An Iterated Local Search Algorithm for Solving the Orienteering Problem with Time Windows

Aldy Gunawan(✉), Hoong Chuin Lau, and Kun Lu

School of Information Systems, Singapore Management University,
80 Stamford Road, Singapore 178902, Singapore
{aldygunawan,hclau,kunlu}@smu.edu.sg

Abstract. The Orienteering Problem with Time Windows (OPTW) is a variant of the Orienteering Problem (OP). Given a set of nodes including their scores, service times and time windows, the goal is to maximize the total of scores collected by a particular route considering a predefined time window during which the service has to start. We propose an Iterated Local Search (ILS) algorithm to solve the OPTW, which is based on several LocalSearch operations, such as swap, 2-opt, insert and replace. We also implement the combination between AcceptanceCriterion and Perturbation mechanisms to control the balance between diversification and intensification of the search. In Perturbation, Shake strategy is introduced. The computational results obtained by our proposed algorithm are compared against optimal solutions or best known solution values obtained by state-of-the-art algorithms. We show experimentally that our proposed algorithm is effective on well-known benchmark instances available in the literature. It is also able to improve the best known solution of some benchmark instances.

Keywords: Orienteering problem · Time windows · Iterated local search

1 Introduction

The Orienteering Problem (OP) was first introduced by Tsiligirides in [1]. The main objective is to select a subset of nodes and define the sequence of selected nodes so that the total collected score is maximized while the maximum total travel time (time budget given) is not exceeded. The recent survey of real-life applications of the OP and its variants is presented by Vansteenwegen et al. in [2].

The Orienteering Problem with Time Windows (OPTW) is a variant of the OP with time window constraints that arise in situations where nodes/locations have to be visited within a predefined time window specified by an earliest and a latest time into which the service has to start [3]. An early arrival to a particular node leads to waiting times, while a late arrival causes infeasibility. Given a set of nodes, each one with a score, the goal is to maximize the total of collected score by a particular route subject to a time budget and time window constraints. The OPTW can be extended to the Team Orienteering Problem with Time Windows (TOPTW) when the number of route considered is more than one route [4].

In this paper, an Iterated Local Search (ILS) algorithm is proposed to solve the OPTW. The algorithm starts with generating an initial solution, which is constructed by inserting nodes subsequently into a route. A set of feasible candidate nodes to be inserted is created and the selection of a node to be inserted is based on roulette-wheel selection [5]. The initial solution is further improved by ILS. We consider components of ILS: LocalSearch, Perturbation, and AcceptanceCriterion. The LocalSearch procedure involves several operations, such as swap, 2-opt, replace and insert.

In Sect. 2, we present the problem description and literature review of the OPTW. Section 3 is devoted to the proposed algorithm. Section 4 provides the computation results together with the analysis of the results. Section 5 concludes the paper and summarizes directions for further research.

2 Problem Description and Literature Review

The OPTW is defined as follows. Let us consider a set of nodes $N = \{1, 2, \cdots, n\}$ where each node $i \in N$ is associated with a score u_i and a service time T_i. The starting and end nodes are assumed to be nodes 1 and n, respectively; therefore, u_1, T_1, u_n, T_n are set to 0. The non-negative travel time between nodes i and j is represented as t_{ij}.

Each node i associates with a time window $[e_i, l_i]$, where e_i and l_i are the earliest and latest times allowed for starting service at node i. We assume that $e_1 = e_n = 0$ and $l_1 = l_n = T^{max}$. For mathematical formulations for the OPTW, we refer to [4,6]. The objective of the OPTW is to maximize the total collected score when visiting a subset of the nodes with respect to following constraints, as listed below:

- The route starts and ends at nodes 1 and n, respectively.
- Each node $i \in N$ is visited at most once.
- The service start time at node i is within a time window $[e_i, l_i]$.
- The time budget is limited by T^{max}.

The initial investigation of the OPTW has been presented by Kantor and Rosenwein in [6]. Since OPTW falls into NP-hard, a heuristic based on the tree heuristic was proposed. The experiments showed that the tree heuristic outperforms the insertion heuristic. Righini and Salani [7] proposed an exact optimization algorithm for the OPTW. The algorithm is based on dynamic programming with decremental state space relaxation. The result shows that there is no domination between the proposed algorithm and the other dynamic programming proposed by Boland et al. in [8] for solving benchmark instances. A new heuristic technique for the initialization of the critical vertex set has also been proposed in order to reduce the number of iterations and the amount of computing time required.

