OPSEARCH
https://doi.org/10.1007/s12597-020-00444-x

APPLICATION ARTICLE

An effective hybrid local search approach for the post enrolment course timetabling problem

Say Leng Goh[1] · Graham Kendall[1,2,3] · Nasser R. Sabar[1,4] · Salwani Abdullah[5]

Accepted: 7 May 2020

Abstract

We address the post enrolment course timetabling (PE-CTT) problem in this paper. PE-CTT is known as an NP-hard problem that focuses on finding an efficient allocation of courses onto a finite number of time slots and rooms. It is one of the most challenging resources allocation problems faced by universities around the world. This work proposes a two-phase hybrid local search algorithm to address the PE-CTT problem. The first phase focuses on finding a feasible solution, while the second phase tries to minimize the soft constraint violations of the generated feasible solution. For the first phase, we propose a hybrid of Tabu Search with Sampling and Perturbation with Iterated Local Search. We test the proposed methodology on the hardest cases of PE-CTT benchmarks. The hybrid algorithm performs well and our results are superior compared to the recent methods in finding feasible solutions. For the second phase, we propose an algorithm called Simulated Annealing with Reheating (SAR) with two preliminary runs (SAR-2P). The SAR algorithm is used to minimize the soft constraint violations by exploiting information collected from the preliminary runs. We test the proposed methodology on three publicly available datasets. Our algorithm is competitive with the standards set by the recent methods. In total, the algorithm attains new best results for 3 cases and new best mean results for 7 cases. Furthermore, it is scalable when the execution time is extended.

Keywords Tabu Search with Sampling and Perturbation (TSSP) · Iterated local search (ILS) · TSSP-ILS · SAR · SAR-2P

1 Introduction

Combinatorial optimization problems (COP) require decision making on the values for variables in a discrete search space, seeking to optimize (maximise or minimize) an objective function. Traveling salesman, vehicle routing, bin packing, timetabling and

✉ Say Leng Goh
gohsayleng@yahoo.com

Extended author information available on the last page of the article

Published online: 20 June 2020

minimal spanning tree are some examples of COP. Timetabling problem involves the placement of resources in time and space in such a way to optimize utilization and satisfy stakeholders' requirements. Instances of this problem include sports timetabling, educational timetabling, nurse rostering, transportation timetabling, etc. We focus on the PE-CTT problem, a type of educational timetabling. In this context, the timetable is a placement of courses into time slots and rooms, fulfilling a set of constraints. University students are allowed to select their favourite courses every semester. Therefore, the general prerequisite is that they can attend all their registered courses without hindrance such as clashing of courses. In other words, events attended by a student should not be allotted to the identical time slot. This requirement actually resembles that of graph coloring problem where two connected nodes cannot be given the same colour. In fact, timetabling problem can be reduced to graph coloring problem (known to be NP-hard) [6,7]. Even et al. [8] and Cooper and Kingston [5] shows that timetable construction is NP-hard. Additional requirements further complicate the process of course timetabling. Therefore, it is complicated and time consuming to manually construct a course timetable for hundreds (possibly thousands) of courses and students. Nowadays, course timetabling is usually automated. It is widely believed that exact methods cannot solve NP-hard problems optimally in polynomial time. Alternatively, heuristic based approximation methods are used to produce satisfactory solutions in decent computational times. In this work, we propose a two-phase hybrid local search algorithm to effectively handle the PE-CTT problem. Our contributions are as follows:

- We test the Tabu Search with Sampling and Perturbation (TSSP) [9] algorithm for the first time on the 60 hardest publicly available cases of PE-CTT benchmarks. Furthermore, we propose to enhance the TSSP by hybridizing it with Iterated Local Search (ILS). We show the effectiveness of this hybrid by comparing it with each individual algorithm run separately. The hybrid is far superior compared to the recent methods found in the scientific literature where it achieved 60 best results and 59 best means out of the 60 cases.
- We also propose an algorithm called Simulated Annealing with Reheating (SAR) with two preliminary runs (SAR-2P) in minimizing the soft constraint violations. The SAR algorithm uses the inputs from the preliminary runs. The first preliminary run employs a reinforcement learning based methodology to estimate a good neighbourhood structure composition. This will be used in the second preliminary run where the average cost changes $\overline{\Delta f}$ is measured. Both the estimated composition and $\overline{\Delta f}$ will then be utilized in SAR-2P in minimizing the soft constraint violations. Unlike conventional SA where preliminary run is embedded in each run, we perform preliminary run for once and utilize the information gathered for multiple runs for each case. We test this novel method on three benchmark datasets for PE-CTT problem and the results are compared with recent methods.

The structure of this paper is as follows. We describe the problem in Sect. 2. The related work is presented in Sect. 3. Our methodology is proposed in Sect. 4. We present the experimental results in Sect. 5. Section 6 gives concluding remarks. Lastly, we provide some suggestions for future work in Sect. 7.

2 Problem description

In PE-CTT problem, we construct a weekly timetable by assigning a set of C courses to 45 time slots (5 days a week, 9 time slots a day) and R rooms. The courses has F features and are attended by S students. The aim is to fulfill all hard constraints (to obtain a feasible solution) and minimize soft constraint violations (to obtain a higher quality solution). Entire datasets have the following hard constraints (HC):

- HC1: Students can only attend one course at a time.
- HC2: Rooms must have specific features required by courses.
- HC3: Rooms must have enough capacity for students attending courses.
- HC4: Only one course is allowed in each room at a time.

ITC07 dataset has extra two hard constraints:

- HC5: Courses may have to be scheduled to predefined time slots.
- HC6: Courses may have to appear in a specific sequence.

All the datasets (except Hard) have the following soft constraints (SC):

- SC1: Students should have more than one course on a day.
- SC2: Students should have less than three consecutive courses.
- SC3: Students should not have a course in the last time slot of the day.

Note that soft constraint violations are omitted for the Hard dataset. Instead the focus is on minimizing the hard constraint violations (finding feasible solutions). The benchmark datasets utilized in this work are publicly available. Optimal timetables (zero hard and soft constraint violations) are believed to exist except Hard where only hard constraints are considered.

- **Hard** comprises 60 cases.
- **Socha** comprises 11 cases [19].
- **ITC02** comprises 20 cases.
- **ITC07** comprises 24 cases.

Refer to Table 1 for the size of the problems we are dealing with, in terms of number of events, rooms, features and students. We use the standard execution time limit (as in ITC02) which is set by executing an application file on the host machine. Our machine is granted 190 s.

3 Related work

Lewis and Paechter employed Grouping Genetic Algorithms (GGA) in constructing feasible solutions for the Hard dataset [13]. The authors treated university course timetabling problem (UCTP) as a grouping problem. Groups of time slots were used for representations and genetic operators. They revamped the recombination operator used in the standard GGA (comprising of point selection, injection, duplicate removal using adaptation and reconstruction). Apart from the recombination operator, they also used mutation and inversion as operators. Several fitness functions were tested. A

Table 1 Description of datasets

Dataset	Hard			Socha			ITC02	ITC07
Case	S01-20	M01-20	B01-20	S01-05	M01-05	L	01-20	01-24
Event	200–225	390–425	1K–1.075K	100	400	400	350–440	100–600
Room	5–6	10–11	25–28	5	10	10	10–11	10–20
Feature	3–10	5–10	10–25	5	5	10	5–10	10–30
Student	200–1K	400–1K	800–1.2K	80	200	400	200–350	300–1K

local search was employed following the mutation operator to improve the algorithm. In a conclusion, local search outperformed GGA on large problem cases.

In addressing the Hard dataset, Tuga et al. applied a sequential heuristic approach (ISheuristic) comprising of largest degree and degree to construct the initial solution [22]. They placed unassigned events into artificial time slots. The unassigned events were considered as soft constraint violations and minimized utilizing Hybrid Simulated Annealing (HSA) with simple, swap and Kempe chain as neighbourhood structures. They set the initial temperature to an adequately high value. They used a cooling equation identical to that utilized by Kostuch [12]. The number of attempts for every temperature was fixed to $a \cdot number\ of\ events$. a is originally set to 10 and linearly raised. After a certain number of iterations, the temperature was altered when no advancement was detected. Their result was comparable to that of Lewis for small cases. They reported good results with many feasible solutions found for medium and big cases.

A clique based algorithm was used by Liu et al. in generating feasible timetables for the Hard dataset [15]. They were inspired by Carter and Johnson who believed that cliques found in the timetabling problems might be beneficial for timetable construction. The algorithm comprises of three steps. Firstly, the 45 cliques (equivalent to 45 time slots) were initialized. A vertex is randomly selected for a clique. The clique is then extended to include more vertices before a maximal matching is executed for room assignment. Secondly, the cliques were expanded further followed by a maximal matching. Thirdly, the algorithm swapped some vertices between two cliques. Superior results were reported in a comparison to other methodologies, particularly for the larger cases.

Recently, Song et al. applied an iterated local search algorithm for the Hard dataset [20]. A greedy heuristic procedure was utilized to generate an initial solution where each period was initialized one by one sequentially before allocating rooms using maximal matching. SA and perturbation procedure were then alternated until the stopping criteria was met. The algorithm outperformed other algorithms.

Ceschia et al. utilised Simulated Annealing (SA) on the Socha cases [3]. In their method, events were moved and swapped. Dummy time slots and rooms were exploited. They utilized a cost function comprising of various other components besides soft constraint violations, such as unscheduled events, precedences and conflicts. Weights were set for each component. Furthermore, specific parameters in simulated annealing were set using the F-race mechanism. Good results were reported compared to solvers by other researchers.

Kostuch was the winner of ITC02 with his SA based heuristic approach [11]. After the competition, he further improved his method on the same set of cases [12]. Graph coloring heuristics and maximum matching were used to construct feasible solutions which were then improved by ordering time slots and swapping pairs of events. Ten dummy events were also introduced (two events at each last time slot of the day) and removed in the final timetable. Breakthrough results were achieved on all the 20 cases.

