6-2017

Well-tuned algorithms for the team orienteering problem with time windows

Aldy GUNAWAN
Singapore Management University, aldygunawan@smu.edu.sg

Hoong Chuin LAU
Singapore Management University, hclau@smu.edu.sg

Kun LU
Singapore Management University, kunlu@smu.edu.sg

Lu KUN

Citation

GUNAWAN, Aldy; LAU, Hoong Chuin; LU, Kun; and KUN, Lu. Well-tuned algorithms for the team orienteering problem with time windows. (2017). *Journal of the Operational Research Society*. 68, (8), 861-876.
Available at: https://ink.library.smu.edu.sg/sis_research/4305

Published in Journal of the Operational Research Society, 2017 August, Volume 68, Issue 8, Pages 861-876
http://doi.org/10.1057/s41274-017-0244-1

Well-tuned algorithms for the Team Orienteering Problem with Time Windows

Aldy Gunawan[1]*, Hoong Chuin Lau[1], Pieter Vansteenwegen[2] and Kun Lu[1]

[1]*School of Information Systems, Singapore Management University, 80 Stamford Road, Singapore 178902, Singapore; and* [2]*KU Leuven Mobility Research Centre - CIB, Celestijnenlaan 300, Box 2422, 3001 Leuven, Belgium*

The Team Orienteering Problem with Time Windows (TOPTW) is the extension of the Orienteering Problem (OP) where each node is limited by a predefined time window during which the service has to start. The objective of the TOPTW is to maximize the total collected score by visiting a set of nodes with a limited number of paths. We propose two algorithms, Iterated Local Search and a hybridization of Simulated Annealing and Iterated Local Search (SAILS), to solve the TOPTW. As indicated in multiple research works on algorithms for the OP and its variants, determining appropriate parameter values in a statistical way remains a challenge. We apply Design of Experiments, namely factorial experimental design, to screen and rank all the parameters thereby allowing us to focus on the parameter search space of the important parameters. The proposed algorithms are tested on benchmark TOPTW instances. We demonstrate that well-tuned ILS and SAILS lead to improvements in terms of the quality of the solutions. More precisely, we are able to improve 50 best known solution values on the available benchmark instances.

Journal of the Operational Research Society (2017) **68(8),** 861–876. doi:10.1057/s41274-017-0244-1;
published online 26 June 2017

Keywords: Orienteering Problem; time windows; Iterated Local Search; Simulated Annealing; hybrid algorithm

1. Introduction

The Orienteering Problem (OP), introduced by Tsiligirides (1984), is a sport in which a competitor has to determine a path from a start point to a final destination, visiting a subset of control points (nodes) along the path. In the context of the Team OP (TOP), a certain number of paths are allowed, instead of only one. The Team Orienteering Problem with Time Windows (TOPTW) is an extension of the TOP (Labadie *et al*, 2012). 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 collected score by visiting a set of nodes with a limited number of paths.

Since the OP has been proven to be NP-hard (Golden *et al*, 1987), it is unlikely that the TOPTW can be solved optimally within polynomial time. Therefore, it is interesting to propose fast algorithms to solve the problem, especially when dealing with large instances. A survey of (T)OP(TW) formulations and applied solution algorithms is published by Vansteenwegen *et al* (2011). Gunawan *et al* (2016) present a comprehensive and thorough survey of recent variants of the OP, including the proposed solution approaches and the most recent applications of the OP. For example, TOPTW has been used as a model in formulating various practical applications, such as the Tourist Trip Design Problem (TTDP) and the mobile-crowdsourcing problem (Gunawan *et al*, 2016). Other applications of the TOP (TW), mentioned in Vansteenwegen *et al* (2009), are the home fuel delivery problem, athlete recruiting from high schools, routing technicians to service customers, etc.

In this paper, we introduce two algorithms for solving the TOPTW. The first algorithm is based on an Iterated Local Search (ILS) algorithm. Iterated Local Search (Louren *et al*, 2003) is a simple but effective metaheuristic. Vansteenwegen *et al* (2009) introduce a simple, fast and effective ILS for the TOPTW. Our ILS differs in the way of generating the initial solution. An initial solution is constructed by inserting nodes subsequently into one of the paths, based on Roulette-Wheel Selection method (Goldberg, 1989). Various components of ILS such as LocalSearch, Perturbation, and AcceptanceCriterion are also included. A simplified version of our ILS has been used to solve the TOPTW, with only one path (Gunawan *et al*, 2015b). Since the TOPTW involves more than one path, we introduce in this paper more operators of LocalSearch, such as swapping two nodes from two different paths and moving one node from one path to another, and an exchange path strategy in Perturbation.

The second algorithm, namely SAILS, is a hybridization of ILS and Simulated Annealing (SA). SA helps to increase the diversification of the search, which is typically needed for obtaining high-quality OP-solutions, as stated by, for instance, Gendreau *et al* (1998) and Vansteenwegen *et al* (2009). The current implementation of ILS easily gets trapped in local optima. The hybridization with SA helps to escape from these local optima. SA has the capability to escape from a local optimum by accepting a worse solution with a probability that changes over time. SA has been applied to a huge variety of combinatorial problems. Preliminary results of SAILS were presented in the 7th Multidisciplinary International Scheduling Conference (MISTA 2015) (Gunawan *et al*, 2015a).

Associated with both ILS and SAILS is a set of parameter values that need to be tuned. It is known that good parameter values can have a significant impact on the performance of an algorithm (Hutter *et al*, 2009). Many proposed algorithms do not pay special attention to it—the underlying parameters are set either rather arbitrarily without any explanation or based on limited preliminary tests, or based on values reported in previous studies.

The problem of setting the parameter values has become an interesting research area, typically divided in two categories, parameter control and parameter tuning (Eiben *et al*, 1999). The former changes the parameter values during an algorithm run while the latter focuses on finding good parameter values before running the algorithm. Some works related to parameter control are presented by Eiben *et al* (2007) and Stützle *et al* (2012). An example of parameter tuning is presented by Eiben and Smit (2012).

In response to this need for a sound statistical approach to obtain parameter values, we focus on the parameter tuning by applying a factorial design experiment to screen and rank all the parameters based on their importance, as described in Gunawan and Lau (2011). In this approach, parameters found to be "unimportant" (in the sense that the solution quality is insensitive to the values of these parameters) are set to some constant values so that the resulting parameter space that needs to be explored is reduced and we can focus on tuning the important parameters. Adenso-Diaz and Laguna (2006) develop CALIBRA which employs a Taguchi fractional experimental design followed by a local search procedure. However, CALIBRA can only handle up to five parameters and focuses on the main effects of parameters without exploiting the interaction effects between parameters.

We test our well-tuned ILS and SAILS on benchmark TOPTW instances. The experiments are run with different scenarios in order to ensure fair comparison. We compare the results with those of the state-of-the-art algorithms: Iterated Local Search (Vansteenwegen *et al*, 2009), Variable Neighborhood Search (Tricoire *et al*, 2010), Ant Colony System (Montemanni and Gambardella, 2009; Montemanni *et al*, 2011), Slow Simulated Annealing (Lin and Yu, 2012), Granular Variable Neighborhood Search, Hybridized Greedy Randomized Iterated Local Search (Souffriau *et al*, 2013) and Iterative Three-Component Heuristic (Hu and Lim, 2014). We show that our proposed algorithms outperform the state-of-the-art algorithms. More precisely, 50 new best known solutions are found for a set of 304 well-known benchmark instances for which no proven optimal solution is available.

The main contributions of this paper are listed below:

- We extend the ILS that has been used for solving the OPTW (Gunawan *et al*, 2015b). More operators of LocalSearch and an exchange path strategy in Perturbation are introduced in order to deal with the TOPTW.
- We propose a hybrid algorithm, SAILS, in order to improve the performance of the ILS. Simulated Annealing is incorporated in order to avoid early termination in local optimality. We show that SA is well suited to improve the ILS.
- We apply the concept of Design of Experiment in order to determine the algorithm parameter values of important parameters.
- We are able to find 50 new best known solutions. This serves as benchmark for future studies and complements our previous work (Gunawan *et al*, 2015b).

The paper is organized as follows. In Section 2, the TOPTW is described, including most recent related works. Section 3 describes the proposed algorithms, ILS and SAILS. We also briefly explain the factorial experimental design for determining the parameter values of both algorithms. Section 4 is devoted to the experimental results and analysis. Finally, conclusions and some ideas for future works are summarized in Section 5.

