Well-Tuned ILS for Extended 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

August, 2015
LARC-TR-01-15

LARC Technical Report Series: http://smu.edu/sg/centres/larc/larc-technical-reports-series

SMU
SINGAPORE MANAGEMENT
UNIVERSITY

Carnegie
Mellon
University

ABSTRACT

The Team Orienteering Problem with Time Windows (TOPTW) is the extension of the Team Orienteering Problem (TOP) by incorporating the time window constraints. 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 fixed number of routes considering a predefined time window during which the service has to start. In this paper, we propose a mathematical model for the TOPTW that incorporates more real-world constraints, such as different total travel time \ budgets, different start and end nodes for routes. This extension is motivated by a real-world application, namely the Tourist Trip Design Problem (TTDP). We then propose an Iterated Local Search (ILS) algorithm to solve both TOPTW and TTDP. While there have been multiple research works on ILS for TOP and variants, the challenge remains in determining appropriate parameter values to solve them. Here we apply Design of Experiments (DOE), namely factorial experiment design, to first screen and rank all the parameters thereby allowing us to focus on the parameter search space of the important parameters. We demonstrate that well-tuned ILS leads to improvements in terms of the quality of the solutions. More precisely, it is able to improve 31 best known solution values on benchmark TOPTW instances. Our proposed algorithm is also effective for solving real-world problem instances of the TTDP within a few seconds on a regular desktop machine.

Keywords: Orienteering Problem, Team Orienteering Problem with Time Windows, Tourist Trip Design Problem, Design of Experiment, Iterated Local Search

Well-Tuned ILS for Extended Team Orienteering Problem with Time Windows

Aldy Gunawan, Hoong Chuin Lau and Kun Lu

School of Information Systems, Singapore Management University, Singapore 178902

Abstract

The Team Orienteering Problem with Time Windows (TOPTW) is the extension of the Team Orienteering Problem (TOP) by incorporating the time window constraints. 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 fixed number of routes considering a predefined time window during which the service has to start. In this paper, we propose a mathematical model for the TOPTW that incorporates more real-world constraints, such as different total travel time budgets, different start and end nodes for routes. This extension is motivated by a real-world application, namely the Tourist Trip Design Problem (TTDP). We then propose an Iterated Local Search (ILS) algorithm to solve both TOPTW and TTDP. While there have been multiple research works on ILS for TOP and variants, the challenge remains in determining appropriate parameter values to solve them. Here we apply Design of Experiments (DOE), namely factorial experiment design, to first screen and rank all the parameters thereby allowing us to focus on the parameter search space of the important parameters. We demonstrate that well-tuned ILS leads to improvements in terms of the quality of the solutions. More precisely, it is able to improve 31 best known solution values on benchmark TOPTW instances. Our proposed algorithm is also effective for solving real-world problem instances of the TTDP within a few seconds on a regular desktop machine.

Keywords: Orienteering Problem, Team Orienteering Problem with Time Windows, Tourist Trip Design Problem, Design of Experiment, Iterated Local Search

1. INTRODUCTION

The Orienteering Problem (OP) was first introduced by Tsiligirides (1984). It originates from the sport game of orienteering (Chao et al., 1996). The main objective is to determine a path that consists of a subset of nodes and define the sequence of the selected nodes so that the total collected score is maximized and the maximum time travelled (a given time budget) is not exceeded. Vansteenwegen et al. (2011a) provide an excellent review of the OP and its variants.

The Team Orienteering Problem with Time Windows (TOPTW) extends the Team Orienteering Problem (TOP) by incorporating time window constraints. Time window constraints arise in some problems where nodes have to be visited within a predefined time window specified by an earliest and a latest time into which the service has to start (Labadie et al., 2012). An early arrival to a particular node leads to waiting times, while a late arrival causes infeasibility. The TOPTW can be reduced to the Orienteering Problem with Time Windows (OPTW) when only one route is considered.

This paper focuses on an extended version of the TOPTW. Given a set of nodes with a score each, the goal is to maximize the total collected score by visiting a set of nodes, taking into account a service time and a time window. A mathematical model for the extension of TOPTW is introduced. In addition, this model addresses the main drawbacks of the TOPTW model: in the standard TOPTW, the start and end nodes are assumed to be the same node. In the proposed mathematical model, we allow different start and end nodes for each route, and each route may have a different maximum travel time. This extension can be linked to the Tourist Trip Design Problem (TTDP) (Vansteenwegen et al., 2009). In terms of application, each route can be interpreted as a day trip in the context of the Tourist Trip Design Problem (TTDP).

