Noname manuscript No.
(will be inserted by the editor)

SAILS: Hybrid Algorithm for the Team Orienteering Problem with Time Windows

Aldy Gunawan · Hoong Chuin Lau · Kun Lu

Received: date / Accepted: date

Abstract The Team Orienteering Problem with Time Windows (TOPTW) is the extended version of the Orienteering Problem where each node is limited by a given time window. The objective is to maximize the total collected score from a certain number of paths. In this paper, a hybridization of Simulated Annealing and Iterated Local Search, namely SAILS, is proposed to solve the TOPTW. The efficacy of the proposed algorithm is tested using benchmark instances. The results show that the proposed algorithm is competitive with the state-of-the-art algorithms in the literature. SAILS is able to improve the best known solutions for 19 benchmark instances.

Keywords Orienteering Problem · Time Windows · Hybrid Algorithm · Simulated Annealing · Iterated Local Search

1 Introduction

The Team Orienteering Problem with Time Windows (TOPTW) is an extension of the Orienteering Problem (OP) [11]. A certain number of paths are required to serve a set of nodes. The visit on each node is limited by a given time window. The score of a particular node will be received once a node is visited within its time window. The main objective of the TOPTW is to maximize the total score from all visited nodes.

Since the OP has been proven as a NP-hard problem [5], it is unlikely that the TOPTW can be solved optimally within polynomial time. It is therefore interesting to propose fast heuristics to solve the problem, especially when we are dealing with real life large-scale applications of TOPTW, e.g. a personalized city trip planner [3, 21].

A. Gunawan, H.C. Lau and K. Lu
School of Information Systems, Singapore Management University
Tel.: +65-68085227
Fax: +65-68280901
E-mail: {aldygunawan, hclau, kunlu}@smu.edu.sg

In this paper, we introduce a hybrid algorithm that combines two well-known metaheuristics, Iterated Local Search (ILS) and Simulated Annealing (SA). Iterated Local Search [15] is a simple but effective metaheuristic. In general, since it accepts only improving solutions or moves, we consider the incorporation of Simulated Annealing to avoid early termination in local optimality. Simulated Annealing [9] has been successfully applied to several combinatorial optimization problems [12–14]. It has the capability to escape from a local optimum by accepting a worse solution with a probability that changes over time. Our proposed algorithm is competitive with the state-of-the-art algorithms. More precisely, we show that it is able to improve the best known solution values of 19 benchmark instances. Hence, our work also serves as benchmark for future studies.

The paper is organized as follows. In Section 2, the TOPTW is briefly explained, including most recent works related to the TOPTW. Section 3 describes the proposed algorithm, SAILS, in detail. Section 4 is devoted to the experimental results and analysis. Finally, conclusions and ideas for future works are summarized in Section 5 .

2 The Team Orienteering Problem with Time Windows

2.1 Problem Description

The TOPTW is defined as follows. We are given an undirected network graph $G = (N,A)$ where $N = \{0,1,2,\ldots,|N|\}$ is the set of nodes, $A = \{(i,j) : i \neq j \in N\}$ refers to the set of arcs connecting two different nodes i and j and $M = \{1,2,\ldots,|M|\}$ is the set of paths. The non-negative travel time between nodes i and j is represented as t_{ij}. Each node $i \in N$ has a positive score u_i that would be collected the first time the node i is visited, a service time S_i and a time window $[e_i,l_i]$. e_i and l_i refer to the earliest and latest times allowed for starting the visit at node i.

In the TOPTW, it is assumed that node 0 is the start and end nodes, therefore $u_0 = S_0 = 0$. The visit to node i is successful if it begins within a time window $[e_i,l_i]$. Each node can only be visited at most once. The visit is allowed to wait until the time window begins in the case of an earlier arrival. In the context of TOPTW, the number of paths is fixed at $|M|$. Each path $m \in M$ is constrained within the time limit $[e_0,l_0]$. We have $e_0 = 0$ and $l_0 = T^{max}$, where T^{max} is the time budget or the maximum duration of the tour. The main objective is to maximize the total collected score of the visited nodes from $|M|$ paths. The mathematical formulation of the TOPTW can be found in [21].

2.2 Literature Review

Vansteenwegen et al. [21] introduced an Iterated Local Search (ILS) algorithm to solve the TOPTW with emphasis on providing a simple, fast and effective algorithm that can be tailored for a realistic Tourist Trip Design Problem (TTDP). Only two operations of ILS, INSERT and SHAKE, are considered in this deterministic algorithm. A metaheuristic algorithm based on Ant Colony System (ACS) was proposed by Montemanni and Gambardella [16]. The algorithm was further improved by Montemanni et al. [17], namely the Enhanced ACS (EACS) algorithm. The EACS algorithm includes two additional operations to overcome the drawbacks of ACS. Both operations are related to the consideration of using the best solution found so far during the construction phase and applying the local search procedure only on those solutions on which the local search has not been recently applied.

In addition, Lin and Yu [13] proposed two different versions of Simulated Annealing, Fast SA (FSA) and Slow SA (SSA), in order to tailor two different scenarios. FSA is mainly for the applications that need quick responses, while SSA is more concerned about the quality of the solutions at the expense of more computational time. Labadie et al. [11] introduced an LP-based Granular Variable Neighborhood Search (GVNS) for solving the TOPTW.

Another ILS algorithm was proposed by Gunawan et al. [6] for solving the OPTW. The problem is also considered as the TOPTW with $|M| = 1$. The algorithm is started by generating an initial feasible solution using a greedy construction heuristic. The initial solution obtained is further improved by ILS. ILS is mainly based on several local search components, such as SWAP, 2-OPT, INSERT and REPLACE. The combination between ACCEPTANCECRITERION and PERTURBATION mechanisms is implemented to control the balance between diversification and intensification of the search. Computational results show that ILS is able to improve 8 best known solutions values of benchmark instances.

The idea of combining some advantages has been brought up by many researchers for solving different combinatorial optimization problems. Several taxonomies related to the hybrid algorithm were introduced by Talbi et al. [20] and Puchinger and Raidl [18]. Labadie et al. [10] introduced a hybridization of a Greedy Randomized Adaptive Search Procedure (GRASP) and an Evolutionary Local Search algorithm (ELS) for the TOPTW. Different constructive heuristics based on GRASP are proposed in order to build the initial solutions. Those initial solutions are further improved by the ELS algorithm. Another hybrid algorithm which is based a local search (LS) procedure, Simulated Annealing (SA) and Route Combination (RR) component is proposed by Hu and Lim [7]. Three components are iteratively incorporated within a certain number of iterations. It is shown that 35 new best solutions are found and more than 83% of instances with optimal solutions can be obtained.

Most recently, Cura [1] proposed an Artificial Bee Colony (ABC) algorithm to solve the TOPTW. Hybridization of SA and a new scout bee search behavior based on a local search procedure is introduced to improve the solution quality of benchmark instances. The proposed method is able to produce high-quality TOPTW solutions and comparable to other approaches. There is no new best found solution reported.