2. The Team Orienteering Problem with Time Windows

In this section, we first present the formal definition of the TOPTW including notations used in this paper. We then summarize a short overview of the related literature on the TOPTW, including the state-of-the-art algorithms. For more details about algorithms, we refer to the original papers.

2.1. Problem description

The TOPTW is defined as follows. Consider an undirected network graph $G = (N, A)$ where $N = \{0, 1, 2, \ldots, v\}$ is the set of nodes and $A = \{(i, j) : i \neq j \in N\}$ refers to the set of arcs connecting two different nodes i and j. The nonnegative travel time between nodes i and j is represented by t_{ij}. Each node $i \in N \setminus \{0\}$ has a positive score u_i that is collected when node i is visited, a service time T_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 often assumed that node 0

is the starting and end nodes; therefore, $u_0 = T_0 = 0$. Take note that each graph G has $(v + 1)$ nodes.

Let $M = \{1, 2, \ldots, \mu\}$ be the set of paths. Each path $j \in M$ starts and ends at node 0. Each path is also constrained within the time limit $[e_0, l_0]$. Each node $i \in N$, except node 0, is visited at most once. The start time at node $i \in N$ in path $j \in M$ is within time window $[e_i; l_i]$. In case of an early arrival, a visit will only start when the time window opens. We have $e_0 = 0$ and $l_0 = T^{max}$, where T^{max} is the time budget or the maximum duration to complete a path. The objective function of the TOPTW is to maximize the total collected score from visited nodes from all paths. For the mathematical model formulation, please refer to the works of Vansteenwegen *et al* (2009, 2011).

2.2. Literature review

Vansteenwegen *et al* (2009) propose an ILS algorithm to solve the TOPTW. Only two operations of ILS, INSERT and SHAKE, are considered in this algorithm. A new dataset, which is generated from Cordeau *et al* (1997) and Solomon (1987), is introduced.

A metaheuristic algorithm based on an Ant Colony System (ACS) is proposed by Montemanni and Gambardella (2009). Montemanni *et al* (2011) further improve it by introducing Enhanced ACS (EACS). The algorithm includes two additional operations to overcome the drawbacks of ACS. Both operations focus on 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.

Variable Neighborhood Search (VNS) with several neighborhood structures is proposed by Tricoire *et al* (2010). Lin and Yu (2012) propose two different versions of Simulated Annealing, Fast SA (FSA) and Slow SA (SSA). FSA is mainly for the application that needs quick responses, while SSA is more concerned about the quality of the solutions at the expense of more computational time.

Another ILS algorithm is proposed by Gunawan *et al* (2015b) for solving the OPTW, i.e., with only one path. The algorithm starts by a greedy construction heuristic to construct an initial feasible solution. The initial solution 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.

The idea of combining some advantages of algorithms has been brought up by many researchers for solving different combinatorial optimization problems. Labadie *et al* (2011) propose a hybridization of the Greedy Randomized Adaptive Search Procedure (GRASP) and Evolutionary Local Search (ELS) for the TOPTW. Different constructive heuristics based on GRASP are introduced for constructing the initial solutions. Those initial solutions are further improved by ELS.

Another hybrid algorithm based on a local search (LS) procedure, Simulated Annealing (SA) and Route Combination (RR) is proposed by Hu and Lim (2014). Three components are iteratively incorporated within a certain number of iterations. Labadie *et al* (2012) introduce an LP-based Granular Variable Neighborhood Search (GVNS). The idea is to include time constraints and profits in addition to pure distances. By including the granularity, the performance of the proposed algorithm is improved.

Most recently, Cura (2014) proposes an Artificial Bee Colony (ABC) algorithm to solve the TOPTW. The 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. More details about the performances of the state-of-the-art algorithms for the TOPTW can be found in a recent survey (Gunawan *et al*, 2016).

3. Algorithms

This section is devoted to the description of our proposed algorithms, ILS and SAILS. We introduce a greedy construction heuristic for providing an initial solution. The initial solution is further improved either by ILS or by SAILS. The details of our proposed algorithms are described in the following subsections.

3.1. Greedy construction heuristic

The greedy construction heuristic is outlined in Algorithm 1. The idea of constructing an initial solution extends the one proposed by Gunawan *et al* (2015b) which is only dedicated for $\mu = 1$. First, we initialize N', N^* and S_0. N' and N^* denote the sets of unscheduled and scheduled nodes, respectively ($N' \cup N^* = N$). N^* is initialized by the starting and end nodes, node 0, while N' consists of unscheduled nodes. Take note that all benchmark instances assume both start and end nodes are the same. S_0 refers to the current feasible solution obtained so far, represented by μ-row vectors. Each row is initialized with starting and end nodes, node 0.

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 non-increasing order based on $ratio_{n,p,m}$
Select the best f elements of F and remove the rest
return F

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
$AccumProb \leftarrow 0$
$U \leftarrow rand(0, 1)$
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 $\langle n^*, p^*, m^* \rangle$

The construction heuristic is started by generating a set of all feasible candidate nodes that can be inserted, F. Each element of F, denoted as $\langle n, p, m \rangle$, represents a feasible insertion of node n in position p of path m. We examine all possibilities of inserting an unscheduled node $n \in N'$ in position p of path m. An insertion $\langle n, p, m \rangle$ is feasible if after the insertion, all scheduled nodes do not violate their respective time windows and the total spent time in path m does not exceed T^{max}.

Algorithm 2 summarizes the algorithm of generating F. Let $P(m)$ be a set of the positions of scheduled nodes on path m. For each possible insertion, the benefit of insertion $ratio_{n,p,m}$ is calculated by Eq. 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 certain number of elements, f, would be kept and considered.

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

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 Eq. 2:

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

The selection of $\langle n^*, p^*, m^* \rangle$ from F is based on the Roulette-Wheel Selection method (Goldberg, 1989). The probability of selection is proportional to the benefit of insertion of an individual, denoted as $ratio_{n,p,m}$. First, a random number $U \sim rand[0, 1]$ is generated. We then select a particular $\langle n, p, m \rangle$ and add the respective probability value $prob_{n,p,m}$ to the value of $AccumProb$. If $(U \leq AccumProb)$, the corresponding $\langle n, p, m \rangle$ is then selected.

The greedy construction heuristic is terminated when $F = \emptyset$. Vansteenwegen *et al* (2009) concluded that due to the time windows, the score of the node considered for insertion is more relevant compared to the time consumption of an insertion. Therefore, the square of the score is applied in Eq. 1.

3.2. ILS

The outline of ILS is presented in Algorithm 4. Three components of ILS: PERTURBATION, LOCALSEARCH and ACCEPTANCECRITERION are taken into consideration.

The initial solution S_0 which is generated by the greedy construction heuristic is treated as the current solution in ILS. LOCALSEARCH is applied in order to generate some possible neighborhood solutions and pick a better one, if any. We then update the current solution S_0. The updated solution is treated as the best found solution so far S^*.

We continue with applying PERTURBATION to S_0. LOCALSEARCH is then applied after a perturbation. If the current solution S_0 is better than S^*, we update the best found solution so far S^*. This part is related to the ACCEPTANCECRITERION component of ILS.

We include the intensification strategy in our ILS. This is the main difference of the standard ILS and our ILS. The idea of the intensification strategy is as follows. If S^* is not updated for a certain number of iterations, ((NOIMPR+1) MOD THRESHOLD1 = 0), we restart the search from the best found solution,

S^*. Finally, the entire algorithm will be run within the computational budget, TIMELIMIT.

Algorithm 4 ILS (N, M)

```
S_0 ← CONSTRUCTION(N, M)
S_0 ← LOCALSEARCH(S_0, N*, N', M)
S* ← S_0
NOIMPR ← 0
while TIMELIMIT has not been reached do
    S_0 ← PERTURBATION(S_0, N*, N', M)
    S_0 ← LOCALSEARCH(S_0, N*, N', M)
    if S_0 better than S* then
        S* ← S_0
        NOIMPR ← 0
    else
        NOIMPR ← NOIMPR + 1
    end if
    if (NOIMPR+1) MOD THRESHOLD1 = 0 then
        S_0 ← S*
    end if
end while
return S*
```

Two components that play important roles for improving the performance of ILS are PERTURBATION and LOCALSEARCH. ILS escapes from local optima by applying PERTURBATION to the current local minimum. LOCALSEARCH is applied to the current solution in order to generate neighborhoods. In the following subsections, we provide the descriptions of PERTURBATION and LOCALSEARCH.

