Improved Local Search Approaches to Solve Post Enrolment Course Timetabling Problem

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

[a] *The University of Nottingham Malaysia Campus, Jalan Broga, 43500 Semenyih, Selangor Darul Ehsan, Malaysia.*
[b] *University of Nottingham, University Park, Nottingham NG7 2RD, United Kingdom.*
[c] *Queensland University of Technology, 2 George Street, Brisbane, QLD 4000, Australia.*

Abstract

In this work, we are addressing the post enrollment course timetabling (PE-CTT) problem. We combine different local search algorithms into an iterative two stage procedure. In the first stage, Tabu Search with Sampling and Perturbation (TSSP) is used to generate feasible solutions. In the second stage, we propose an improved variant of Simulated Annealing (SA), which we call Simulated Annealing with Reheating (SAR), to improve the solution quality of feasible solutions. SAR has three features: a novel neighborhood examination scheme, a new way of estimating local optima and a reheating scheme. SAR eliminates the need for extensive tuning as is often required in conventional SA. The proposed methodologies are tested on the three most studied datasets from the scientific literature. Our algorithms perform well and our results are competitive, if not better, compared to the benchmarks set by the state of the art methods. New best known results are provided for many instances.

Keywords: Timetabling, Combinatorial Optimization, Local Search, Tabu Search with Sampling and Perturbation (TSSP), Simulated Annealing with Reheating (SAR).

*Corresponding author
Email addresses: gohsayleng@yahoo.com (Say Leng Goh), Graham.Kendall@nottingham.edu.my (Graham Kendall), nasser.sabar@gmail.com (Nasser R. Sabar)

1. Introduction

University course timetabling problems involve assigning a set of courses to a limited set of time slots, and rooms, whilst satisfying a set of constraints [1]. It is an NP-complete problem, meaning that approaches which are guaranteed to provide an optimal solution are often too time consuming so that heuristic and meta-heuristic approaches are often utilized. Assigning courses to time slots alone is equivalent to that graph coloring problem which is also NP-complete. de Werra shows the reduction of timetabling to graph coloring problem [2]. Timetabling construction has also been shown to be NP-complete in several other ways [3]. Timetabling has an increased level of difficulty as courses have to be assigned to rooms in addition to time slots.

In this work, Tabu Search with Sampling and Perturbation (TSSP) is used to find feasible solutions. The feasible solution is then improved in terms of soft constraint violations by using an enhanced version of Simulated Annealing (SA) called SAR which eliminates the need for tuning as is often the case for a conventional SA. We do not use Tabu Search (TS) to improve the soft cost of the solutions in stage 2 as we feel TS is too restrictive and may affect the connectivity of search space. The proposed method is tested on three benchmark datasets for university course timetabling problems and the results are compared with other state of the art methods.

In this paper, the problem description is given in Section 2. Related work is reviewed in Section 3. In Section 4, we provide details of the *base* algorithm that we use as the basis for our proposed algorithm (the refinements are presented in Section 5.1.1). The proposed methodology is described in Section 5 and the experimental results are presented in Section 6. Performance and behavior of the algorithms are discussed in section 7. Concluding remarks are given in Section 8. Finally, suggestions for future work are given in Section 9.

2. Problem Description

There are many variants of the course timetabling problem, with different requirements expressed as either hard or soft constraints, across institu-

tions of higher learning around the globe. Different implementations have reported varying degrees of success. However, it is difficult to compare the effectiveness of different algorithms if they are executed on different problem instances. Researchers have shared datasets so that algorithm comparison is more objective. The datasets utilized in this research are publicly available and regarded as the standard benchmarks:

- **Socha with 11 instances[1].** The instances (5 small, 5 medium and 1 large) are generated using an algorithm developed by Ben Paechter. The time limit for the small, medium, and large instances is set to 90, 900, 9000 seconds respectively [4]. Even this is problematical as different machine specifications means that running for 900 seconds is not a fair comparison. Refer to Table 1 for the benchmark statistics.

- **International Timetabling Competition 2002 (ITC2002) with 20 instances[2].** This competition was organized by the Metaheuristic Network and the instances were generated by Ben Paechter. The time limit is benchmarked by running a program on the host machine, which enables a fair comparison. Refer to Table 2 for the benchmark statistics.

- **International Timetabling Competition 2007 (ITC2007) with 24 instances[3].** The time limit is benchmarked in the same way as ITC2002. Refer to Table 3 for the benchmark statistics.

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

Table 1: Statistics for the Socha dataset

[1] http://iridia.ulb.ac.be/supp/IridiaSupp2002-001/index.html Last accessed: May 23, 2017.

[2] http://www.idsia.ch/Files/ttcomp2002/ Last accessed: May 23, 2017.

[3] http://www.cs.qub.ac.uk/itc2007/ Last accessed: May 23, 2017.

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

Table 2: Statistics for the ITC2002 dataset

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 3: Statistics for the ITC2007 dataset

Solving the problem involves assigning a set of C courses (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 minimize soft constraint violations as far as possible. Perfect solutions are known to exist for the datasets.

The hard constraints for all the datasets are:

- HC1: No student can be assigned more than one course at the same time.
- HC2: The room should satisfy the features required by the course.
- HC3: The number of students attending the course should be less than or equal to the capacity of the room.
- HC4: No more than one course is allowed for each time slot in each room.

There are two additional hard constraints for ITC2007 namely:

- HC5: A course can only be assigned to some preset time slots
- HC6: Where specified, a course should be scheduled to occur in the correct order.

The soft constraints for all the datasets are:

- SC1: A student should not have a single course on a day.
- SC2: A student should not have more than two consecutive courses.
- SC3: A student should not have a course scheduled in the last time slot of the day.

3. Related Work

In this paper we are using a multi-stage approach. Although different to hybridized approaches, there are good reasons for using multiple algorithms within an overarching approach. Hybridization has led to good quality results in previous research (e.g. [5, 6]). More recent work has validated these earlier findings. For example, [7] hybridized mixed integer linear programming, a greedy heuristic, two local search strategies and three metaheuristics for a vehicle routing problem, reporting positive results. Zeb et al.

[8] hybridized simulated annealing (SA) and a genetic algorithm (GA). The GA was used as an exploration operator, SA was used to intensify the search. The algorithm was evaluated on 35 cell formulation benchmark instances, producing 24 best results, and two new best results. A genetic algorithm was also used in [9]. Their abstract states "*Meta-heuristics still suffers from several problems that remains open as the variability of their performance depending on the problem or instance being solved. One of the approaches to deal with these problems is the hybridization of techniques.*" They concluded that their hybridized approach is the best performing method for instances with high dimensionality.

Metropolis introduced the Metropolis algorithm to simulate the evolution of solid in a heat bath to thermal equilibrium [10]. Kirkpatrick applied the concepts of annealing to optimization problems [11]. SA accepts all improving moves or those equivalent to the current solution. It also accepts worse moves with probability of:

$$e^{\Delta f/T} \tag{1}$$

where Δf is the change in solution quality and T is the temperature. Usually, T is initialized with a sufficiently high value and gradually reduced as the search progresses. In practice, the search ends when the temperature exceeds a predefined end temperature or any other stopping condition is met.