3 Hybrid Algorithm

In this section, we describe the proposed algorithm that combines Simulated Annealing (SA) and Iterated Local Search (ILS), namely SAILS. Instead of starting with a randomly generated initial solution which is commonly used in SA, we introduce a greedy construction heuristic for providing an initial solution. The initial solution is further improved by SAILS. By using SA, a new solution with a worse objective

Algorithm 1 CONSTRUCTION (N, M)

```
$N^* \leftarrow$node 0
$N' \leftarrow N \backslash$node 0
Initialize $S_0 \leftarrow N^*$
$F \leftarrow$ UPDATEF$(N', M)$
while $F \neq \emptyset$ do
    $\langle n^*, p^*, m^* \rangle \leftarrow$ SELECT$(F)$
    $S_0 \leftarrow \langle n^*, p^*, m^* \rangle$
    Update $P(m)$
    $N' \leftarrow N' \setminus \{n^*\}$
    $N^* \leftarrow N^* \cup \{n^*\}$
    $F \leftarrow$ UPDATEF$(N', M)$
end while
return $S_0$
```

Algorithm 2 UPDATEF (N', M)

```
$F \leftarrow \emptyset$
for all $n \in N'$ do
    for all $m \in M$ do
        for all $p \in P(m)$ do
            if insert node $n$ in position $p$ of path $m$ is feasible then
                calculate $ratio_{n,p,m}$
                $F \leftarrow F \cup \langle n, p, m \rangle$
            end if
        end for
    end for
end for
Sort all elements of $F$ in descending order based on $ratio_{n,p,m}$
Select the best $f$ elements of $F$ and remove the rest
return $F$
```

function value may be accepted with a certain probability. The possible neighborhoods are generated by implementing ILS. The details of the SAILS algorithm are described in the following sub-sections.

3.1 Greedy Construction Heuristic

The greedy construction heuristic is outlined in Algorithm 1. The idea of generating an initial solution is adopted from the one proposed by Gunawan et al. [6]. The earlier version is only dedicated for $|M| = 1$. Here, the heuristic is extended for $|M| > 1$. N' and N^* denote the sets of unscheduled and scheduled nodes, respectively ($N' \cup N^* = N$). N^* is initialized by the start and end nodes, node 0, while N' consists of all unscheduled nodes. S_0 refers the current feasible solution obtained so far, represented as m-row vectors. Each row is initialized with start and end nodes, node 0.

The construction heuristic is started by generating a set of all feasible candidate nodes to be inserted, F. Each element of F, which represents a feasible insertion of node n in position p of path m, is represented as $\langle n, p, m \rangle$. All possibilities of inserting an unscheduled node in position p of path m are examined. A insertion $\langle n, p, m \rangle$ is

Algorithm 3 SELECT (F)

```
$SumRatio \leftarrow 0$
for all $\langle n,p,m\rangle \in F$ do
    $SumRatio \leftarrow SumRatio + ratio_{n,p,m}$
end for
for all $\langle n,p,m\rangle \in F$ do
    $prob_{n,p,m} \leftarrow ratio_{n,p,m}/SumRatio$
end for
$U \leftarrow rand(0,1)$
$AccumProb \leftarrow 0$
for all $\langle n,p,m\rangle \in F$ do
    $AccumProb \leftarrow AccumProb + prob_{n,p,m}$
    if $U \leq AccumProb$ then
        $\langle n^*,p^*,m^*\rangle \leftarrow \langle n,p,m\rangle$
        break
    end if
end for
return $\langle n^*,p^*,m^*\rangle$
```

feasible if after the insertion, all scheduled nodes do not violate their respective time windows and the total spent time of path m does not exceed T^{max}.

Let $P(m)$ be a set of positions of scheduled nodes on path m. For each possible insertion, the benefit of insertion $ratio_{n,p,m}$ is calculated by equation 1. $Diff_{n,p,m}$ represents the difference between the total time spent before and after the insertion of node n in position p of path m. All elements of F are then sorted in descending order based on $ratio_{n,p,m}$ values. Only a subset of elements, f, would be kept. Algorithm 2 summarizes the algorithm of generating F.

$$ratio_{n,p,m} = \left(\frac{u_n^2}{Diff_{n,p,m}}\right) \tag{1}$$

If F is not an empty set, Algorithm 3 is run in order to select which $\langle n^*, p^*, m^*\rangle$ to be inserted. Each $\langle n, p, m\rangle$ corresponds to a particular probability value, $prob_{n,p,m}$. The probability is calculated by Equation 2:

$$prob_{n,p,m} = \left(\frac{ratio_{n,p,m}}{\sum_{\langle i,j,k\rangle \in F} ratio_{i,j,k}}\right) \tag{2}$$

The selection of $\langle n^*, p^*, m^*\rangle$ from F is based on Roulette-Wheel selection concept [4]. This method assumes that the probability of selection is proportional to the benefit of insertion of an individual, $ratio_{n,p,m}$. The accumulative of probability values, *AccumProb*, is initially set to 0. A random number $U \sim rand[0,1]$ is generated. We then select a particular $\langle n^*, p^*, m^*\rangle$ and update the value of *AccumProb* iteratively. This loop will be terminated when $(U \leq AccumProb)$ and the corresponding $\langle n^*, p^*, m^*\rangle$ is then selected. S_0, N' and N^* will also be updated. The greedy construction heuristic is terminated when $F = \emptyset$.

3.2 SAILS

Given the initial solution generated from the greedy construction heuristic, we propose a hybridization between Simulated Annealing (SA) and Iterated Local Search (ILS) to further improve the quality of the initial solution. The outline of SAILS is presented in Algorithm 4. The SA algorithm requires three parameters T_0, α and INNERLOOP. T_0 refer to the initial temperature. α is a coefficient used to control the speed of the cooling schedule. INNERLOOP denotes the number of iterations at a particular temperature.

Let S_0, S^* and S' be the current solution, the best found solution so far and the starting solution for each iteration, respectively. At the beginning, the current temperature $Temp$ is equal to T_0 and will be decreased after INNERLOOP iterations by using the following formula: $Temp = Temp \times \alpha$ $(0 < \alpha < 1)$.

At a particular value of temperature, we apply two components of ILS: PERTURBATION and LOCALSEARCH in order to explore neighborhoods of S_0. For each iteration, we calculate the difference between two solutions S_0 and S', denoted as δ. If δ is greater than 0, which implies that the improvement of the objective function does exist, S' is replaced by S_0. If S_0 also improves S^*, S^* is then replaced by S_0. On the other hand, if the solution generated is worse, a random number between 0 and 1, r, is generated and compared with $\exp(\delta/Temp)$. If this worse solution is accepted ($r < \exp(\delta/Temp)$), we update S'; otherwise, we return to S'. For each iteration, if there is no improvement of S^*, we increase the number of no improvement NOIMPR by one. In [21], the solution will only be accepted if it is better than the best found, otherwise the number of non-improvement iteration will be increased by one.

The main difference of the standard SA and our SAILS lies in the additional strategy applied. We include the intensification strategy. The idea of this strategy is as follows. If there is no improvement of the solution obtained after a certain number of iterations LIMIT, we focus the search once again starting from the best solution obtained S^*. Finally, the entire algorithm will be run within the computational budget TIMELIMIT.