3.2.1. Perturbation PERTURBATION is applied to S_0 in order to escape from local optima. Two different steps implemented are EXCHANGEPATH and SHAKE. If the number of iterations without improvement, NOIMPR, is larger than THRESHOLD2 and (NOIMPR + 1) Mod THRESHOLD3 is equal to 0, EXCHANGEPATH is executed; otherwise, SHAKE is selected. THRESHOLD2 and THRESHOLD3 are constant parameters.

By implementing EXCHANGEPATH, we restructure the current solution in order to provide opportunities for operators of LOCALSEARCH. In this step, we change the order of the paths in the solution by swapping two adjacent paths every time. The MOVE operator of LOCALSEARCH, explained in Section 3.2.2, is applied to paths in ascending order, e.g., from path one to the last path. Exchanging the order of the paths therefore creates extra opportunities for improvement by the MOVE operator.

The strategy of selecting two different paths is based on generating permutations by adjacent transposition method (Johnson, 1963). Each permutation is derived from its predecessor by a single interchange of two paths in adjacent positions. For example, with $\mu = 3$, the selected permutation is run once in this order: 1 **2 3**, **1 3** 2, 3 **1 2**, **3 2** 1, 2 **3 1** and **2 1** 3. Bold numbers represent two adjacent paths need to be swapped. Figure 1 provides an example of the EXCHANGEPATH step. If the selected permutation is 1 **2 3**, all nodes from path 2 are exchanged to path 3 and vice versa (Figure 1a, b).

The SHAKE step is adopted from the one proposed by Vansteenwegen *et al* (2009) with some modifications. 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 in a particular path. If we reach the last scheduled node, the process will continue to the first node after the starting node. Figure 2 illustrates the example of the SHAKE step with CONS equals to 2 and POST equals to 3. Two nodes are removed starting from position 3 of each path. For path 1, since we have reached the last node, node 4, we continue to remove node 1.

Both CONS and POST are initially set to one. After each SHAKE step, POST is increased by CONS. CONS is also increased by one after a fixed number of consecutive iterations, e.g., two iterations. If POST is greater than the number of scheduled

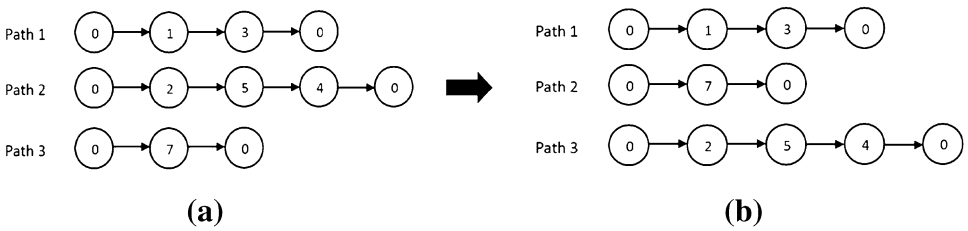

Figure 1 Example of EXCHANGEPATH. **a** Before EXCHANGEPATH, **b** After EXCHANGEPATH.

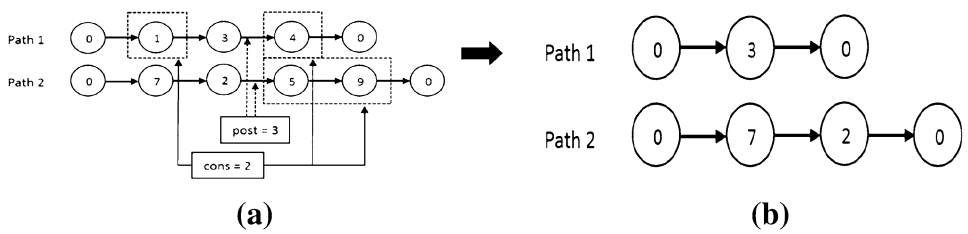

Figure 2 Example of SHAKE. **a** Before SHAKE, **b** After SHAKE.

Table 1 LocalSearch operations

Operation	*Description*
Swap1	Exchange two nodes within one path
Swap2	Exchange two nodes within two paths
2-Opt	Reverse 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

nodes of the shortest path, Post is subtracted with the size of the shortest 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.

This is different from the one in Vansteenwegen *et al* (2009) where Cons is always increased by one for each iteration and it would be set to one if it equals to $\frac{v}{3\times\mu}$. 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 by using Algorithm 3. This is repeated until $F = \emptyset$.

3.2.2. Local search In LocalSearch, we run six different operations consecutively, as shown in Table 1. The first four operators (Swap1, Swap2, 2-Opt and Move) restructure the current solution by increasing the remaining travel time. The remaining travel time refers to the difference between T^{max} and the time in which the path arrives to the end node. The order of the four operators is decided after conducting some preliminary experiments. Two other operators, Insert and Replace, contribute to improve the quality of the solution. The main reason why we put the first four operators first is that they could provide opportunities for Insert and Replace to make improvements. When $\mu = 1$, only Swap1, 2-Opt, Insert and Replace are considered.

Swap1 is defined by swapping two scheduled nodes within one particular path with the lowest remaining travel time. We examine all possible combinations of swapping two different nodes. Swap1 is executed if it increases the remaining travel time of the path.

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, after exchanging the nodes, is increased. Both Swap1 and Swap2 would be terminated if we cannot find two nodes to be swapped.

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 improving move is found.

Move is performed by reallocating one node from one path to another path. It is started from the first scheduled node n^* from the first path m^*. We try to insert node n^* to another path. 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 the selected node is moved successfully or the last scheduled node of the last path μ is examined.

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. This will continue until there is no possible replacement.

3.3. SAILS

The outline of SAILS is presented in Algorithm 5. The SA algorithm requires three parameters T_0, α and InnerLoop. T_0 refers to the initial temperature. α is a coefficient used to control the speed of the cooling schedule ($0 < \alpha < 1$). 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 would be decreased after InnerLoop iterations by using the following formula: $Temp = Temp \times \alpha$. SAILS consists of three components of ILS: Perturbation, LocalSearch and AcceptanceCriterion that have been described in Section 3.2.

Algorithm 5 SAILS (N, M)

```
S0 ← Construction(N, M)
S* ← S0
S' ← S0
Temp ← T0
NoImpr ← 0
while TimeLimit has not been reached do
    InnerLoop = 0
    while InnerLoop < MaxInnerLoop do
        S0 ← Perturbation(S0, N*, N', M)
        S0 ← LocalSearch(S0, N*, N', M)
        δ ← obj value of S0 - obj value of S'
        if δ > 0 then
            S' ← S0
            if S0 is better than S* then
                S* ← S0
                NoImpr ← 0
            else
                NoImpr ← NoImpr + 1
            end if
        else
            r ← rand[0, 1]
            if r < exp(δ/Temp) then
                S' ← S0
            else
                S0 ← S'
            end if
            NoImpr ← NoImpr + 1
        end if
        InnerLoop ← InnerLoop + 1
    end while
    Temp ← Temp × α
    if NoImpr > Limit then
        S0 ← S*
        S' ← S0
        NoImpr ← 0
    end if
end while
return S*
```

At a particular value of temperature *Temp*, 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 the objective function values of the solutions S_0 and S', denoted as δ. If δ is greater than 0, which implies that the objective function value is improved, S' is replaced by S_0. If S_0 also improves S^*, S^* is then replaced by S_0. We follow the same step of SA in accepting a worse solution, as shown in Algorithm 5.

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. For each iteration, if there is no improvement of S^*, we increase the number of no improvement NoImpr by one. If the number of no improvement NoImpr reaches a threshold 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.

3.4. Parameter tuning

We implement a sequential experimental approach which is grounded on the Design of Experiment (DOE) methodology for screening algorithm parameters (Gunawan and Lau, 2011). We briefly explain the idea of this approach.

Given a set of parameters *PR*, it is assumed that the initial range value of each parameter $par \in PR$ is known and bounded by a numerical interval $[LB_{par}, UB_{par}]$. We apply a $2^{|PR|}$ factorial design to screen and rank the parameters. A complete design requires $rep \times 2^{|PR|}$ observations where *rep* represents the number of replicates for one particular set of parameter values. The purpose of screening is to determine which parameters are statistically significant. By doing so, the number of parameters that require tuning is reduced.

