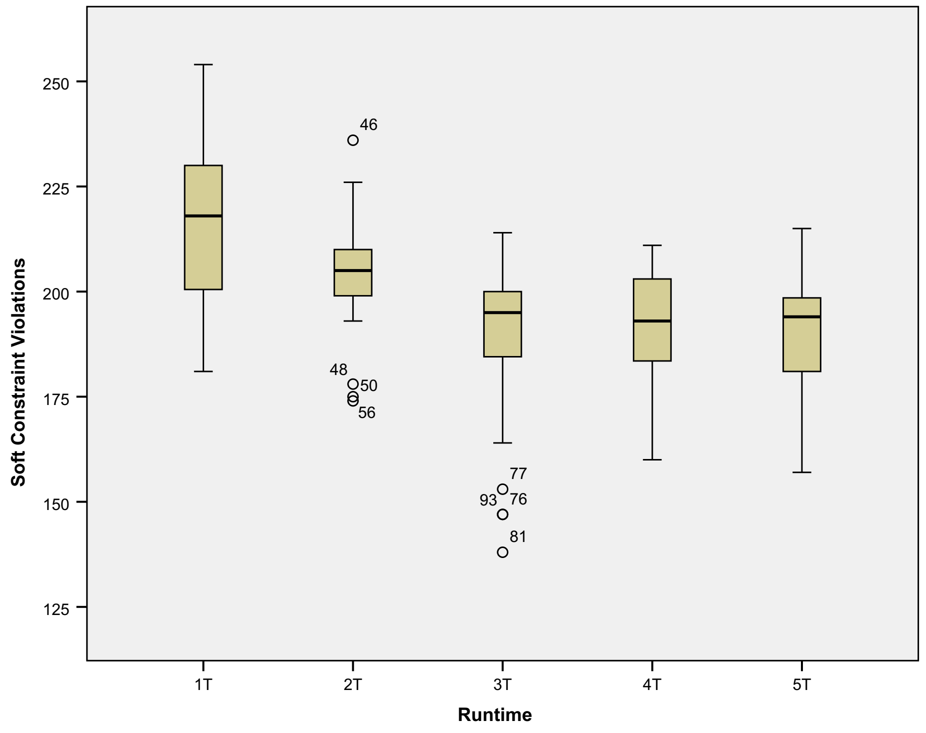

Figure 1. Box plot showing the soft constraint violations for SAIRL with extended runtime on Socha-L instance. Note: n=31 runs.

runtime between SAR and SAIRL for selected instances. As it took around 8 hours to run each instance for 31 times, we selected only one instance from each dataset namely Socha-L, ITC02-1 and ITC07-1. SAIRL is comparable to SAR as shown is Table 16. In fact, SAIRL is preferred based on the average of means for the selected instances. There is no significant difference between the means of both algorithms for ITC02-1. SAR is more effective than SAIRL for instance Socha-L. Meanwhile, SAIRL performed better than SAR for instance ICT2007-1.

5.5. Discussion

In order for a conventional SA to produce good results, certain parameters have to be tuned for specific instances e.g. initial temperature, final temperature, Markov chain length, and decay rate. SAIRL, proposed in this