The neighborhoods of the current solution is generated by ILS. Two components of ILS are considered: PERTURBATION and LOCALSEARCH. Two different steps implemented in PERTURBATION are: EXCHANGEPATH and SHAKE. If the number of iterations without improvement, NOIMPR, is larger than THRESHOLD1 and (NOIMPR + 1) Mod THRESHOLD2 = 0, EXCHANGEPATH would be executed; otherwise, SHAKE would be selected. THRESHOLD1 and THRESHOLD2 are two pre-set parameters. In EXCHANGEPATH step, all nodes from two different paths are selected and swapped. The strategy of selecting two different paths are based on generating of permutations by adjacent transposition method [8]. EXCHANGEPATH will only be implemented if the number of paths is more than one. Otherwise, we implement SHAKE.

The SHAKE step is adopted from [21]. One or more nodes will be removed from each path m, which depends on two integer values, CONS and POST. CONS indicates how many consecutive nodes to remove for a particular path while POST indicates the first position of the removing process on a particular path. If we reach the last scheduled node, the process will then be back to the first node after the start node,

Algorithm 4 SAILS (N,M)

```
$S_0 \leftarrow$ CONSTRUCTION$(N,M)$
$S^* \leftarrow S_0$
$S' \leftarrow S_0$
$Temp \leftarrow T_0$
NOIMPR $\leftarrow 0$
while TIMELIMIT has not been reached do
    INNERLOOP = 0
    WHILE INNERLOOP < MAXINNERLOOP DO
        $S_0 \leftarrow$ PERTURBATION$(S_0,N^*,N',M)$
        $S_0 \leftarrow$ LOCALSEARCH$(S_0,N^*,N',M)$
        $\delta \leftarrow S_0 - S'$
        IF $\delta > 0$ THEN
            $S' \leftarrow S_0$
            IF $S_0$ IS BETTER THAN $S^*$ THEN
                $S^* \leftarrow S_0$
                NOIMPR $\leftarrow 0$
            ELSE
                NOIMPR $\leftarrow$ NOIMPR + 1
            END IF
        ELSE
            $r \leftarrow rand[0,1]$
            IF $r < \exp(\delta/Temp)$ THEN
                $S' \leftarrow S_0$
            ELSE
                $S_0 \leftarrow S'$
            END IF
            NOIMPR $\leftarrow$ NOIMPR + 1
        END IF
        INNERLOOP $\leftarrow$ INNERLOOP + 1
    END WHILE
    $Temp \leftarrow Temp \times \alpha$
    IF NOIMPR > LIMIT THEN
        $S_0 \leftarrow S^*$
        $S' \leftarrow S_0$
        NOIMPR $\leftarrow 0$
    END IF
end while
return $S^*$
```

node 0. Both CONS and POST are initially set to 1. After each SHAKE step, POST is increased by CONS. CONS would also be increased by 1 after a fixed number of consecutive iterations, e.g. 2 iterations.

If POST is greater than the size of the smallest path, POST is subtracted with the size of the smallest path to determine the new position POST. If CONS is greater than the size of the largest path, or S^* is updated, CONS is reset to one. Take note that CONS is always increased by 1 for each iteration and would be set to 1 if it equals to $\frac{n}{3\times|M|}$ in [21]. After removing CONS nodes, we update N' and N^* accordingly. F is then regenerated based on Algorithm 2 and an unscheduled node that needs to be inserted is selected using Algorithm 3. This is repeated until $F = \emptyset$.

Table 1 presents six operations in LOCALSEARCH that are run consecutively and applied to S_0. When m = 1, only SWAP1, 2-OPT, INSERT and REPLACE are con-

Table 1: LOCAL SEARCH operations.

Operations	Descriptions
SWAP1	Exchange two nodes within one path
SWAP2	Exchange two nodes within two paths
2-OPT	Reorder the sequence of certain nodes within one path
MOVE	Move one node from one path to another path
INSERT	Insert nodes into a path
REPLACE	Replace one scheduled node with one unscheduled node

sidered. SWAP1 is applied by exchanging two scheduled nodes within one particular path with the lowest remaining travel time. We examine all possible combinations of selecting two different nodes. SWAP1 is executed if it is able to increase the remaining travel time of selected path and there is no constraint violation. The idea of SWAP1 is extended to two different paths with the lowest and the second lowest remaining travel times, namely SWAP2. This operation will be accepted if the total remaining travel times from both paths is increased. Both SWAP1 and SWAP2 would be terminated when there is no further improvement in terms of the remaining travel times.

2-OPT is started by selecting one path with the lowest remaining travel time. All possible combinations of selecting two different nodes are enumerated and the sequence of scheduled nodes is reversed as long as there is no constraint violation. It has to increase the remaining travel time of the selected path. This would be terminated until no further improvement in terms of the total of remaining travel time of the selected path.

MOVE is performed by reallocating one node from one path to another path. It is started from the first scheduled node n^* from first path m^*. We try to insert node n^* in another path. First, F is generated by using Algorithm 2 where $N' = \{n^*\}$ and $M = M \setminus \{m^*\}$. If $F \neq \emptyset$, node n^* would be reallocated using Algorithm 3. Otherwise, the process will continue to the next scheduled node. This operation would be terminated if node n^* is moved successfully or the last scheduled node of the last path $|M|$ is reached.

The purpose of INSERT is to insert one unscheduled node to a particular path. It is started by generating F based on Algorithm 2 and selecting node $i \in N'$ to be inserted by using Algorithm 3. After the insertion, S_0, N', N^* and F are updated accordingly. This is repeated until $F = \emptyset$. In the last operation REPLACE, one scheduled node $i \in N^*$ is replaced with one unscheduled node $j \in N'$. The operation is started by selecting path m with the highest remaining travel time, followed by selecting one node $j \in N'$ with the highest score u_j. We then check each position p of the selected path and examine whether selected node j can replace the node in position p. Once this operation is successful, the process will continue to the next unscheduled node j and repeat the operation. Otherwise, the operation would be terminated.

Table 2: Benchmark Instances

References	Names	Instance Sets	$\lvert N\rvert$	$\lvert M\rvert$
[19]	Solomon	c100, r100, rc100	100	1 to 4
	Cordeau	pr01 - pr10	[48, 288]	
[16]	Solomon	c200, r200, rc200	100	1 to 4
	Cordeau	pr11 - pr20	[48, 288]	
[21]	Solomon	c100, r100, rc100	100	
		c200, r200, rc200	100	up to number of vehicles
	Cordeau	pr01 - pr10	[48, 288]	

Table 3: Estimation of single-thread performance

Algorithm	Experimental environment	Estimate of single-thread performance
IterILS	Intel Core 2 with 2.5 GHz processor	0.92
ACS	Dual AMD Opteron 250 2.4 gigahertz CPU, 4 gigabytes RAM	0.39
SSA	Intel Core 2 CPU, 2.5 gigahertz	0.92
GVNS	Intel Pentium (R) IV, 3 gigahertz CPU	0.39
I3CH	Intel Xeon E5430 CPU clocked at 2.66 gigahertz, 8 gigabytes RAM	1.16
SAILS	Intel(R) Core(TM) i5 CPU with 3.2 GHz processor, 12 GB RAM	1

4 Computational Results

4.1 Benchmark Instances and Approach Comparison

The benchmark instances are categorized into three groups, as listed in Table 2. All benchmark instances can be accessed at `http://www.mech.kuleuven.be/en/cib/op`. The first two groups are considered as ”INST-M” which contain four instance sets: ”Solomon 100”, ”Solomon 200”, ”Cordeau 1-10” and ”Cordeau 11-20”. The last group is known as ”OPT”. The optimal solution for each instance in this group is known as the total score of all nodes on the network graph [7].