Cambazard et al. won the PE-CTT of the ITC07 [2]. Feasible solutions were constructed by using a tabu based method utilizing the neighbourhood structures such as transferring events, interchanging events, interchanging time slots, matching (where events are reassigned within a time slot), transfering events with matching and Hun-

garian moves. The feasible solutions were then enhanced in regard to soft constraint violations by SA. Post competition, Lewis and Thompson utilized constructive heuristics and their PARTIALCOL algorithm to obtain feasible solutions [14]. The feasible solutions were enhanced by using SA. The beginning temperature was automatically set as the standard deviation of the cost of sample moves. To fulfill the time limit, the cooling rate was adjusted during the run. They found that a Kempe chain operator was effective for the cases. Strong results were attained.

One notable observation from the scientific literature is that most of the successful methods were based on SA. They are either highly tuned [3] or focused on certain cases [14], indicating the requirement of intensive parameter tuning in simulated annealing to obtain high quality results. In fact, most of them are conventional SA where the initial temperature is critical and has to be set sufficiently high for each case (a preliminary run is usually required for setting this value). In addition, the end temperature is critical in the performance of SA and needs to be set for a specific case. Furthermore, for problems restricted by a time limit, decay rate or Markov chain length needs to be set for each case.

Based on these limitations, Goh et al. proposed the SAR algorithm for timetabling problems [9]. It drew inspiration from the idea that the search should explore more when the current cost is high and exploit more when the current cost is low. Search exploration is directed by the current cost (through the initial and reheated temperature). The methodology does not require setting the end temperature because the temperature is reheated in case the search is stuck. In addition, no setting is required for the algorithm to run with differing time limits. The method showed competitive results and removed the requirement for rigorous tuning of certain parameters (beginning temperature, the end temperature and the Markov chain length), which is often required in a conventional SA. However, it has a scaling factor C needs to be set.

The SAR algorithm has limitations. It requires manual setting for neighbourhood structure composition to obtain good results. Goh et al. further improved the algorithm and proposed an algorithm termed as Simulated Annealing with Improved Reheating and Learning (SAIRL) [10]. A reinforcement learning method was used to adjust the composition of neighbourhood structures on the fly during the search. In addition, the average cost changes was incorporated into the reheated temperature function. Overall, improved results were reported.

Lately, Nagata employed a 2-stage approach for the problem [16]. In stage one, a tabu search based algorithm was used to construct an initial solution. In stage two, Random Partial Neighbourhood Search (RPNS) was used to reduce the soft constraint violations. RPNS is based on Tabu Search with exception that the tabu tenure is set to zero. The author found RPNS worked better when the neighbourhood sizes were varied during the course of the search. RPNS were configured for each dataset in terms of neighbourhood structure, ratio, update strategy and number of iterations. Highly competitive results were reported for the datasets tested especially ITC07.

Table 2 Comparing average cost changes $\overline{\Delta f}$ for different settings in terms of NES, NS and AC

NES	NS			AC
	Transfer	Swap	Kempe	
Random	4.43	4.07	7.35	Accept all
	5.39	4.97	9.60	Reject all
Deterministic	4.46	4.06	7.23	Accept all
	5.25	4.87	8.48	Reject all

$N = 1$ run for ITC02-1 case

4 Proposed methodology

In the SAR algorithm [9], the right composition of neighbourhood structures needs to be manually set for specific datasets to obtain good results. In this work, we propose a preliminary run to evaluate the neighbourhood structures so that a good composition can be obtained automatically.

All improving or equivalent moves relative to the current solution are accepted by SA based methods including SAR. Worse moves are also accepted with probability:

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

where Δf is the change in cost (solution quality) and T is the temperature. Thus, T can be derived as:

$$T = \frac{-\Delta f}{\ln P} \tag{2}$$

From the Eq. (2), T is proportional to Δf. Intuitively, we try to integrate the average cost changes $\overline{\Delta f}$ into the initial and reheated temperature function as we feel that the use of current cost solely to dictate the initial and reheated temperature, as the case in SAR, is rather limited.

We perform several runs of random walks on ITC02-1 case and found that the average cost changes $\overline{\Delta f}$ varies for different settings in terms of neighbourhood structure (NS), neighbourhood examination scheme (NES) and acceptance criteria (AC), as shown in Table 2. This suggests how search exploration can vary for different settings. Therefore, we propose to measure $\overline{\Delta f}$ in a preliminary run that resembles the actual run (SAR algorithm). This is to ensure the right $\overline{\Delta f}$ is obtained, thus the right temperature (initial and reheated) can be set according to the exploration level required.

An overview of the proposed timetabling is given in Algorithm 1. In the PRELIMINARYRUN1, we evaluate and estimate the right composition of each neighbourhood structure in NS for a specific problem case. This composition will be used in the PRELIMINARYRUN2 to measure the average cost changes ($\overline{\Delta f}$) to estimate the fitness landscape of the search space. Both the NS composition and $\overline{\Delta f}$ will be then utilized in the HYBRIDLOCALSEARCH.

Algorithm 1

```
1: procedure TIMETABLING
2:     E ← list of events
3:     NS ← {ns_1, ..., ns_|NS|}
4:     Δf̄ ← 0
5:
6:     PRELIMINARYRUN1(E, NS)                    ▷ Evaluating NS
7:     PRELIMINARYRUN2(E, NS, Δf̄)                ▷ Measuring Δf̄
8:     for run = 1 to 31 do
9:         HYBRIDLOCALSEARCH(E, NS, Δf̄)
10:    end for
11: end procedure
```

4.1 HYBRIDLOCALSEARCH

HYBRIDLOCALSEARCH is shown in Algorithm 2, with further details below. In phase 1 (Sect. 4.1.1), TSSP-ILS is utilized in constructing a feasible solution. When a feasible solution is constructed, it is improved with regard to soft constraint violations in phase 2 (Sect. 4.1.2), utilizing the SAR- 2P algorithm by using $\overline{\Delta f}$ and NS calculated earlier. Note that we do not reuse the feasible solution to ensure that initial solutions are scattered across the search space in the hope that the search space is fully explored and an optimal solution can be found.

Algorithm 2

```
1: procedure HYBRIDLOCALSEARCH(E, NS, Δf̄)
2:     bestSol ← vacant
3:     unassigned ← E
4:
5:     TSSP- ILS(bestSol, unassigned)            ▷ Phase 1: Constructing a feasible solution
6:     if unassigned is vacant then
7:         SAR- 2P(bestSol, E, NS, Δf̄)           ▷ Phase 2: Improving soft constraint violations
8:     end if
9: end procedure
```

4.1.1 Phase 1: constructing a feasible solution

A feasible solution is constructed by using Tabu Search with Sampling and Perturbation (TSSP) hybridized with Iterated Local Search (ILS), hence the term TSSP-ILS. This is shown in Algorithm 3. If TSSP fails to find a feasible solution (*unassigned* is not vacant) in $\frac{3}{4}$ of the execution time t, the best solution is passed to ILS for further processing using the remaining time.

Algorithm 3

```
1: procedure TSSP- ILS(bestSol, unassigned)
2:     TSSP(bestSol, unassigned)
3:     if unassigned is not vacant then
4:         ILS(bestSol, unassigned)
5:     end if
6: end procedure
```

Tabu Search with Sampling and Perturbation (TSSP)

The TSSP procedure was previously applied to course timetabling problems [9]. For continuity and clarity, the TSSP procedure is restated in Algorithm 4. A neighbour move is equivalent to transfering an event from the list of unplaced events *unplaced* to a time slot in the current solution *curSol*. At each iteration, we randomly select a certain number (0.25% of the total events) of events from *unplaced* and add them to the *sample* list. We evaluate neighbor moves by assessing all non-tabu suitable time slots for each event in the *sample* (lines 10–24). We temporarily remove the event e from *unplaced*. To feasibly move an event into a particular time slot, we move minimal conflicting events (violated clash or precedence constraint) from *curSol* to *unplaced*. We use matching sparingly for room assignment. In case matching cannot get a room for the event e, we randomly choose a suitable room and move the related event from *curSol* to *unplaced*. The candidate solution cost *f(canSol)* is calculated as the number of unplaced events plus the clash ratio as shown in Eq. (3). *clashSum* is the total number of clashes of all events.

$$\sum_{e \in unplaced} 1 + \frac{clash[e]}{clashSum} \tag{3}$$

We prefer the candidate solution with the least number of unplaced events and clashes with other events. Before evaluating the next non-tabu suitable time slot, we move the events conflicting with e in *unplaced* back to *curSol*. Before considering the next event, we move e back to *unplaced* after evaluating all the non-tabu time slots. We record the best neighbour move as *bestEvent* and *bestSlot* (lines 16–17).

We apply the best neighbour move to *curSol* by moving the *bestEvent* to the *bestSlot* (line 26) after extracting the conflicting events from *curSol*. We update *bestSol* and *f(bestSol)* if *f(curSol)* is better than *f(bestSol)*. We prevent the extracted events from returning to their original time slots for a number of iterations (line 33) according to the tabu tenure in Eq. (4).

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

where $|unplaced|$ is the number of unplaced events. Next, we remove the *bestEvent* from *unplaced* and put the extracted events into *unplaced*.

Algorithm 4 [9]

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

At certain iteration intervals ($i = ITER = room^3$), we perturb (Algorithm 5) *curSol*, reset i to 0 and reset tabu list. We perturb *curSol* by trying to move each assigned event to each time slot in *slotList* (shuffled randomly) utilizing either a swap or Kempe operator. We move the event only if it is feasible (not violating any hard constraints) to do so.

Algorithm 5

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

The PERTURB procedure is using the following neighbourhood structures:

- Swap: We attempt to swap *e* with the event in each room (room list shuffled randomly) in *slot*. The swap is implemented provided all the hard constraints are satisfied.
- Kempe: We attempt the Kempe chain interchange [4,14,21]. At first, *e* is added to a chain. Events in the time slot occupied by *e* and *slot* which clash with any event in the chain are gradually appended to the chain. When the chain is complete, the events in both time slots are interchanged if all the hard constraints are satisfied.