The following example provides an illustration of the screening phase. Suppose there are two different parameters, par_1 and par_2. The 2^2 factorial design consists of four experimental units where each unit is run *rep* times:

- Set par_1 at LB_{par_1} and par_2 at LB_{par_2}
- Set par_1 at LB_{par_1} and par_2 at UB_{par_2}
- Set par_1 at UB_{par_1} and par_2 at LB_{par_2}
- Set par_1 at UB_{par_1} and par_2 at UB_{par_2}

A factorial experiment is then analyzed by using analysis of variance (ANOVA) to evaluate the main effect for a particular parameter. Further diagnosis related to normality, constant variance, time-dependent effects, and model significance are conducted. The test of significance of the main effect of the parameters with a significance level ($\alpha = 5\%$) is conducted for determining the importance of parameters. The important parameters are ranked by comparing the absolute values of the main effects of those parameters. Each unimportant (non-significant) parameter is then set to a constant value. The details of the statistical test can be found in Gunawan and Lau (2011) and Montgomery (2005).

4. Computational experiments

In this section, we first present the benchmark TOPTW instances and the state-of-the-art algorithms for comparison purpose. In Section 4.2, we describe how we determine and set the parameter values for the proposed algorithms. Comprehensive results are analyzed in Section 4.3.

4.1. Benchmark instances and approach comparison

The benchmark OPTW instances are initially proposed by Righini and Salani (2009). 58 problem instances are generated from Solomon's instances (Solomon, 1987), and 10 instances are adapted from Cordeau's instances (Cordeau *et al*, 1997). Montemanni and Gambardella (2009) develop another set of 37 instances for the OPTW. The TOPTW instances are designed by extending the OPTW instances with different values of paths: $\mu = 2$, 3 and 4. These instances are classified as the "INST-M" category (Hu and Lim, 2014).

Subsequently, Vansteenwegen *et al* (2009) add more instances based on instances of Solomon (1987) and Cordeau *et al* (1997). Those instances are considered more difficult instances, with the number of paths μ set to the number of vehicles. Due to the specific setting on the number of provided vehicles, the optimal solutions for these instances are known and are equal to the total score collected from all customers. Hu and Lim (2014) include these instances into the "OPT" category. Table 2 summarizes the benchmark TOPTW instances into three groups. All benchmark instances can be downloaded from http://www.mech.kuleuven.be/en/cib/op.

To evaluate the performance of our proposed algorithms, we consider the state-of-the-art algorithms, as listed in Table 3. The last two rows refer to our proposed algorithms, ILS and SAILS. In order to ensure fair comparisons due to different experimental environments, we use the same approach used by Hu and Lim (2014), namely the *SuperPi* benchmark. This method adjusts the computational time to the speed of the computers used in other approaches. 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. Therefore, the computational times shown in this paper have been adjusted accordingly.

We propose two different scenarios for conducting experiments. Each scenario is related to a different purpose that would be explained below. For the first scenario, we focus on the quality of the solution rather than the computational time. We notice that among the state-of-the-art algorithms, only ACS uses the computational budget, while the rest uses the number of iterations. The computational budget required by ACS to solve each instance is 1 h, which is considered large. For this scenario, we decide to run experiments with three different settings (computational budgets): 3600 s, 35% $\times$ 3600 s and 10% $\times$ 3600 s. The main purpose is to show that our proposed algorithms are able to perform well with shorter computational times.

The computational budgets per run for each instance using our processor are then set to 100% $\times$ 0.22 $\times$ 3600 s ($\approx$792 s), 35% $\times$ 0.22 $\times$ 3600 s ($\approx$ 277 s and 10% $\times$ 0.22 $\times$ 3600 s ($\approx$ 79.2 s), respectively. Our algorithms are then named as ILS^{100}, ILS^{35}, ILS^{10}, SAILS^{100}, SAILS^{35} and SAILS^{10}, respectively.

In the second scenario, we conduct experiments in which both ILS and SAILS are set to the same computational time as the other algorithms, e.g., I3CH (Hu and Lim, 2014) and VNS (Tricoire *et al*, 2010) so that our approach can be compared on the same base. The main purpose is to show that our algorithms are able to improve the solutions of the state-of-the-art algorithms on average.

It has been proven that I3CH outperforms other algorithms, such as IterILS, SSA and GVNS (Hu and Lim, 2014). We also select the computational time of VNS (Tricoire *et al*, 2010) since the computational time of VNS is much longer than the one of I3CH. For our proposed algorithms, each instance is executed in 10 runs with different random seeds. ACS is executed in 5 runs, whereas VNS, GVNS and GRILS are also executed 10 runs. IterILS, SSA and I3CH are only executed once.

4.2. Parameter tuning

Our proposed algorithm ILS consists of 4 parameters: f, THRESHOLD1, THRESHOLD2 and THRESHOLD3, as listed in Table 4. In this section, we present details of our parameter tuning experiments. We follow the same scenario used by Hu and Lim (2014) for selecting some training instances. The chosen instances are "c203," "c207," "pr02," "pr07," "pr12," "pr16," "r102," "r105," "rc107" and "rc204" with $\mu = 4$. Hu and Lim (2014) carried out experiments to tune some parameters by setting other parameters to constant values.

We implement the concept of Design of Experiment (DOE) as described in Section 3.4. In our problem, the DOE result is shown in Figure 3. Two statistically significant parameters with p value <5% are f and THRESHOLD1. These two parameters are considered as important parameters. By referring to the absolute effect values, the direction of adjustment for each parameter can be determined. For example, the most important parameter, f, has the highest absolute effect value of -0.984. Since our objective is to maximize the total collected score, we should set f to the lower range so we decide to adjust the range to [1, 5].

A similar idea is applied to the second highest important parameter, THRESHOLD1. On the other hand, for nonsignificant parameters (e.g., THRESHOLD2 and THRESHOLD3), we set to a

Table 2 Benchmark instances

Reference	*Name*	*Instance set*	*Number of instances*	*Number of nodes v*	*Number of paths μ*
Righini and Salani (2009)	Solomon	c100, r100, rc100	29 × 4	100	1–4
	Cordeau	pr01–pr10	10 × 4	48–288	
Montemanni and Gambardella (2009)	Solomon	c200, r200, rc200	27 × 4	100	1–4
	Cordeau	pr11–pr20	10 × 4	48–288	
Vansteenwegen *et al* (2009)	Solomon	c100, r100, rc100	29 × 1	100	3–20
		c200, r200, rc200	27 × 1	100	
	Cordeau	pr01–pr10	10 × 1	48–288	

Table 3 Estimation of single-thread performance

Algorithm	*Abbreviation*	*Experimental environment*	*Estimate of single-thread performance*
Iterated Local Search	IterILS (Vansteenwegen *et al*, 2009)	Intel Core 2 with 2.5 GHz CPU	0.92
Variable Neighborhood Search	VNS (Tricoire *et al*, 2010)	Intel Pentium 4 with 2.4 GHz CPU, 4 GB RAM	0.39
Ant Colony System	ACS (Montemanni and Gambardella, 2009; Montemanni *et al*, 2011)	Dual AMD Opteron 250 with 2.4 GHz CPU, 4 GB RAM	0.39
Slow Simulated Annealing	SSA (Lin and Yu, 2012)	Intel Core 2 with 2.5 GHz CPU	0.92
Granular Variable Neighborhood Search	GVNS (Labadie *et al*, 2012)	Intel Pentium (R) IV with 3 GHz CPU	0.39
Hybridized Greedy Randomized Iterated Local Search	GRILS (Souffriau *et al*, 2013)	Intel Xeon with 2.5 GHz CPU, 4 GB RAM	0.39
Iterative Three-Component Heuristic	I3CH (Hu and Lim, 2014)	Intel Xeon E5430 with 2.66 GHz CPU, 8 GB RAM	1.16
Iterated Local Search	ILS	Intel (R) Core(TM) i5 with 3.2 GHz CPU, 12 GB RAM	1
Hybridized Simulated Annealing—Iterated Local Search	SAILS	Intel (R) Core(TM) i5 with 3.2 GHz CPU, 12 GB RAM	1

Table 4 Parameter values of ILS

Parameter par	*Initial range* $[LB_{par}, UB_{par}]$	*Final range* $[LB_{par}, UB_{par}]$
f	[1, 10]	[1, 5]
THRESHOLD1	[1, 10]	[5, 10]
THRESHOLD2	[10, 20]	20
THRESHOLD3	[1, 3]	3

Estimated Effects and Coefficients for Obj (coded units)

Term	Effect	Coef	SE Coef	T	p
Constant		4.610	0.03312	139.19	0.000
f	-0.984	-0.492	0.03312	-14.85	0.000
Threshold1	0.492	0.246	0.03312	7.43	0.000
Threshold2	0.050	0.025	0.03312	0.75	0.453
Threshold3	0.992	0.496	0.03312	14.98	0.720

Figure 3 Design of Experiments result for ILS.

constant value by referring to the sign of the effect value. Both are considered as unimportant parameters. For instance, THRESHOLD2 has a positive effect; therefore, we set to its higher value, which is 20. The final range or the final value for each parameter can be referred to the last column of Table 4. Since our main concern is to screen the parameters, we ignore the effect of interaction among parameters (Montgomery, 2005).