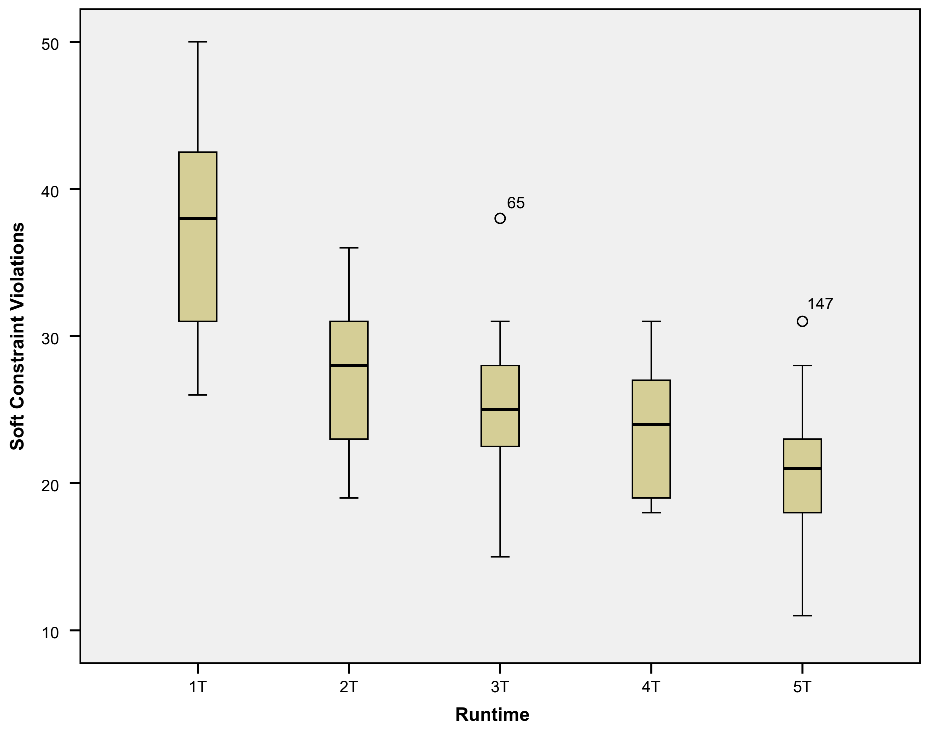

Figure 2. Box plot showing the soft constraint violations for SAIRL with extended runtime on ITC02-1 instance. Note: *n*=31 runs.

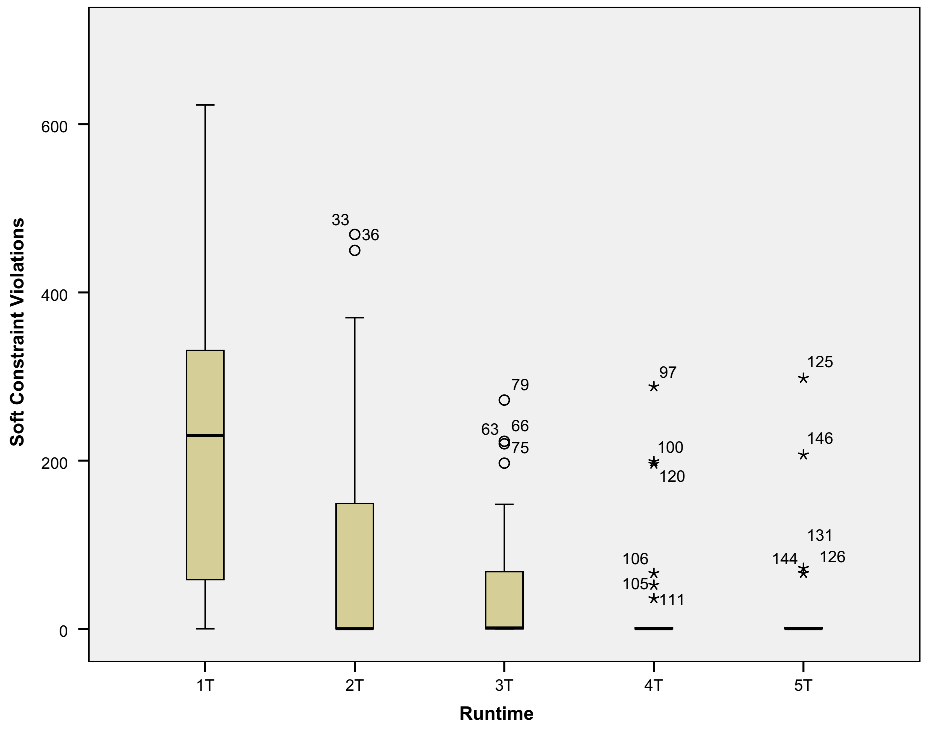

Figure 3. Box plot showing the soft constraint violations for SAIRL with extendedruntime on ITC07-1 instance. Note: *n*=31 runs.

paper, not only eliminates the requirement for tuning those parameters but also adjusting the neighbourhood structure composition (data-set-specific) in SAR. Nevertheless, the algorithm produces results comparable to the state of the art methods which were either heavily tuned or limited in terms of data-sets considered. For SAIRL, we merely set the decay rate β to 0.9995 and the constant D to 0.001. The same settings were used and worked well for all the instances without tuning. Meanwhile the initial temperature is simply set as 1% of the initial cost. Unlike conventional SA, the initial temperature is not critical for SAIRL as reheating allows the search to reset itself when it detects that it is stuck in a local optima.