In this paper, we consider the use of Iterated Local Search (ILS) to solve the problem. An initial solution is constructed by inserting nodes subsequently into one of the routes, based on roulette-wheel selection method (Goldberg, 1989). Various components of ILS LOCALSEARCH, PERTURBATION, and ACCEPTANCECRITERION will be discussed. Associated with ILS is a set of parameter values that need to be tuned. Even though it is known that good parameter values have 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 arbitrarily without explanation, or conveniently choose parameter values that have been reported in previous studies. In response to the need for a principled approach to find good parameter settings, we propose the

application of a factorial design experiment to first screen and rank all the parameters based on their importance, as described in Gunawan et al. (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.

In summary, the contribution of the paper is fourfold:

- We introduce an extension mathematical model of the Team Orienteering Problem with Time Windows that addresses the main drawbacks of the classical TOPTW model.
- We propose an Iterated Local Search algorithm for solving the classical TOPTW. We show experimentally that our proposed ILS can improve 31 best known solution values of the TOPTW benchmark instances.
- We propose the use of a factorial experiment design that screens and ranks the algorithm's parameters. The screening process helps us to identify parameters that can be set to constant values. By focusing on important parameters, we reduce the parameter search space and target our search on the promising regions of the important parameter search space.
- We show experimentally that our well-tuned ILS algorithm is effective in providing good solutions for a real world application of the extended TOPTW, namely the Tourist Trip Design Problem (TTDP).

The remainder of this paper is organized as follows. Section 2 provides a literature review. In Section 3, we present the description and the mathematical model of the Team Orienteering Problem with Time Windows (TOPTW). Section 4 is devoted to the proposed algorithm to solve the TOPTW benchmark instances and real-world problems. A brief explanation of a factorial experiment design used for determining the algorithm parameter values is also included. Section 5 provides the computation results together with the analysis of the results. Section 6 concludes the paper and summarizes directions for further research.

2. LITERATURE REVIEW

As described in Section 1, the TOPTW is a variant of the OP. Time-window constraints arise in the routing problems where customers have to be visited within

a predefined time interval specified by an earliest and a latest time into which the service has to start.

Vansteenwegen et al. (2009) proposed a simple, fast and effective Iterated Local Search (ILS) algorithm to solve the OPTW and the TOPTW. The algorithm is started by inserting one node at a time to a route by ensuring the feasibility of other nodes that have been scheduled on a route. This insertion is repeated until a local optimum is reached. The shake step is then used to escape from the local optima and to explore other possible solutions. During this step, a certain number of nodes will be removed. A new data set with more difficult instances was designed to analyse the performance of the proposed algorithm and to be used as a benchmark for further research. High quality results are obtained within a short computation time and new best known solutions for 31 instances have been found.

Montemanni and Gambardella (2009) introduced a heuristic based on an Ant Colony System (ACS) algorithm. It includes a local search procedure by exchanging two sub-chains of nodes of the giant tour. The algorithm is used to solve the OPTW and the TOPTW. For the OPTW, ACS is able to retrieve the best known solutions for Solomon's instances. Experimental results on benchmark instances of the TOPTW have proven the effectiveness of the algorithm. Montemanni et al. (2011) identified some drawbacks of ACS (Montemanni and Gambardella, 2009) and proposed an Enhanced ACS algorithm (EACS) that provides some directions for overcoming the drawbacks. Two operations to enhance the performance of ACS are: considering 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. EACS is able to improve the results of ACS in average and to retrieve new best known results for some instances.

A new problem for scheduling sales representatives to visit the customers, namely the Multi-Period Orienteering Problem with multiple Time Windows, was introduced by Tricoire et al. (2010). The MuPOPTW allows multiple time windows. Each customer may be visited on different days, on different time slots and each customer may have multiple time windows for a given day. A Variable Neighborhood Search (VNS) algorithm was introduced in order to provide good solutions in reasonable computation time. The VNS was adapted to the OPTW and TOPTW benchmark instances. VNS is able to find the optimal solution for each instance at least once over 10 runs for the OPTW instances. VNS also provides very good solutions but it requires more computation time for solving the TOPTW instances.

A hybridization of a Greedy Randomized Adaptive Search Procedure (GRASP) and an Evolutionary Local Search algorithm (ELS) was proposed by Labadie et al.