Our main attention now is on important parameters, f and THRESHOLD1. We select some integer values within their final ranges and re-run ILS. In our design, the experiment is repeated six times with $f \in \{1, 3, 5\}$ and THRESHOLD1 $\in \{5, 10\}$. For each run, the percentage gap between the solution value achieved by ILS and the best known solution is computed. Table 5 summarizes the average gap for each possible combination of f and THRESHOLD1. Based on the preliminary testing, the following parameter values have the best performance within a reasonable computational time: f = 5, THRESHOLD1 = 10, THRESHOLD2 = 20 and THRESHOLD3 = 3.

Table 5 Parameter tuning on f and THRESHOLD1

	$f = 1$	$f = 3$	$f = 5$
THRESHOLD1 = 5	2.58	2.33	2.76
THRESHOLD1 = 10	3.18	2.45	1.99

For the second proposed algorithm, SAILS, we adopt the following parameter values from ILS: f = 5, THRESHOLD2 = 20 and THRESHOLD3 = 3. Other parameter values: α, T_0 and MAXINNERLOOP, are determined by implementing the DOE to the same set of training instances. Figure 4 shows the experimental result of the DOE.

From Figure 3, parameter MAXINNERLOOP seems not significant with a p value >5%. We then set its value to its lower value, which is 50. Another parameter, LIMIT, depends on the values of MAXINNERLOOP and a constant value c, using the formula: $\lceil$ LIMIT = $c \times$ MAXINNERLOOP $\rceil$. In this paper, we use $c = 0.05$. Other two parameters, α and T_0, are statistically significant with p value <5%. By referring to the positive sign of their effect values, we conclude that both algorithms perform better if we focus on the range with higher values. For example, the initial range for T_0 is [100, 1000] which is adjusted to [500, 1000]. Table 6 presents the initial and final

Estimated Effects and Coefficients for Obj (coded units)

Term	Effect	Coef	SE Coef	T	p
Constant		4.610	0.02259	204.07	0.000
Alpha	0.884	0.442	0.02259	19.57	0.000
T0	0.495	0.248	0.02259	10.98	0.000
MaxInnerLoop	-0.092	-0.046	0.02259	-2.03	0.276

Figure 4 Design of Experiments result for SAILS.

Table 6 Parameter values of SAILS

Parameter par	*Initial range* $[LB_{par}, UB_{par}]$	*Final range* $[LB_{par}, UB_{par}]$
α	[0.1, 0.9]	[0.5, 0.9]
T_0	[100, 1000]	[500, 1000]
MaxInnerLoop	[50, 100]	50
Limit	–	$\lceil 0.05 \times 50 = 3 \rceil$

ranges for significant parameters and the final values for nonsignificant parameters.

We continue the parameter tuning experiment by generating different values for each significant parameter in order to determine the final value of α and T_0. The values of parameters considered are as follows: $\alpha \in \{0.5, 0.75, 0.9\}$ and $T_0 \in \{500, 600, 700, 800, 900, 1000\}$. All possible combinations are run in order to obtain the final parameter values, as shown in Table 7. We conclude that the following values: α = 0.75, T_0 = 1000, MaxInnerLoop = 50 and Limit = 3, perform best.

4.3. Computational results

We report a comprehensive analysis of the results from two different scenarios. The last subsection consolidates all new best known solutions.

4.3.1. First scenario Table 8 summarizes the comparison of our proposed algorithms with ACS (Montemanni and Gambardella, 2009; Montemanni *et al*, 2011) for all instances from "INST-M." The *Numb* column provides the number of instances in a set. The *BG* and *AG* columns report the best and average percentage gaps with the best known solutions (BKs). All BKs are collected from the state-of-the-art algorithms listed in Table 3. A negative value indicates an improvement of the current best known solutions. As described in Section 4.1, both ILS and SAILS are run with three different computational budgets (settings), while ACS is run for 3600 s ($\approx$792 s with our processor).

We observe that the average percentage gaps (AG values) over all "INS-M" instances achieved by ILS^{100} and $SAILS^{100}$ are 0.81 and 0.80%, respectively. They outperform ACS at least by 1.52%. As computational budget becomes less, ILS and SAILS become less effective in solving the TOPTW instances. However, both ILS and SAILS are still better than ACS in terms of the average percentage gap values (*AG* values).

We also record the best solution found from 10 runs and calculate the percentage gap with the best known solution (*BG* values). On average, ILS and SAILS with different computational budgets still outperform the ACS. The worst average is around 0.73% (by ILS^{10}) which is still better than the one of the ACS which is 1.69%. We also conclude that SAILS outperforms ILS in terms of *BG* and *AG* values in all different computational budgets.

In order to support the above-mentioned paragraph about the performance of SAILS, we summarize the average improvement of the initial solution generated by $SAILS^{35}$, as shown in Table 9. Take note that $SAILS^{10}$ and $SAILS^{100}$ show similar performance. $SAILS^{35}$ is able to improve the initial solution generated by the greedy construction heuristic between 0.60 and 19.41%. $SAILS^{35}$ performs best for μ = 1 where the average values of percentage of improvement vary from 9.51 to 19.41%. We observe that the higher the value of μ, the lower the average value is.

For other state-of-the-art algorithms, we provide Table 10 for comparison purpose. The comparison should be done carefully since some results are obtained after one run while others are based on multiple runs. Our algorithms use the computational budget mentioned in Section 4.1, while other state-of-the-art algorithms use the number of iterations. We compare the average results of multiple runs with the results of the single-run algorithms. The average results correspond to the average expected quality and computational time of the multiple-run algorithm when it is executed only once (Souffriau *et al*, 2013).

Both ILS and GRILS are suitable for problems in real time since both are solved very fast although the solution quality is worse than other algorithms. I3CH outperforms SSA and GVNS although it requires more computational times. Both ACS and VNS do not perform well since they also require more computational times but the solution quality is worse. In conclusion, I3CH is considered as the best algorithm among the state-of-the-art algorithms (Hu and Lim, 2014).

We first compare $SAILS^{100}$ and ILS^{100} against the state-of-the-art algorithms in terms of solution quality and computational time. Please note that, as discussed in Section 4.1, the computational time for $SAILS^{100}$ and ILS^{100} is 792 s per run. Both outperform the state-of-the-art algorithms, except I3CH, and require a higher computational time. $SAILS^{35}$, ILS^{35}, $SAILS^{10}$ and ILS^{35} are able to outperform VNS in terms of solution quality and computational time. On the other hand, they perform worse than SSA.

Table 7 Parameter tuning on α and T_0

	$T_0 = 500$	$T_0 = 600$	$T_0 = 700$	$T_0 = 800$	$T_0 = 900$	$T_0 = 1000$
$\alpha = 0.5$	2.20	2.43	2.51	2.23	1.91	1.94
$\alpha = 0.75$	2.18	2.52	2.86	2.22	2.71	1.76
$\alpha = 0.9$	2.84	2.39	2.05	2.49	3.56	2.41

Table 8 Overall comparison of ILS and SAILS to ACS on "INST-M"