Table 12. Comparison of soft constraint violations between SAIRL with runtime of *T* and 5*T* on selected instances.

	t=*T*		*t*=5*T*		*t*-test
Inst.	best	mean	best	mean	(*p* value)
Socha-L	181	215.19	157	190.42	0.000
ITC02-1	26	37.03	11	20.84	0.000
ITC07-1	0	209.39	0	23.06	0.000
Avg	–	153.87	–	**78.11**	

Note: *n* = 31 runs.

Table 13. Descriptive statistics for SAIRL with extended runtime on Socha-L instance.

	Runtime				
Soft Constraint Violations	1*T*	2*T*	3*T*	4*T*	5*T*
Min	181	174	138	160	157
Max	254	236	214	211	215
Median	218.00	205.00	195.00	193.00	194.00
Mean	215.19	204.97	189.65	191.58	190.42

Note: *n* = 31 runs.

Table 14. Descriptive statistics for SAIRL with extended runtime on ITC02-1 instance.

	Runtime				
Soft Constraint Violations	1*T*	2*T*	3*T*	4*T*	5*T*
Min	26	19	15	18	11
Max	50	36	38	31	31
Median	38.00	28.00	25.00	24.00	21.00
Mean	37.03	27.84	25.03	23.65	20.84

Note: *n* = 31 runs.

Table 15. Descriptive statistics for SAIRL with extended runtime on ITC07-1 instance.

	Runtime				
Soft Constraint Violations	1*T*	2*T*	3*T*	4*T*	5*T*
Min	0	0	0	0	0
Max	623	469	272	288	298
Median	230.00	0.00	1.00	0.00	0.00
Mean	209.39	90.23	48.29	27.23	23.06

Note: *n* = 31 runs.

Table 16. Comparison of soft constraint violations between SAR (5*T*) and SAIRL (5*T*) on selected instances.

	SAR (5*T*)		SAIRL (5*T*)		*t*-test
Inst.	best	mean	best	mean	(*p* value)
Socha-L	103	139.39	157	190.42	0.000
ITC02-1	10	21.03	11	20.84	0.867
ITC07-1	0	134.94	0	23.06	0.000
Avg	–	98.45	–	**78.11**	

Note: *n* = 31 runs.

Multiple operator implementations are generally better than any single operator variant because the solution space is more connected. However, selecting operators with equal probability for multiple operators is suboptimal because in reality, the operators have different acceptance ratio and computational costs for different data instances. The effect of RL is shown in Figure 4.

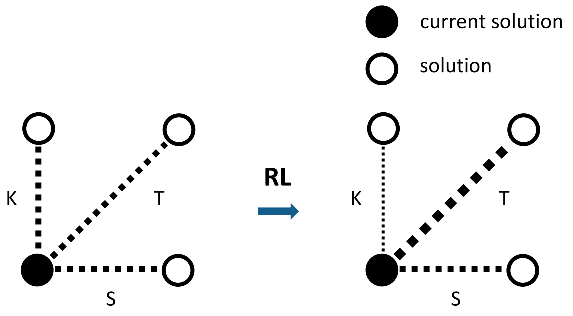

Figure 4. The use of Reinforcement Learning (RL) to adjust the neighbourhood structure composition.

We use dashed instead of solid lines to indicate that the solution space is not necessarily connected by any of the operators as move acceptance is determined by feasibility as well as the temperature. Meanwhile, the thickness of the dashed lines represent the selection probabilities for the operators. Initially, the operators have an equal chance of being selected. As the search progresses, the probabilities are increased or decreased depending on the acceptance ratio and computational cost of the operators. Unlike other methods which reward operators that improve the current or best solution, our RL-based method rewards operators that change the current solution (improving moves or equal cost moves or worsening moves). Effectively, operators with relatively high values (cumulative mean of rewards) will have a higher tendency to be selected. The probabilistic selection that we use prevents the domination of any operator as low valued operators can still be selected. As a result, the number of transitions (accepted moves) per time unit is maximised and the solution space connectivity is (we hope) improved. The movement of neighbourhood structure composition for Socha-L, ITC02-1 and ITC07-1 is shown in Fig. 5–7 respectively. The composition of Kempe operators is higher for ITC07-1 compared to Socha-L and ITC02-1. For Socha and ITC02 instances, the search spaces are well connected by transfer and swap operators. Therefore, a Kempe operator is redundant for these instances. Furthermore, the Kempe operator is computationally more expensive. Meanwhile, for ITC07 instances, the search space is poorly connected by transfer and swap operators. Thus, a higher composition of a Kempe operator is worthwhile for ITC07 instances.