The Tourist Trip Design Problems (TTDP) can be formulated as the OPTW and the TOPTW [4]. A simple, fast and effective Iterated Local Search (ILS) was proposed to solve both problems. The proposed algorithm only combines insertion and shaking operations to generate the solutions. New data set was designed to analyse the performance of the proposed algorithm and to be used as a benchmark for further research. Montemanni and Gambardella [9] proposed a heuristic approach based on Ant Colony System (ACS). It includes a local search procedure by exchanging two subchains of nodes of the giant tour. Experimental results on benchmark instances have proven the effectiveness of the algorithm. For other related works with further improvement of benchmark instances' results, we can refer to [10,11].

A Simulated Annealing-based heuristic was proposed by Lin and Yu in [12] for solving both OPTW and TOPTW. Two different versions, fast SA (FSA) and slow SA (SSA), were developed in order to tailor two different scenarios. The former is mainly for the applications that need quick responses while the latter is more concerned about the quality of the solutions. The SSA heuristic is able to find 33 new best solutions. A heuristic based on a Variable Neighborhood Search (VNS) was proposed in order to tackle the OPTW and the TOPTW [3]. The idea of granularity that includes time constraints and profits in addition to pure distances is introduced. The proposed algorithm has been able to improve 25 best known solution values.

Hu and Lim [13] proposed an iterative framework which is based on three components: a local search procedure, a Simulated Annealing procedure and Route Recombination. The first two components are used to explore the solution space and discover a set of routes. The last component which focuses on combining the routes to identify high quality solutions is included. 35 new best solutions are found and more than 83 % of instances with optimal solutions can be found.

3 Proposed Algorithm

This section presents the description of our proposed algorithm. The algorithm is started by generating an initial feasible solution using a greedy construction heuristic. The initial solution is further improved by Iterated Local Search (ILS). Components of ILS: LocalSearch, Perturbation and AcceptanceCriterion, are taken into consideration. The differences between our ILS and ILS proposed by Vansteenwegen et al. [4] would be described below.

3.1 Greedy Construction Heuristic

The greedy construction heuristic builds an initial solution from scratch. The idea is to insert a node subsequently to a route until no more feasible insertion can be found. A node insertion is feasible if all scheduled nodes after the insertion still satisfy their respective time windows and the total spent time does not exceed T^{max}.

Let N' and N^* be the sets of unscheduled and scheduled nodes respectively ($N' \cup N^* = N$). The greedy construction heuristic is outlined in Algorithm 1. N^* is initialized by nodes 1 and n, while N' consists of the remaining unscheduled nodes. S_0 represents the current feasible solution obtained so far, represented as a vector $(1 \times |N^*|)$.

Let F be the set of feasible candidate nodes to be inserted. F is generated iteratively in order to store feasible candidate unscheduled nodes to be inserted. The idea of generating F is summarized in Algorithm 2. P is denoted as the set of all positions of a route. We examine all possibilities of inserting an unscheduled node in position $p \in P$. Each element in F, which represents a feasible insertion of node n in position p of a route, is represented as $\langle n, p \rangle$. For each possible insertion, we calculate the benefit of insertion $ratio_{n,p}$ by using Eq. (1). $\Delta_{n,p}$ represents the difference between the total time spent before and after the insertion of node n in position p. For example, if the total time spent before the insertion of node n in position p is 700 time units and the total time spent after the insertion is increased to 720 time units, the value of $\Delta_{n,p}$ is $720 - 700 = 20$ time units. All elements would be sorted in descending order based on $ratio_{n,p}$ values and we only keep f elements in F and remove the rest.