Thompson and Dowsland applied simulated annealing for the examination timetabling problem [12]. The authors mentioned the difficulty of setting weights in a single phased method. Therefore, a multi-phased method was used where more important objectives were considered in earlier phases, while other objectives were considered in later phases. The optimized objectives in the early phases were considered as binding constraints in later phases. The authors note that their method is not perfect as the solution space may be disconnected. Three ways were introduced to deal with solution space connectivity, but only two were deemed successful, namely; using different starting solutions and changing the neighborhood structure

(Kempe chain operator).

Abramson, Amoorthy and Dang compared two cooling schedules (geometric cooling and multiple cooling) and four reheating schemes (geometric reheating, enhanced geometric reheating, non-monotonic cooling and reheating as cost function) on randomly generated school timetabling problems [13]. Cost based reheating was found to be superior in finding global minima and also faster compared to other schemes.

3.1. Specific approaches applied to Socha instances

One of the earliest approaches on these instances was based on an ant system [4]. In fact, this benchmark is named after the author. Ants follow a list of ordered events and choose time slots randomly based on probabilities that depend on pheromone and heuristic information. A matching algorithm was applied for room assignment. The candidate solution was further improved by a local search as an exploitation mechanism. A global best solution was maintained. Pheromone corresponding to the global best was increased while the rest were reduced using an evaporation co-efficient. The pheromone levels were reduced to allow exploration in the search space. Parameter tuning was required for each instance type. The method was reported to be better than random restart local search. Interested readers can refer to its variants [14] [15].

Ceschia et al. applied simulated annealing on the problem and achieved breakthrough results in very short time relative to methods used by other researchers [16]. Two neighborhood structures were used; moving an event from one space to another and swapping events. Dummy time slots and dummy rooms were used. The cost function was evaluated based on unscheduled events, precedences and conflicts in addition to soft constraint violations which prompted the need to set the proper weights for each component. In addition, parameters specific to simulated annealing had to be set. The author used an F-race mechanism to tune the related parameters. The author attributed the good results to the preprocessing and constraint reformulation step which improved the efficacy of the local search. Their implementation produced the best results in terms of best and mean results

as reported in the literature.

3.2. Specific approaches applied to ITC2002 instances

Kostuch employed a simulated annealing based heuristic approach, becoming the winner of International Timetabling Competition 2002 (ITC2002) [17]. In the preprocessing, the author defined the event room matrix and incidence matrix. The incidence matrix was further updated with 1-room events. In finding an initial feasible solution, events were ordered and each assigned a time slot with a minimum number of events, provided that the number of events in the time slot did not exceed the number of rooms. Unassigned events were placed in a pool. Maximum matching was run for room assignment and unassigned events were removed from the time slots and placed in a pool. Next, in an *improvement* phase, unplaced events were refitted into the time slots where events were removed during the room assignment phase. In *shuffling* phase, every event from the pool of unplaced events was assigned to a random time slot and maximum matching was run for room assignment. The newly unassigned event was hopefully different from the initially unassigned event. The *improvement* phase was rerun. Next in a *blow-up* phase, the unassigned events were placed into a time slot and all current events in that time slot were removed. Then rooms were assigned and unplaced events were kept in a pool. The *improvement* and *shuffling* phase were rerun. The still unplaced events, if there were any, were distributed over the last time slots. The feasible solution was then improved with simulated annealing by sequencing the time slots and exchanging pairs of events. Finally, simulated annealing was run with lower acceptance probability on the best solution until the time limit was reached. The search was confined to the vicinity of solution.

Chiarandini et al. employed a strategy which combined construction heuristics, variable neighborhood descent and simulated annealing which outperformed the winner of ITC2002 [18]. The authors used a racing algorithm to iteratively select and configure algorithms. Candidate algorithms were evaluated and discarded when sufficient statistical evidence was gathered against them. Local search and tabu search were utilized to obtain a

feasible solution. The feasible solution is improved in terms of soft constraint violations by using variable neighborhood descent and simulated annealing. The authors claimed that the method reduced the number of experiments and was well suited for the engineering of meta-heuristics. Findings highlighted the importance of local search in ant colony optimization and genetic algorithms and that variable neighborhoods strongly enhanced the local search. Solving hard and soft constraints separately was also found to be preferable than weighting constraints in an evaluation function. The authors also highlighted that tabu search was not suitable for optimizing soft constraints. Population based meta-heuristics did not perform better than a single solution based approach and the importance of problem specific knowledge was emphasized. The method obtained better results than the ITC2002 winner on 18 out of 20 instances.

Kostuch further improved his method and achieved the best results on all 20 instances [19]. Feasible solutions were constructed using graph coloring and maximum matching. The feasible solution was improved by sequencing the time slots and exchanging pairs of events. To keep the neighborhood structure simple, the author also introduced 10 dummy events, 2 at each end of day time slots which were removed in the final timetable. His implementation is the current state of the art for the problem instances.

3.3. Specific approaches applied to ITC2007 instances

The submission by Cambazard et al. won the post enrolment based course timetabling of ITC2007 [20]. A few approaches were studied. In the first approach, local search is performed on randomly generated solutions to find a feasible one. A tabu list is maintained to prevent an event from being assigned the same time slots for the last k iterations. Among the neighborhood structures used were; moving an event to empty space, swapping two events, swapping two time slots, matching where events are reassigned within a time slot, moving an event with matching and Hungarian move. The feasible solution is optimized by simulated annealing with reheating. Moving an event with matching is the only neighborhood structure considered in this phase. The second approach presented was also based on local

search but with a relaxation on room allocation. It was termed LS with coloring. Four stages were involved. In stage 1, a feasible solution is found ignoring room allocation. In stage 2, soft constraint violations were minimized, again ignoring room allocation. In stage 3, the solution is repaired into a feasible solution. In stage 4, the solution is improved in terms of soft constraint violations. The same neighborhood structures were used but without matching during room allocation. The author reported LS with coloring was the best approach in finding feasible solutions. Also LS with coloring worked best for highly constrained problems. Constraint programming was also developed but was less competitive compared to the local search approach and it was unable to find feasible solutions for instances 1,2,9 and 10.

Nothegger et al. applied Ant Colony Optimization (ACO) to ITC2007 achieving fourth place in the competition [21]. They proposed two separate matrices to store pheromone information instead of the traditional single matrix. They showed that a two matrices representation produced better results in terms of distance to feasibility (DTF) and soft constraints penalty (SCP) as it is less expensive computationally thus allowing more iterations per time unit. Events were considered in random order and assigned to time slots and rooms based on pheromone information. Heuristic information in a typical ACO was not used to promote randomness. Each constructed solution were improved locally by an ejection chain. Pheromone information is updated by solutions with lowest DTF scores and better than average SCP scores. The pheromone levels were reduced by evaporation. The authors also presented a parallel ACO with simulated annealing as the local search procedure.

Lewis and Thompson achieved 100% feasibility on all instances of ITC2007 by using constructive heuristics and followed by their PARTIALCOL algorithm which uses a tabu mechanism for the remaining unassigned events [22]. The authors further improved the method by performing perturbations in the form of random walk and resetting the tabu list after 5000 idle iterations. The feasible solution was improved by using simulated annealing. The initial temperature was set automatically as the standard deviation of

the cost of sample moves. The cooling rate was altered during the run. The authors also used the feasibility ratio to gauge the connectivity of the search space of various neighborhood operators on the instances. A Kempe chain operator was found to be particularly suitable for the instances. Strong results were achieved. The authors also showed that the use of dummy rooms did not improve the results.