Instance set	*Numb*	*ACS*		*ILS^{100}*		*ILS^{35}*		*ILS^{10}*		*SAILS^{100}*		*SAILS^{35}*		*SAILS^{10}*	
		BG	*AG*	*BG*	*AG*	*BG*	*AG*	*BG*	*AG*	*BG*	*AG*	*BG*	*AG*	*BG*	*AG*
$\mu = 1$															
Cordeau 1–10	10	1.06	1.22	0.24	0.36	0.34	0.74	0.37	0.76	0.25	0.73	0.44	0.93	0.34	0.84
Cordeau 11–20	10	11.13	11.87	1.21	1.53	1.33	2.12	1.56	2.16	1.27	1.82	1.14	2.28	1.52	2.45
Solomon 100	29	0.00	0.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.02	0.00	0.03
Solomon 200	27	1.38	2.07	0.08	0.40	0.04	0.71	0.27	0.91	−0.01	0.52	0.12	0.83	0.30	0.83
$\mu = 2$															
Cordeau 1–10	10	2.59	3.57	0.48	1.30	0.77	2.30	1.37	2.52	0.19	1.15	0.56	2.09	1.08	1.83
Cordeau 11–20	10	5.00	6.15	1.19	2.13	1.66	3.38	2.36	3.48	1.12	2.17	1.40	3.11	1.59	2.82
Solomon 100	29	0.17	0.62	−0.01	0.00	−0.01	0.07	−0.01	0.08	−0.01	0.04	−0.01	0.09	0.01	0.12
Solomon 200	27	2.46	3.18	0.23	0.83	0.47	1.39	0.91	1.65	0.37	0.93	0.46	1.40	0.68	1.45
$\mu = 3$															
Cordeau 1–10	10	2.96	4.21	1.12	2.29	2.13	3.59	1.79	3.17	1.03	2.17	1.26	2.85	1.89	3.40
Cordeau 11–20	10	5.40	7.24	1.76	2.94	2.07	3.59	2.41	3.98	0.92	2.24	2.02	3.39	1.83	2.99
Solomon 100	29	0.31	1.07	0.06	0.22	0.05	0.51	0.16	0.44	0.07	0.30	0.08	0.51	0.15	0.60
Solomon 200	27	0.50	0.86	0.14	0.30	0.21	0.52	0.34	0.71	0.08	0.25	0.16	0.43	0.21	0.49
$\mu = 4$															
Cordeau 1–10	10	2.76	3.42	2.34	3.48	2.33	4.00	3.00	4.45	1.67	2.79	2.08	3.58	2.49	3.69
Cordeau 11–20	10	5.53	6.34	2.14	3.47	2.91	4.42	3.43	4.55	1.85	3.12	2.05	3.97	2.71	3.97
Solomon 100	29	0.65	1.79	0.34	0.84	0.28	1.13	0.46	1.20	0.28	0.86	0.27	1.23	0.53	1.17
Solomon 200	27	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Grand Mean		1.69	2.33	0.42	0.81	0.54	1.19	0.73	1.28	0.34	0.80	0.46	1.14	0.61	1.15

Table 9 The improvement of the greedy construction heuristic result by SAILS (in %)

Instance set	$\mu = 1$	$\mu = 2$	$\mu = 3$	$\mu = 4$
Cordeau 1–10	18.57	18.86	15.70	12.80
Cordeau 11–20	19.41	18.63	14.26	11.43
Solomon 100	12.42	13.08	13.20	13.30
Solomon 200	9.51	8.29	4.56	0.60
Grand Mean	12.87	12.59	10.50	8.17

Table 11 reports the overall comparisons on the instances in the "OPT" category. Take note that ACS, VNS and GRILS are not tested on those instances. SAILS^{100} and ILS^{100} outperform IterILS, SSA and GVNS, but use more computational time. The I3CH achieves the smallest average gap over all "OPT" instances.

4.3.2. Second scenario For the purpose of ensuring a fair comparison among our proposed algorithms, SAILS and ILS, and other algorithms, I3CH and VNS, we now conduct experiments in which the ILS and SAILS were set to the same computational time as the other approaches, adjusted by their computers' speed (Table 3).

In this scenario, we follow the same scenario which is used in Hu and Lim (2014). The experiments are carried out on the "INST-M" instances. The computational time and the number of runs for each instance were set to the ones required by I3CH. We also include the computational time of VNS for comparison purpose since I3CH has not been compared with the computational time of VNS before Hu and Lim (2014). To differentiate with other scenarios, our proposed algorithms are named as ILS$^{\text{I3CH}}$, ILS$^{\text{VNS}}$, SAILS$^{\text{I3CH}}$ and SAILS$^{\text{VNS}}$, respectively.

Table 12 summarizes the performance of ILS$^{\text{I3CH}}$ and SAILS$^{\text{I3CH}}$ in solving all instances from "INST-M." ILS$^{\text{I3CH}}$ outperforms I3CH on instance sets with μ = 1. SAILS$^{\text{I3CH}}$ performs well on the instance sets with μ = 1 and 2. I3CH, ILS$^{\text{I3CH}}$ and SAILS$^{\text{I3CH}}$ achieve average percentage gaps of 0.69, 1.20 and 0.61%, respectively.

When we set the computational times of ILS$^{\text{VNS}}$ and SAILS$^{\text{VNS}}$ to the one of VNS, we conclude that both ILS$^{\text{VNS}}$ and SAILS$^{\text{VNS}}$ outperform VNS. However, we found that

Table 10 Overall comparison of ILS and SAILS to the state-of-the-art algorithms on "INST-M"

Instance set	*Numb*	*IterILS*		*VNS*		*SSA*		*GVNS*		*GRILS*		*I3CH*		ILS^{100}	ILS^{35}	ILS^{10}	$SAILS^{100}$	$SAILS^{35}$	$SAILS^{10}$
		AG	*AT*	*AG*	*AT*	*AG*	*AT*	*AG*	*AT*	*AG*	*AT*	*AG*	*AT*	*AG*	*AG*	*AG*	*AG*	*AG*	*AG*
$\mu = 1$																			
Cordeau 1–10	10	4.70	1.7	1.10	753.6	0.98	103.2	1.62	4.8	3.74	3.8	1.07	126.8	0.36	0.74	0.76	0.73	0.93	0.84
Cordeau 11–20	10	8.40	1.8	3.38	958.9	3.71	149.3	4.26	9.3	7.96	4.7	4.28	151.5	1.53	2.12	2.16	1.82	2.28	2.45
Solomon 100	29	1.82	0.2	0.07	78.3	0.05	20.5	2.46	25.7	0.27	1.3	0.69	31.1	0.00	0.00	0.00	0.03	0.02	0.03
Solomon 200	27	2.87	1.5	0.89	786.4	0.85	40.8	2.88	16.8	2.41	7.7	1.34	153.7	0.40	0.71	0.91	0.52	0.83	0.83
$\mu = 2$																			
Cordeau 1–10	10	6.00	4.4	3.88	481.1	2.45	159.9	1.79	15.1	6.76	10.8	1.11	287.4	1.30	2.30	2.52	1.15	2.09	1.83
Cordeau 11–20	10	7.10	4.8	3.63	567.2	3.88	185.4	2.18	31.8	11.28	12.4	2.70	354.3	2.13	3.38	3.48	2.17	3.11	2.82
Solomon 100	29	1.88	0.8	1.07	63.1	0.15	31.7	1.74	28.5	1.73	3.7	0.49	80.6	0.00	0.07	0.08	0.04	0.09	0.12
Solomon 200	27	3.06	2.4	1.00	746.0	0.96	70.7	1.46	7.6	3.74	12.6	0.47	539.5	0.83	1.39	1.65	0.93	1.40	1.45
$\mu = 3$																			
Cordeau 1–10	10	6.40	8.5	3.63	433.8	2.34	181.1	1.13	33.2	7.58	16.9	0.36	493.2	2.29	3.59	3.17	2.17	2.85	3.40
Cordeau 11–20	10	8.50	8.9	3.35	474.5	3.81	231.5	1.80	58.2	11.21	18.6	1.11	578.1	2.94	3.59	3.98	2.24	3.39	2.99
Solomon 100	29	1.92	1.4	1.22	63.1	0.44	42.2	1.87	35.2	3.19	5.8	0.20	158.0	0.22	0.51	0.44	0.30	0.51	0.60
Solomon 200	27	1.10	1.6	0.14	295.5	0.48	48.0	0.27	2.8	1.65	13.0	0.02	103.8	0.30	0.52	0.71	0.25	0.43	0.49
$\mu = 4$																			
Cordeau 1–10	10	6.50	13.0	3.57	369.6	2.23	235.0	1.60	49.1	8.10	23.3	0.36	659.0	3.48	4.00	4.45	2.79	3.58	3.69
Cordeau 11–20	10	6.90	12.6	3.49	374.1	3.95	261.1	2.81	89.8	10.34	25.1	0.45	847.6	3.47	4.42	4.55	3.12	3.97	3.97
Solomon 100	29	2.83	2.2	1.66	61.2	0.58	53.6	1.95	33.4	5.06	8.2	0.16	232.1	0.84	1.13	1.20	0.86	1.23	1.17
Solomon 200	27	0.00	0.9	0.00	129.4	0.00	37.2	0.00	0.2	0.00	13.1	0.00	0.2	0.00	0.00	0.00	0.00	0.00	0.00
Grand Mean		3.20	2.8	1.42	344.3	1.09	81.2	1.74	24.8	3.87	9.7	0.69	233.7	0.81	1.19	1.28	0.80	1.14	1.15

Table 11 Overall comparison of ILS and SAILS to the state-of-the-art algorithms on "OPT"