Algorithm 1. CONSTRUCTION (N)

```
$N^* \leftarrow$ nodes 1 and $n$
$N' \leftarrow N \setminus$ nodes 1 and $n$
Initialize $S_0 \leftarrow N^*$
$F \leftarrow$ UPDATEF$(N')$
while $F \neq \emptyset$ do
    $\langle n^*, p^* \rangle \leftarrow$ SELECT$(F)$
    $S_0 \leftarrow \langle n^*, p^* \rangle$
    $N' \leftarrow N' \setminus \{n^*\}$
    $N^* \leftarrow N^* \cup \{n^*\}$
    $F \leftarrow$ UPDATEF$(N')$
end while
return $S_0$
```

Algorithm 2. UPDATEF (N')

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

Algorithm 3. Select (F)

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

$$ratio_{n,p} = \left(\frac{u_n^2}{\Delta_{n,p}}\right) \ \forall n \in N', p \in P \tag{1}$$

If $F \neq \emptyset$, Algorithm 3 is run in order to select which $\langle n^*, p^*\rangle$ to be inserted. Each $\langle n, p\rangle$ corresponds to probability value $prob_{n,p}$. The probability is calculated by Eq. (2):

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

Instead of always selecting an inserted node with the highest value of $ratio_{n,p}$ [4], our approach is different. Selecting $\langle n^*, p^*\rangle$ from F is based on roulette-wheel selection [5]. This method assumes that the probability of selection a particular $\langle n, p\rangle$ is proportional to the benefit of its insertion, $ratio_{n,p}$. A random number $U \sim [0, 1]$ is generated. The accumulative of probability values, $AccumProb$, is initially set to 0. We select a particular $\langle n^*, p^*\rangle$ and update the value of $AccumProb$ iteratively. This loop will be terminated when $(U \leq AccumProb)$ and the corresponding $\langle n^*, p^*\rangle$ is selected. S_0, N' and N^* will then be updated. The greedy construction heuristic is terminated when there is no further feasible insertion $(F = \emptyset)$.

Due to the time windows, the score of a node insertion is more relevant compared against the time consumption of an insertion. By removing the square, the obtained results are worse [4]. Therefore, the square of score is then applied in Eq. (1). Another main reason is by using the square of score, we increase the probability of selecting a particular node with a higher ratio (Eq. (2)) since the main objective is to maximize the collected score.

3.2 Iterated Local Search

Given the initial solution S_0 generated by the greedy construction heuristic, we propose an Iterated Local Search (ILS) algorithm to further improve the quality of S_0. Three components of ILS: Perturbation, LocalSearch and AcceptanceCriterion, are taken into consideration. Let S^* be the best found solution so far. The outline of ILS is presented in Algorithm 4.

Algorithm 4. ILS (N)

```
$S_0 \leftarrow$ Construction($N$)
$S_0 \leftarrow$ LocalSearch($S_0, N^*, N'$)
$S^* \leftarrow S_0$
NoImpr $\leftarrow 0$
while TimeLimit has not been reached do
  $S_0 \leftarrow$ Perturbation($S_0, N^*, N'$)
  $S_0 \leftarrow$ LocalSearch($S_0, N^*, N'$)
  if $S_0$ better than $S^*$ then
    $S^* \leftarrow S_0$
    NoImpr $\leftarrow 0$
  else
    NoImpr $\leftarrow$ NoImpr $+ 1$
  end if
  if (NoImpr+1) Mod Threshold1 $= 0$ then
    $S_0 \leftarrow S^*$
  end if
end while
return $S^*$
```

Perturbation is applied to S_0 in order to escape from local optima. In this paper, we implement Shake operation. The Shake operation is adopted from [4] with some modifications. During Shake operation, one or more nodes will be removed, which depends on two integer values. The first one indicates how many consecutive nodes to be removed (denoted as *cons*), while the second one indicates the first position of the removed nodes (denoted as *post*). If the last scheduled node is reached and there are still some nodes to be removed, we go back to the start node and include nodes after the start node. Both *cons* and *post* are initially set to 1. After each shake operation, *post* is increased by *cons*. *cons* remains the same for a fixed number of consecutive iterations, e.g. 2 iterations and it is then increased by 1 subsequently. In [4], *cons* will always be increased by 1 for each iteration. If *post* is greater than the size of the smallest route, *post* is subtracted with the size of the smallest route in order to determine the new position. If *cons* is greater than the size of the largest route, or S^* is updated, *cons* is reset to one. Again, this differs from [4] where *cons* is set to 1 if *cons* is equal to $n/3$. After removing *cons* nodes, we generate F based on Algorithm 2 and select a node to be inserted using Algorithm 3. N' and N^* are then updated accordingly. This is repeated until $F = \emptyset$.