4. Tabu Search (TS)

TS was introduced by Glover [23] as an extension to hill climbing to overcome local optima. TS in general selects the best admissible neighborhood move (non-tabu or allowed by aspiration). The move with the lowest cost function is selected even if it increases the cost function of the current solution. If the current solution is better than the best solution, the best solution is updated. The reversal of the selected move is then set tabu for some time to prevent cycling.

PARTIALCOL [24], which was initially used for solving graph coloring problems, was adapted by [25], [26] and [22] in solving course timetabling problems. The TS procedure, presented in Algorithm 1, is based on PARTIALCOL. A neighbor 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, all the neighborhood moves are evaluated (line 7-21) by considering all non-tabu suitable time slots for all the events in *unplacedE*.

The event e is temporarily removed from *unplacedE*. To feasibly move e into a particular time slot, events conflicting with e (violated clash or precedence constraint) are temporarily moved from *current* to *unplacedE*. By comparison, [22] only removed events which violated a clash constraint from the time slots. As maximal matching is computationally expensive, it is used for room assignment only when necessary. If matching could not find a room for the event under consideration, a room is chosen randomly from among the suitable rooms and the related event is moved from *current* to *unplacedE*.

Algorithm 1

```
1: procedure TS(best, unassignedE)
2:     unplacedE ← unassignedE
3:     current ← best
4:     f(best) ← f(current)
5:     while unplacedE is not empty AND time.elapsed() < T  do
6:         min ← ∞
7:         for all e ∈ unplacedE do
8:             unplacedE ← unplacedE − e
9:             for all s ∈ S | S non-tabu slot suitable for e do
10:                current ← current − {events conflicting e}
11:                unplacedE ← unplacedE ∪ {events conflicting e}
12:                if f(candidate) < min then
13:                    bestEvent ← e
14:                    bestSlot ← s
15:                    min ← f(candidate)
16:                end if
17:                unplacedE ← unplacedE − {events conflicting e}
18:                current ← current ∪ {events conflicting e}
19:            end for
20:            unplacedE ← unplacedE ∪ e
21:        end for
22:        current ← current − {events conflicting bestEvent}
23:        current ← current ∪ bestEvent                    ▷ bestSlot
24:        f(current) ← min
25:        if f(current) < f(best) then
26:            best ← current
27:            f(best) ← f(current)
28:            unassignedE ← unplacedE
29:        end if
30:        set tabu {events conflicting bestEvent} from original time slots
31:        unplacedE ← unplacedE − bestEvent
32:        unplacedE ← unplacedE ∪ {events conflicting bestEvent}
33:    end while
34: end procedure
```

The cost function of the candidate solution *f(candidate)* is based on the

number of unplaced events:

$$\sum_{e \in unplacedE} 1 \tag{2}$$

Effectively, the candidate solution with the lowest number of unplaced events is preferred. As a comparison, the cost function used in [26] was the number of students required to attend the unplaced events:

$$\sum_{e \in unplacedE} size[e] \tag{3}$$

The events conflicting with e are moved back from *unplacedE* to *current* before evaluating the next non-tabu suitable time slot for the event under consideration. When all the non-tabu time slots are evaluated, e is placed back to *unplacedE* before the next event is considered. Ultimately, the neighbor move with the lowest candidate cost *f(candidate)* is recorded as *bestEvent* and *bestSlot*.

Events conflicting with *bestEvent* are extracted from *current* (line 22). The best neighbor move is applied where the *bestEvent* is moved to the *bestSlot* of *current* (line 23). *best*, *f(best)* and *unassignedE* are updated if *f(current)* is better than *f(best)*. The events conflicting with *bestEvent* are set tabu from returning to their original time slots for a number of iterations (line 30) according to the tabu tenure

$$\text{RANDOM}[10) + |unplacedE| \tag{4}$$

where $|unplacedE|$ is the number of unplaced events. A value of 10 is used in the random element of the tabu tenure length. We use this value as the same value was used in [24], [22] and [26]. The value works well for all the datasets that we consider. The value of tabu tenure determines the level of exploration for the search. When the value of tabu tenure is set too high, most of the available moves are not reachable and may restrict the search. When the value is too low, cycling tends to occur which may stall the search.

bestEvent is removed from *unplacedE* while the events conflicting with *bestEvent* are added to *unplacedE*. The iteration continues until *unplacedE* is empty (feasible solution is found) or the elapsed time exceeds the allowed time limit, T.

5. Proposed Methodology

The timetable is constructed by employing a two stage approach. 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 terms of the soft constraint violations in the second stage. The algorithm is shown in Algorithm 2, with further details below.

Algorithm 2

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

5.1. Stage 1: Finding a Feasible Solution

In stage 1, a feasible solution is built constructively by using Tabu Search with Sampling and Perturbation (TSSP). Only if a feasible solution is found (*unassignedE* is empty), is it passed to Simulated Annealing with Reheating (SAR) for soft constraint improvement.

5.1.1. Tabu Search with Sampling and Perturbation (TSSP)

We propose several enhancements to TS. The procedure is shown in Algorithm 3. It is important to note that no parameter tuning is required in this algorithm.

Algorithm 3

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

Instead of evaluating all the non-tabu time slots for all the events, we only evaluate the non-tabu time slots for certain sampled events. At the start of the iteration, S number of events are selected randomly from *unplacedE* and added to *sampleE* list (line 8). S is set as $0.0025 \times$ number of events. The event sample size is proportional to the number of events, e.g. event sample size is 1, 2 and 3 for the number of events between 1-400, 401-800 and 801-1200 respectively.

Rather than using the cost function based solely on the number of unplaced events, we propose a novel cost function which is based on the number of unplaced events plus the clash ratio:

$$\sum_{e \in unplacedE} 1 + \frac{clash[e]}{clashSum} \tag{5}$$

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

Unlike other implementations which tracked idle iterations before performing perturbation, we perturbed the current solution at certain iteration intervals $ITER$ (line 36-40). If $i = ITER$, *current* is perturbed, i is reset to 0 and tabu list is reset. In the PERTURB procedure (Algorithm 4), we attempt to move each assigned event to each time slot (except the time slot currently occupied by the event) in *slotList* (shuffled randomly) by using either a Swap or a Kempe operator. Maximal matching is used sparingly for room assignment. The event is moved only if the time slot under consideration is suitable for the event (not violating any hard constraints). Perturbation is used to explore the search space and does not affect the current solution as no event is extracted. However, when used too often it may slow down the search of a feasible solution. $ITER$ is set as $room^3$ (line 5). Essentially, the search is allowed to progress longer when the search space is larger before perturbation is initiated.