Instance set	*Numb*	*IterILS*		*SSA*		*GVNS*		*I3CH*		ILS^{100}	ILS^{35}	ILS^{10}	$SAILS^{100}$	$SAILS^{35}$	$SAILS^{10}$
		AG	*AT*	*AG*	*AT*	*AG*	*AT*	*AG*	*AT*	*AG*	*AG*	*AG*	*AG*	*AG*	*AG*
Cordeau 1–10	10	2.32	28.0	1.04	520.3	1.25	19.8	0.78	380.0	0.96	1.11	1.23	0.92	1.05	1.05
Solomon 100	29	1.80	3.0	0.59	83.4	1.14	11.4	0.03	458.1	0.47	0.18	0.19	0.39	0.12	0.12
Solomon 200	27	0.39	1.4	0.08	44.5	0.12	1.3	0.04	147.9	0.02	1.68	1.84	0.09	1.53	1.43
Grand Mean		1.30	6.1	0.45	133.7	0.74	8.5	0.15	319.4	0.36	0.82	0.90	0.35	0.75	0.73

Table 12 Overall comparison of ILS and SAILS to I3CH and VNS on "INST-M"

Instance set	*Numb*	*I3CH* *AG*	ILS^{I3CH} *AG*	$SAILS^{I3CH}$ *AG*	*AT*	*VNS* *AG*	ILS^{VNS} *AG*	$SAILS^{VNS}$ *AG*	*AT*
$\mu = 1$									
Cordeau 1–10	10	1.07	0.66	0.24	126.8	1.10	0.57	0.66	753.6
Cordeau 11–20	10	4.28	2.28	1.83	151.5	3.38	1.70	1.84	958.9
Solomon 100	29	0.69	0.00	0.00	31.1	0.07	0.00	0.03	78.3
Solomon 200	27	1.34	0.88	0.32	153.7	0.89	0.47	0.52	786.4
$\mu = 2$									
Cordeau 1–10	10	1.11	2.32	0.87	287.4	3.88	1.90	1.43	481.1
Cordeau 11–20	10	2.70	2.78	1.83	354.3	3.63	2.73	2.38	567.2
Solomon 100	29	0.49	0.12	0.06	80.6	1.07	0.01	0.12	63.1
Solomon 200	27	0.47	1.18	0.55	539.5	1.00	1.12	0.90	746.0
$\mu = 3$									
Cordeau 1–10	10	0.36	2.79	1.46	493.2	3.63	2.78	2.26	433.8
Cordeau 11–20	10	1.11	3.21	1.79	578.1	3.35	3.20	2.49	474.5
Solomon 100	29	0.20	0.45	0.25	158.0	1.22	0.28	0.63	63.1
Solomon 200	27	0.02	1.65	0.66	103.8	0.14	0.33	0.33	295.5
$\mu = 4$									
Cordeau 1–10	10	0.36	3.48	1.98	659.0	3.57	3.61	3.30	369.6
Cordeau 11–20	10	0.45	3.53	1.89	847.6	3.49	3.88	3.24	374.1
Solomon 100	29	0.16	1.11	0.48	232.1	1.66	0.85	1.27	61.2
Solomon 200	27	0.00	0.18	0.06	0.2	0.00	0.00	0.00	129.4
Grand Mean		0.69	1.20	0.61	233.7	1.42	0.95	0.93	344.3

$SAILS^{VNS}$ displays better performance on 8 of the 16 instances sets. ILS^{VNS} performs well for $\mu = 1$.

4.3.3. New best known solution During the different scenarios explained in previous subsections, we have found 50 new best known solutions that can serve as benchmarks for future studies. Forty-two percent of them are benchmark instances with $\mu = 2$. Around 30% of new *BK*s are from Cordeau et al.'s datasets which is harder to solve compared against Solomon's datasets (Duque *et al*, 2015).

Table 13 presents the new best known results including their respective algorithms. In total, we have ten different settings for both ILS and SAILS. Some of new best known solutions are identical with those found in our earlier works (Gunawan *et al*, 2015a, b). We found that two benchmark instances (r107 with $\mu = 2$ and r104 with $\mu = 3$) can be solved with almost all settings. The details of new solutions can be downloaded from https://unicen.smu.edu.sg/oplib-orienteering-problem-library.

We summarize the performance of the algorithms in Table 14. We observe that most of new best known solutions are discovered by using the computational times of 100% of ACS, I3CH and VNS. SAILS dominates ILS in all different settings. The computational time of VNS is higher than the one of I3CH; therefore, ILS^{VNS} outperforms ILS^{I3CH} by 14.2%. On the other hand, $SAILS^{VNS}$ outperforms $SAILS^{I3CH}$ by 4.4%. In conclusion, SAILS is able to discover more new best known solutions than ILS does.

Table 13 New best solution values discovered by ILS and SAILS

Instance	μ	*Old BK*	*New BK*	*Algorithm*	*Instance*	μ	*Old BK*	*New BK*	*Algorithm*
r203	1	1021	1028	ILS^{VNS}	rc201[a]	2	1384	1385	$SAILS^{100}$, $SAILS^{I3CH}$
r204[a]	1	1086	1093	ILS^{35}, $SAILS^{I3CH}$, ILS^{VNS}	rc202	2	1509	1512	ILS^{I3CH}
r206[a]	1	1029	1032	ILS^{100}, $SAILS^{100}$, $SAILS^{35}$, $SAILS^{VNS}$	rc206	2	1546	1552	ILS^{35}, ILS^{VNS}
r207	1	1072	1076	ILS^{100}, $SAILS^{VNS}$	rc207	2	1587	1599	$SAILS^{I3CH}$
r208	1	1112	1118	$SAILS^{10}$, $SAILS^{VNS}$	rc208	2	1691	1692	$SAILS^{100}$
r209	1	950	961	ILS^{100}, $SAILS^{VNS}$	pr13[a]	2	832	843	$SAILS^{35}$, $SAILS^{100}$, $SAILS^{I3CH}$
r210	1	987	1000	$SAILS^{100}$	pr15	2	1219	1220	ILS^{35}
r211	1	1046	1051	ILS^{100}, ILS^{VNS}, $SAILS^{VNS}$	pr18	2	938	953	ILS^{VNS}
rc202[a]	1	936	938	ILS^{100}, $SAILS^{100}$, $SAILS^{I3CH}$, $SAILS^{VNS}$	pr20	2	1232	1241	$SAILS^{VNS}$
rc206[a]	1	895	899	ILS^{100}, $SAILS^{100}$, ILS^{I3CH}, $SAILS^{I3CH}$, ILS^{VNS}, $SAILS^{VNS}$	r104[a]	3	777	778	ILS^{10}, ILS^{35}, ILS^{100}, ILS^{VNS}, $SAILS^{10}$, $SAILS^{35}$, $SAILS^{100}$, $SAILS^{I3CH}$, $SAILS^{VNS}$
rc208[a]	1	1053	1057	$SAILS^{100}$, ILS^{100}, ILS^{35}, ILS^{100}, $SAILS^{VNS}$	rc104[a]	3	834	835	ILS^{35}, ILS^{100}, ILS^{I3CH}, ILS^{VNS}, $SAILS^{35}$, $SAILS^{I3CH}$
pr13	1	466	467	ILS^{100}, $SAILS^{100}$, $SAILS^{VNS}$	pr02	3	942	943	$SAILS^{35}$, $SAILS^{100}$, $SAILS^{VNS}$
pr15	1	707	708	ILS^{100}, ILS^{I3CH}, ILS^{VNS}	r201	3	1441	1442	$SAILS^{I3CH}$
r107[a]	2	536	538	ILS^{35}, ILS^{I3CH}, ILS^{VNS}, $SAILS^{10}$, $SAILS^{35}$, $SAILS^{100}$, $SAILS^{I3CH}$, $SAILS^{VNS}$	pr13	3	1145	1152	$SAILS^{I3CH}$
pr04[a]	2	925	926	ILS^{100}, ILS^{I3CH}, ILS^{VNS}, $SAILS^{35}$	pr15	3	1654	1659	$SAILS^{100}$
pr09[a]	2	905	909	$SAILS^{35}$, $SAILS^{I3CH}$	pr18	3	1281	1282	$SAILS^{100}$
pr10	2	1129	1134	$SAILS^{100}$	pr20	3	1684	1690	$SAILS^{I3CH}$
c204[a]	2	1480	1490	ILS^{100}, ILS^{VNS}, $SAILS^{35}$, $SAILS^{100}$	r104	4	972	975	$SAILS^{I3CH}$
r201	2	1254	1256	$SAILS^{VNS}$	r108	4	994	995	ILS^{VNS}, $SAILS^{VNS}$
r202	2	1347	1348	$SAILS^{VNS}$	r112	4	971	974	$SAILS^{100}$
r203	2	1416	1418	$SAILS^{I3CH}$	rc103[a]	4	974	975	ILS^{100}, ILS^{I3CH}, ILS^{VNS}, $SAILS^{35}$, $SAILS^{VNS}$
r205	2	1380	1386	$SAILS^{I3CH}$	rc104	4	1064	1065	ILS^{10}
r206	2	1440	1450	$SAILS^{I3CH}$	rc107	4	980	987	ILS^{10}, ILS^{100}, ILS^{I3CH}, ILS^{VNS}, $SAILS^{100}$, $SAILS^{VNS}$
r209	2	1405	1414	ILS^{VNS}	pr14	4	1670	1674	ILS^{I3CH}
r210	2	1423	1427	$SAILS^{VNS}$	pr19	4	1750	1760	$SAILS^{VNS}$

[a] Same results with either Gunawan *et al* (2015a) or Gunawan *et al* (2015b).