(2011). Five simple constructive heuristics are introduced in order to build the initial solutions. The first three heuristics, which focus on adding one node into the solution at each iteration, are different in the way of selecting the best insertion. The other two heuristics are based on sweep algorithms that focus on creating clusters of nodes and constructing a tour for each cluster. All different constructive heuristics are used in the GRASP to generate distinct initial solutions that would be further improved by the ELS algorithm. GRASP-ELS is able to provide 150 new best known solutions of both the OPTW and the TOPTW.

A Simulated Annealing-based heuristic was proposed by Lin and Yu (2012). Two different versions, fast SA (FSA) and slow SA (SSA), are developed in order to tailor two different scenarios. FSA, which is based on the computational time used, is mainly for the applications that need quick responses. On the other hand, SSA is more concerned about the quality of the solutions; therefore, it will only be terminated if the current best solution has not been improved for a certain number of consecutive temperature decreases. Both implement a standard SA procedure with a random neighborhood structure that features swap, insertion and inversion operations. The SSA heuristic is able to find 33 new best solutions. The FSA is comparable to the ILS (Vansteenwegen et al., 2009) in terms of the average computational time, but the solution quality of the FSA is slightly better than that of the ILS. The FSA obtains better solutions compared to the ACS (Montemanni and Gambardella, 2009) within a much shorter computational time.

Labadie et al. (2012) introduced an LP-based Granular Variable Neighborhood Search (GVNS) to solve the OPTW and the TOPTW. The idea of granularity that includes time constraints and profits in addition to pure distances is introduced. The granularity aims at reducing the size of the analyzed neighborhoods without losing its effectiveness. The dual optimal solutions of a LP-problem is used to construct granular neighborhoods. is comparable to the state-of-the-art algorithms (Labadie et al., 2011; Montemanni and Gambardella, 2009; Vansteenwegen et al., 2009) and it is able to improve 25 best known solution values.

A hybrid algorithm based on a Greedy Randomized Adaptive Search Procedure (GRASP) and Iterated Local Search (ILS), namely GRILS, was introduced by Souffriau et al. (2013) for solving the Multiconstraint Team Orienteering Problem with Multiple Time Windows (MC-TOPMTW), a variant of the TOPTW. In this problem, some nodes may have one or more time windows. A certain number of additional knapsack constraints are included in the problem. Since there is no benchmark instances for MC-TOPMTW, the performance of the proposed algorithm is evaluated by using three related problems, the TOPTW, the Selective Vehicle Routing Problem with Time Windows (SVRPTW) (Boussier et al., 2007)

and the Multiconstraint Team Orienteering Problem with Time Windows (MC-TOPTW) (Garcia et al., 2009). Here, we only focus on the TOPTW results. The results obtained by GRILS are compared with those of ACS (Montemanni and Gambardella, 2009), ILS (Vansteenwegen et al., 2009) and VNS (Tricoire et al., 2010). Although VNS generates better results with regard to solution quality, the computation time of VNS is much higher than those of ILS and GRILS. GRILS outperforms ILS at a cost of extra computation time. Three new best known solutions have been achieved by the GRILS algorithm.

Another iterative framework, namely I3CH, has been proposed by Hu and Lim (2014). It is based on two components: a local search (LS) procedure and a Simulated Annealing (SA) procedure. Both components are used to explore the solution space and discover a set of routes. An eliminator operator is mainly used to generate neighborhood solutions. First, some nodes from some routes are removed and some unscheduled nodes are then inserted. Only improved solution quality would be considered. Such newly generated solutions are further improved through a post-processing procedure that consists of seven different operators, such as EXCHANGE, 2-OPT and RELOCATE operators. Both LS and SA send the solutions into *POOL*. The last component of the framework, namely Route Recombination (RR), which focuses on combining the routes from *POOL* to identify high quality solutions is included. RR solves a set packing formulation to produce the best combination. The final solution is then used to assist LS and SA. The entire framework would be run iteratively within a certain number of iterations, I_{max}. It is shown that 35 new best solutions are found and more than 83% of instances with optimal solutions can be obtained.

Cura (2014) proposed a relatively new technique called artificial bee colony (ABC) approach to solve the TOPTW. A new food source acceptance criterion based on a Simulated Annealing (SA) heuristic and a new scout bee search behavior based on a local search procedure are introduced in order to improve the solution quality of benchmark instances. The proposed method is able to produce high-quality TOPTW solutions and comparable to other approaches. There is no new best found solution reported.