The performances of SAILS are compared against the state-of-the-art algorithms: Iterated Local Search (IterILS) [21], Ant Colony System (ACS) [16,17], Slow Simulated Annealing (SSA) [13], Granular Variable Neighborhood Search (GVNS) [11] and Iterative Three-Component Heuristic (I3CH) [7]. In order to ensure the fairness among algorithms, we also follow the same approach by using the *SuperPi* benchmark [7] to adjust the computational time to the speed of the computers used in other solutions. The main idea is to set the performance of our machine to be 1 and estimate the single-thread performance of other processors by multiplying with the single-thread performance estimation, as shown in Table 3.

We propose two different scenarios for running SAILS. In the first scenario, we refer to the computational time used by ACS since we are more concerned about the quality of the solution rather than the solution time. Only ACS uses the computational budget, while the rest use the number of iterations. Our experiments use 35% of ACS’s computational budget (= 3600 seconds). Therefore, the computational budget for each instance is set to 35% $\times$ 0.39 $\times$ 3600 seconds $\approx$ 492 seconds using our

Table 4: New best known solution values found by SAILS (first scenario)

Instance	*m*	Old *BK*	New *BK*	Instance	*m*	Old *BK*	New *BK*
r206	1	1029	1032	pr18	2	938	946
r208	1	1112	1115	r104	3	777	778
rc206	1	895	899[‡]	rc104	3	834	835
r107	2	536	538	pr02	3	942	943
pr04	2	925	926	r104	4	972	973
pr09	2	905	909	rc103	4	974	975
c204	2	1480	1490	rc107	4	980	985
pr13	2	832	843				

[‡] Same result with that of ILS [6]

processor (refer to Table 3). In the second scenario, we conduct experiments in which SAILS is set to the same computational time of I3CH. It has been proven that I3CH outperforms other algorithms, such as IterILS, SSA and GVNS [7].

For SAILS, each instance is executed in 10 runs with different random seeds. ACS was executed in 5 runs whereas GVNS was also executed 10 runs. IterILS, SSA and I3CH were only executed once and reported one solution for each instance. Some parameter settings adopted from [6] are as follows: f = 5, THRESHOLD1 = 20 and THRESHOLD2 = 3. Other SA parameters have been selected according to preliminary experiments using a subset of instances. The values of parameters considered are as follows: $\alpha \in \{0.5, 0.75, 0.9\}$, $Temp \in \{500, 1000, 1500, 2000\}$ and MAXINNERLOOP $\in \{50, 100\}$. Only one parameter is set to a constant value, using the formula: LIMIT = 0.05 $\times$ MAXINNERLOOP. All possible combinations were run in order to obtain the final parameter values: α = 0.75, T_0 = 1000 and MAXINNERLOOP = 50.

4.2 Computational Results

We report a comprehensive analysis of the results obtained by SAILS. Table 4 presents 15 new best known solutions (*BK*s) obtained by SAILS, 40% of them are from instances with $m = 2$ while each of other m values has 20% of new *BK*s. Around 33% of new *BK*s are from Cordeau et al.'s datasets which is harder to solve compared against Solomon's datasets [2]. We only report the results of Cordeau et al.'s datasets for m = 1 to 4 due to space constraints, as shown in Tables 5-8. The complete results is available at `http://centres.smu.edu.sg/larc/Orienteering-Problem-Library`.

Tables 5 - 8 consist of two identical structure parts. The first column shows the instance name. The second column contains the best known solution value *BK* from one of the state-of-the-art algorithms: IterILS, ACS, SSA, GVNS and I3CH. The following three columns present maximum, average and minimum solution values obtained by SAILS from 10 runs. The "*BG* (%)" column refers to the percentage gap between *BK* and the maximum (best) solution obtained by a particular algorithm. "*AG* (%)" provides the percentage gap between *BK* and the average solution obtained by a particular algorithm. The last three columns show maximum, average and minimum computational times (in seconds) required to obtain the best found within the given computational time. The new *BK* are highlighted in **bold**.

Table 5: Detailed results of SAILS on Cordeau et al.'s instances with $m = 1$

Instance	*BK*	SAILS			*BG*	*AG*	*Time*			Instance	*BK*	SAILS			*BG*	*AG*	*Time*		
		Max	Avg	Min	(%)	(%)	Max	Avg	Min			Max	Avg	Min	(%)	(%)	Max	Avg	Min
pr01	*308*	308	308	308	0.0	0.0	5.7	2.4	0.0	pr11	353	353	353	353	0.0	0.0	412.3	124.7	10.6
pr02	*404*	404	404	404	0.0	0.0	230.8	43.4	6.8	pr12	442	441	440	439	0.2	0.5	87.9	48.2	6.2
pr03	*394*	394	394	394	0.0	0.0	40.7	18.7	0.4	pr13	466	458	455.9	455	1.7	2.2	407.8	152.3	17.7
pr04	*489*	489	482.8	471	0.0	1.3	425.0	120.3	16.2	pr14	567	552	545.2	525	2.6	3.8	408.5	128.1	26.2
pr05	*595*	592	591.1	590	0.5	0.7	481.6	189.5	34.3	pr15	707	707	686.4	662	0.0	2.9	299.2	156.8	52.6
pr06	591	579	565	553	2.0	4.4	262.5	90.0	14.5	pr16	674	650	636	621	3.6	5.6	462.3	135.2	44.2
pr07	*298*	298	298	298	0.0	0.0	13.9	3.5	0.1	pr17	362	362	362	362	0.0	0.0	74.4	37.0	7.8
pr08	*463*	463	463	463	0.0	0.0	30.6	10.1	2.6	pr18	539	539	538	533	0.0	0.2	490.9	98.9	13.3
pr09	*493*	493	491.8	490	0.0	0.2	153.8	70.8	5.9	pr19	562	560	546.1	536	0.4	2.8	397.3	151.2	4.1
pr10	594	583	577.7	563	1.9	2.7	490.2	208.2	30.5	pr20	667	648	635.5	627	2.8	4.7	356.5	151.0	48.1