ILS proposed by Vansteenwegen et al. [4] only considers Insert and Shake operations for generating the solutions. In our LocalSearch, we consider four different operations that would be explained as follows. Swap is applied by exchanging two scheduled nodes within a route. All possible combinations of selecting two different scheduled nodes are examined. Swap is executed if it increases the remaining travel time and there is no constraint violation. 2-opt is started by selecting two positions of two scheduled nodes. The sequence of scheduled nodes is reversed as long as there is no constraint violation and there is an improvement of the remaining travel time. This would be terminated if no further improvement in terms of total of remaining travel time.

Insert is applied in order to insert one unscheduled node to a route. It is started by generating F based on Algorithm 2 and selecting node $i \in N'$ to be inserted using Algorithm 3. This is repeated until $F = \emptyset$. The idea is the same with the one introduced in the greedy construction heuristic. The last operation, Replace, tries to replace one scheduled node $i \in N^*$ with one unscheduled node $j \in N'$ with the highest score u_j. We then check each position p and examine whether selected node j can replace the node in position p. The feasibility of the solution and the improvement of total score are considered in this operation. Once this operation is successful, we continue with the next unscheduled node j with the second highest score u_j. Otherwise, the operation would be terminated.

Acceptance criterion is described as follows. The new local optimum solution is always accepted as the initial solution for the next run of local search. However, if there is no improvement of S^* obtained for a certain number of iterations, ((NoImpr+1) Mod Threshold1 = 0), the search is continued by applying an intensification strategy. This strategy focuses the search once again starting from the best found solution, S^* in order to improve the probability of hitting the global optimum. Finally, the entire algorithm will be run within the computational budget, TimeLimit.

4 Computational Experiments

4.1 Benchmarks and Experimental Setup

The test problems for the OPTW in the literature were initially proposed by Righini and Salani in [7], which are generated from Solomon's [14] and Cordeau et al.'s instances [15]. 48 Solomon's instances contain 100 nodes of series (c100, r100 and rc100). Cordeau et al.'s instances consists of 10 instances with different number of nodes, varying from 48 to 288 nodes (pr01–pr10). Those instances were designed for the Vehicle Routing Problem with Time Windows (VRPTW) and the Multi Depot Periodic VRPTW respectively. In this paper, we only concern with benchmark instances with the number of route = 1, which related to the OPTW problem. 37 additional instances were created [9]. 27 instances are converted from Solomon's dataset (c200, r200 and rc200) and 10 instances are converted from Cordeau et al.'s dataset (pr11–pr20).

Table 1. Estimation of single-thread performance [13].

Algorithm	Experimental environment	*SuperPi*	Estimate of single-thread performance
IterLS	Intel Core 2 with 2.5 gigahertz CPU, 3.45 gigabytes RAM	18.6	0.53
ACS*	Dual AMD Opteron 250 2.4 gigahertz CPU, 4 gigabytes RAM	Unknown	0.22
SSA	Intel Core 2 CPU, 2.5 gigahertz	18.6	0.53
GVNS	Intel Pentium (R) IV, 3 gigahertz CPU	44.3	0.22
I3CH	Intel Xeon E5430 CPU clocked at 2.66 gigahertz, 8 gigabytes RAM	14.7	0.67
ILS	Intel Core i7-4770 with 3.4 GHz processor, 16 gigabytes RAM	9.8	1

The experiments were carried out on a personal computer Intel Core i7 - 4770 with 3.4 GHz processor and 16 GB RAM. Vansteenwegen et al. [4] discussed the difficulty of solving the instances by a commercial solver (CPLEX). ILS was tested by performing 10 runs with different random seeds per each instance. The performances of the proposed ILS are compared to the state-of-the-art methods: Iterated Local Search (IterLS) [4], Ant Colony System (ACS) [9], Enhanced Ant Colony System (Enhanced ACS) [11], Slow Simulated Annealing (SSA) [12], Granular Variable Neighborhood Search (GVNS) [3] and Iterative Three-Component Heuristic (I3CH) [13]. Enhanced ACS [11] has empirically outperformed the original ACS [9]. In this paper, we refer to the results of both, whichever is better and denote them as ACS*.