The pulse algorithm (Lozano et al., 2014) was adapted for solving the OPTW (Duque et al., 2015). The pulse algorithm was presented as a general-purpose pulse framework for hard shortest path problems. The algorithm was tested on a part of the well-known benchmark instances. It was also tested over large-scale instances on the multiobjective orienteering problem (MOOP) (Schilde et al., 2009). The proposed method outperforms the state-of-the-art algorithm (Righini and Salani, 2008) in terms of the computation speed of obtaining the optimal

solutions. However the experiments were only conducted on a part of the benchmark instances. It is also able to obtain optimal solutions to large-scale problems within reasonable computation times.

In most published works, how to methodically choose parameter values are rarely reported. They only reported recommended parameter values, such as works by Vansteenwegen et al. (2009) and Lin and Yu (2012). In some other works, some researchers use a One-Factor-At-A-Time (OFAT) approach in order to determine the parameter values of the proposed algorithms. This approach examines the effect of a subset of parameters while holding all the rest constant, e.g. parameter tuning in ABC approach (Cura, 2014) and I3CH (Hu and Lim, 2014).

Ridge and Kudenko (2007) presented an in-depth Design of Experiments (DOE) methodology for the performance analysis of a stochastic heuristic for solving the Traveling Salesman Problem (TSP). The weaknesses of OFAT approach have also been discussed. They also demonstrates the advantages of Design of Experiments (DOE) over the OFAT approach, such as DOE require fewer resources, in terms of experiments, time and material, for the amount of information obtained. Gunawan et al. (2011) proposed another automated tuning framework based on the DOE approach and applied to two combinatorial optimization problems, Traveling Salesman Problem (TSP) and Quadratic Assignment Problem (QAP).

The TOPTW can be extended to model the Tourist Trip Design Problem (TTDP) (Vansteenwegen et al., 2009). In the TTDP, integrated holiday plans consisting of the most valuable points of interest (POIs) are suggested to the tourists based on the tourists' preferences. The score of a particular POI represents the interest of a tourist in that POI. Vansteenwegen et al. (2011b) presented the City Trip Planner as a website that is able to propose city trips tailored to the user's context and personal interests. The system has been implemented as web site for five historic cities in Flanders, Belgium. Garcia et al. (2013) extended the idea by integrating public transportation in personalised tourist trips. The problem is viewed as the Time-Dependent Team Orienteering Problem with Time Windows (TD-TOPTW).

3. PROBLEM DESCRIPTION

The extended TOPTW 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 score u_i and service time T_i. The non-negative travel time between nodes i and j is represented as t_{ij}. We also define a set of routes $M = \{1, 2, \cdots, |M|\}$. Each route $m \in M$ is constrained by a given time budget T_m^{max}.

Other new sets are defined as follows:

- $N^{start} = \{n_1^{start}, n_2^{start}, \cdots, n_{|M|}^{start}\}$, where n_m^{start} represents the start node in route m ($n_m^{start} \in N, m \in M$).
- $N^{end} = \{n_1^{end}, n_2^{end}, \cdots, n_{|M|}^{end}\}$, where n_m^{end} represents the end node in route m ($n_m^{end} \in N, m \in M$).

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 the service at node i. We assume that $e_{n_m^{start}} = 0$ $(\forall n_m^{start} \in N^{start}, m \in M)$, $e_{n_m^{end}} = 0$ $(\forall n_m^{end} \in N^{end}, m \in M)$, $l_{n_m^{start}} = T_m^{max}$ $(\forall n_m^{start} \in N^{start}, m \in M)$, $l_{n_m^{end}} = T_m^{max}$ $(\forall n_m^{end} \in N^{end}, m \in M)$. T_i is set to 0 $(\forall i \in (N^{start} \cup N^{end}))$.

The following decision variables are defined:

- X_{ijm} = 1 if a visit to node i is followed by a visit to node j in route m, 0 otherwise.
- Y_{im} = 1 if a visit to node i in route m.
- S_{im} = the start time at node i in route m.
- W_{im} = the waiting time before entering node i in route m.

The TOPTW is formulated as an integer programming model:

$$Maximize \sum_{m \in M} \sum_{i \in N \setminus (N^{start} \cup N^{end})} u_i Y_{im} \tag{1}$$

The objective function is to maximize the total collected score from visited nodes from all routes.

$$\sum_{j \in N \setminus N^{start}} X_{ijm} = 1 \ , \forall m \in M, i \in N^{start} \tag{2}$$

$$\sum_{i \in N \setminus N^{end}} X_{ijm} = 1 \ , \forall m \in M, j \in N^{end} \tag{3}$$

Constraints 2 and 3 ensure that each route m starts from its start node n_m^{start} and ends at its end node n_m^{end}.