Algorithm 4

```
1: procedure PERTURB(solution)
2:     for all e ∈ solution do
3:         SHUFFLE(slotList)
4:         for all slot ∈ slotList do
5:             if RANDOM[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 neighborhood 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 is attempted [12], [18], [22]. A chain is built between events in time slot occupied by e (time slot A) and events in *slot* (time slot B). Initially, e is added to the chain. Next, events in time slot B clashing with e are added to the chain. Next, events in time slot A clashing with the chained events in time slot B are added to the chain. Then, events in time slot B clashing with the chained events in time slot A are added to the chain. The process is repeated until a complete chain is obtained. Subsequently, the chained events in time slot A are moved to time slot B and vice versa, assuming that all the hard constraints are satisfied.

As in TS, the algorithm is exited when a feasible solution is found or the time limit is exceeded. Therefore, the time or iteration number to find

a feasible solution varies for each instance. The time limit is determined by running a program on the executing machine (our machine is entitled to 190s).

5.2. Stage 2: Improving Soft Constraint Violations

In stage 2, we improved the feasible solution in terms of soft constraint violations by using a method based on SA. SA has been very effective in solving combinatorial optimization problems, particularly timetabling problems. In fact, all state of the art methods for the instances considered in this work are based on SA.

5.2.1. Simulated Annealing with Reheating (SAR)

The temperature in SA is used to determine the acceptance of uphill moves. In geometric cooling, as temperature gradually decreases, the search gradually switches from exploring to exploiting the search space.

We propose an improved SA called Simulated Annealing with Reheating (SAR). The method is 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. In SAR, we rely on the current cost to determine the initial temperature (rigorous setting of the initial temperature is bypassed) and how much to reheat when the search is stuck. In fact, we also rely on the current cost to determine whether the search is stuck in a local optima (inactive current cost through Markov chains indicates the search is stuck). As the temperature is reheated when a local optima is estimated at a certain low temperature, the setting of an end temperature as required in conventional SA is omitted. If the search is still stuck after the previous reheating, a higher temperature is applied for the next reheating. We estimate whether the search is still stuck in the previous local optima by utilizing the current and best cost. The approach is novel as the closest cost based reheating in the literature is based on the best cost and specific heat [13]. The details of SAR is shown in Algorithm 5.

Algorithm 5