Table 6: Detailed results of SAILS on Cordeau et al.'s instances with $m = 2$

Instance	*BK*	SAILS			*BG*	*AG*	Time			Instance	*BK*	SAILS			*BG*	*AG*	Time		
		Max	Avg	Min	(%)	(%)	Max	Avg	Min			Max	Avg	Min	(%)	(%)	Max	Avg	Min
pr01	502	502	502	502	0.0	0.0	91.5	22.3	2.2	pr11	566	566	563.5	559	0.0	0.4	195.1	35.1	5.8
pr02	715	712	701.8	693	0.4	1.8	451.4	136.6	18.6	pr12	774	766	757.8	743	1.0	2.1	260.4	130.4	26.3
pr03	742	742	732.3	710	0.0	1.3	437.4	252.7	96.9	pr13	832	**843**	826.8	811	-1.3	0.6	476.5	174.7	48.8
pr04	925	**926**	909.4	898	-0.1	1.7	431.0	218.5	24.6	pr14	1017	986	965.4	924	3.0	5.1	471.4	275.4	88.2
pr05	1101	1092	1068.7	1038	0.8	2.9	352.7	218.5	75.7	pr15	1219	1206	1175.6	1130	1.1	3.6	470.1	323.2	195.6
pr06	1076	1045	1022.2	993	2.9	5.0	482.3	303.0	39.7	pr16	1231	1149	1132.7	1107	6.7	8.0	406.2	265.8	131.3
pr07	566	566	566	566	0.0	0.0	372.9	90.2	5.2	pr17	652	646	644.5	643	0.9	1.2	291.6	128.6	13.5
pr08	834	830	821.7	809	0.5	1.5	455.0	202.2	19.9	pr18	938	**946**	924.3	909	-0.9	1.5	439.5	220.0	33.9
pr09	905	**909**	888.2	861	-0.4	1.9	453.0	192.4	35.2	pr19	1034	1023	993.9	956	1.1	3.9	484.9	321.1	135.9
pr10	1129	1111	1074.6	1044	1.6	4.8	424.0	232.9	71.6	pr20	1232	1203	1172.7	1121	2.4	4.8	489.4	314.4	152.4

Table 7: Detailed results of SAILS on Cordeau et al.'s instances with $m = 3$

Instance	*BK*	SAILS			*BG*	*AG*	*Time*			Instance	*BK*	SAILS			*BG*	*AG*	*Time*		
		Max	Avg	Min	(%)	(%)	Max	Avg	Min			Max	Avg	Min	(%)	(%)	Max	Avg	Min
pr01	622	619	614.5	606	0.5	1.2	390.0	95.6	2.6	pr11	654	654	652.1	649	0.0	0.3	205.6	57.3	5.5
pr02	942	**943**	932.4	920	-0.1	1.0	487.9	246.1	28.2	pr12	1002	993	976.4	959	0.9	2.6	454.4	218.2	33.8
pr03	1010	1004	992.3	972	0.6	1.8	429.4	187.3	23.7	pr13	1145	1132	1119.1	1108	1.1	2.3	360.5	155.6	44.8
pr04	1294	1281	1259.8	1243	1.0	2.6	466.6	282.7	71.6	pr14	1372	1341	1317.2	1294	2.3	4.0	454.8	289.6	143.8
pr05	1482	1459	1434.3	1414	1.6	3.2	490.2	350.1	141.2	pr15	1654	1621	1600.5	1578	2.0	3.2	484.0	408.6	264.7
pr06	1514	1474	1428.6	1392	2.6	5.6	492.4	325.0	136.5	pr16	1668	1574	1554.5	1532	5.6	6.8	472.5	375.7	252.0
pr07	744	741	736.2	732	0.4	1.0	402.6	152.1	7.3	pr17	841	835	827.4	817	0.7	1.6	331.5	92.9	13.5
pr08	1139	1124	1107.4	1082	1.3	2.8	451.5	237.6	33.8	pr18	1281	1250	1230.4	1202	2.4	4.0	397.4	234.2	77.6
pr09	1275	1244	1225.3	1203	2.4	3.9	492.0	260.1	69.0	pr19	1417	1372	1350.4	1311	3.2	4.7	488.9	344.1	137.2
pr10	1573	1537	1490	1448	2.3	5.3	384.6	233.1	75.8	pr20	1684	1651	1608.9	1575	2.0	4.5	472.5	333.2	117.7

Table 8: Detailed results of SAILS on Cordeau et al.'s instances with $m = 4$

Instance	*BK*	SAILS			*BG*	*AG*	*Time*			Instance	*BK*	SAILS			*BG*	*AG*	*Time*		
		Max	Avg	Min	(%)	(%)	Max	Avg	Min			Max	Avg	Min	(%)	(%)	Max	Avg	Min
pr01	657	657	657	657	0.0	0.0	55.1	8.9	2.0	pr11	657	657	657	657	0.0	0.0	7.0	4.3	0.3
pr02	1079	1068	1057.3	1041	1.0	2.0	476.4	170.9	13.9	pr12	1132	1112	1100.9	1083	1.8	2.7	467.8	174.8	25.5
pr03	1232	1228	1191.7	1157	0.3	3.3	439.7	185.2	79.6	pr13	1386	1363	1332.4	1285	1.7	3.9	398.3	178.7	60.8
pr04	1585	1543	1518.5	1474	2.6	4.2	443.8	222.9	53.7	pr14	1670	1660	1606.5	1564	0.6	3.8	492.7	327.3	160.8
pr05	1838	1774	1735.3	1659	3.5	5.6	453.8	328.9	160.4	pr15	2065	1953	1924.4	1867	5.4	6.8	495.6	389.0	264.6
pr06	1860	1796	1765.8	1729	3.4	5.1	481.6	325.2	140.0	pr16	2065	1945	1914.8	1877	5.8	7.3	481.0	382.4	237.1
pr07	876	869	860	842	0.8	1.8	477.2	167.2	7.6	pr17	934	930	918.6	906	0.4	1.6	488.3	210.5	12.1
pr08	1382	1349	1333.6	1316	2.4	3.5	427.8	227.9	66.0	pr18	1539	1501	1474.9	1450	2.5	4.2	463.4	261.8	82.5
pr09	1619	1573	1540.6	1516	2.8	4.8	467.2	371.4	169.1	pr19	1750	1721	1677.2	1638	1.7	4.2	476.1	375.9	150.8
pr10	1943	1869	1835.6	1802	3.8	5.5	491.9	335.0	166.5	pr20	2062	2047	1954.6	1903	0.7	5.2	459.4	389.7	181.4

Table 9: Overall ”Average” Comparison of SAILS to the state-of-the-art algorithms on ”INST-M” instances