In SAR, the reheated temperature is set proportional to the current cost. When the current cost is low, the temperature is set proportionally low. In effect, the search is guided to operate in the vicinity of the current solution with the hope of finding the optimal solution (exploitation). Meanwhile, when the current cost is high, the temperature is set proportionally high and the search is allowed to explore more. However, setting the reheated temperature based on the current cost alone is not sufficient as the search landscape for each

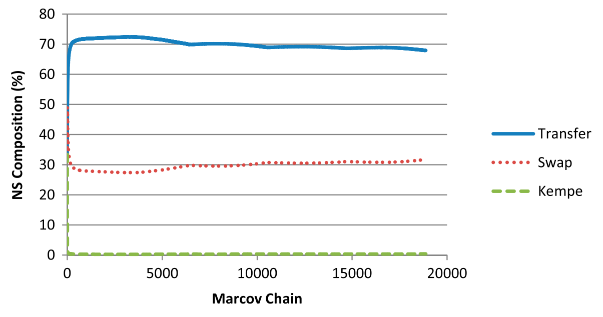

Figure 5. Movement of neighbourhood structure composition for Socha-L.

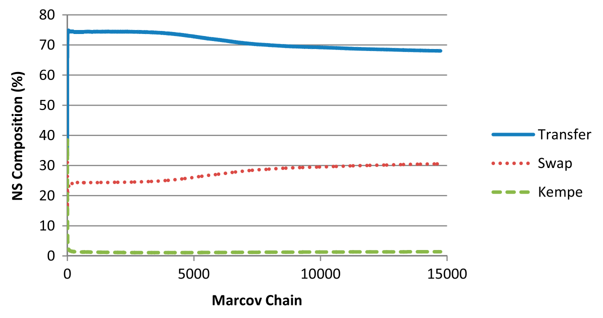

Figure 6. Movement of neighbourhood structure composition for ITC02-1.

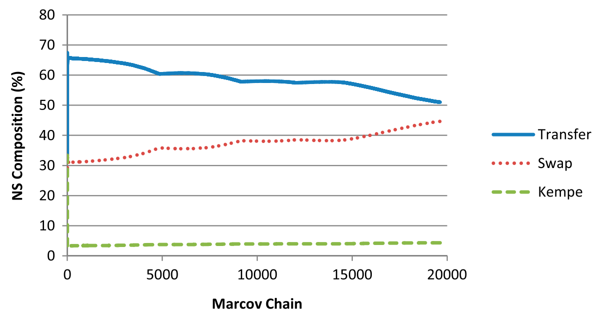

Figure 7. Movement of neighbourhood structure composition for ITC07-1.

instance is different. In SAIRL, the average cost changes (which provides insight into the gradient of the search landscape) is incorporated into the reheated temperature function since the temperature determines the acceptance of uphill moves based on the cost changes. In effect, we are using the information on the gradient of the search landscape to determine the exploration level for the search.

6. Conclusion

This work proposed a two-stage approach for the post enrolment course timetabling problem. The proposed approach utilised tabu search algorithm in the first stage to generate a feasible solution. We propose enhanced variant of the simulated annealing that uses an improved reheating and learning strategies to further improve the generated solutions in the second stage. We have conducted extensive experimental tests to compare the performance of several variant of simulated annealing: simulated annealing with reheating, simulated annealing with reheating with learning and simulated annealing with improved reheating and learning strategies on all the instances. The results demonstrated that the proposed enhancements did improve the performance of traditional simulated annealing algorithm. We also compared the performance of proposed algorithm with state of the art methods. The results show that the proposed algorithm is comparable or better than other state of the art methods. Finally, we have shown that the proposed algorithm is scalable when the runtime is extended.

7. Future work

We are looking forward to utilise the proposed algorithm on other educational timetabling problems such as school timetabling (ITC11) and examination timetabling (examination track of ITC07 and Toronto data-set). Adaptations should be minimal considering the common features and structures shared by the educational timetabling problems. The algorithm could also be applied to other scheduling problems (transport scheduling, sports scheduling and nurse rostering) and possibly other combinatorial optimisation problems (bin packing, vehicle routing). Working on different problems not only provides a platform to test the robustness and general applicability of the proposed algorithm but also invaluable experience to further improve the algorithm.