$$\sum_{i \in N \setminus N^{end}} X_{ikm} = Y_{km} \ , \forall k \in N \setminus (N^{start} \cup N^{end}), m \in M \tag{4}$$

$$\sum_{j \in N \setminus N^{start}} X_{kjm} = Y_{km} \ , \forall k \in N \setminus (N^{start} \cup N^{end}), m \in M \tag{5}$$

Constraints 4 and 5 determine the connectivity of each route m.

$$\sum_{m \in M} Y_{im} \leq 1 \ , \forall i \in N \setminus (N^{start} \cup N^{end}) \tag{6}$$

Constraint 6 ensures that each node i is visited at most once.

$$Y_{im} = 1 \ , \forall m \in M, i \in (N^{start} \cup N^{end}) \tag{7}$$

Constraint 7 ensures that the start node n_m^{start} and end node n_m^{end} in each route m are visited.

$$S_{im} \geq e_i \ , \forall m \in M, i \in N \tag{8}$$

$$S_{im} \leq l_i \ , \forall m \in M, i \in N \tag{9}$$

Constraints 8 and 9 ensure that the start time at node i in route m is within time window $[e_i, l_i]$.

$$S_{im} + T_i + t_{ij} - S_{jm} \leq \hat{L}(1 - X_{ijm}) \ , \forall i, j \in N, m \in M \tag{10}$$

$$S_{jm} - (S_{im} + T_i + t_{ij}) \leq \hat{L}(1 - X_{ijm}) + W_{jm} \ , \forall i, j \in N, m \in M \tag{11}$$

Constraint 10 ensures that if nodes i and j are visited consecutively, then the start time at node j has to be greater than or equal to the start time at node i plus the service time at node i and the travel time from nodes i to j. In constraint 11, the waiting time before entering node j in route m, W_{jm}, is at least equal to the start time at node j minus the arrival time at node j (i.e. sum of the start time at node i, the service time at node i and the travel time from nodes i to j). Both constraints ensure the timeline of each route m. Note that $\hat{L}$ is a very large constant value.

$$\sum_{i \in N} T_i Y_{im} + \sum_{i \in N \setminus N^{end}} \sum_{j \in N \setminus N^{start}} t_{ij} X_{ijm} + \sum_{i \in N} W_{im} \leq T_m^{max} \ , \forall m \in M \tag{12}$$

Constraint 12 limits the time budget for each route m by T_m^{max}.

$$S_{im}, W_{im} \geq 0 \ , \forall i \in N, m \in M \tag{13}$$

Constraint 13 is the non-negativity condition for S_{im} and W_{im}.

$$X_{ijm}, Y_{im} \in \{0,1\}\ , \forall i, j \in N, m \in M \tag{14}$$

The binary conditions for X_{ijm} and Y_{im} are constrained by equation 14.

The above mathematical model is considered as an extension of the TOPTW when the start and end nodes for each route m can be different and the given time budget T_m^{max} for each route m may vary. Simpler mathematical formulations of the TOPTW, when the start and end nodes are the same and the values of T_m^{max} for all routes are constant, can be found in Labadie et al. (2012) and Vansteenwegen et al. (2009).

In the TTDP context, e_i and l_i refer to the opening and closing times of a particular POI i, respectively. We need to ensure that the visitor has to leave the POI before l_i. In this scenario, constraint 9 is replaced by constraint 15.

$$S_{im} + T_i \leq l_i\ , \forall m \in M, i \in N \tag{15}$$

4. ALGORITHM

In this section, we propose a simple yet effective heuristic algorithm that improves the state-of-the-art results and works well in real-world problem instances. In a nutshell, our algorithm constructs an initial feasible solution via a greedy heuristic. The initial solution is further improved by Iterated Local Search (ILS), which comprises the components LOCALSEARCH, PERTURBATION and ACCEPTANCECRITERION. The details of our proposed algorithm will be described in the following sub-sections.

4.1. Greedy Construction Heuristic

The basic idea is to begin from an empty solution and insert nodes iteratively to the current solution until there is no node can be inserted. The greedy construction heuristic is outlined in Algorithm 1.

Let N' and N^* be the sets of unscheduled and scheduled nodes, respectively ($N' \cup N^* = N$) and let F be the set of feasible candidate nodes to be inserted. N^* is initialized by N^{start} and N^{end}, while N' consists of unscheduled nodes. S_0 refers the current feasible solution obtained so far, represented as m-vector. Each row m is initialized with $n_m^{start} \in N^{start}$ and $n_m^{end} \in N^{end}$.