Instance Set	*Numb*	IterILS		ACS		SSA		GVNS		I3CH		SAILS	
		$\overline{BG}$ (%)	$\overline{Time}$	$\overline{AG}$ (%)	$\overline{Time}^{‡}$	$\overline{BG}$ (%)	$\overline{Time}$	$\overline{AG}$ (%)	$\overline{Time}$	$\overline{BG}$ (%)	$\overline{Time}$	$\overline{AG}$ (%)	$\overline{Time}^{‡}$
m = 1													
c100	9	1.11	0.3	0.00	2.4	0.00	19.4	1.22	64.3	0.00	29.3	0.00	1.0
r100	12	1.90	0.2	0.24	148.0	0.11	21.5	2.68	11.4	0.56	33.3	0.00	5.9
rc100	8	2.92	0.2	0.00	55.3	0.00	20.4	3.51	3.8	1.66	29.7	0.06	4.8
c200	8	2.28	1.6	0.58	132.2	0.13	34.5	1.11	74.3	0.40	98.2	0.14	68.3
r200	11	2.90	1.6	3.17	600.9	1.30	42.1	3.38	13.1	1.05	204.9	1.05	281.6
rc200	8	3.43	1.5	2.04	596.2	0.96	46.2	3.96	6.2	2.68	138.8	1.20	158.8
pr01-10	10	4.74	1.7	1.22	627.9	0.98	103.2	1.62	4.8	1.07	126.8	0.93	75.7
pr11-20	10	9.56	1.8	11.87	342.6	3.71	149.3	4.26	9.3	4.28	151.5	2.28	118.3
m = 2													
c100	9	0.94	1.0	0.14	309.9	0.00	24.3	0.72	53.9	0.00	101.2	0.03	19.8
r100	12	2.36	0.8	0.55	532.2	0.23	33.7	1.80	23.3	0.58	73.3	0.08	83.7
rc100	8	2.47	0.6	1.27	504.1	0.19	37.2	2.80	7.8	0.90	68.5	0.17	60.4
c200	8	2.54	3.2	1.81	539.7	1.18	49.3	0.58	13.0	0.68	466.7	0.90	124.4
r200	11	2.74	2.1	3.71	1055.8	0.58	84.0	1.30	5.7	0.21	612.9	1.35	270.1
rc200	8	4.14	2.0	3.83	904.3	1.25	73.6	2.57	4.9	0.62	511.5	1.96	233.1
pr01-10	10	6.22	4.4	3.57	729.4	2.45	159.9	1.79	15.1	1.11	287.4	2.09	186.9
pr11-20	10	7.86	4.8	6.15	920.5	3.88	185.4	2.18	31.8	2.70	354.3	3.11	218.9
m = 3													
c100	9	2.55	1.4	0.79	402.7	0.33	32.4	0.95	63.7	0.11	221.3	0.50	75.7
r100	12	1.79	1.6	1.30	681.6	0.39	51.5	2.27	28.5	0.21	137.6	0.57	108.0
rc100	8	3.14	1.0	1.07	458.7	0.64	39.3	2.32	13.0	0.27	117.4	0.41	66.0
c200	8	1.93	2.0	1.63	509.7	1.24	54.9	0.19	3.0	0.00	14.3	0.86	154.1
r200	11	0.30	1.3	0.24	452.3	0.08	38.6	0.20	2.7	0.01	105.6	0.13	98.4
rc200	8	1.44	1.6	0.94	588.8	0.27	54.2	0.44	2.9	0.04	190.8	0.40	82.9
pr01-10	10	6.58	8.5	4.21	818.9	2.34	181.1	1.13	33.2	0.36	493.2	2.85	237.0
pr11-20	10	9.19	8.9	7.24	997.7	3.81	231.5	1.80	58.2	1.11	578.1	3.39	250.9
m = 4													
c100	9	3.11	2.2	1.27	476.8	0.55	45.5	1.63	51.4	0.10	304.5	1.38	108.0
r100	12	3.31	2.4	1.92	684.3	0.73	53.7	2.28	32.7	0.16	214.4	1.40	117.7
rc100	8	3.18	1.8	2.18	656.3	0.37	62.6	1.79	14.2	0.23	177.3	0.80	107.9
c200	8	0.00	0.9	0.07	3.0	0.00	38.4	0.00	0.2	0.00	0.1	0.00	14.0
r200	11	0.00	0.8	0.00	48.8	0.00	36.5	0.00	0.1	0.00	0.2	0.00	20.9
rc200	8	0.00	1.0	0.01	249.6	0.00	36.9	0.01	0.3	0.00	0.2	0.00	27.3
pr01-10	10	7.08	13.0	3.42	966.6	2.23	235.0	1.60	49.1	0.36	659.0	3.58	234.3
pr11-20	10	8.47	12.6	6.34	997.2	3.95	261.1	2.81	89.8	0.45	847.6	3.97	269.4
Grand Mean		3.50	2.8	2.33	541.5	1.09	81.2	1.74	24.8	0.69	233.8	1.14	124.1

‡ Average computational time to obtain the best found (in seconds)

Table 10: Overall ”Best” Comparison of SAILS to the state-of-the-art algorithms on ”INST-M” instances

Instance Set	*Numb*	IterILS	ACS	SSA	GVNS	I3CH	SAILS
		$\overline{BG}$ (%)	$\overline{BG}$ (%)	$\overline{BG}$ (%)	$\overline{BG}$ (%)	$\overline{BG}$ (%)	$\overline{BG}$ (%)
$m = 1$							
c100	9	1.11	0.00	0.00	0.56	0.00	0.00
r100	12	1.90	0.00	0.11	1.72	0.56	0.00
rc100	8	2.92	0.00	0.00	1.88	1.66	0.00
c200	8	2.28	0.40	0.13	0.55	0.40	0.00
r200	11	2.90	2.19	1.30	2.45	1.05	0.13
rc200	8	3.43	1.23	0.96	2.53	2.68	0.23
pr01-10	10	4.74	1.06	0.98	0.56	1.07	0.44
pr11-20	10	9.56	11.13	3.71	3.17	4.28	1.14
$m = 2$							
c100	9	0.94	0.00	0.00	0.47	0.00	0.00
r100	12	2.36	0.20	0.23	1.19	0.58	-0.03
rc100	8	2.47	0.33	0.19	0.78	0.90	0.00
c200	8	2.54	1.27	1.18	0.25	0.68	0.25
r200	11	2.74	3.16	0.58	0.67	0.21	0.46
rc200	8	4.14	2.70	1.25	1.68	0.62	0.68
pr01-10	10	6.22	2.59	2.45	0.82	1.11	0.56
pr11-20	10	7.86	5.00	3.88	1.21	2.70	1.40
$m = 3$							
c100	9	2.55	0.22	0.33	0.45	0.11	0.11
r100	12	1.79	0.36	0.39	1.22	0.21	0.11
rc100	8	3.14	0.35	0.64	0.91	0.27	-0.01
c200	8	1.93	1.10	1.24	0.07	0.00	0.35
r200	11	0.30	0.13	0.08	0.11	0.01	0.04
rc200	8	1.44	0.42	0.27	0.32	0.04	0.13
pr01-10	10	6.58	2.96	2.34	0.36	0.36	1.26
pr11-20	10	9.19	5.40	3.81	1.02	1.11	2.02
$m = 4$							
c100	9	3.11	0.36	0.55	1.04	0.10	0.38
r100	12	3.31	0.78	0.73	1.22	0.16	0.39
rc100	8	3.18	0.78	0.37	0.95	0.23	-0.01
c200	8	0.00	0.00	0.00	0.00	0.00	0.00
r200	11	0.00	0.00	0.00	0.00	0.00	0.00
rc200	8	0.00	0.00	0.00	0.00	0.00	0.00
pr01-10	10	7.08	2.76	2.23	1.08	0.36	2.08
pr11-20	10	8.47	5.53	3.95	2.05	0.45	2.05
Grand Mean		3.50	1.69	1.09	1.00	0.69	0.46