For each instance, ACS* was executed in 5 runs whereas ILS and GVNS were executed 10 times. On the other hand, IterLS, SSA and I3CH were only executed once and reported only the best found solutions. For comparison purpose, the solutions of our ILS were compared against the best known solutions (BKs) of IterLS, ACS*, SSA, GVNS and I3CH. In order to ensure the fair comparisons, we refer to the same approach [13] to compare the speed of the computers used in obtaining the solutions, as shown in Table 1. *SuperPi* is a single-threaded program that computes the first 1 million digits of π of a particular processor. The comparability of processors used by ACS* and GVNS is shown in [10] since the *SuperPi* for ACS* is not available.

By setting the performance of our machine to be 1, we then estimated the single-thread performance of other processors by multiplying with the single-thread performance estimation (last column of Table 1). For the details, please refer to [13]. Among all algorithms, only ACS* used one hour of the computational time for each instance, while the rest use the number of iterations. In this paper, we are more concerned with solution quality, we then used ACS* as our reference. Instead of using 100 % of ACS*'s computational time, we only

Table 2. New best known solution values found by ILS.

Instance	Old BK	New BK	Instance	Old BK	New BK	Instance	Old BK	New BK
r203	1021	1026	r209	950	956	rc206	895	899
r204	1086	1093	r211	1046	1049	rc208	1053	1057
r208	1112	1113	rc202	936	938			

use 35 % of it. The computational time for each instance is then set to $35\,\% \times 0.22 \times 3600 = 272$ s using our processor. Based on the preliminary testing, the following parameter values seem to have the best performance within a reasonable computational time: $f = 5$ and THRESHOLD1 = 10.

4.2 Experimental Results

Table 2 reports the new best known solutions (BKs) obtained by ILS. We discovered 8 new best known solution values for Solomon's instances. Partial results obtained by ILS on benchmark instances are reported in Table 3. We only report the results of Solomon's instances due to space constraints. The detailed results can be found online at http://centres.smu.edu.sg/larc/Orienteering-Problem-Library.

Table 3 consists of two identical structure parts. The first column contains the instance name, the second column reports the best known solution value BK from references. The following three columns show maximum, average and minimum solution values obtained by our ILS. The "BG (%)" column provides the best relative percentage deviation, which refers to the percentage gap between BK and the best solution obtained by ILS. "AG (%)" provides the average relative percentage deviation, which refers to the percentage gap between BK and the average solution obtained by ILS. Finally, the last three columns show maximum, average and minimum computational times required to obtain the best found.

Take note that the optimal value is indicated in *italic* and the new BK obtained by ILS is indicated in **bold**. There are still 27 and 12 instances of Solomon's and Cordeau et al.'s instances where the optimal values are unknown. ILS is able to obtain 41 out 56 (≈73.2 %) best known solutions (BKs) on Solomon's instances. It also improved the best known solutions of 8 out 27 instances (≈30.0 %). For Cordeau et al.'s instances, 12 out 20 (≈60.0 %) BKs can be found by ILS.

Table 4 summarizes the results of IterLS, ACS*, GVNS, SSA, I3CH and our ILS results. The *numb* column provides the number of instances in a particular instance set. The table reports the average of AG for each instance set ($\overline{AG}$(%)). However, IterLS, SSA and I3CH only reported their best known solution obtained; therefore, we report the average of BG ($\overline{BG}$(%)) as well. The best known solutions (BKs) were collected from IterILS, ACS*, SSA, GVNS and I3CH results. The computational time ($\overline{CPU}$) for ACS* and ILS for one particular instance set reports the average of time spent to obtain the best