The set F is iteratively generated to store feasible candidate unscheduled nodes to be inserted into S_0. We examine all possibilities of inserting an unscheduled

Algorithm 1 CONSTRUCTION (N, M)

$N' \leftarrow N \setminus \{N^{start} \cup N^{end}\}$
$N^* \leftarrow \{N^{start} \cup N^{end}\}$
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

node in position p of route m. Let $P(m)$ represents a set of positions of scheduled nodes in route m. A node insertion is feasible if all scheduled nodes after an insertion still satisfy their respective time windows and the total spent time of each route m does not exceed T_m^{max}.

Each element in F, which represents a feasible insertion of node n in position p of route m, is represented as $\langle n, p, m \rangle$. For each possible insertion, we calculate the benefit of insertion $ratio_{n,p,m}$ by using equation 16. $\Delta_{n,p,m}$ represents the difference between the total time spent before and after the insertion of node n in position p of route m. For example, if the total time spent of a particular route is increased from 700 to 720 time units after inserting node n in position p of route m, the value of $\Delta_{n,p,m}$ is 720 - 700 = 20 time units. All elements in F would be sorted in descending order based on $ratio_{n,p,m}$ values and we only keep f elements. The idea of generating F is summarized in Algorithm 2.

$$ratio_{n,p,m} = \left(\frac{u_n^2}{\Delta_{n,p,m}} \right) \tag{16}$$

If F is not an empty set, Algorithm 3 is run in order to select $\langle n^*, p^*, m^* \rangle$ to be inserted. Each $\langle n, p, m \rangle$ corresponds to a probability value, $prob_{n,p,m}$, which is calculated by Equation 17:

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 route m is feasible **then**
        calculate $ratio_{n,p,m}$
        $F \leftarrow F \cup \langle n, p, m \rangle$
      end if
    end for
  end for
end for
Sort all elements of F in descending order based on $ratio_{n,p,m}$
Select the best f number of elements of F and remove the rest
return F

$$prob_{n,p,m} = \left(\frac{ratio_{n,p,m}}{\sum_{\langle i,j,k \rangle \in F} ratio_{i,j,k}} \right) \tag{17}$$

Selecting $\langle n^*, p^*, m^* \rangle$ from F is based on the Roulette-Wheel selection (Goldberg, 1989). This method assumes that the probability of selecting a candidate insertion is proportional to its benefit, $ratio_{n,p,m}$. Random number $U \sim [0,1]$ is generated. The accumulative of probability values, *AccumProb*, is initially set to 0. We subsequently select a candidate $\langle n^*, p^*, m^* \rangle$ and update the value of *AccumProb*. This loop will be terminated when $(U \leq AccumProb)$ and the corresponding $\langle n^*, p^*, m^* \rangle$ is then selected. S_0, N' and N^* will be updated after the insertion. The greedy construction heuristic is terminated when there is no further feasible candidate insertion (when $F = \emptyset$).

Vansteenwegen et al. (2009) concluded that due to the time windows, the score of a candidate insertion is more relevant compared against its time consumption. Therefore, the square of score is imposed in Equation 16. Besides, by using the square of score, we increase the probability of selecting a candidate insertion node with a higher ratio (refer to Equation 17) which aligns with the main objective of maximizing the total collected score.

Algorithm 3 SELECT (F)

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

4.2. Iterated Local Search

Given the initial solution S_0 generated from the greedy construction heuristic, we propose Iterated Local Search (ILS) 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.

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

In the EXCHANGEROUTE step, all nodes of one route are moved to the other route in the same order and vice versa. The strategy of selecting two different routes are based on generating of permutations by adjacent transposition method (Johnson, 1963). The basic idea is each permutation is derived from its predecessor by a single interchange of two routes in adjacent positions. For example, with m = 3, the selected permutation is run once in this order: 1 **2 3**, **1 3** 2, 3 **1 2**, **3 2** 1,

Algorithm 4 ILS (N, M)

$S_0 \leftarrow$ CONSTRUCTION(N, M)
$S_0 \leftarrow$ LOCALSEARCH(S_0, N^*, N', M)
$S^* \leftarrow S_0$
NOIMPR $\leftarrow 0$
while TIMELIMIT has not been reached **do**
  $S_0 \leftarrow$ PERTURBATION(S_0, N^*, N', M)
  $S_0 \leftarrow$ LOCALSEARCH(S_0, N^*, N', M)
  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^*

2 **3 1** and 2 1 3. Two bold numbers represent the routes need to be swapped when EXCHANGEROUTE is called.