Table 11: Comparison of SAILS to the state-of-the-art methods on "OPT" instances

Instance Set	*Numb*	IterILS		SSA		GVNS		I3CH		SAILS		
		$\overline{BG}$(%)	$\overline{Time}$	$\overline{BG}$(%)	$\overline{Time}$	$\overline{AG}$(%)	$\overline{Time}$	$\overline{BG}$(%)	$\overline{Time}$	$\overline{BG}$(%)	$\overline{AG}$(%)	$\overline{Time}$‡
c100	9	1.41	2.8	1.04	71.4	0.47	3.0	0.00	55.4	0.43	0.87	93.6
r100	12	1.93	2.7	0.42	96.2	1.55	15.3	0.07	1021.0	0.45	1.18	203.7
rc100	8	2.06	3.5	0.35	77.8	1.29	15.2	0.00	66.7	0.54	1.11	147.1
c200	8	0.00	1.0	0.00	38.5	0.00	0.2	0.00	0.7	0.00	0.00	19.8
r200	11	0.62	1.5	0.16	53.6	0.17	2.1	0.07	201.7	0.01	0.22	166.6
rc200	8	0.47	1.6	0.07	38.0	0.16	1.1	0.04	221.2	0.07	0.10	75.0
pr01-pr10	10	2.32	28.0	1.04	520.3	1.25	19.8	0.78	380.0	1.22	1.53	260.1
Grand Mean		1.30	6.1	0.45	133.7	0.74	8.5	0.15	319.4	0.40	0.75	146.3

‡ Average computational time to obtain the best found (in seconds)

Table 12: New best known solution values found by SAILS (second scenario)

Instance	*m*	Old *BK*	New *BK*	Instance	*m*	Old *BK*	New *BK*
r207	1	1072	1074	rc201	2	1384	1385
rc202	1	936	938‡	r112	4	971	972

‡ Same result with that of ILS [6]

Tables 9 reports the average of AG ($\overline{AG}$ (%)) and the average computational time (in seconds) ($\overline{Time}$) for each instance set of "INST-M". Since IterILS, SSA and I3CH were only run once, we also include their average of BG ($\overline{BG}$ (%)) although we cannot directly compare with $\overline{AG}$ (%). The *num* column provides the number of instances in a particular instance set. The values of $\overline{Time}$ for ACS and SAILS refer to the average of computational time (in seconds) in order to obtain the best found from all runs. On the other hand, the ones for IterILS, SSA, GVNS and I3CH refer to the average of computational time (in seconds) for solving one particular instance set. All values reported have been adjusted according to the computer's speed as listed in Table 3.

In general, SAILS is competitive with the state-of-the-art algorithms. IterILS is an algorithm with the main purpose of providing good solutions very quickly, whereas SAILS focuses on finding better solutions at the cost of larger computational times. SAILS outperforms ACS in terms of the computational time and the solution quality. ACS requires 1 hour ($\approx$ 1404 seconds using our PC) while SAILS only requires 492 seconds for solving one instance. The Grand Mean of $\overline{Time}$ of SAILS is around 23% of ACS's Grand Mean. In terms of the solution quality, SAILS is able to reduce the Grand Mean of $\overline{AG}$ up to 48.9%. SAILS also outperforms GVNS in terms of the solution quality. The $\overline{AG}$'s Grand Mean of SAILS and GVNS are 1.14% and 1.74%, respectively although SAILS spends more computational time compared against that of GVNS.

Tables 10 summarizes the comparison among algorithms in terms of the values of $\overline{BG}$. All algorithms except IterILS are able to provide the Grand Mean of $\overline{BG}$ below 1.7%. SAILS is the best compared against other algorithms where the grand Mean of $\overline{BG}$ is only 0.46%. It also has a narrow range of -0.03% to 2.08%. Three instance sets give negative values, meaning that SAILS achieves some improvements of some *BK*s in those instance sets. Two of them are from rc100 instance sets with m = 3 and 4. Table 11 reports the results obtained on "OPT" instances [21]. SAILS outperforms other algorithms, except I3CH in terms of the Grand Mean of $\overline{BG}$. SAILS provides better results with greater computational time. The Grand Mean values of $\overline{AG}$ for GVNS and SAILS are 0.74% and 0.75%, respectively. Thus, we can conclude that SAILS provides the trade-off between the solution quality and computational time, on average.

At first glance, SAILS requires more computational time compared against those of other algorithms except ACS. Therefore, we implement the following second s-cenario. Additional experiments were done by setting the computational time as the one of I3CH. It has been shown that I3CH outperforms other approaches when us-ing the same computational time [7]. We encountered four additional new *BK*s, as shown in Table 12. The results of using the same computational time are presented in Tables 13 and 14. We observed that SAILS overall average performance in terms of $\overline{AG}$ is 0.12% better than that of I3CH. I3CH has a wider range for $\overline{BG}$ values. SAILS and I3CH ranges from -0.01% to 3.18% and from 0.00% to 4.28%, respectively. For "OPT" instances, I3CH performs best with the lowest Grand Mean of $\overline{BG}$. The value is only 0.15%. The computational time using I3CH is less than the one used in the first scenario, except for r100 instance set.

Table 15 summarizes the percentage improvement of the solution quality (in av-erage) for all instance sets. In general, we can conclude that SAILS is able to improve

Table 13: Comparison with the same computational time on ”INST-M” instances

m	Instance Set	I3CH	SAILS		$\overline{Time}$
		$\overline{BG}$(%)	$\overline{BG}$(%)	$\overline{AG}$(%)	(seconds)
1	c100	0.00	0.00	0.00	29.3
	r100	0.56	0.00	0.03	33.3
	rc100	1.66	0.00	0.10	29.7
	c200	0.40	0.00	0.32	98.8
	r200	1.05	0.33	1.36	207.5
	rc200	2.68	0.52	1.44	140.1
	pr01-10	1.07	0.37	1.02	126.9
	pr11-20	4.28	1.49	3.13	152.0
2	c100	0.00	0.00	0.18	101.0
	r100	0.58	-0.01	0.31	73.2
	rc100	0.90	0.02	0.32	68.4
	c200	0.68	0.25	0.88	466.7
	r200	0.21	0.51	1.48	616.0
	rc200	0.62	0.51	1.90	512.5
	pr01-10	1.11	0.73	1.81	287.2
	pr11-20	2.70	1.54	2.96	355.4
3	c100	0.11	0.22	0.77	220.9
	r100	0.21	0.14	0.70	137.4
	rc100	0.27	0.00	0.60	117.2
	c200	0.00	3.18	4.21	16.1
	r200	0.01	0.27	0.61	109.8
	rc200	0.04	0.52	1.49	192.3
	pr01-10	0.36	1.17	2.92	492.7
	pr11-20	1.11	1.41	3.05	578.8
4	c100	0.10	0.58	1.52	303.9
	r100	0.16	0.43	1.52	214.0
	rc100	0.23	0.17	0.99	177.0
	c200	0.00	0.00	0.00	1.4
	r200	0.00	0.02	0.13	4.0
	rc200	0.00	0.15	0.34	2.1
	pr01-10	0.36	1.74	3.72	658.2
	pr11-20	0.45	1.92	3.41	847.6
Grand Mean		0.69	0.57	1.36	234.4

the initial solution generated by the Greedy Construction Heuristic. The values range from 0.30% to 19.41%. SAILS performs best for m = 1 where the percentage of improvement is varied from 6.20% to 19.41%. Figure 1 shows the Grand Mean values obtained in terms of percentage improvement, as shown in Table 15. We observe that the higher the value of m, the lower the Grand Mean value. It is expected since the problem is more difficult for higher values of m. ”OPT” instance sets are the most difficult to solve.