The TSSP procedure is halted when a feasible solution is attained (*unplaced* is vacant) or elapsed time passes $\frac{3}{4}$ of the execution time *t*. Note that this algorithm does not require parameter tuning.

Iterated local search (ILS)

In case there are remaining events in *unassigned* (Algorithm 3), ILS (Algorithm 6) will be initiated. The best solution *bestSol* generated in TSSP is utilized in this phase. At the beginning of each iteration, we perturb *bestSol* (line 3) by randomly shuffling the assigned events using the PERTURB procedure (Algorithm 5). Next, we attempt to swap each unassigned event *e1* in *unassigned* with each assigned event *e2* in *bestSol* (line 4–13). The swap will be implemented if *e2* has lesser number of clashes than *e1* and the time slot occupied by *e2* suits (not violating any hard constraints) *e1*. Subsequently, *unassigned* will have a group of easier events for assignment later. Next, we sort *unassigned* by number of clashes in descending order (line 14). Effectively, harder events will be scheduled first. We attempt to transfer every event in *unassigned* to every time slot in *bestSol* (line 15–22). The event *e* will be transferred to the time slot if no hard constraint is violated. The iteration stops when *unassigned* is vacant (feasible solution is found) or execution time *t* is exceeded.

Algorithm 6

```
1: procedure ILS(bestSol, unassigned)
2:     while unassigned is not vacant AND time.elapsed() < t do
3:         PERTURB(bestSol)
4:         for all e1 ∈ unassigned do
5:             for all e2 ∈ bestSol do
6:                 if clash[e2] < clash[e1] then
7:                     if slot of e2 is suitable for e1 then
8:                         swap e1 and e2
9:                         break
10:                    end if
11:                end if
12:            end for
13:        end for
14:        sort unassigned by number of clashes (descending order)
15:        for all e ∈ unassigned do
16:            for slot = 1 to 45 do
17:                if slot is suitable for e then
18:                    transfer e from unassigned to slot of bestSol
19:                    break
20:                end if
21:            end for
22:        end for
23:    end while
24: end procedure
```

4.1.2 Phase 2: improving soft constraint violations

SAR- 2P algorithm is shown in Algorithm 7. The changes to the original SAR are highlighted in boxes. Unlike SAR, it is utilizing the neighbourhood structure composition estimated in the PRELIMINARYRUN1 and average cost changes $\overline{\Delta f}$ measured in the PRELIMINARYRUN2. We set the initial temperature as the the initial cost $f(curSol)$ multiplied by $\overline{\Delta f}$ and a constant C as shown in the Eq. (5). The exploration level is dictated by the coefficient C.

$$temp \leftarrow f(curSol) \times \overline{\Delta f} \times C \tag{5}$$

A Markov chain is created (at each temperature) where each event $e \in E$ is attempted in each time slot (except the one occupied by e) utilizing a neighbourhood structure chosen in a probabilistic manner from a set of neighbourhood structures according to the composition estimated earlier. For room assignment, maximal matching is used (only when necessary).

If a candidate solution *canSol* (a solution that fulfills all the hard constraints) exists, it is evaluated where the improving or equal cost solution is accepted while the worsening solution is accepted with a certain probability. If accepted, the candidate solution will become the current solution. The best solution is updated provided the current solution is better than the best. The neighbourhood structures used are:

Algorithm 7

1: **procedure** SAR- 2P($curSol$, E, $\overline{\Delta f}$, NS)
2: $temp \leftarrow \overline{\Delta f} \times f(curSol) \times C$
3: $heat \leftarrow 0$
4: $bestSol \leftarrow curSol$
5: $previousCost \leftarrow f(curSol)$
6: $currentStagnantCount \leftarrow 0$
7: $stuckedBestCost \leftarrow f(curSol)$
8: $stuckedCurrentCost \leftarrow f(curSol)$
9:
10: **while** $terminationCondition$ = false **do**
11: **for all** $e \in E$ **do**
12: $moved \leftarrow$ false
13: **for** $slot = 1$ to 45 **do**
14: $ns_k \leftarrow$ SELECTNEIGHBOURHOODSTRUCTURE(NS)
15: $canSol \leftarrow$ GETCANDIDATE($curSol, e, slot, ns_k$)
16: **if** $canSol$ exists **then**
17: $\Delta f \leftarrow f(canSol) - f(curSol)$
18: **if** RANDOM[0,1) $\leq \exp(-\frac{\Delta f}{temp})$ **then**
19: $moved \leftarrow$ true
20: $curSol \leftarrow canSol$
21: **if** $f(curSol) < f(bestSol)$ **then**
22: $bestSol \leftarrow curSol$
23: **end if**
24: **end if**
25: **end if**
26: **if** $moved$ **then**
27: break
28: **end if**
29: **end for**
30: **end for**
31: **if** STUCK($f(curSol), previousCost, currentStagnantCount$) **then**
32: **if** $f(bestSol) = stuckBestCost$ **then**
33: **if** $f(curSol) - stuckCurrentCost < 2\%$ **then**
34: $heat = heat + 1$
35: **else**
36: $heat \leftarrow 0$
37: **end if**
38: **else**
39: $heat \leftarrow 0$
40: **end if**
41: $temp \leftarrow \overline{\Delta f} \times [heat \times 0.2 \times f(curSol) + f(curSol)] \times C$
42: $stuckBestCost \leftarrow f(bestSol)$
43: $stuckCurrentCost \leftarrow f(curSol)$
44: **else**
45: $temp \leftarrow temp \times \beta$
46: **end if**
47: $previousCost \leftarrow f(curSol)$
48: **end while**
49: **end procedure**

- Transfer: We attempt to transfer e into $slot$. A feasible transfer is evaluated for acceptance as a candidate solution.

- Swap: We attempt to swap e with event in each room (incrementing order) in $slot$. The first feasible swap is evaluated for acceptance as a candidate solution.
- Kempe: A Kempe chain interchange is attempted. As previously described, a complete chain is built between events in time slot occupied by e and events in $slot$. The events in both time slots are then exchanged (provided all the hard constraints are fulfilled) and evaluated for acceptance as a candidate solution.

After each Markov chain, the STUCK procedure (Algorithm 8) is initiated to check in case the search is stuck in a local optima. If the search is stuck, we reheat the temperature according to the Eq. (6) where $heat$ is an incremental step. Otherwise, we cool the initial temperature according to $T_{i+1} = T_i \times \beta$.

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

After reheating, the temperature is cooled again until the search is stuck in another local optima. A higher temperature is applied for the next reheating if the search is stuck in the previous local optima. A series of cooling and reheating is repeated until the $terminationCondition$ is true when either the elapsed time passes the execution time t or an optimal solution is obtained (all cases are known to have a zero-cost solution). The decay rate β is set to 0.9995, meanwhile coefficient C is set to 0.008.

Algorithm 8

```
1: procedure STUCK(f(curSol), previousCost, currentStagnantCount)
2:     if f(curSol) − previousCost < 1% then
3:         currentStagnantCount = currentStagnantCount + 1
4:     else
5:         currentStagnantCount ← 0
6:     end if
7:     if currentStagnantCount > 5 then
8:         return true
9:     else
10:        return false
11:    end if
12: end procedure
```

4.2 PRELIMINARYRUN1

The details of the PRELIMINARYRUN1 is shown in Algorithm 9. A feasible solution is built constructively by using the TSSP-ILS (Algorithm 3). When a feasible solution (which satisfies all the hard constraints) is constructed, it is passed to the SAR algorithm to evaluate the neighbourhood structures in use.

Like the original SAR, the initial and reheated temperature is set according to the Eq. (7).

$$temp \leftarrow [heat \times 0.2 \times f(curSol) + f(curSol)] \times C \tag{7}$$

Algorithm 9

```
1: procedure PRELIMINARYRUN1(E, NS)
2:     bestSol ← vacant
3:     unassigned ← E
4:
5:     TSSP- ILS(bestSol, unassigned)                ▷ Constructing a feasible solution
6:     if unassigned is vacant then
7:         SAR(bestSol, E, NS)                       ▷ Evaluating NS
8:     end if
9: end procedure
```

Unlike the original SAR (where the neighbourhood structure composition is preset manually), we adjust the composition by using a reinforcement learning method. Implementation wise, a $visit$ and a $value$ are maintained for each neighbourhood structure ns_k. Each time the neighbourhood structure ns_k is selected, $ns_k.visit$ is incremented by 1. Meanwhile, $ns_k.value$ is updated as a cumulative mean of rewards:

$$ns_k.value \leftarrow ns_k.value + \frac{reward - ns_k.value}{ns_k.visit} \tag{8}$$

where the reward is defined as:

$$reward = \begin{cases} 0, & \text{if candidate is accepted} \\ \text{CPU time}, & \text{otherwise} \end{cases} \tag{9}$$

At the beginning, all neighbourhood structures have an equal probability of being selected. Gradually, the probability varies according to:

$$P_{ns_k} = \frac{\frac{1}{ns_k.value}}{\sum_{k=1}^{|NS|} \frac{1}{ns_k.value}} \tag{10}$$

If a candidate solution is accepted, a reward of 0 is awarded to ns_k. Otherwise, the ns_k is penalized with CPU time (elapsed time since selection) in case the candidate is rejected or the candidate does not exist because a move is not feasible. As different neighbourhood structures may have different acceptance rates and computational costs for different cases, the objective is to maximize the number of accepted moves per time unit, with the hope that solution space connectivity can be improved. The neighbourhood structures used are transfer, swap and Kempe.

4.3 PRELIMINARYRUN2

The PRELIMINARYRUN2 is described in Algorithm 10. Again, the TSSP-ILS (Algorithm 3) is used to find a feasible solution. If a feasible solution is constructed, it is

passed to the SAR algorithm to measure the average cost changes $\overline{\Delta f}$ to estimate the fitness landscape.