The SHAKE step is adopted from Vansteenwegen et al. (2009) with some modifications. During SHAKE step, one or more nodes will be removed in each route m, which depends on two integer values, CONS and POST. CONS indicates how many consecutive nodes to remove for a particular route while POST indicates the first position of the removing process in a particular route. If we reach the last scheduled node, the process will then be back to the first node after the start node n_m^{start}. Both CONS and POST are initially set to 1. After each SHAKE step, POST is increased by CONS. CONS would also be increased by one after a fixed number of consecutive iterations, e.g. 2 iterations. If POST is greater than the size of the smallest route, POST is subtracted with the size of the smallest route to determine the new position POST. If CONS is greater than the size of the largest route, or S^* is updated, CONS is reset to one. After removing CONS nodes, we update N' and N^* accordingly. F is then regenerated based on Algorithm 2 and an unscheduled node that needs to be inserted is selected using Algorithm 3. This is repeated until $F = \emptyset$.

Table 1: LOCAL SEARCH operations

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

In LOCALSEARCH, we run six different operations consecutively, as shown in Table 1. SWAP1 is applied by exchanging two scheduled nodes within one particular route. It is started by selecting one route m with the lowest unused time budget. All possible combinations of selecting two different nodes are examined. SWAP1 is executed if it increases the unused time budget of route m and there is no constraint violation. The idea of swapping two nodes is extended to two different routes with the lowest and the second lowest unused time budget, namely SWAP2. The swapping is executed if the total of unused time budgets of both selected routes is increased. Both swapping operations would be terminated when there is no further improvement.

2-OPT is started by selecting one route m with the lowest unused time budget. By considering all possible combinations of selecting two different nodes of route m, the sequence of scheduled nodes in between is reversed as long as there is no constraint violation and it increases the unused time budget of route m. This would be terminated until no further improvement in terms of the total of unused time budget of route m.

MOVE is performed by reallocating one node from one route to another one. It is started from the first scheduled node n^* from the first route m^*. The idea is to generate F using Algorithm 2 where $N' = \{n^*\}$ and $M = M \setminus \{m^*\}$. If $F \neq \emptyset$, node n^* would be reallocated based on Algorithm 3. Otherwise, we continue to the next scheduled node. This operation would be terminated if node n^* is moved or we have reached the last scheduled node of the last route $|M|$. INSERT is applied in order to subsequently insert unscheduled nodes into the current solution. It is started by generating F based on Algorithm 2 and selecting candidate node $i \in N'$ to be inserted using Algorithm 3. After the insertion, N', N^* and F are updated accordingly. This is repeated until $F = \emptyset$.

The last operation, REPLACE, tries to replace one scheduled node $i \in N^*$ with

one unscheduled node $j \in N'$. The operation is started by selecting route m with the highest unused time budget, followed by selecting one node $j \in N'$ with the highest score u_j. We then check each position p of selected route m 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, the solution is updated. We then continue with the next unscheduled node j and repeat the operation. If the operation is not successful for any selected node j, the operation would be terminated. All six operations in Table 1 are run consecutively when we call the LOCALSEARCH component.

If S^* is not updated for a certain number of iterations,((NOIMPR+1) MOD THRESHOLD1 = 0), we apply an intensification strategy. This strategy restarts the search from the best found solution, S^*. Finally, the entire algorithm will be run within the computational budget, TIMELIMIT.

4.3. Parameter Tuning

As described in Subsections 4.1 and 4.2, the ILS algorithm uses four parameters that need to be determined: f, THRESHOLD1, THRESHOLD2 and THRESHOLD3. It is very time consuming if we use OFAT to determine their values. We implement a sequential experimental approach which is grounded on the Design of Experiment (DOE) methodology for screening algorithm parameters that has been proposed by Gunawan et al. (2011).

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 $[LB_{par}, UB_{par}]$. LB_{par} and UB_{par} represent the upper and lower bound values of parameter par, respectively. We apply a $2^{|PR|}$ factorial experiment 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 parameter value. The screening is performed in order to determine which parameters are statistically important. By doing so, the number of parameters that require tuning are then reduced. The following example provides an illustration how the screening phase is applied.