```
1: procedure SAR(current, E)
2:     temp ← f(current) × C
3:     heat ← 0
4:     best ← current
5:     previousCost ← f(current)
6:     currentStagnantCount ← 0
7:     stuckedBestCost ← f(current)
8:     stuckedCurrentCost ← f(current)
9:
10:    while current is not optimal AND time.elapsed() < T do
11:        for all e ∈ E do
12:            moved ← false
13:            for slot = 1 to 45 do
14:                ns ← SELECTNEIGHBOURSTRUCTURE()
15:                candidate ← GETCANDIDATE(current, e, slot, ns)
16:                if candidate exists then
17:                    if RANDOM[0,1) ≤ exp(−(f(candidate)−f(current))/temp) then
18:                        moved ← true
19:                        current ← candidate
20:                        if f(current) < f(best) then
21:                            best ← current
22:                        end if
23:                    end if
24:                end if
25:                if moved then
26:                    break
27:                end if
28:            end for
29:        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 ← 0
                end if
            else
                heat ← 0
            end if
            temp ← [heat × 0.2 × f(current) + f(current)] × C
            stuckBestCost ← f(best)
            stuckCurrentCost ← f(current)
        else
            temp ← temp × β
        end if
        previousCost ← f(current)
    end while
end procedure
```

Algorithm 6

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

Algorithm 7

1: **procedure** SELECTNEIGHBOURSTRUCTURE( )
2: return a neighbourhood structure selected probalistically (Roulette Wheel) from a set of neighbourhood structures with predefined composition.
3: **end procedure**

At each temperature, a Markov chain is generated by deterministically trying to move each event $e \in E$ into each time slot (except the time slot currently occupied by e) using a neighbourhood structure selected probalistically from a set of neighbourhood structures with predefined composition as shown in Table 4. We use maximal matching (only when necessary) for room assignment.

Candidate solutions are feasible solutions which satisfy all the hard constraints. If a candidate solution exists, it 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, the best solution is updated.

Dataset	Neighbourhood Structure Composition (%)
Socha	Transfer: 70, Swap: 29, Kempe: 1
ITC02	Transfer: 70, Swap: 29, Kempe: 1
ITC07	Transfer: 70, Swap: 20, Kempe: 10

Table 4: Neighbourhood structure composition for dataset

The neighborhood structures used are:

- Transfer: Attempt to transfer e into *slot*. A feasible transfer is returned as a candidate for acceptance evaluation.
- Swap: A swap is attempted between e with event in each room (incrementing order) in *slot*. The first feasible swap is returned as a candidate for acceptance evaluation.

- Kempe: Same as described in section 5.1.1 except that a candidate is returned for acceptance evaluation.

In our implementation, a number of variables are maintained, namely *previousCost* (cost after each Markov chain is completed), *currentStagnantCount* (number of consecutive times current cost remains the same), *stuckBestCost* (best cost when the search is stuck) and *stuckCurrentCost* (current cost when the search is stuck).

The initial temperature is set as the initial cost multiplied by a constant C. This bypasses the manual setting of an initial temperature which is critical in conventional SA. The temperature is cooled according to an update rule $T_{i+1} = T_i \times \beta$.

After each Markov chain, we check whether the search is stuck in a local optima. In the STUCK procedure (Algorithm 6), we observe the changes of the current cost between Markov chains. *currentStagnantCount* is incremented by 1 if the difference between *f(current)* and *previousCost* is less than 1%. Otherwise, *currentStagnantCount* is set to 0.

If the search is stuck (*currentStagnantCount* is more than a threshold value set as 5), the temperature is reheated according to

$$temp \leftarrow [heat \times 0.2 \times f(current) + f(current)] \times C \tag{6}$$

where C is a constant and *heat* is the incremental step. For the first reheating, *heat* is usually set to 0 (line 38) thus the temperature is reheated to $f(current) \times C$ before being cooled again until the search is stuck in another local optima.

If the search is still stuck in the previous local optima (no new best since the previous reheating AND $f(current) - stuckCurrentCost < 2\%$), *heat* is incremented by 1 (line 33). Essentially, a higher temperature is applied for the next reheating so that the search can explore more in order to escape from the previous local optima.

If the search has escaped from the previous local optima ([a new best since the previous reheating] OR [no new best since the previous reheating AND $f(current) - stuckCurrentCost \geq 2\%$]), *heat* is set to 0 (line 35 and

38). In effect, the temperature is reheated to $f(current) \times C$ in order for the search to escape from the current local optima. Note that the setting of end temperature is omitted as the temperature is reheated when the search is stuck.

The series of cooling and reheating is repeated until an optimal solution is obtained or the elapsed time exceeds the time limit, T. We set the decay rate β to 0.9995 and the constant C to 0.01. The same values are used across all instances in our experiments.

6. Experimental Results

The experiments are performed on an Intel Xeon (3.1 GHz) with 4Gb RAM machine. The algorithms were coded in Java. The computation time limit allowed by running the benchmark program[4] is T=190 seconds for each single run. When a feasible solution is found, the focus is switched to minimizing 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.

6.1. Stage 1: Finding a Feasible Solution

6.1.1. The Effect of Sampling

Here, we present the effect of sampling on TS. Event sampling $S \propto$ event number, is compared with no sampling at one continuum end and $S = 1$ at the other continuum end. TS with sampling is more effective than TS without sampling for all the datasets in terms of the average number of unassigned events (Table 5) and average time to feasibility (Table 6). Dash symbols indicate that average time to feasibility is invalid as feasibility is not 100%. TS with sampling ($S \propto$ Event number) achieved 100% feasibility for all the datasets and therefore is preferred and used onwards.

[4]http://www.idsia.ch/Files/ttcomp2002/ Last accessed: May 23, 2017.

	Event Sampling		
Dataset	None	S=1	$S \propto$ Event number
Socha	0.00	0.00	0.00
ITC02	0.01	0.00	0.00
ITC07	0.03	0.00	0.00

Table 5: Average number of unassigned events

	Event Sampling		
Dataset	None	S=1	$S \propto$ Event number
Socha	1.0923	0.0243	0.0326
ITC02	-	-	0.0408
ITC07	-	0.8040	0.8007

Table 6: Average time to feasibility (s)

6.1.2. The Effect of Cost Functions and Perturbation

In this section, we present the effect of using different cost functions with or without perturbation on TS with sampling. The average number of unassigned events is given in Table 7. 100% feasibility is achieved when perturbation is used regardless of cost functions. The average time to feasibility is shown in Table 8. Dash symbols in the table indicate that the average time to feasibility is invalid because there are unassigned events. On the whole, perturbation improves the average time to feasibility. As evident in Table 8, $\sum_{e \in unplacedE} 1 + \frac{clash[e]}{clashSum}$ is the most effective cost function when used with or without perturbation. When the cost function is paired with perturbation, the average time to feasibility is further improved and in fact the lowest in a comparison.

	Cost Functions					
	$\sum_{e \in unplacedE} 1$		$\sum_{e \in unplacedE} size[e]$		$\sum_{e \in unplacedE} 1 + \frac{clash[e]}{clashSum}$	