It is also possible to add more complex operators into the neighbourhood structure composition such as Hungarian method (Kuhn, 1955), double Kempe (Lu & Hao, 2010), etc. The complex operators may be computationally expensive but are worthwhile provided the connectivity of the search space is improved. The Reinforcement Learning used in the proposed algorithm will adjust the neighbourhood structure composition accordingly based on the acceptance ratio and computational cost (CPU time) of the operators.

The proposed algorithm can be hybridised with a Tabu Search mechanism. In the proposed algorithm, every time slot is attempted for each event unless the event is successfully moved to a time slot. An event can be prohibited from moving to certain slots after moving out of the time slots recently. Instead of attempting to move an event to every time slot, only certain non-tabu time slots are attempted. It is hoped that the exploration of the search will improve. Tabu tenure determines the prohibition duration (number of iterations) of a particular time slot for a particular event. We could set the tabu tenure as a function proportional to the current cost. This dynamic tabu tenure is expected to allow the search to explore and exploit the search space accordingly during the search process. This idea is inspired by the principle that the search should explore more when the current cost is high and exploit more when the current cost is low.

The proposed algorithm can also be hybridised with any population-based algorithms (GA) which are well known for their exploration capability. For instance, when the search gets stuck during the search, GA can be initiated for the search to escape from being stuck in a local optima. We would suggest that the number of iteration for the genetic algorithm to be set proportional to the current cost. After a certain number of iterations, the mode of execution is returned back to the proposed algorithm. Then the temperature is set proportional to the current cost and cooled until it is stuck again. The execution of the proposed algorithm and GA are alternated until the time limit is reached.

Currently, the proposed algorithm utilises a static cooling schedule. We look forward to test the proposed algorithm with various adaptive cooling schedules as proposed in Van Laarhoven and Aarts (1987), Romeo, Sangiovanni, and Huang (1986), Otten and van Ginneken (2003) and Triki et al. (1998). The adaptive cooling schedules worked well for the respective domains. However, a parameter value has to be set for them to work effectively.

ORCID

Graham Kendall http://orcid.org/0000-0003-2006-5103

Disclosure statement

No potential conflict of interest was reported by the authors.

References

Abdullah, S., Burke, E. K., & McCollum, B. (2005). *An investigation of variable neighbourhood search for university course timetabling*. The 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA) (p. 413-427).

Abdullah, S., Burke, E. K., & McCollum, B. (2007). *A hybrid evolutionary approach to the university course timetabling problem*. In Evolutionary Computation (CEC 2007, pp. 1764-1768). IEEE.

Abdullah, S., Shaker, K., McCollum, B., & McMullan, P. (2009). *Construction of course timetables based on great deluge and tabu search*. Metaheuristics Int. Conf., VIII Meteheuristic (pp. 13-16).

Arntzen, H., & Lokketangen, A. (2003). A local search heuristic for a university timetabling problem. *nine*, *1*(T2), T45

Burke, E., Bykov, Y., Newall, J., & Petrovic, S. (2003). A time-predefined approach to course timetabling. *The Yugoslav Journal of Operations Research*, *13*(2), 139–151. ISSN: 0354–0243 EISSN: 2334–6043.

Burke, E., Jackson, K., Kingston, J. H., & Weare, R. (1997). Automated university timetabling: The state of the art. *The Computer Journal, 40*(9), 565–571.

Burke, E. K., Kendall, G., & Soubeiga, E. (2003). A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics, 9*(6), 451–470.

Cambazard, H., Hebrard, E., OSullivan, B., & Papadopoulos, A. (2012). Local search and constraint programming for the post enrolment-based course timetabling problem. *Annals of Operations Research, 194*(1), 111–135.

Ceschia, S., Di Gaspero, L., & Schaerf, A. (2012). Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. *Computers & Operations Research, 39*(7), 1615–1624.

Chiarandini, M., Birattari, M., Socha, K., & Rossi-Doria, O. (2006). An effective hybrid algorithm for university course timetabling. *Journal of Scheduling, 9*(5), 403–432.

Chiarandini, M., Fawcett, C., & Hoos, H. H. (2008). *A modular multiphase heuristic solver for post enrollment course timetabling*. Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008).

Cooper, T. B., & Kingston, J. H. (1996). *The complexity of timetable construction problems*. Berlin: Springer.

Cordeau, J. F., Jaumard, B., & Morales, R. (2003). Efficient timetabling solution with tabu search. In International Timetabling Competition.

de Werra, D. (1985). An introduction to timetabling. *European Journal of Operational Research, 19*(2), 151–162.