Algorithm 10

```
1: procedure PRELIMINARYRUN2(E, NS, Δf̄)
2:     bestSol ← vacant
3:     unassigned ← E
4:
5:     TSSP- ILS(bestSol, unassigned)          ▷ Constructing a feasible solution
6:     if unassigned is vacant then
7:         SAR(bestSol, E, NS, Δf̄)             ▷ Measuring Δf̄
8:     end if
9: end procedure
```

The SAR algorithm here is utilizing the neighbourhood structure composition estimated in the PRELIMINARYRUN1. A *sum* and a *count* are initialized and updated during the run. Only uphill and downhill moves are considered for the computation of *sum* and *count*. Sideway moves are omitted to gain better insights on the fitness landscape of the search space. The $\overline{\Delta f}$ is calculated as $\frac{sum}{count}$.

5 Experimental results

We code the algorithms in Java and perform the experiments on machines (Intel Xeon) with 3.1 GHz clock speed and 4 Gb RAM. Each machine is entitled to a computational time limit of $T = 190$ s (dictated by executing a benchmark program provided by the ITC02 organizer). When a feasible solution is found, the remaining available time is used to either evaluate neighbourhood structures (in PRELIMINARYRUN1) or measure $\overline{\Delta f}$ (in PRELIMINARYRUN2) or improve the soft constraint violations (in HYBRIDLOCALSEARCH). We execute the HYBRIDLOCALSEARCH procedure for a total of 31 times for each case.

5.1 Phase 1: finding a feasible solution

5.1.1 Comparing TSSP-ILS with TSSP and ILS

The effectiveness of TSSP-ILS and TSSP in constructing feasible solutions is compared. Both methods are comparable in terms of performance on all the small and medium cases with 100% feasibility (Tables 3 and 4).

For the big cases, TSSP-ILS is more effective than TSSP in terms of feasibility and the number of unassigned events as shown in Table 5. TSSP-ILS manages to construct feasible solutions for 17 cases comparing to 15 for TSSP. TSSP-ILS also achieves a higher feasibility (%) for the cases B5, B6, B10, B17 and B18. Furthermore, TSSP-ILS improves the mean of unassigned events for the cases B5, B6, B7, B10, B11, B17, B18, B19 and B20. A *t*-test is conducted to compare the means between both

Table 3 Comparing feasibility (%), best(mean) of unassigned events among TSSP-ILS, TSSP and ILS on Hard:Small cases

Case	TSSP	ILS	TSSP-ILS	*t*-test (*p* value)	
				TSSP	ILS
S01	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
S02	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
S03	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
S04	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
S05	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
S06	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
S07	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
S08	100, 0 (0.00)	97, 0 (0.03)	100, 0 (0.00)	–	0.321
S09	100, 0 (0.00)	10, 0 (4.65)	100, 0 (0.00)	–	0.000
S10	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
S11	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
S12	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
S13	100, 0 (0.00)	45, 0 (2.52)	100, 0 (0.00)	–	0.000
S14	100, 0 (0.00)	0, 12 (20.16)	100, 0 (0.00)	–	0.000
S15	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
S16	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
S17	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
S18	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
S19	100, 0 (0.00)	3, 0 (27.77)	100, 0 (0.00)	–	0.000
S20	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–

$N = 31$ runs

algorithms for each case. The last column of the table shows the respective p values. t cannot be computed for some cases (as indicated by dash symbols) as the standard deviations of both groups are 0 (means are 0). The p values (less than 0.05) show a significant difference between the means (unassigned events) of TSSP and TSSP-ILS for all the other cases except B11.

We also compare TSSP-ILS and ILS in constructing feasible solutions. Apparently, TSSP-ILS is superior than ILS on all the cases (small, medium and big) in regard to feasibility and the number of unassigned events as presented in Table 3, 4 and 5. A significant difference is observed between the means (unassigned events) of ILS and TSSP-ILS for all the cases except S8 as indicated by the p values of t-tests.

5.1.2 Comparing TSSP-ILS with state of the art methods

Here, the performance of TSSP-ILS and the state of the art methods is compared. As shown in Tables 6 and 7, TSSP-ILS performs better than other methods for small and medium cases. Best results are in bold. TSSP-ILS also outperforms other methods for big cases as shown in Table 8. Feasible solutions are found for 17 out of the 20 cases utilizing a benchmarked execution time t of $T = 190$ s. Note that the results of the solver

Table 4 Comparing feasibility (%), best(mean) of unassigned events among TSSP-ILS, TSSP and ILS on Hard:Medium cases

Case	TSSP	ILS	TSSP-ILS	*t*-test (*p* value)	
				TSSP	ILS
M01	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
M02	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
M03	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
M04	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
M05	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
M06	100, 0 (0.00)	19, 0 (11.87)	100, 0 (0.00)	–	0.000
M07	100, 0 (0.00)	0, 72 (79.26)	100, 0 (0.00)	–	0.000
M08	100, 0 (0.00)	0, 21 (31.71)	100, 0 (0.00)	–	0.000
M09	100, 0 (0.00)	0, 29 (40.19)	100, 0 (0.00)	–	0.000
M10	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
M11	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
M12	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
M13	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
M14	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
M15	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
M16	100, 0 (0.00)	0, 57 (75.81)	100, 0 (0.00)	–	0.000
M17	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
M18	100, 0 (0.00)	0, 24 (61.32)	100, 0 (0.00)	–	0.000
M19	100, 0 (0.00)	10, 0 (42.13)	100, 0 (0.00)	–	0.000
M20	100, 0 (0.00)	48, 0 (8.45)	100, 0 (0.00)	–	0.000

$N = 31$ runs

D5 shown here is based on execution time of 200 s (small cases), 500 s (medium cases) and 1000 s (big cases). For reference, D5 is executed utilizing Pentium IV 2.66 GHz CPU and 1.0 Gb RAM machine. Details of the solvers are given in Table 9.

5.1.3 Extended execution time for TSSP-ILS

In this section, we measure the performance of TSSP-ILS using an extended execution time. We manage to construct feasible solutions for 19 of the 20 cases when the execution time t is doubled to $2T$ (380 s) as shown in Table 10. In a comparison, solver D5 finds feasible solutions for 18 cases. Furthermore, we conduct a t-test to compare the means of TSSP-ILS with execution time t of T and $2T$. The p values (less than 0.05) indicate a significant difference between the means of TSSP-ILS for both execution times.

We further extend the execution time t for cases with non zero means (B6, B7, B17, B18, B19, B20). The descriptive statistics are given in Table 11 and illustrated in Fig. 1. In fact, we manage to obtain best known results for all the cases when execution time is $4T = 760$ s except for the mean produced by the solver D5 for the case B20.

Table 5 Comparing feasibility (%), best(mean) of unassigned events among TSSP-ILS, TSSP and ILS on Hard:Big cases

Case	TSSP	ILS	TSSP-ILS	*t*-test (*p* value)	
				TSSP	ILS
B01	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
B02	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
B03	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
B04	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
B05	87, 0 (0.71)	100, 0 (0.00)	100, 0 (0.00)	0.047	–
B06	0, 1 (10.77)	0, 70 (90.13)	16, 0 (1.87)	0.000	0.000
B07	0, 11 (64.61)	0, 185 (191.65)	0, 23 (29.55)	0.000	0.000
B08	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
B09	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
B10	74, 0 (1.42)	100, 0 (0.00)	100, 0 (0.00)	0.006	–
B11	94, 0 (0.26)	100, 0 (0.00)	100, 0 (0.00)	0.314	–
B12	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
B13	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
B14	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
B15	100, 0 (0.00)	0, 130 (143.48)	100, 0 (0.00)	–	0.000
B16	100, 0 (0.00)	100, 0 (0.00)	100, 0 (0.00)	–	–
B17	0, 8 (13.45)	0, 313 (325.48)	6, 0 (3.97)	0.000	0.000
B18	23, 0 (5.06)	0, 181 (195.65)	48, 0 (0.84)	0.000	0.000
B19	0, 15 (35.39)	0, 301 (309.32)	0, 1 (16.97)	0.000	0.000
B20	0, 8 (18.87)	0, 167 (188.23)	0, 2 (6.65)	0.000	0.000

$N = 31$ runs

Meanwhile, the average time required to find feasible solutions (using TSSP-ILS algorithm) is less than 1 s for all the cases in Socha, ITC02 and ITC07 except the case ITC07-22 which requires approximately 2 s.

5.2 Phase 2: improving soft constraint violations

5.2.1 Comparing SAR-1P and SAR-2P

In this section, we compare the average of soft constraint violations between SAR-1P and SAR-2P using different values for the constant C. Note that SAR-1P is a SAR algorithm with one preliminary run. Specifically, the preliminary run consists of a TSSP algorithm and a SAR algorithm. Both the neighbourhood structure composition and fitness landscape are simultaneously estimated in the preliminary run. As evident in Table 12, SAR-2P is superior compared to SAR-1P regardless of the value of C. The lowest total average of soft constraint violations is recorded for SAR-2P with $C = 0.008$. Therefore this value is used in the other experiments.