Dataset	-	Perturbation	-	Perturbation	-	Perturbation
Socha	0.00	0.00	0.00	0.00	0.00	0.00
ITC02	0.00	0.00	0.07	0.00	0.00	0.00
ITC07	0.00	0.00	0.00	0.00	0.00	0.00

Table 7: Average number of unassigned events

	Cost Functions					
	$\sum_{e \in unplacedE} 1$		$\sum_{e \in unplacedE} size[e]$		$\sum_{e \in unplacedE} 1 + \frac{clash[e]}{clashSum}$	
Dataset	-	Perturbation	-	Perturbation	-	Perturbation
Socha	0.0326	0.0205	0.0139	0.0146	0.0132	0.0133
ITC02	0.0408	0.0197	-	0.0271	0.0211	0.0190
ITC07	0.8007	0.5612	0.2610	0.3113	0.2021	0.1953

Table 8: Average time to feasibility (s)

6.1.3. Comparing TS and TSSP

We compare the performance of TS and TSSP in finding feasible solutions. For Socha instances, both algorithms performed well with 100% feasibility. However, TSSP is faster as shown in Table 9. On average, the algorithm managed to find feasible solutions in less than one-tenth of a second. The p values (less than 0.05) reveal a significant difference between the means (time to feasibility) of TS and TSSP for all the instances.

Inst.	TS	TSSP	t-test (p value)
S1	0.0361	0.0032	0.000
S2	0.0268	0.0019	0.000
S3	0.0290	0.0013	0.000
S4	0.0397	0.0019	0.000
S5	0.0313	0.0019	0.000
M1	2.2906	0.0210	0.000
M2	2.0184	0.0242	0.000
M3	1.9681	0.0194	0.000
M4	1.8355	0.0203	0.000
M5	2.0655	0.0219	0.000
L	1.6742	0.0287	0.000

Table 9: Average time to feasibility for Socha instances

For ITC02, TSSP is more effective than TS in terms of feasibility and the number of unassigned events as shown in Table 10. TSSP achieved 100% feasibility for all the instances. As a comparison, TS achieved 100% feasibility for all the instances except instance 7 (87%). The p value of 0.039 (less than 0.05) revealed a significant difference between the means (unassigned events) of TS and TSSP for instance 7. Note that, the rest of the instances are omitted in Table 10 as they have means equivalent to 0. In addition, TSSP is faster than TS as shown in Table 11. On average, the algorithm managed to find feasible solutions in less than one-tenth of a second. The p values (less than 0.05) revealed a significant difference between the means (time to feasibility) of TS and TSSP for all the instances. Note: dash symbols in Table 11 indicate that the average time to feasibility is invalid (feasibility is not 100%) and therefore p value is invalid.

	TS			TSSP			
		Unassigned			Unassigned		t-test
Inst.	Fea.(%)	best	mean	Fea.(%)	best	mean	(p value)
7	87	0	0.13	100	0	0.00	0.039

Table 10: Number of unassigned events for ITC02 instances

Inst.	TS	TSSP	t-test (p value)
1	1.7419	0.0239	0.000
2	1.7355	0.0200	0.000
3	1.1168	0.0142	0.000
4	1.7348	0.0203	0.000
5	0.6163	0.0103	0.000
6	0.9790	0.0148	0.000
7	-	0.0181	-
8	1.8723	0.0213	0.000
9	1.9865	0.0319	0.000
10	1.5881	0.0232	0.000
11	1.8161	0.0203	0.000
12	1.5023	0.0226	0.000
13	1.5506	0.0177	0.000
14	1.0461	0.0148	0.000
15	1.0119	0.0135	0.000
16	2.1765	0.0339	0.000
17	0.4819	0.0084	0.000
18	1.5652	0.0187	0.000
19	1.4961	0.0200	0.000
20	1.0377	0.0129	0.000

Table 11: Average time to feasibility for ITC02 Instances

For ITC07, TSSP performed better than TS on average as shown in Table 12. TSSP managed to achieve 100% feasibility for all the instances. Meanwhile, TS achieved 100% feasibility for all the instances except instance 11 (87%), instance 19 (81%) and instance 23 (94%). The p value of 0.015 (less than 0.05) revealed a significant difference between the means (unassigned events) of TS and TSSP for instance 19. TSSP is also faster than TS as shown in Table 13. The algorithm managed to obtain feasible solutions

in less than one second except instance 22. The p values (less than 0.05) revealed a significant difference between the means (time to feasibility) of TS and TSSP for all the instances except instance 3.

	TS			TSSP			
		Unassigned			Unassigned		t-test
Inst.	Fea.(%)	best	mean	Fea.(%)	best	mean	(p value)
11	87	0	0.26	100	0	0.00	0.053
19	81	0	0.29	100	0	0.00	0.015
23	94	0	0.06	100	0	0.00	0.156

Table 12: Number of unassigned events for ITC07 instances

Inst.	TS	TSSP	t-test (p value)
1	3.0639	0.1797	0.000
2	10.0226	0.4126	0.000
3	3.1600	0.0055	0.299
4	0.1277	0.0142	0.000
5	1.1787	0.0229	0.000
6	1.1800	0.0281	0.000
7	0.3955	0.0090	0.000
8	0.1310	0.0052	0.000
9	30.1587	0.4516	0.000
10	27.8168	0.8026	0.000
11	-	0.0119	-
12	1.2755	0.0161	0.045
13	1.3423	0.0355	0.000
14	1.2823	0.0313	0.000
15	0.0906	0.0058	0.000
16	0.0855	0.0032	0.000
17	0.0232	0.0013	0.000
18	0.1894	0.0129	0.000
19	-	0.2139	-
20	0.9274	0.0181	0.000
21	1.9823	0.0690	0.000
22	61.9365	2.1113	0.000
23	-	0.1894	-
24	2.0416	0.0371	0.000

Table 13: Average time to feasibility for ITC07 instances

Indeed TSSP is able to always find feasible solutions for all the datasets. As TSSP is shown to be more effective, our focus will be on TSSP from here onwards.

6.1.4. Comparing TSSP with State of the Art Methods

Our method performed generally faster on average time (especially instances 10, 19, 23 and 24) than the Improved PARTIACOL by Lewis [22] while being equally effective (100% feasibility) in finding feasible solutions as shown in Table 14. As a comparison, the allowed time for Lewis's method was 247s.

	LS-Colouring [20]		I. PARTIALCOL [22]		TSSP	
Inst.	Time(s)	Fea.(%)	Time(s)	Fea.(%)	Time(s)	Fea.(%)
1	7.31	100	0.25	100	0.18	100
2	15.80	100	0.79	100	0.41	100
3	0.47	100	0.02	100	0.01	100
4	0.48	100	0.02	100	0.01	100
5	2.77	100	0.06	100	0.02	100
6	3.47	100	0.08	100	0.03	100
7	0.59	100	0.03	100	0.01	100
8	0.49	100	0.01	100	0.01	100
9	14.78	100	0.68	100	0.45	100
10	53.87	98	2.03	100	0.80	100
11	0.63	100	0.03	100	0.01	100
12	0.73	100	0.04	100	0.02	100
13	3.86	100	0.08	100	0.04	100
14	3.75	100	0.11	100	0.03	100
15	0.60	100	0.01	100	0.01	100
16	0.50	100	0.01	100	0.00	100
17	-	-	0.00	100	0.00	100
18	-	-	0.02	100	0.01	100
19	-	-	0.71	100	0.21	100
20	-	-	0.01	100	0.02	100
21	-	-	0.08	100	0.07	100
22	-	-	3.80	100	2.11	100
23	-	-	1.10	100	0.19	100
24	-	-	0.18	100	0.04	100

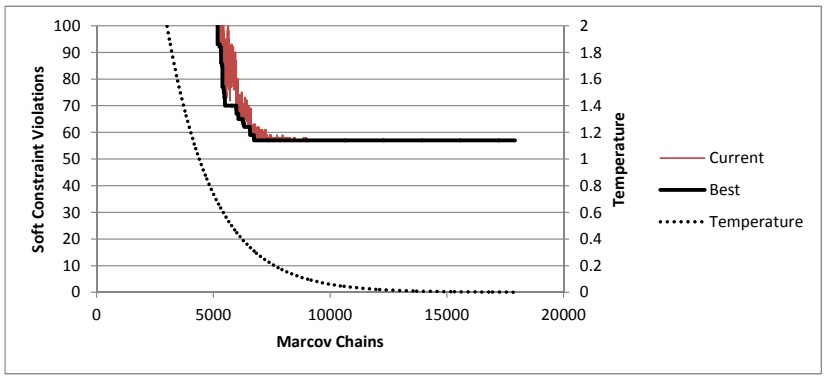
Table 14: Comparison of TSSP with state of the art methods on ITC07

6.2. Stage 2: Improving Soft Constraint Violations

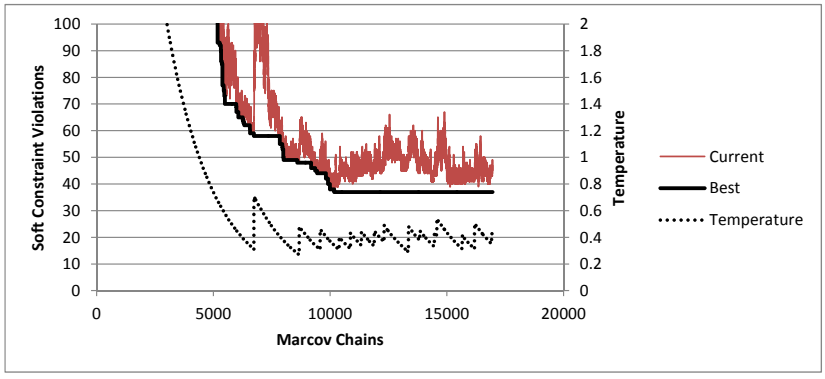
6.2.1. The Effect of Reheating in SAR

We run SAR with seed 1 on ITC02-1 instance. To compare the effect of reheating, we disabled the reheating part and run the algorithm with the same seed. The variation in temperature, current cost and best cost for both settings is presented in Figure 1b and 1a. The current cost and best cost for the algorithm with reheating disabled, becomes idle early in the search. Meanwhile, the current cost for the algorithm with reheating is active even towards the end of the search. In a comparison, a lower soft

constraint violation is achieved by the algorithm with reheating.



(a) SAR (reheating disabled)



(b) SAR

Figure 1: The effect of reheating on ITC02-1 instance

6.2.2. Comparing SAR with State of the Art Methods

We now compare SAR with the best results in the literature. Table 15 summarizes the details of the solvers.

Solver	Reference	Technique
A	Socha et al. [4]	Ant System
B	Burke et al. [27]	Tabu Search Hyperheuristic
C	McMullan [28]	Extended Great Deluge
D	Abdullah et al. [29]	Great Deluge + Tabu Search
E	Obit et al. [30]	Non Linear Great Deluge + Learning
F	Turabieh et al. [31]	Fish Swarm
G	Shaker and Abdullah [32]	Round Robin Multi Algorithms
H	Sabar et al. [33]	Honey Bee Mating
I	Ceschia et al. [16]	Simulated Annealing
J1	Kostuch [17]	Simulated Annealing
J2	Kostuch [19]	Simulated Annealing
K	Cordeau et al. [34]	Tabu Search
L	Burke et al. [35]	Great Deluge
M	DiGaspero and Schaerf [36]	Local Search + Tabu Search
N	Chiarandini et al. [18]	Hybrid Algorithm
O	Cambazard et al. [20]	Simulated Annealing
P	Nothegger et al. [21]	Ant Colony Optimization
Q	Lewis and Thompson [22]	Simulated Annealing

Table 15: Solver details