Table 3. Detailed results of ILS on Solomon's instances

Instance	*BK*	ILS			*BG*(%)	*AG*(%)	*CPU*(seconds)			Instance	*BK*	ILS			*BG*(%)	*AG*(%)	*CPU*(seconds)		
		Max	Avg	Min			Max	Avg	Min			Max	Avg	Min			Max	Avg	Min
c101	*320*	320	320.0	320	0.0	0.0	0.5	0.2	0.0	c201	870	870	870.0	870	0.0	0.0	157.8	36.7	1.6
c102	*360*	360	360.0	360	0.0	0.0	0.8	0.3	0.0	c202	930	930	930.0	930	0.0	0.0	185.0	59.0	20.8
c103	*400*	400	400.0	400	0.0	0.0	0.4	0.2	0.1	c203	960	960	960.0	960	0.0	0.0	247.9	137.2	20.8
c104	*420*	420	420.0	420	0.0	0.0	1.0	0.4	0.1	c204	980	980	974.0	970	0.0	0.6	246.5	217.6	104.9
c105	*340*	340	340.0	340	0.0	0.0	1.3	0.4	0.1	c205	910	910	908.0	900	0.0	0.2	249.3	56.2	11.8
c106	*340*	340	340.0	340	0.0	0.0	0.9	0.5	0.1	c206	930	930	927.0	920	0.0	0.3	219.8	111.5	5.8
c107	*370*	370	370.0	370	0.0	0.0	0.4	0.1	0.0	c207	930	930	930.0	930	0.0	0.0	141.5	68.1	12.5
c108	*370*	370	370.0	370	0.0	0.0	0.9	0.5	0.1	c208	950	950	950.0	950	0.0	0.0	68.3	33.3	4.9
c109	*380*	380	380.0	380	0.0	0.0	25.4	6.8	0.6										
r101	*198*	198	198.0	198	0.0	0.0	0.4	0.1	0.0	r201	797	794	788.7	784	0.4	1.0	243.4	133.7	34.4
r102	*286*	286	286.0	286	0.0	0.0	0.5	0.2	0.0	r202	930	921	910.3	896	1.0	2.1	269.8	165.6	25.4
r103	*293*	293	293.0	293	0.0	0.0	3.9	1.4	0.2	r203	1021	**1026**	1011.3	996	-0.5	1.0	268.7	213.5	148.1
r104	*303*	303	303.0	303	0.0	0.0	6.2	1.5	0.1	r204	1086	**1093**	1082.8	1071	-0.6	0.3	266.9	171.0	56.6
r105	*247*	247	247.0	247	0.0	0.0	1.3	0.7	0.0	r205	953	953	948.4	942	0.0	0.5	253.2	169.9	28.8
r106	*293*	293	293.0	293	0.0	0.0	0.6	0.2	0.0	r206	1029	1022	1012.4	1002	0.7	1.6	246.4	126.5	34.2
r107	*299*	299	299.0	299	0.0	0.0	1.5	0.5	0.0	r207	1072	1067	1059.5	1049	0.5	1.2	248.3	174.0	77.1
r108	*308*	308	308.0	308	0.0	0.0	2.4	0.9	0.1	r208	1112	**1113**	1107.6	1100	-0.1	0.4	267.9	165.6	47.6
r109	*277*	277	277.0	277	0.0	0.0	0.4	0.2	0.0	r209	950	**956**	949.7	938	-0.6	0.0	231.6	145.8	76.3
r110	*284*	284	284.0	284	0.0	0.0	3.8	1.3	0.0	r210	987	978	970.8	962	0.9	1.6	263.0	171.8	14.5
r111	*297*	297	297.0	297	0.0	0.0	50.3	10.9	0.4	r211	1046	**1049**	1040.4	1025	-0.3	0.5	256.0	145.7	3.3
r112	*298*	298	298.0	298	0.0	0.0	10.6	3.3	0.0										
rc101	*219*	219	219.0	219	0.0	0.0	0.5	0.2	0.0	rc201	795	795	795.0	795	0.0	0.0	132.1	63.5	9.0
rc102	*266*	266	266.0	266	0.0	0.0	1.7	0.4	0.0	rc202	936	**938**	929.0	918	-0.2	0.7	262.1	156.2	31.1
rc103	*266*	266	266.0	266	0.0	0.0	9.6	2.0	0.1	rc203	1003	999	989.8	969	0.4	1.3	243.9	111.5	27.2
rc104	*301*	301	301.0	301	0.0	0.0	0.7	0.3	0.2	rc204	1140	1136	1131.3	1128	0.4	0.8	263.4	165.0	16.4
rc105	*244*	244	244.0	244	0.0	0.0	10.0	4.3	0.2	rc205	859	859	854.7	849	0.0	0.5	219.0	100.5	11.4
rc106	*252*	252	252.0	252	0.0	0.0	1.0	0.3	0.0	rc206	895	**899**	894.1	883	-0.4	0.1	255.9	152.0	52.1
rc107	*277*	277	277.0	277	0.0	0.0	0.9	0.3	0.0	rc207	983	983	952.1	941	0.0	3.1	252.9	129.9	14.7
rc108	*298*	298	298.0	298	0.0	0.0	0.2	0.1	0.0	rc208	1053	**1057**	1040.7	1020	-0.4	1.2	158.9	85.6	15.0

Table 4. Overall "Average" Comparison of ILS to the state-of-the-art methods.