Table 6 Comparing best(mean) of unassigned events among solvers on Hard:Small cases

Case	A1	A2	A3	A4	A5	TSSP-ILS
S01	**0**	**0**(**0.00**)	**0**(**0.00**)	–	**0**(**0.00**)	**0**(**0.00**)
S02	**0**	**0**(**0.00**)	**0**(**0.00**)	–	**0**(**0.00**)	**0**(**0.00**)
S03	**0**	**0**(**0.00**)	**0**(**0.00**)	–	**0**(**0.00**)	**0**(**0.00**)
S04	**0**	**0**(**0.00**)	**0**(**0.00**)	–	**0**(**0.00**)	**0**(**0.00**)
S05	**0**	**0**(**0.00**)	**0**(**0.00**)	–	**0**(**0.00**)	**0**(**0.00**)
S06	**0**	**0**(**0.00**)	**0**(**0.00**)	–	**0**(**0.00**)	**0**(**0.00**)
S07	**0**	**0**(**0.00**)	**0**(0.20)	–	**0**(**0.00**)	**0**(**0.00**)
S08	**0**	**0**(1.90)	**0**(0.30)	–	**0**(**0.00**)	**0**(**0.00**)
S09	**0**	**0**(3.85)	**0**(0.15)	–	**0**(0.55)	**0**(**0.00**)
S10	**0**	**0**(**0.00**)	**0**(**0.00**)	–	**0**(**0.00**)	**0**(**0.00**)
S11	**0**	**0**(**0.00**)	**0**(**0.00**)	–	**0**(**0.00**)	**0**(**0.00**)
S12	**0**	**0**(**0.00**)	**0**(**0.00**)	–	**0**(**0.00**)	**0**(**0.00**)
S13	**0**	**0**(1.00)	**0**(**0.00**)	–	**0**(0.10)	**0**(**0.00**)
S14	**0**	3(5.95)	**0**(0.70)	–	**0**(0.05)	**0**(**0.00**)
S15	**0**	**0**(**0.00**)	**0**(**0.00**)	–	**0**(**0.00**)	**0**(**0.00**)
S16	**0**	**0**(**0.00**)	**0**(0.30)	–	**0**(**0.00**)	**0**(**0.00**)
S17	**0**	**0**(**0.00**)	**0**(**0.00**)	–	**0**(**0.00**)	**0**(**0.00**)
S18	**0**	**0**(0.45)	**0**(0.70)	–	**0**(**0.00**)	**0**(**0.00**)
S19	**0**	**0**(1.20)	**0**(**0.00**)	–	**0**(0.25)	**0**(**0.00**)
S20	**0**	**0**(**0.00**)	**0**(0.15)	–	**0**(**0.00**)	**0**(**0.00**)

$N = 31$ runs

5.2.2 Comparing SAR-2P with SAR and SAIRL

In SAR algorithm, parameter values (neighbourhood structure composition) are set manually. SAIRL is a type of dynamic optimization where parameter values (neighbourhood structure composition and average cost changes used in the reheated temperature function) are determined and utilized on the run. Meanwhile, SAR-2P is a type of static optimization where parameter values are determined automatically in the preliminary runs. As we also developed SAR and SAIRL, we have the privilege to compare the performance between SAR, SAIRL and SAR-2P in relation to the average of soft constraint violations as presented in Table 13. SAR-2P is particularly effective for Socha and ITC07 datasets. Overall, SAR-2P is the most effective algorithm with the lowest total average of soft constraint violations.

We compare the performance of SAR, SAIRL and SAR-2P in terms of best and mean for Socha cases in Table 14. *t*-test results are presented in the last two columns between (SAR-2P and SAR) and between (SAR-2P and SAIRL) respectively. The *t*-tests are run with a confidence level of 95% ($\alpha = 0.05$). The *p* values reveal that there is no significant difference between the means of SAR and SAR-2P for all the cases. The same is true of SAIRL and SAR-2P except case L where SAR-2P performs significantly better. *P* values less than 0.05 are in bold.

Table 7 Comparing best(mean) of unassigned events among solvers on Hard:Medium cases

Case	A1	A2	A3	A4	A5	TSSP-ILS
M01	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)
M02	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)
M03	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)
M04	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)
M05	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)
M06	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (0.90)	**0** (**0.00**)	**0** (**0.00**)
M07	14	1 (4.15)	**0** (3.55)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)
M08	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (0.30)	**0** (**0.00**)	**0** (**0.00**)
M09	2	**0** (4.90)	**0** (2.15)	**0** (0.35)	0 (0.65)	**0** (**0.00**)
M10	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)
M11	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)
M12	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (0.60)	**0** (**0.00**)	**0** (**0.00**)
M13	**0**	**0** (0.50)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)
M14	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (0.05)	**0** (**0.00**)	**0** (**0.00**)
M15	**0**	**0** (0.05)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)
M16	1	1 (5.15)	**0** (0.30)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)
M17	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (0.15)	**0** (**0.00**)	**0** (**0.00**)
M18	**0**	**0** (6.05)	**0** (**0.00**)	**0** (0.30)	**0** (**0.00**)	**0** (**0.00**)
M19	**0**	**0** (5.45)	**0** (**0.00**)	**0** (0.50)	**0** (**0.00**)	**0** (**0.00**)
M20	3	2 (10.60)	**0** (0.65)	**0** (0.55)	**0** (**0.00**)	**0** (**0.00**)

$N = 31$ runs

For ITC02 cases, the *t*-tests show that SAR performs significantly better than SAR-2P for cases 1, 4, 9 and 16 as shown in Table 15. Meanwhile, SAR-2P is more effective than SAIRL for case 3. There is no significant difference between the means between SAR-2P and SAR as well as the means between SAR-2P and SAIRL, for the rest of the cases.

For ITC07 cases, SAR-2P performs significantly better compared to SAR for the cases 1, 2, 9, 10, 11, 15, 16, 19, 24 as shown in Table 16. SAR is better than SAR-2P for the case 14. Meanwhile, SAR-2P is better than SAIRL for the case 23. No significant difference is observed, neither between the means of SAR-2P and SAR nor between the means of SAR-2P and SAIRL, for the rest of the cases.

Overall, SAR-2P is significantly better than SAR on 9 cases. Meanwhile, SAR is significantly better than SAR-2P on 5 cases. SAR-2P is significantly better than SAIRL on 3 cases.

5.2.3 Comparing SAR-2P with state of the art methods

We now compare SAR-2P with the state of the art methods in the literature. Table 17 summarizes the details of the solvers.

The performance of SAR-2P is superior to that of all the solvers except solver R and solver S as shown in Table 18. Its performance is comparable to solver R and

Table 8 Comparing best(mean) of unassigned events among solvers on Hard:Big cases

Case	A1	A2	A3	A4	A5	TSSP-ILS
B01	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (0.15)	**0** (**0**)	**0** (**0.00**)
B02	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (0.60)	**0** (**0**)	**0** (**0.00**)
B03	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (1.45)	**0** (**0**)	**0** (**0.00**)
B04	8	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0**)	**0** (**0.00**)
B05	30	**0** (1.10)	1 (3.20)	**0** (**0.00**)	**0** (**0**)	**0** (**0.00**)
B06	77	5 (8.45)	10 (15.40)	1 (2.85)	**0** (**0.35**)	**0** (1.87)
B07	150	47 (58.30)	39 (46.65)	**21** (**29.25**)	22 (32)	23 (29.55)
B08	5	**0** (**0.00**)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0**)	**0** (**0.00**)
B09	3	**0** (0.05)	**0** (**0.00**)	**0** (**0.00**)	**0** (**0**)	**0** (**0.00**)
B10	24	**0** (1.25)	**0** (1.95)	**0** (**0.00**)	**0** (**0**)	**0** (**0.00**)
B11	22	**0** (0.35)	**0** (2.35)	**0** (**0.00**)	**0** (**0**)	**0** (**0.00**)
B12	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (1.15)	**0** (**0**)	**0** (**0.00**)
B13	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (1.15)	**0** (**0**)	**0** (**0.00**)
B14	**0**	**0** (**0.00**)	**0** (**0.00**)	**0** (1.20)	**0** (**0**)	**0** (**0.00**)
B15	**0**	**0** (**0.00**)	**0** (**0.00**)	1 (3.5)	**0** (**0**)	**0** (**0.00**)
B16	19	**0** (2.00)	**0** (**0.00**)	**0** (0.65)	**0** (**0**)	**0** (**0.00**)
B17	163	76 (89.90)	**0** (**2.05**)	12 (22.00)	**0** (4.0)	**0** (3.97)
B18	164	53 (62.60)	**0** (1.70)	8 (13.55)	**0** (**0**)	**0** (0.84)
B19	232	109 (127.00)	40 (53.20)	37 (52.85)	12 (24.55)	**1** (**16.97**)
B20	149	40 (46.70)	9 (14.05)	11 (15.05)	**0** (**0**)	2 (6.65)

$N = 31$ runs

Table 9 Details of solvers

Solver	Methodology	References
A1	GA	Lewis and Paechter [13]
A2	Hybrid SA	Tuga et al. [22]
A3	Clique Based Algorithm	Liu et al. [15]
A4	SA	Ceshia et al. [3]
A5	ILS	Ting Song et al. [20]

S. SAR-2P obtains optimal solutions for 9 out of 11 cases. Furthermore, SAR-2P achieves new best and mean for the case L.

For ITC02, our results are competitive or better than the other solvers on all the cases as shown in Table 19. SAR-2P obtains optimal solutions for 7 out of 20 cases in comparison to the solver J2 (four), solver R (seven) and solver S (seven). SAR-2P also achieves two new best results (cases 12 and 17) and three new means (cases 6, 12 and 14).

Table 20 shows the results comparison for ITC07. Our results are good compared to the other solvers. SAR-2P finds nineteen optimal solutions compared to the solver Q

Table 10 Comparing feasibility (%), best(mean) of unassigned events between TSSP-ILS with execution time t of T and $2T$

Case	$t=T$	$t=2T$	t-test (p value)
B01	100, 0(0.00)	100, 0(0.00)	–
B02	100, 0(0.00)	100, 0(0.00)	–
B03	100, 0(0.00)	100, 0(0.00)	–
B04	100, 0(0.00)	100, 0(0.00)	–
B05	100, 0(0.00)	100, 0(0.00)	–
B06	16, 0(1.87)	35, 0(1.00)	0.006
B07	0, 23(29.55)	0, 21(26.87)	0.006
B08	100, 0(0.00)	100, 0(0.00)	–
B09	100, 0(0.00)	100, 0(0.00)	–
B10	100, 0(0.00)	100, 0(0.00)	–
B11	100, 0(0.00)	100, 0(0.00)	–
B12	100, 0(0.00)	100, 0(0.00)	–
B13	100, 0(0.00)	100, 0(0.00)	–
B14	100, 0(0.00)	100, 0(0.00)	–
B15	100, 0(0.00)	100, 0(0.00)	–
B16	100, 0(0.00)	100, 0(0.00)	–
B17	6, 0(3.97)	19, 0(1.45)	0.000
B18	48, 0(0.84)	81, 0(0.23)	0.004
B19	0, 1(16.97)	3, 0(9.16)	0.000
B20	0, 2(6.65)	6, 0(3.35)	0.000

$N = 31$ runs

Table 11 Comparing best(mean) of unassigned events for TSSP-ILS with extended execution time t on selected Hard instances

Case	$t=1T$	$t=2T$	$t=3T$	$t=4T$	$t=5T$
Hard-B06	0(1.87)	0(1.00)	0(0.39)	0(0.35)	0(0.26)
Hard-B07	23(29.55)	21(26.87)	19(24.58)	17(23.74)	15(23.13)
Hard-B17	0(3.97)	0(1.15)	0(0.71)	0(0.61)	0(0.32)
Hard-B18	0(0.84)	0(0.23)	0(0.03)	0(0.00)	0(0.00)
Hard-B19	1(16.97)	0(9.16)	0(5.19)	0(3.39)	0(2.06)
Hard-B20	2(6.65)	0(3.35)	0(1.90)	0(1.45)	0(0.68)

$N=31$ runs

(seventeen), solver R (fifteen), solver S (eighteen) and solver T(nineteen). In addition, SAR-2P achieves three new means (cases 2, 9 and 17).