SAR outperformed (best results are in bold) all the other solvers for all Socha instances as shown in Table 16. It is interesting to note that our averages are far better than the best produced by other solvers over all instances. We found optimal solutions for 9 out of the 11 instances. Note that solver A was run according to time limit set initially (small instances: 90s, medium instances: 900s and large instances: 900s). For solver B-H, no time limit was followed in their implementations.

Inst.	Solver									
	A	B	C	D	E	F	G	H	I	SAR
S1	1	1	**0**(0.8)	**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**)
S3	1	**0**	**0**(1.3)	**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**)
S5	**0**	**0**	**0**(0.2)	**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**)
M2	184	173	105(116.9)	92	37	40	108	88	15(25.9)	**0**(**2.2**)
M3	248	267	139(162.1)	135	60	61	135	129	36(49.0)	**7**(**13.4**)
M4	164.5	169	88(108.8)	75	39	35	75	74	12(23.8)	**0**(**0.7**)
M5	219.5	303	88(119.7)	68	55	49	160	64	3(10.9)	**0**(**1.2**)
L	851.1	1166	730(834.1)	556	638	407	589	523	208(259.8)	**165**(**206.6**)

Table 16: Results comparison for Socha instances. Depicted is best(mean) for n=31 runs. Note that some authors only reported their best results.

Results comparison for ITC02 is given in Table 17. J1 was the official winner of ITC02. The solver N appeared post competition and was competitive with the solver J1. Not long after that, J2 was presented and became the state of the art method with the best known results for all the instances. J2 was an improvement of J1 by the same author. Since then, no other solvers are able to beat the results of the solver J2 on any of the ITC02 instances. We have managed to achieve that. Our results are competitive or better than the other solvers on all the instances. In fact, we managed to get optimal solutions for 7 out of 20 instances in comparison to the four of the solver J2.

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

Table 17: Results comparison for ITC02 instances. Depicted is best(mean) for n=31 runs. Note that some authors only reported their best results.

Table 18 shows the results comparison for ITC07. The solver O was the

official winner of ITC07. The solver P is based on Ant Colony Optimization and is the only competitve algorithm which is not SA based. However, SA was present in their approach and played a critical role in performance. As presented, our results are competitive compared to the other solvers We managed to obtain optimal solutions for 15 out of 24 instances. The solver O and P did not attempt their methods on instances 17-24.

	Solver				
Inst.	O	P	I	Q	SAR
1	15(547.0)	**0**(613.0)	59(399.2)	**0**(377.0)	**0**(**307.6**)
2	9(403.0)	**0**(556.0)	**0**(142.2)	**0**(382.2)	**0**(**63.4**)
3	174(254.0)	**110**(680.0)	148(209.9)	122(**181.8**)	163(199.4)
4	249(361.0)	53(580.0)	25(349.6)	**18**(**319.4**)	242(328.8)
5	**0**(26.0)	13(92.0)	**0**(7.7)	**0**(7.5)	**0**(**2.7**)
6	**0**(16.0)	**0**(212.0)	**0**(**8.6**)	**0**(22.8)	**0**(33.2)
7	1(8.0)	**0**(4.0)	**0**(4.9)	**0**(5.5)	5(18.0)
8	**0**(**0.0**)	**0**(61.0)	**0**(1.5)	**0**(0.6)	**0**(**0.0**)
9	29(1167.0)	**0**(202.0)	**0**(258.8)	**0**(514.4)	**0**(**100.7**)
10	2(1297.0)	**0**(**4.0**)	3(186.4)	**0**(1202.4)	**0**(65.3)
11	178(361.0)	143(774.0)	142(269.5)	**48**(**202.6**)	161(244.3)
12	14(380.0)	**0**(538.0)	267(400.0)	**0**(340.2)	**0**(**318.2**)
13	**0**(135.0)	5(360.0)	1(120.0)	**0**(79.0)	**0**(99.5)
14	**0**(15.0)	**0**(41.0)	**0**(3.6)	**0**(0.5)	**0**(**0.2**)
15	**0**(47.0)	**0**(**29.0**)	**0**(48.0)	**0**(139.9)	**0**(192.0)
16	1(58.0)	**0**(101.0)	**0**(**50.1**)	**0**(105.2)	10(105.8)
17	-	-	**0**(**0**)	**0**(0.1)	**0**(0.8)
18	-	-	**0**(41.1)	**0**(**2.2**)	**0**(12.5)
19	-	-	**0**(951.5)	**0**(**346.1**)	**0**(516.7)
20	-	-	543(700.2)	557(724.5)	586(**650.7**)
21	-	-	5(35.9)	1(32.1)	**0**(**12.5**)
22	-	-	5(**19.9**)	4(1790.1)	**1**(136.0)
23	-	-	1292(1707.7)	**0**(514.1)	11(**504.4**)
24	-	-	**0**(**105.3**)	18(328.2)	5(192.6)

Table 18: Results comparison for ITC07 instances. Depicted is best(mean) for n=31 runs.

6.2.3. Extended Runtime for SAR

Up to this point, all the experiments were conducted according to the time limit provided by the competition benchmark program. Out of curiosity, we also performed some experiments to see the effects of an extended runtime with regard to soft constraint violations. We selected one hard instance from each dataset namely Socha:Large, ITC02:1, ITC07:1 and ran for five times time limit or $5 \times T$. It took around 8 hours to run each instance for 31 times. As shown in Table 19, the algorithm is scalable as the best and average cost improved significantly when the run time is extended. In fact, we managed to obtain the best known results for the instances. It is important to note that we simply reset the run time in the algorithm without tuning any parameters, as is often required in a conventional SA e.g. decay rate. The p values ($0.000 < 0.05$) of t-tests failed to reject the null hypotheses $H_0 : \mu_{190s} = \mu_{1900s}$ and revealed a statistically difference between the mean between the runtime of 190s and 1900s.

	T=190s		$5\,T$		t-test
Inst.	best	mean	best	mean	(p value)
Socha: Large	165	206.61	103	139.39	0.000
ITC02: 1	23	32.61	10	21.03	0.000
ITC07: 1	0	307.55	0	134.94	0.000

Table 19: Comparison between different runtime on selected instances

7. Discussion

TSSP does not require parameter tuning as the values such as event sample size S and $ITER$ in the algorithm are determined automatically based on the characteristics of the specific instances.

TSSP is not only effective but also fast in finding feasible solutions. In our opinion, the sampling of events reduces computational cost as less evaluation is needed before a move is made, permitting more moves per time unit.

In our implementation, the sampling ratio (event sample size : number of unassigned events) increases as more events are assigned. For instance,

the event sample size for 1000 unassigned events is 3. Initially, the sampling ratio is 0.0025 (0.25%). When the number of unassigned events decreases to 6, the sampling ratio is 0.5 (50%). When the number of unassigned events decreases to 3 or below, sampling ratio is 1.0 (100%) where all the events will be selected for evaluation without sampling. Naturally, the sampling that we adopted allows the search to switch from diversification (exploration) to intensification (exploitation) and vice versa depending on the number of unassigned events.

Meanwhile, the proposed cost function increases the probability of unassigned events to be assigned later as they have the least number of clash with other events.

We believe perturbation enhances the diversification capability of TS, allowing the algorithm to explore the search space better which in turn alleviates the phenomena of cycling which is problematic in TS. Instead of trying to move random assigned events to random time slots, we attempt to move all assigned events to random time slots. As a result, a solution is thoroughly perturbed and the search is able to explore other areas of the search space effectively or possibly escape from local optima (if the search is stuck).

TSSP assisted SAR in obtaining the good results. As only a fraction of time is used by TSSP to find feasible solutions, more time is allocated for SAR to improve the soft constraint violations.

The right neighborhood structure composition used in SAR contributed to the good results. For Socha and ITC02 instances, the search spaces are well connected by transfer and swap operators. Therefore, the Kempe operator is redundant for these instances. Furthermore, the Kempe operator is computationally more expensive, thus reducing the number of transitions attempted. Meanwhile, for the ITC07 instances, the search space is poorly connected by transfer and swap operators. Thus, a higher composition of Kempe operators is worthwhile for these instances as it increases the connectivity of search space. In fact, ITC07 instances are more constrained compared to the instances of Socha and ITC07 as there are two additional hard constraints for ITC07 instances (order of events and preset time slots).