Table 14: Comparison with the same computational time on ”OPT” instances

Instance Set	I3CH	SAILS		$\overline{Time}$
	$\overline{BG}$(%)	$\overline{BG}$(%)	$\overline{AG}$(%)	(seconds)
c100	0.00	2.15	2.92	55.6
r100	0.07	0.79	1.47	1018.6
rc100	0.00	1.15	1.90	66.8
c200	0.00	0.00	0.00	1.7
r200	0.07	0.31	0.97	204.8
rc200	0.04	0.38	0.96	222.5
pr01-10	0.78	1.46	1.89	382.1
Grand Mean	0.15	0.90	1.46	320.1

Table 15: The solution quality improvement by SAILS (in %)

Instance Set	”INST-M”				”OPT”
	$m = 1$	$m = 2$	$m = 3$	$m = 4$	
c100	9.43	10.47	9.45	9.03	5.08
r100	11.13	13.30	14.75	14.68	6.25
rc100	17.73	15.67	15.10	16.04	7.77
c200	6.20	6.15	6.71	0.40	0.30
r200	9.09	7.93	2.39	0.32	3.74
rc200	13.41	10.91	5.39	1.18	4.04
pr01-10	18.57	18.86	15.70	12.80	5.03
pr11-20	19.41	18.63	14.26	11.43	-
Grand Mean	12.87	12.59	10.50	8.17	4.50

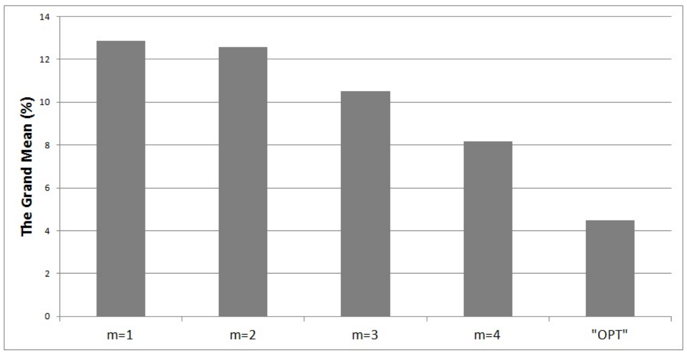

Fig. 1: The Grand Mean values for m = 1 to 4

5 Conclusion

In this paper, we present a hybridization of Simulated Annealing and Iterated Local Search, namely SAILS, to solve the TOPTW. The proposed algorithm is run in two different scenarios. The first scenario is to run SAILS with longer computational time since we are more concerned with the solution quality. The second scenario is mainly tailored for the comparison purpose with the-state-of-the-art algorithms. This is done by setting the computational times to those of one of the-state-of-the-art algorithms, I3CH. Both scenarios are applied to benchmark instances.

Computational results show that SAILS is competitive with the-state-of-the-art algorithms. Simulated Annealing is able to improve the performance of Iterated Local Search by discovering 19 new best known solutions. Two areas of future work can be considered. Using different scenarios for building the initial solutions in order to observe the effect of Simulated Annealing would be one interesting area. And since the Orienteering Problem and its variants have attracted more attention in recent years, SAILS may be potentially applied to solve them.

Acknowledgements This research is supported by Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office, Media Development Authority (MDA).

References

1. Cura, T.: An artificial bee colony algorithm approach for the team orienteering problem with time windows. Computers and Industrial Engineering **74**, 270–290 (2014)
2. Duque, D., Lozano, L., Medaglia, A.: Solving the orienteering problem with time windows via the pulse framework. Computers and Operations Research **54**, 168–176 (2015)
3. Garcia, A., Arbelaitz, O., Vansteenwegen, P., Souffriau, W., Linaza, M.T.: Hybrid approach for the public transportation time dependent orienteering problem with time windows. In: E. Corchado, M. Romay, A. Savio (eds.) Hybrid Artificial Intelligence Systems, *Lecture Notes in Computer Science*, vol. 6077, pp. 151–158. Springer, Berlin, Germany (2010)
4. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, Massachusetts (1989)
5. Golden, B., Levy, L., Vohra, R.: The orienteering problem. Naval Research Logistics **34**(3), 307–318 (1987)
6. Gunawan, A., Lau, H.C., Lu, K.: An iterated local search algorithm for solving the orienteering problem with time windows. In: G. Ochoa, F. Chicano (eds.) proceedings of the 15th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoStar 2015), 8-10 April 2015, Copenhagen, Denmark, *Lecture Notes in Computer Science*, vol. 9026, pp. 61–73. Springer-Verlag, Berlin, Germany (2015)
7. Hu, Q., Lim, A.: An iterative three-component heuristic for the team orienteering problem with time windows. European Journal of Operational Research **232**(2), 276–286 (2014)
8. Johnson, S.M.: Generation of permutations by adjacent transposition. Mathematics of Computation **17**(83), 282–285 (1963)
9. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
10. Labadie, N., Mansini, R., Melechovskỳ, J., Calvo, R.W.: Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. Journal of Heuristics **17**(6), 729–753 (2011)
11. Labadie, N., Mansini, R., Melechovskỳ J.and Calvo, R.: The team orienteering problem with time windows: an LP-based granular variable neighborhood search. European Journal of Operational Research **220**(1), 15–27 (2012)
12. Lee, D.H., Cao, Z., Meng, Q.: Scheduling of two-transtainer systems for loading outbound containers in port container terminals with simulated annealing algorithm. International Journal of Production Economics **107**(1), 115–124 (2007)
13. Lin, S.W., Yu, V.F.: A simulated annealing heuristic for the team orienteering problem with time windows. European Journal of Operational Research **217**(1), 94–107 (2012)
14. Lin, S.W., Yu, V.F., Chou, S.Y.: Solving the truck and trailer routing problem based on a simulated annealing heuristic. Computers and Operations Research **36**(5), 1683–1692 (2009)

15. Lourenço, H., Martin, O., Stützle, T.: Iterated local search. In: Handbook of metaheuristics, pp. 320–353. Springer (2003)
16. Montemanni, R., Gambardella, L.M.: Ant colony system for team orienteering problem with time windows. Foundations of Computing and Decision Sciences **34**(4), 287–306 (2009)
17. Montemanni, R., Weyland, D., Gambardella, L.M.: An enhanced ant colony system for the team orienteering problem with time windows. In: Proceedings of 2011 International Symposium on Computer Science and Society (ISCCS), pp. 381–384. Kota Kinabalu, Malaysia (2011)
18. Puchinger, J., Raidl, G.R.: Combining metaheuristics and exact algorithms in combinatorial optimization: a survey and classification. In: J. Mira, J.R. Alvarez (eds.) Artificial Intelligence and Knowledge Engineering Applications: First International Work-Conference on the Interplay between Natural and Artificial Computation, *Lecture Notes in Computer Science*, vol. 3562, pp. 41–53. Springer (2005)
19. Righini, G., Salani, M.: Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. Computers and Operations Research **36**(4), 1191–1203 (2009)
20. Talbi, E.G., Hafidi, Z., Geib, J.M.: A parallel adaptive tabu search approach. Parallel Computing **24**(14), 2003–2019 (1998)
21. Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D.: Iterated local search for the team orienteering problem with time windows. Computers and Operations Research **36**(12), 3281–3290 (2009)