5.3 Extended execution time for SAR-2P

We run the algorithm for fivefold the time limit ($5T$ = 950 s), only on selected cases (Socha-L, ITC02-1 and ITC07-1) as it took around 8 hours to run each case for 31

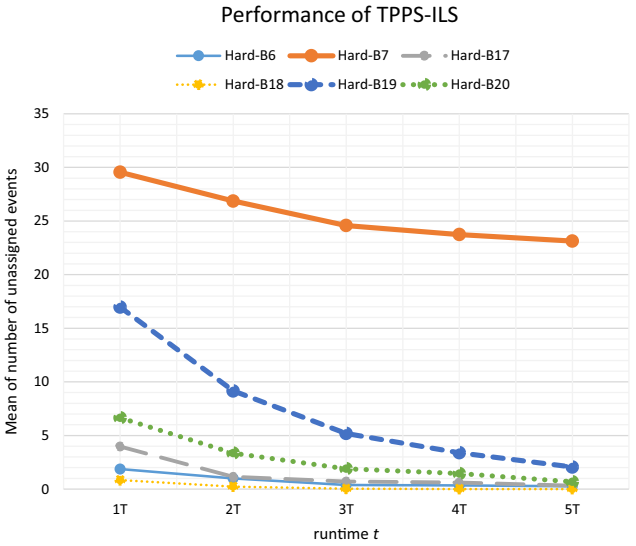

Fig. 1 Line graph showing the number of unassigned events for TSSP-ILS with extended execution time t on selected Hard cases. $N = 31$ runs

Table 12 Comparing average of soft constraint violations between SAR-1P and SAR-2P on datasets

Dataset	$C = 0.007$		$C = 0.008$		$C = 0.009$	
	SAR-1P	SAR-2P	SAR-1P	SAR-2P	SAR-1P	SAR-2P
Socha	20.72	20.79	20.71	**20.20**	20.94	20.92
ITC02	25.29	24.68	**24.17**	24.52	24.57	24.36
ITC07	126.96	122.44	125.51	**102.06**	129.07	124.87
Total	68.74	66.56	67.70	**64.21**	69.44	67.53

$N = 31$ runs for each case in the dataset

Table 13 Comparing average of soft constraint violations among SAR, SAIRL, and SAR-2P on datasets

Dataset	SAR	SAIRL	SAR-2P
Socha	20.52	21.67	**20.20**
ITC02	**24.17**	24.60	24.52
ITC07	171.05	126.38	**102.06**
Total	87.53	68.43	**64.21**

$N = 31$ runs for each case in the dataset

times. The best and average cost are remarkably improved when the execution time t is extended as evident in Table 21. Notice that we do not adjust any parameters. We just simply reset the execution time t. The p values ($0.000 < 0.05$) of t-tests reveal a statistically significant difference between the means.

Table 22 shows the comparison of soft constraint violations between SAR, SAIRL and SAR-2P with execution time t of $5T$ on the cases Socha-L, ITC02-1 and ITC07-1. t-tests are performed between (SAR-2P and SAR) and (SAR-2P and SAIRL) and the

Table 14 Comparing best(mean) of soft constraint violations among SAR, SAIRL and SAR-2P on Socha cases

Case	SAR	SAIRL	SAR-2P	*t*-test (*p* value)	
				SAR	SAIRL
S01	**0** (**0.0**)	**0** (**0.0**)	**0** (**0.0**)	–	–
S02	**0** (**0.0**)	**0** (**0.0**)	**0** (**0.0**)	–	–
S03	**0** (**0.0**)	**0** (**0.0**)	**0** (**0.0**)	–	–
S04	**0** (**0.0**)	**0** (**0.0**)	**0** (**0.0**)	–	–
S05	**0** (**0.0**)	**0** (**0.0**)	**0** (**0.0**)	–	–
M01	**0** (**1.5**)	**0** (2.3)	**0** (2.3)	0.087	0.951
M02	**0** (**2.2**)	**0** (3.6)	**0** (2.7)	0.251	0.090
M03	7 (**13.4**)	6 (14.4)	**5** (14.1)	0.599	0.802
M04	**0** (**0.7**)	**0** (1.4)	**0** (1.2)	0.212	0.679
M05	**0** (**1.2**)	**0** (1.4)	**0** (1.7)	0.230	0.514
L	165 (206.6)	181 (215.2)	**164** (**200.2**)	0.259	**0.003**

$N = 31$ runs

Table 15 Comparing best(mean) of soft constraint violations among SAR, SAIRL and SAR-2P on ITC02 cases

Case	SAR	SAIRL	SAR-2P	*t*-test (*p* value)	
				SAR	SAIRL
01	**23** (**32.6**)	26 (35.7)	29 (36.8)	**0.001**	0.441
02	7 (**13.7**)	6 (16.3)	**2** (16.2)	0.088	0.931
03	26 (36.4)	27 (38.2)	**24** (**34.3**)	0.205	**0.006**
04	50 (**63.1**)	47 (69.0)	**46** (70.7)	**0.031**	0.626
05	38 (58.6)	**36** (**51.8**)	43 (55.0)	0.081	0.139
06	**0** (0.8)	**0** (0.8)	**0** (**0.4**)	0.178	0.104
07	**0** (2.6)	**0** (**2.4**)	**0** (**2.4**)	0.562	0.940
08	**0** (**1.4**)	**0** (1.5)	**0** (2.2)	0.099	0.169
09	**0** (**4.6**)	**0** (6.4)	**0** (6.5)	**0.011**	0.938
10	28 (40.9)	22 (40.4)	**26** (**39.2**)	0.235	0.495
11	10 (**17.7**)	10 (19.0)	**9** (19.7)	0.141	0.583
12	53 (64.5)	47 (64.1)	**46** (**63.9**)	0.790	0.951
13	38 (53.3)	**33** (**51.0**)	40 (51.2)	0.258	0.935
14	5 (12.9)	4 (13.6)	**4** (**12.1**)	0.540	0.199
15	**0** (**4.0**)	**0** (4.8)	**0** (4.4)	0.574	0.487
16	**0** (**0.5**)	**0** (2.2)	**0** (1.6)	**0.006**	0.204
17	26 (41.6)	25 (36.8)	**24** (**38.7**)	0.177	0.317
18	**2** (**9.7**)	3 (12.5)	4 (11.7)	0.054	0.393
19	11 (24.7)	15 (25.6)	**9** (**23.6**)	0.502	0.199
20	**0** (**0.0**)	**0** (**0.0**)	**0** (**0.0**)	–	–

$N = 31$ runs

Table 16 Comparing best(mean) of soft constraint violations among SAR, SAIRL and SAR-2P on ITC07 cases

Case	SAR	SAIRL	SAR-2P	*t*-test (*p* value)	
				SAR	SAIRL
01	**0** (307.6)	**0** (209.4)	**0** (**191.8**)	**0.011**	0.680
02	**0** (63.4)	**0** (10.1)	**0** (**1.7**)	**0.016**	0.352
03	163 (199.4)	141 (**188.6**)	**137** (189.8)	0.105	0.820
04	242 (328.8)	192 (320.9)	**24** (**315.5**)	0.383	0.743
05	**0** (**2.7**)	**0** (2.9)	**0** (2.9)	0.890	0.959
06	**0** (**33.2**)	**0** (54.7)	**0** (37.6)	0.718	0.110
07	5 (18.0)	**4** (**14.5**)	5 (16.2)	0.803	0.767
08	**0** (**0.0**)	**0** (1.6)	**0** (5.7)	0.069	0.219
09	**0** (100.7)	**0** (15.2)	**0** (**2.6**)	**0.001**	0.393
10	**0** (65.3)	**0** (30.5)	**0** (**16.3**)	**0.040**	0.258
11	161 (244.3)	136 (201.6)	**21** (**199.6**)	**0.001**	0.870
12	**0** (318.2)	**0** (303.5)	**0** (**258.1**)	0.123	0.229
13	**0** (99.5)	**0** (90.4)	**0** (**85.9**)	0.395	0.780
14	**0** (**0.2**)	**0** (25.6)	**0** (17.8)	**0.017**	0.452
15	**0** (192.0)	**0** (12.5)	**0** (**9.3**)	**0.000**	0.785
16	10 (105.8)	**0** (45.8)	**0** (**40.2**)	**0.000**	0.642
17	**0** (0.8)	**0** (0.5)	**0** (**0.1**)	0.099	0.366
18	**0** (12.5)	**0** (**7.7**)	**0** (15.5)	0.633	0.081
19	**0** (516.7)	**0** (**11.0**)	**0** (79.6)	**0.000**	0.099
20	586 (**650.7**)	**555** (664.0)	579 (661.5)	0.329	0.847
21	**0** (**12.5**)	**0** (25.7)	**0** (14.8)	0.714	0.118
22	1 (136.0)	**0** (**5.8**)	**0** (22.6)	0.154	0.197
23	11 (**504.4**)	56 (713.6)	**0** (531.7)	0.718	**0.006**
24	5 (192.6)	**0** (**77.5**)	**0** (102.1)	**0.005**	0.347

$N = 31$ runs

p values are presented on the two rightmost columns. SAR-2P is significantly better than SAIRL (Socha-L case) and SAR (ITC07-1 case). Meanwhile, SAR is better than SAR-2P on Socha-L case.