Table 14 Tabulation of new best solution values discovered by ILS and SAILS

Algorithm	μ	*Base algorithm*					*Total*	*Percentage*
		ACS-10%	*ACS-35%*	*ACS-100%*	*I3CH*	*VNS*		
ILS	1	0	2	8	2	5	17	34.69
	2	0	3	2	3	6	14	28.57
	3	1	2	2	1	2	8	16.33
	4	2	0	2	3	3	10	20.41
Total		3	7	14	9	16	49	100
Percentage		6.1	14.3	28.6	18.4	32.6	100	
SAILS	1	1	1	6	3	9	20	29.41
	2	1	5	6	8	5	25	36.76
	3	1	3	4	5	2	15	22.06
	4	0	1	2	1	4	8	11.76
Total		3	10	18	17	20	68	100
Percentage		4.4	14.7	26.5	25.0	29.4	100	

5. Conclusion

We propose two algorithms, Iterated Local Search (ILS) and a hybridization of Simulated Annealing and Iterated Local Search (SAILS), for solving the TOPTW. We also implement a factorial design experiment to find good parameter values for both algorithms.

The proposed algorithms are run by using two different scenarios. The first scenario concerns with the solution quality; therefore, ILS and SAILS are run with longer computational times. The second scenario is mainly tailored for comparison purposes with the state-of-the-art algorithms. Both well-tuned ILS and SAILS are run by using the computational times of several state-of-the-art algorithms.

All scenarios are applied to benchmark TOPTW instances. The computational results show that both algorithms outperform all but one state-of-the-art algorithm and perform comparable to the I3CH algorithm. They are able to discover 50 new best known solutions. The new results can serve as benchmarks for future studies.

Several areas of future work can be considered. Using different operators for constructing neighborhood solutions, such as inserting or removing two or more nodes simultaneously, would be an interesting area. The recent trend of hybridizing exact algorithms, such as Lagrangian Relaxation, and metaheuristics seems also a promising future research area for solving the TOPTW more efficiently. The idea of parameter control can be considered for future work in order to further improve the quality of the solutions. Since the OP and its variants, such as the Arc Orienteering Problem and Multiconstraint Team Orienteering Problem with Multiple Time Windows, have attracted more attention in recent years, ILS and SAILS can be potentially tailored to solve these as well.

Acknowledgements—This research project is funded by National Research Foundation Singapore under its Corp Lab @ University scheme and Fujitsu Limited.

References

Adenso-Diaz B and Laguna M (2006). Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research* **54**(1):99–114.

Cordeau J, Grendreau M and Laporte G (1997). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks* **30**(2):105–119.

Cura T (2014). An artificial bee colony algorithm approach for the team orienteering problem with time windows. *Computers and Industrial Engineering* **74**:270–290.

Duque D, Lozano L and Medaglia A (2015). Solving the orienteering problem with time windows via the pulse framework. *Computers and Operations Research* **54**:168–176.

Eiben AE and Smit SK (2012). Evolutionary algorithm parameters and methods to tune them. In Hamadi Y, Monfroy E and Saubion F (Eds.) *Autonomous Search*. Springer: Berlin, pp. 15–36.

Eiben AE, Michalewicz Z, Schoenauer M and Smith JE (2007). Parameter control in evolutionary algorithms. In Lobo FG, Lima CF and Michalewicz Z (Eds.) *Parameter Setting in Evolutionary Algorithms*. Springer: Berlin, pp. 19–46.

Eiben AE, Hinterding R and Michalewicz Z (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* **3**(2):124–141.

Gendreau M, Laporte G and Semet F (1998). A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research* **106**(2–3):539–545.

Goldberg DE (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley: Reading.

Golden B, Levy L and Vohra R (1987). The orienteering problem. *Naval Research Logistics* **34**(3):307–318.

Gunawan A, Lau HC and Lindawati (2011). Fine-tuning algorithm parameters using the design of experiments approach. In Coello CAC (Ed.) *Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION5)*, Vol. 6683 of Lecture Notes in Computer Science. Springer: Berlin, pp. 278–292.

Gunawan A, Lau HC and Lu K (2015a). SAILS: hybrid algorithm for the team orienteering problem with time windows. In *Proceedings of 7th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2015)*, Prague, Czech Republic, pp. 276–295.

Gunawan A, Lau HC and Lu K (2015b). An iterated local search algorithm for solving the orienteering problem with time windows. In Ochoa G and Chicano F (Eds.) *Proceedings of the 15th*

European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoStar 2015), Vol. 9026 of Lecture Notes in Computer Science, 8–10 April 2015, Copenhagen, Denmark. Springer: Berlin, pp. 61–73.

Gunawan A, Lau HC and Vansteenwegen P (2016). Orienteering problem: a survey of recent variants, solution approaches and applications. *European Journal of Operational Research* **255**(2):315–332.

Hu Q and Lim A (2014). An iterative three-component heuristic for the team orienteering problem with time windows. *European Journal of Operational Research* **232**(2):276–286.

Hutter F, Hoos, Leyton-Brown K and Stützle T (2009). ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research* **36**:267–306.

Johnson SM (1963). Generation of permutations by adjacent transposition. *Mathematics of Computation*.**1717**(83):282–285.

Labadie N, Mansini R, Melechovskỳ J and Calvo RW (2011). Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. *Journal of Heuristics* **17**(6):729–753.

Labadie N, Mansini R, Melechovskỳ R and Calvo J (2012). The team orienteering problem with time windows: an LP-based granular variable neighborhood search. *European Journal of Operational Research* **220**(1):15–27.

Lin S-W and Yu VF (2012). A simulated annealing heuristic for the team orienteering problem with time windows. *European Journal of Operational Research* **217**(1):94–107.

Lourenço H, Martin O and Stützle T (2003). Iterated local search. In Glover FW and Kochenberger GA (Eds.) *Handbook of Metaheuristics*. Springer: Berlin, pp. 320–353.

Montemanni R and Gambardella LM (2009). Ant colony system for team orienteering problem with time windows. *Foundations of Computing and Decision Sciences* **34**(4):287–306.

Montemanni R, Weyland D and Gambardella LM (2011). 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)*, Kota Kinabalu, Malaysia, pp. 381–384.

Montgomery D (2005). *Design and Analysis of Expeirments*, 6th Edn. Wiley: London.

Righini G and Salani M (2009). 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.

Solomon M (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research* **35**(2):254–265.

Souffriau W, Vansteenwegen P, Berghe G and van Oudheusden D (2013). The multiconstraint team orienteering problem with multple time windows. *Transportation Science* **47**(1):53–63.

Stützle T, López-Ibáñez M, Pellegrini P, Maur M, Montes de Oca M, Birattari M and Dorigo M (2012). Parameter adaptation in ant colony optimization. In Hamadi Y, Monfroy E and Saubion F (Eds.) *Autonomous Search*. Springer: Berlin, pp. 191–215.

Tricoire F, Romauch M, Doerner KF and Hartl RF (2010). Heuristics for the multi-period orienteering problem with multiple time windows. *Computers and Operations Research* **37**(2):351–367.

Tsiligirides T (1984). Heuristic methods applied to orienteering. *Journal of the Operational Research Society* **35**(9):797–809.

Vansteenwegen P, Souffriau W, Vanden Berghe G and Van Oudheusden D (2009). Iterated local search for the team orienteering problem with time windows. *Computers and Operations Research* **36**(12):3281–3290.

Vansteenwegen P, Souffriau W and Van Oudheusden D (2011). The orienteering problem: a survey. *European Journal of Operational Research* **209**(1):1–10.