The neighborhood examination scheme applied in SAR, played a key role in achieving the good results. Attempts were made to deterministically move each event to each time slot (1-45) in each Markov chain. The scheme allows neighbors to be examined thoroughly and systematically with lesser redundancy.

SAR is able to estimate whether the search is stuck in a local optima precisely, thus allows reheating to be applied at the right time. In addition, incremental reheating provides exploration opportunities for the search to escape from local optima while ensuring that the current solution does not stray too much thus preserving the previous search effort. As a result, the search is effective in escaping from local optima.

The rigorous setting of initial and end temperature in conventional SA is bypassed in SAR. We set the decay rate β and the constant C as 0.9995 and 0.01 respectively. Nonetheless, the values work fine for the all the instances considered in this work.

8. Conclusion

We have presented the effect of sampling on TS. We have compared the effect of using different cost functions with or without perturbation on TS with sampling.

TSSP is shown to be more effective in finding feasible solution for the benchmark timetabling problem compared to TS. The number of unassigned events and average time to feasibility are presented for all the datasets. In addition, t-tests are conducted to compare the means for these values between TS and TSSP. TSSP managed to find 100% feasibility for all Socha, ITC02 and ITC07 instances in relatively short time compared to existing methods in the scientific literature.

We also presented SA with reheating (SAR) to improve the soft constraint violations of the feasible solution. The effect of reheating in SAR is displayed. Overall, competitive results in terms of soft constraint violations are reported in all datasets tested. Moreover, SAR is also shown to be scalable when the runtime is extended.

9. Future Work

We are looking forward to utilize the methods on other timetabling instances (ITC2011) or possibly other optimization problems to test the robustness and general applicability of the algorithms proposed in this paper.

Since the composition of neighborhood structures play an important role and were set manually, we are looking at the possibility of varying the composition automatically as the search progresses.

We are aware of the limitation of using the current cost to determine the level of reheated temperature as different instances may need different level of exploration to search effectively. Therefore, we are considering to incorporate the average change in cost of all uphill moves into the temperature reheating function.

10. References

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

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

[3] T. B. Cooper, J. H. Kingston, The complexity of timetable construction problems, Springer, 1996.

[4] K. Socha, J. Knowles, M. Sampels, A max-min ant system for the university course timetabling problem, in: Ant algorithms, Springer, 2002, pp. 1–13.

[5] R. Qu, N. Pham, R. Bai, G. Kendall, Hybridising heuristics within an estimation distribution algorithm for examination timetabling, Applied Intelligence 42 (4) (2015) 679–693.

[6] R. Bai, E. K. Burke, G. Kendall, J. Li, B. McCollum, A Hybrid Evolutionary Approach to the Nurse Rostering Problem, IEEE Transactions on Evolutionary Computation 14 (4) (2010) 580–590.

[7] S. Haddadene, N. Labadie, C. Prodhon, A GRASP x ILS for the vehicle routing problem with time windows, synchronization and precedence constraints, Expert Systems with Applications 66 (2016) 274–294.

[8] A. Zeb, M. Khan, N. Khan, A. Tariq, L. Ali, F. Azam, S. H. I. Jaffery, Hybridization of simulated annealing with genetic algorithm for cell formation problem, International Journal of Advanced Manufacturing Technology 86 (5-8) (2016) 2243–2254.

[9] P. Lopez-Garcia, E. Onieva, E. Osaba, A. D. Masegosa, A. Perallos, GACE: A meta-heuristic based in the hybridization of Genetic Algorithms and Cross Entropy methods for continuous optimization, Expert Systems with Applications 55 (2016) 508–519.

[10] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, Equation of state calculations by fast computing machines, The journal of chemical physics 21 (6) (1953) 1087–1092.

[11] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, et al., Optimization by simmulated annealing, science 220 (4598) (1983) 671–680.

[12] J. M. Thompson, K. A. Dowsland, Variants of simulated annealing for the examination timetabling problem, Annals of Operations research 63 (1) (1996) 105–128.

[13] D. Abramson, M. K. Amoorthy, H. Dang, Simulated annealing cooling schedules for the school timetabling problem, Asia-Pacific Journal of Operational Research 16 (1) (1999) 1–22.

[14] N. Ejaz, M. Y. Javed, A hybrid approach for course scheduling inspired by die-hard co-operative ant behavior, in: Automation and Logistics, 2007 IEEE International Conference on, IEEE, 2007, pp. 3095–3100.

[15] G. M. Jaradat, M. Ayob, An elitist-ant system for solving the post-enrolment course timetabling problem, in: Database Theory and Application, Bio-Science and Bio-Technology, Springer, 2010, pp. 167–176.

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

[17] P. Kostuch, Timetabling competition-sa-based heuristic, International Timetabling Competition.

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

[19] P. Kostuch, The university course timetabling problem with a three-phase approach, in: Practice and Theory of Automated Timetabling V, Springer, 2005, pp. 109–125.

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

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

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

[23] F. Glover, Future paths for integer programming and links to artificial intelligence, Computers & operations research 13 (5) (1986) 533–549.

[24] I. Blöchliger, N. Zufferey, A graph coloring heuristic using partial solutions and a reactive tabu scheme, Computers & Operations Research 35 (3) (2008) 960–975.

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

[26] L. A. Taylor, Local search methods for the post enrolment-based course timetabling problem, Ph.D. thesis, Cardiff University (2013).

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

[28] P. Mcmullan, An extended implementation of the great deluge algorithm for course timetabling, in: Computational Science–ICCS 2007, Springer, 2007, pp. 538–545.

[29] S. Abdullah, K. Shaker, B. McCollum, P. McMullan, Construction of course timetables based on great deluge and tabu search, in: Metaheuristics Int. Conf., VIII Meteheuristic, 2009, pp. 13–16.

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

[31] H. Turabieh, S. Abdullah, B. McCollum, P. McMullan, Fish swarm intelligent algorithm for the course timetabling problem, in: Rough Set and Knowledge Technology, Springer, 2010, pp. 588–595.

[32] K. Shaker, S. Abdullah, Controlling multi algorithms using round robin for university course timetabling problem, in: Database Theory and Application, Bio-Science and Bio-Technology, Springer, 2010, pp. 47–55.

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

[34] J. F. Cordeau, B. Jaumard, R. Morales, Efficient timetabling solution with tabu search, International Timetabling Competition.

[35] E. Burke, Y. Bykov, J. Newall, S. Petrovic, A time-predefined approach to course timetabling, The Yugoslav Journal of Operations Research ISSN: 0354-0243 EISSN: 2334-6043 13 (2).

[36] L. Di Gaspero, A. Schaerf, Timetabling competition ttcomp 2002: solver description, International Timetabling Competition.