Di Gaspero, L., & Schaerf, A. (2003). Timetabling competition ttcomp 2002: solver description. (International Timetabling Competition).

Ejaz, N., & Javed, M.Y. (2007). *A hybrid approach for course scheduling inspired by die-hard co-operative ant behavior*. Automation and Logistics, 2007 IEEE International Conference on (p. 3095–3100). IEEE.

Goh, S. L., Kendall, G., & Sabar, N. R. (2017). Improved local search approaches to solve post enrolment course timetabling problem. *European Journal of Operational Research, 261*(1), 17–29.

Jaradat, G. M., & Ayob, M. (2010). An elitist-ant system for solving the post-enrolment course timetabling problem. *Database Theory and Application, Bio-Science and Bio-Technology* (pp. 167–176). Berlin: Springer.

Kostuch, P. (2003). Timetabling competition-sa-based heuristic. International Timetabling Competition.

Kostuch, P. (2005). The university course timetabling problem with a three-phase approach. *Practice and Theory of Automated Timetabling* (pp. 109–125). Berlin: Springer.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly, 2*(1), 83–97.

Landa-Silva, D., & Obit, J.H. (2008). *Great deluge with non-linear decay rate for solving course timetabling problems*. Intelligent Systems, 2008. IS'08. 4th International IEEE Conference (Vol. 1, pp. 8-11). IEEE.

Lewis, R. (2008). A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum, 30*(1), 167–190.

Lewis, R. (2012). A time-dependent metaheuristic algorithm for post enrolment-based course timetabling. *Annals of Operations Research, 194*(1), 273–289.

Lewis, R., & Thompson, J. (2015). Analysing the effects of solution space connectivity with an effective metaheuristic for the course timetabling problem. *European Journal of Operational Research, 240*(3), 637–648.

Lu, Z. & Hao, J.-K. (2010). Adaptive tabu search for course timetabling. *European Journal of Operational Research, 200*(1), 235–244.

McMullan, P. (2007). An extended implementation of the great deluge algorithm for course timetabling. *Computational Science-ICCS 2007* (pp. 538–545). Berlin: Springer.

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

Nothegger, C., Mayer, A., Chwatal, A., & Raidl, G. R. (2012). Solving the post enrolment course timetabling problem by ant colony optimization. *Annals of Operations Research, 194*(1), 325–339.

Obit, J., Landa-Silva, D., Ouelhadj, D., & Sevaux, M. (2009). *Non-linear great deluge with learning mechanism for solving the course timetabling problem*. 8th Metaheuristics International Conference (MIC 2009).

Otten, R. H., & van Ginneken, L. P. (2003). Floorplan design using annealing. *The Best of ICCAD* (pp. 479–488). Berlin: Springer.

Petrovic, S., & Burke, E. K. (2004). University timetabling. *Handbook of scheduling: algorithms, models, and performance analysis, 45*, 1–23.

Romeo, F., Sangiovanni, V. A., & Huang, M. (1986). An efficient general cooling schedule for simulated annealing. *Proceeding of IEEE International Conference on Computer Aided Design* (pp. 396–404).

Sabar, N. R., Ayob, M., Kendall, G., & Qu, R. (2012). A honey-bee mating optimization algorithm for educational timetabling problems. *European Journal of Operational Research, 216*(3), 533–543.

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

Shaker, K., & Abdullah, S. (2010). Controlling multi algorithms using round robin for university course timetabling problem. *Database Theory and Application, Bio-Science and Bio-Technology* (pp. 47–55). Berlin: Springer.

Socha, K., Knowles, J., & Sampels, M. (2002). A max-min ant system for the university course timetabling problem. In *Ant algorithms* (pp. 1–13). Berlin: Springer.

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

Triki, E., Collette, Y., & Siarry, P. (1998). A theoretical study on the behavior of simulated annealing leading to a new cooling schedule. *European Journal of Operational Research, 166*(1), 77–92.

Turabieh, H., Abdullah, S., McCollum, B., & McMullan, P. (2010). Fish swarm intelligent algorithm for the course timetabling problem. In *Rough Set and Knowledge Technology* (pp. 588–595). Berlin: Springer.

Van Laarhoven, P. J., & Aarts, E. H. (1987). *Simulated annealing: theory and applications* (Vol. 37), Berlin: Springer Science & Business Media.