5.4 Discussion

A synergy is achieved by hybridizing TSSP and ILS. TSSP is excellent in solution space exploration and exploitation. The best solution resulting from TSSP is further refined by ILS en route to the search end. The search is guided by the ILS to exploit the surrounding area of the best solution with the hope to obtain an optimal solution. As ILS only accepts improving or equal cost solutions, it is greedy in nature. To improve the solution space connectivity, transfer and swap operators are randomly utilized in ILS.

Table 17 Details of solvers

Solver	Methodology	References
E	Genetic Algorithm + Local Search	Abdullah et al. [1]
F	Fish Swarm	Turabieh et al. [23]
H	Honey Bee Mating	Sabar et al. [18]
I	SA	Ceschia et al. [3]
J2	SA	Kostuch [12]
N	Hybrid Algorithm	Chiarandini et al. [4]
O	SA	Cambazard et al. [2]
P	Ant Colony Optimization	Nothegger et al. [17]
Q	SA	Lewis and Thompson [14]
R	SAR	Goh et al. [9]
S	SAIRL	Goh et al. [10]
T	RPNS	Nagata [16]

SAR-2P not only eliminated the need for manual setting of neighbourhood structure composition in SAR but also performed better in minimizing soft constraint violations. The good performance is attributed to the use of average cost changes $\overline{\Delta f}$ (measured in the preliminary run to reflect the exploration level required) in the functions for setting the temperature (initial and reheated) utilized in SAR-2P. In effect, we are exploiting the information on fitness landscape in setting the temperature. Setting the right temperature (initial and reheated) is crucial to ensure that the search is supplied with the necessary exploration capability to explore the search space or perhaps escape from local optima when it is stuck.

SAR-2P is computationally faster compared to SAIRL because it does not calculate the parameter values on the run and it avoids the use of computation expensive neighbourhood structures by utilizing a good neighbourhood structure composition from the start of the search. Hence, SAR-2P achieved a higher number of transitions in a given time limit which explained the better performance. SAR-2P also has the advantage of using average cost changes $\overline{\Delta f}$ (fitness landscape information) in determining not only the reheated temperature but also the initial temperature (better search exploration).

The superiority of SAR-2P compared to SAR-1P (irrrespective of C values used), affirms our hypothesis that preliminary run used to measure the average cost changes $\overline{\Delta f}$, should mirror the actual run. For SAR-1P, neighbourhood structures are evaluated in the same preliminary run where $\overline{\Delta f}$ is measured (unlike SAR-2P where these values are obtained in two separate preliminary runs). As the dynamic neighbourhood structure composition in the preliminary run differs from the static neighbourhood structure composition in the actual run, the $\overline{\Delta f}$ measured may not reflect the actual exploration requirement of the actual run. Subsequently, inaccurate temperatures (initial and reheated) are set for the actual run and this could be the explanation for the inferior performance shown by SAR-1P.

Table 18 Comparing best(mean) of soft constraint violations for Socha cases

Case	Solver							
	E	F	H	I	R	S	T	SAR-2P
S01	**0** (**0.0**)	**0**	**0**	**0** (**0.0**)	**0** (**0.0**)	**0** (**0.0**)	–	**0** (**0.0**)
S02	**0** (**0.0**)	**0**	**0**	**0** (**0.0**)	**0** (**0.0**)	**0** (**0.0**)	–	**0** (**0.0**)
S03	**0** (**0.0**)	**0**	**0**	**0** (**0.0**)	**0** (**0.0**)	**0** (**0.0**)	–	**0** (**0.0**)
S04	**0** (**0.0**)	**0**	**0**	**0** (0.1)	**0** (**0.0**)	**0** (**0.0**)	–	**0** (**0.0**)
S05	**0** (**0.0**)	**0**	**0**	**0** (**0.0**)	**0** (**0.0**)	**0** (**0.0**)	–	**0** (**0.0**)
M01	221 (224.8)	45	75	9 (26.5)	**0** (**1.5**)	**0** (2.3)	1 (5.9)	**0** (2.3)
M02	147 (150.6)	40	88	15 (25.9)	**0** (**2.2**)	**0** (3.6)	1 (4.8)	**0** (2.7)
M03	246 (252.0)	61	129	36 (49.0)	7 (**13.4**)	6 (14.4)	**4** (14.0)	5 (14.1)
M04	165 (167.8)	35	74	12 (23.8)	**0** (**0.7**)	**0** (1.4)	0 (4.4)	**0** (1.2)
M05	130 (135.4)	49	64	3 (10.9)	**0** (**1.2**)	**0** (1.4)	4 (13.9)	**0** (1.7)
L	529 (552.4)	407	523	208 (259.8)	165 (206.6)	181 (215.2)	248 (310.2)	**164** (**200.2**)

$N = 31$ runs. Some authors only reported best results

Table 19 Comparing best(mean) of soft constraint violations for ITC02 cases

Case	Solver						
	N	J2	I	R	S	T	SAR-2P
01	45	**16** (**30.2**)	45 (57.1)	23 (32.6)	26 (35.7)	25 (35.2)	29 (36.8)
02	14	**2** (**11.4**)	20 (33.2)	7 (13.7)	6 (16.3)	7 (15.9)	**2** (16.2)
03	45	**17** (**31.0**)	43 (53.2)	26 (36.4)	27 (38.2)	27 (37.6)	24 (34.3)
04	71	**34** (**60.8**)	87 (109.9)	50 (63.1)	47 (69.0)	61 (84.5)	46 (70.7)
05	59	42 (72.1)	71 (91.7)	38 (58.6)	**36** (**51.8**)	53 (75.1)	43 (55.0)
06	1	**0** (2.4)	2 (14.1)	**0** (0.8)	**0** (0.8)	**0** (1.8)	**0** (**0.4**)
07	3	2 (8.9)	2 (13.7)	**0** (2.6)	**0** (**2.4**)	3 (11.0)	**0** (**2.4**)
08	1	**0** (2.0)	9 (20.0)	**0** (**1.4**)	**0** (1.5)	2 (10.3)	**0** (2.2)
09	8	1 (5.8)	15 (21.9)	**0** (**4.6**)	**0** (6.4)	7 (16.7)	**0** (6.5)
10	52	21 (**35.0**)	41 (60.7)	28 (40.9)	22 (40.4)	**18** (37.8)	26 (39.2)
11	30	5 (**12.9**)	24 (38.2)	10 (17.7)	10 (19.0)	**4** (15.6)	9 (19.7)
12	75	55 (76.3)	62 (83.7)	53 (64.5)	47 (64.1)	74 (95.4)	**46** (**63.9**)
13	55	**31** (**47.1**)	59 (78.0)	38 (53.3)	33 (51.0)	52 (66.3)	40 (51.2)
14	18	11 (22.3)	21 (34.2)	5 (12.9)	**4** (13.6)	20 (29.8)	**4** (**12.1**)
15	8	2 (8.4)	6 (11.8)	**0** (**4.0**)	0 (4.8)	1 (7.2)	**0** (4.4)
16	55	**0** (3.4)	6 (16.7)	**0** (**0.5**)	0 (2.2)	1 (4.8)	**0** (1.6)
17	46	37 (54.0)	42 (56.5)	26 (41.6)	25 (**36.8**)	38 (55.6)	**24** (38.7)
18	24	4 (**9.4**)	11 (25.9)	**2** (9.7)	3 (12.5)	6 (13.9)	4 (11.7)
19	33	**7** (**16.4**)	56 (73.0)	11 (24.7)	15 (25.6)	16 (29.6)	9 (23.6)
20	**0**	**0** (0.5)	**0** (1.8)	**0** (**0.0**)	**0** (**0.0**)	**0** (**0.0**)	**0** (**0.0**)

$N = 31$ runs. Some authors only reported best results

6 Conclusion

We have compared the performance of TSSP-ILS with TSSP and ILS in constructing feasible solutions for the hardest standard timetabling problem. The TSSP-ILS method has a better performance than either TSSP or ILS alone as well as other recent methods. Moreover, we observed a remarkable result improvement when the execution time is extended, indicating the extendability of TSSP-ILS. Furthermore, TSSP-ILS does not require parameter tuning, making it one of the best algorithms in constructing feasible solutions for the PE-CTT problems.

A SAR algorithm with 2 preliminary runs (SAR-2P) is presented. We tested different values for the coefficient C for both SAR-1P and SAR-2P. The performance of SAR-2P, SAR, SAIRL and other recent methods are compared in terms of soft constraint violations. SAR-2P is comparable to SAR, SAIRL and other state of the art methods in the scientific literature. SAR-2P is also shown to be scalable in runs with extended execution time.