Instance Set	*Numb*	IterLS		ACS*		SSA		GVNS		I3CH		ILS	
		$\overline{BG}$(%)	$\overline{CPU}$	$\overline{AG}$(%)	$\overline{CPU}$‡	$\overline{BG}$(%)	$\overline{CPU}$	$\overline{AG}$(%)	$\overline{CPU}$	$\overline{BG}$(%)	$\overline{CPU}$	$\overline{AG}$(%)	$\overline{CPU}$‡
c100	9	1.11	0.2	0.00	1.4	0.00	11.1	1.22	36.8	0.00	16.8	0.00	1.0
r100	12	1.90	0.1	0.24	84.8	0.11	12.3	2.68	6.5	0.56	19.1	0.00	1.8
rc100	8	2.92	0.1	0.00	31.7	0.00	11.7	3.51	2.2	1.66	17.0	0.00	1.0
c200	8	2.28	0.9	0.58	75.8	0.13	19.8	1.11	42.6	0.40	56.3	0.14	90.0
r200	11	2.90	0.9	3.17	344.4	1.30	24.1	3.38	7.5	1.05	117.4	0.93	162.1
rc200	8	3.43	0.9	2.04	341.7	0.96	26.5	3.96	3.5	2.68	79.6	0.97	120.5
pr01-10	10	4.74	0.9	1.22	359.8	0.98	59.1	1.62	2.7	1.07	72.7	0.74	50.4
pr11-20	10	9.56	1.0	11.87	196.4	3.71	85.6	4.26	5.4	4.28	86.8	2.12	97.9
Grand mean		3.64	0.6	2.49	183.9	0.94	31.9	2.73	12.6	1.44	59.1	0.63	65.6

‡ Average computation time to obtain the best found

found BK (in seconds) from all runs since the experiments were based on the computational time. On the other hand, IterLS, SSA, GVNS and I3CH report the average of the computational time for solving each instance (in seconds). The computational time of all approaches were adjusted according the their computer's speed as summarized in Table 1.

From Table 4, we observe that ILS can produce better solutions against those of ACS* and GVNS. The $\overline{AG}$'s grand mean of ILS is only 0.63 %, whereas those of ACS* and GVNS are 2.49 % and 0.94 %, respectively. In terms of the computational time required to obtain the best found, ILS is much faster than ACS*. ILS only requires 65.6 s while ACS* requires 183.9 s, on average. Comparing against SSA, ILS finds better solutions at the expense of more computational time. IterLS is the fastest algorithm but the reported grand mean of $\overline{BG}$ is the largest.

Table 5 reports the best solutions obtained for each algorithm. ILS is the best compared against other methods. The grand mean of $\overline{BG}$ is only 0.23 %. The $\overline{BG}$s of ILS ranges from −0.04 % to 1.33 %, while the ones of IterLS and I3CH have wider ranges from 1.11 % to 9.56 % and 0.00 % to 4.28 %, respectively. ILS

Table 5. Overall "Best" Comparison of ILS to the state-of-the-art methods.

Instance set	*Numb*	IterLS	ACS*	SSA	GVNS	I3CH	ILS
		$\overline{BG}$(%)	$\overline{BG}$(%)	$\overline{BG}$(%)	$\overline{BG}$(%)	$\overline{BG}$(%)	$\overline{BG}$(%)
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	0.11
rc200	8	3.43	1.23	0.96	2.53	2.68	−0.04
pr01–10	10	4.74	1.06	0.98	0.56	1.07	0.34
pr11–20	10	9.56	11.13	3.71	3.17	4.28	1.33
Grand mean		3.64	2.09	0.94	1.71	1.44	0.23

Table 6. Comparison with the same computational time.