Table 20 Comparing best(mean) of soft constraint violations for ITC07 cases

Case	Solver						
	P	I	Q	R	S	T	SAR-2P
01	**0** (613.0)	59 (399.2)	**0** (377.0)	**0** (307.6)	0 (209.4)	**0** (**81.7**)	**0** (91.8)
02	**0** (556.0)	**0** (142.2)	**0** (382.2)	**0** (63.4)	0 (10.1)	**0** (48.0)	**0** (**1.7**)
03	110 (680.0)	148 (209.9)	122 (181.8)	163 (199.4)	141 (188.6)	**55** (**155.0**)	137 (189.8)
04	53 (580.0)	25 (349.6)	18 (319.4)	242 (328.8)	192 (320.9)	**10** (**254.1**)	24 (315.5)
05	13 (92.0)	**0** (7.7)	**0** (7.5)	**0** (2.7)	**0** (2.9)	**0** (**0.0**)	**0** (2.9)
06	**0** (212.0)	**0** (**8.6**)	**0** (22.8)	**0** (33.2)	**0** (54.7)	**0** (**0.0**)	**0** (37.6)
07	**0** (4.0)	**0** (4.9)	**0** (5.5)	5 (18.0)	4 (14.5)	**0** (**3.6**)	5 (16.2)
08	**0** (61.0)	**0** (1.5)	**0** (0.6)	**0** (**0.0**)	**0** (1.6)	**0** (**0.0**)	**0** (5.7)
09	**0** (202.0)	**0** (258.8)	**0** (514.4)	**0** (100.7)	**0** (15.2)	**0** (58.9)	**0** (**2.6**)
10	**0** (**4.0**)	3 (186.4)	**0** (1202.4)	**0** (65.3)	**0** (30.5)	**0** (6.4)	**0** (16.3)
11	143 (774.0)	142 (269.5)	48 (202.6)	161 (244.3)	136 (201.6)	**3** (**140.4**)	21 (199.6)
12	**0** (538.0)	267 (400.0)	**0** (340.2)	**0** (318.2)	**0** (303.5)	**0** (**33.1**)	0 (258.1)
13	5 (360.0)	1 (120.0)	**0** (79.0)	**0** (99.5)	**0** (90.4)	**0** (**0.0**)	**0** (85.9)
14	**0** (41.0)	**0** (3.6)	**0** (0.5)	**0** (0.2)	**0** (25.6)	**0** (**0.0**)	**0** (17.8)
15	**0** (29.0)	**0** (48.0)	**0** (139.9)	**0** (192.0)	**0** (12.5)	**0** (**0.0**)	**0** (9.3)
16	**0** (101.0)	**0** (50.1)	**0** (105.2)	10 (105.8)	**0** (45.8)	**0** (**1.5**)	**0** (40.2)
17	–	**0** (**0.0**)	**0** (0.1)	**0** (0.8)	**0** (0.5)	**0** (0.2)	**0** (**0.0**)
18	–	**0** (41.1)	**0** (2.2)	**0** (12.5)	**0** (7.7)	**0** (**0.5**)	**0** (15.5)
19	–	**0** (951.5)	**0** (346.1)	**0** (516.7)	**0** (**11.0**)	**0** (616.8)	**0** (79.6)
20	–	543 (700.2)	557 (724.5)	586 (650.7)	555 (664)	**438** (**482.0**)	579 (661.5)
21	–	5 (35.9)	1 (32.1)	**0** (12.5)	**0** (25.7)	**0** (**0.1**)	**0** (14.8)
22	–	5 (19.9)	4 (1790.1)	1 (136.0)	**0** (**5.8**)	**0** (35.0)	**0** (22.6)
23	–	1292 (1707.7)	**0** (514.1)	11 (**504.4**)	56 (713.6)	777 (1083.5)	**0** (531.7)
24	–	**0** (105.3)	18 (328.2)	5 (192.6)	**0** (77.5)	**0** (**1.0**)	**0** (102.1)

$N = 31$ runs

Table 21 Comparing best(mean) of soft constraint violations for SAR-2P with execution time $t = T$ and $t = 5T$ on selected cases

Case	$t = T$	$t = 5T$	t-test (p value)
Socha-L	164 (200.16)	136 (172.74)	**0.000**
ITC02-01	29 (36.84)	12 (19.87)	**0.000**
ITC07-01	0 (191.84)	0 (25.68)	**0.000**
Total	(142.95)	(72.76)	

$N = 31$ runs

Table 22 Comparing best(mean) of soft constraint violations among SAR, SAIRL and SAR-2P with execution time $t = 5T$ on selected cases

Case	SAR	SAIRL	SAR-2P	t-test (p value)	
				SAR	SAIRL
Socha-L	103 (**139.39**)	157 (190.42)	136 (172.74)	**0.000**	**0.000**
ITC02-01	10 (21.03)	11 (20.84)	12 (**19.87**)	0.357	0.413
ITC07-01	0 (134.94)	0 (**23.06**)	0 (25.68)	**0.000**	0.861

$N = 31$ runs

7 Future work

It will be interesting to test the robustness and general applicability of the SAR-2P algorithm proposed in this paper, on other educational timetabling problems (school timetabling and examination timetabling), other scheduling problems (transport scheduling, sports scheduling and nurse rostering) and possibly other combinatorial optimisation problems (bin packing, vehicle routing etc.).

As the proposed method is still purely based on SA, we are open to the idea of hybridizing it with methods that exploit on the usage of memory (tabu mechanisms and ant systems) or even evolutionary methodologies such as genetic algorithms which are well known for their explorative capability.

Acknowledgements Funding was provided by Universiti Malaysia Sabah (Grant No. SLB0170-2018).

References

1. Abdullah, S., Burke, E.K., McCollum, B.: A hybrid evolutionary approach to the university course timetabling problem. In: IEEE Congress on Evolutionary Computation, 2007. CEC 2007, pp. 1764–1768. IEEE (2007)
2. Cambazard, H., Hebrard, E., O'Sullivan, B., Papadopoulos, A.: Local search and constraint programming for the post enrolment-based course timetabling problem. Ann. Oper. Res. **194**(1), 111–135 (2012)
3. Ceschia, S., Di Gaspero, L., Schaerf, A.: Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. Comput. Oper. Res. **39**(7), 1615–1624 (2012)
4. Chiarandini, M., Birattari, M., Socha, K., Rossi-Doria, O.: An effective hybrid algorithm for university course timetabling. J. Sched. **9**(5), 403–432 (2006)
5. Cooper, T.B., Kingston, J.H.: The Complexity of Timetable Construction Problems. Springer, Berlin (1996)

6. de Werra, D.: An introduction to timetabling. Eur. J. Oper. Res. **19**(2), 151–162 (1985)
7. de Werra, D., Asratian, A.S., Durand, S.: Complexity of some special types of timetabling problems. J. Sched. **5**(2), 171–183 (2002)
8. Even, S., Itai, A., Shamir, A.: On the complexity of time table and multi-commodity flow problems. In: 16th Annual Symposium on Foundations of Computer Science (SFCS 1975), pp. 184–193 (1975)
9. Goh, S.L., Kendall, G., Sabar, N.R.: Improved local search approaches to solve post enrolment course timetabling problem. Eur. J. Oper. Res. **261**(1), 17–29 (2017)
10. Goh, S.L., Kendall, G., Sabar, N.R.: Simulated annealing with improved reheating and learning for the post enrolment course timetabling problem. J. Oper. Res. Soc. **70**, 1–16 (2018)
11. Kostuch, P.: Timetabling competition-SA-based heuristic. In: International Timetabling Competition (2003)
12. Kostuch, P.: The university course timetabling problem with a three-phase approach. In: Burke, E., Trick, M. (eds.) Practice and Theory of Automated Timetabling V, pp. 109–125. Springer, Berlin (2005)
13. Lewis, R., Paechter, B.: Finding feasible timetables using group-based operators. IEEE Trans. Evolut. Comput. **11**(3), 397–413 (2007)
14. Lewis, R., Thompson, J.: Analysing the effects of solution space connectivity with an effective metaheuristic for the course timetabling problem. Eur. J. Oper. Res. **240**(3), 637–648 (2015)
15. Liu, Y., Zhang, D., Chin, F.Y.: A clique-based algorithm for constructing feasible timetables. Optim. Methods Softw. **26**(2), 281–294 (2011)
16. Nagata, Y.: Random partial neighborhood search for the post-enrollment course timetabling problem. Comput. Oper. Res. **90**, 84–96 (2018)
17. Nothegger, C., Mayer, A., Chwatal, A., Raidl, G.R.: Solving the post enrolment course timetabling problem by ant colony optimization. Ann. Oper. Res. **194**(1), 325–339 (2012)
18. Sabar, N.R., Ayob, M., Kendall, G., Qu, R.: A honey-bee mating optimization algorithm for educational timetabling problems. Eur. J. Oper. Res. **216**(3), 533–543 (2012)
19. Socha, K., Knowles, J., Sampels, M.: A MAX–MIN ant system for the university course timetabling problem. In: Dorigo, M., Di Caro, G., Sampels, M. (eds.) Ant Algorithms, pp. 1–13. Springer, Berlin (2002)
20. Song, T., Liu, S., Tang, X., Peng, X., Chen, M.: An iterated local search algorithm for the university course timetabling problem. Appl. Soft Comput. **68**, 597–608 (2018)
21. Thompson, J.M., Dowsland, K.A.: Variants of simulated annealing for the examination timetabling problem. Ann. Oper. Res. **63**(1), 105–128 (1996)
22. Tuga, M., Berretta, R., Mendes, A.: A hybrid simulated annealing with Kempe chain neighborhood for the university timetabling problem. In: 6th IEEE/ACIS International Conference on Computer and Information Science, 2007. ICIS 2007, pp. 400–405. IEEE (2007)
23. Turabieh, H., Abdullah, S., McCollum, B., McMullan, P.: Fish swarm intelligent algorithm for the course timetabling problem. In: Yu, J., Greco, S., Lingras, P., Wang, G., Skowron, A. (eds.) Rough Set and Knowledge Technology, pp. 588–595. Springer, Berlin (2010)

Affiliations

Say Leng Goh[1] · Graham Kendall[1,2,3] · Nasser R. Sabar[1,4] · Salwani Abdullah[5]

Graham Kendall
Graham.Kendall@nottingham.edu.my

Nasser R. Sabar
nasser.sabar@gmail.com

Salwani Abdullah
salwani@ukm.edu.my

[1] Optimization Research Group (OPT), Faculty of Computing and Informatics, Universiti Malaysia Sabah Labuan International Campus, Jalan Sungai Pagar, 87000 Labuan F.T., Malaysia

[2] The University of Nottingham Malaysia Campus, Jalan Broga, 43500 Semenyih, Selangor Darul Ehsan, Malaysia

[3] University of Nottingham, University Park, Nottingham NG7 2RD, UK

[4] Department of Computer Science and Information Technology, La Trobe University, Melbourne, VIC 3086, Australia

[5] Universiti Kebangsaan Malaysia, 43600 UKM Bangi, Selangor, Malaysia