Instance set	$Numb$	I3CH	ILS		$\overline{CPU}$(s)
		$\overline{BG}$(%)	$\overline{BG}$(%)	$\overline{AG}$(%)	
c100	9	0.00	0.00	0.00	16.8
r100	12	0.56	0.00	0.00	19.1
rc100	8	1.66	0.00	0.03	17.0
c200	8	0.40	0.00	0.29	56.3
r200	11	1.05	0.36	1.24	117.4
rc200	8	2.68	0.20	1.30	79.6
pr01–10	10	1.07	0.30	0.80	72.7
pr11–20	10	4.28	1.29	2.30	86.8
Grand mean		1.44	0.28	0.76	59.1

obtains best known solutions for all instances of the first four instances sets. For rc200 instance set, a negative value of $\overline{BG}$ represents the improvement of some BKs.

I3CH outperforms SSA when using the same computational time that had been adjusted by their computers' speed [13]. We also compare the performance of ILS againts I3CH. As shown in Table 6, we found that the $\overline{BG}$'s grand mean of ILS is 1.16 % better than that of I3CH. ILS is able to obtain 0.00 % of $\overline{BG}$ for 4 out of 8 instance sets. In terms of $\overline{AG}$, ILS can reduce the grand mean of $\overline{AG}$ by almost 50 % compared against the one of $\overline{BG}$ of I3CH.

5 Conclusion

In this paper, we study the Orienteering Problem with Time Windows (OPTW). An algorithm based on Iterated Local Search (ILS) is proposed to solve the problem. Computational results have shown that the proposed algorithm is an effective algorithm. The algorithm has been able to improve 8 best known solution values of benchmark instances.

Other various mechanisms of ILS could be investigated. For instance, using other construction heuristics and restarting the algorithm from a new initial solution. It would also be interesting to consider applying ILS to other variants of the OP, e.g. the Team Orienteering Problem with Time Windows (TOPTW), the Time Dependent Orienteering Problem (TDOP) and the Tourist Trip Design Problem (TTDP).

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. Tsiligirides, T.: Heuristic methods applied to orienteering. J. Oper. Res. Soc. **35**(9), 797–809 (1984)
2. Vansteenwegen, P., Souffriau, W., Van Oudheusden, D.: The orienteering problem: a survey. Eur. J. Oper. Res. **209**(1), 1–10 (2011)
3. Labadie, N., Mansini, R., Melechovskỳ, J., Calvo, R.W.: The team orienteering problem with time windows: an LP-based granular variable neighborhood search. Eur. J. Oper. Res. **220**(1), 15–27 (2012)
4. Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D.: Iterated local search for the team orienteering problem with time windows. Comput. Operat. Res. **36**(12), 3281–3290 (2009)
5. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading (1989)
6. Kantor, M.G., Rosenwein, M.B.: The orienteering problem with time windows. J. Oper. Res. Soc. **43**(6), 629–635 (1992)
7. Righini, G., Salani, M.: Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. Comput. Oper. Res. **36**(4), 1191–1203 (2009)
8. Boland, N., Dethridge, J., Dumitrescu, I.: Accelerated label setting algorithms for the elementary resource constrained shortest path. Oper. Res. Lett. **34**(1), 58–68 (2006)
9. Montemanni, R., Gambardella, L.M.: Ant colony system for team orienteering problem with time windows. Found. Comput. Decis. Sci. **34**(4), 287–306 (2009)
10. Labadie, N., Mansini, R., Melechovskỳ, J., Calvo, R.W.: Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. J. Heuristics **17**(6), 729–753 (2011)
11. 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 (2011)
12. Lin, S.W., Yu, V.F.: A simulated annealing heuristic for the team orienteering problem with time windows. Eur. J. Oper. Res. **217**(1), 94–107 (2012)
13. Hu, Q., Lim, A.: An iterative three-component heuristic for the team orienteering problem with time windows. Eur. J. Oper. Res. **232**(2), 276–286 (2014)
14. Solomon, M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Oper. Res. **35**(2), 254–265 (1987)
15. Cordeau, J.F., Grendreau, M., Laporte, G.: A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks **30**(2), 105–119 (1997)