Suppose we wish to study the effect of two different parameters, par_1 and par_2. The 2^2 factorial experiment design would consist of four experimental units where each unit is run rep times:

- unit (1): set par_1 at LB_{par_1} and par_2 at LB_{par_2}
- unit (par_1): set par_1 at LB_{par_1} and par_2 at UB_{par_2}

- unit (par_2): set par_1 at UB_{par_1} and par_2 at LB_{par_2}
- unit ($par_1 par_2$): set par_1 at UB_{par_1} and par_2 at UB_{par_2}

The four experimental units above use abbreviations: (1), (par_1), (par_2) and ($par_1 par_2$). The presence of parameter names (e.g. par_2) indicates that that parameter par_2 is at its UB value, while par_1 is at its LB value. A factorial experiment is then analyzed by using Analysis of Variance (ANOVA) to estimate the main effect for a particular parameter. 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 ranking of the important parameters is then done by comparing the absolute values of the main effects of those parameters. Each unimportant (non-significant) parameter is then set to a constant value. For more details, the reader may refer to Gunawan et al. (2011).

5. COMPUTATIONAL EXPERIMENTS

5.1. Instances and Experimental Setup

Since the test problems for the extended TOPTW are not available, benchmark instances of the OPTW and TOPTW problems are used to examine the performance of ILS. The OPTW instances were initially proposed by Righini and Salani (2009). 48 test problems were generated from Solomon's instances (Solomon, 1987) and 10 instances were adapted from Cordeau's instances (Cordeau et al., 1997). Both Solomon's and Cordeau's instances were originally designed for vehicle routing problems with time windows and multi-depot vehicle routing problems, respectively. Montemanni and Gambardella (2009) developed another set of 37 instances for the OPTW. The TOPTW instances are constructed by extending the OPTW instances with different values of routes: $m = 2$, 3 and 4.

Subsequently, Vansteenwegen et al. (2009) added more extra instances based on instances of Solomon (1987) and Cordeau et al. (1997). Those instances are considered more difficult instances, with the number of routes $|M|$ is set to the number of vehicles. The optimal solution for these instances are known due to the specific setting on the number of provided vehicles, which is equal to the total score collected from all customers.

In this paper, another set of real-world instances related to the extended TOPTW is introduced. Instances are based on 50 Point of Interests (POIs) in Singapore. The major different characteristics from benchmark instances are as follows. The start and end nodes differ for each route m and the time budget for each route

m, T_m^{max}, may vary. The definition of time window $[e_i, l_i]$ for POI i is related to the opening and closing times of POI i. It means that the user has to leave POI i before the closing time l_i. The score related to the attractiveness of POI u_i is assumed to be known. The travel time between two POIs, the service time, the opening and closing time for POIs are also known. Table 2 summarizes the details of the instances. Hu and Lim (2014) classified the first two sets of instances as the "INST-M" category while the third one is classified into the "OPT" category. The last one represents the real-world instances.

Table 2: Instances

References	Instances	Names	$\|N\|$	$\|M\|$
Righini and Salani (2009)	Solomon	c^*_100, r^*_100, rc^*_100	100	1 to 4
	Cordeau	pr01 - pr10	[48, 288]	
Montemanni and Gambardella (2009)	Solomon	c^*_200, r^*_200, rc^*_200	100	1 to 4
	Cordeau	pr11 - pr20	[48, 288]	
Vansteenwegen et al. (2009)	Solomon	c^*_100, r^*_100, rc^*_100	100	up to number of vehicles
		c^*_200, r^*_200, rc^*_200	100	
	Cordeau	pr01 - pr10	[48, 288]	
-	Real-world	R1-R5	50	1 to 5

The experiments were carried out on a personal computer Intel Core i7 - 4770 with 3.4 GHz processor and 16 GB RAM. The difficulty of solving the instances by a commercial solver (CPLEX) has been mentioned by Vansteenwegen et al. (2009). Our ILS was tested by performing 10 runs with different random seeds for each instance. The performances of the proposed ILS are compared to the state-of-the-art algorithms: Iterated Local Search (IterILS) (Vansteenwegen et al., 2009), Variable Neighborhood Search (VNS) (Tricoire et al., 2010), Ant Colony System (ACS) (Montemanni and Gambardella, 2009), Enhanced Ant Colony System (Enhanced ACS) (Montemanni et al., 2011), Slow Simulated Annealing (SSA) (Lin and Yu, 2012), Granular Variable Neighborhood Search (GVNS) (Labadie et al., 2012), hybridized Greedy Randomized Iterated Local Search (GRILS) (Souffriau et al., 2013) and Iterative Three-Component Heuristic (I3CH) (Hu and Lim, 2014). The Enhanced ACS algorithm proposed by Montemanni et al. (2011) has empirically outperformed the original ACS (Montemanni and Gambardella, 2009). In this paper, we refer and consolidate the results of both ACS and Enhanced ACS and denote them as ACS*.

ACS* was executed in 5 runs for each instance whereas VNS, GVNS, GRILS and ILS were executed in 10. On the other hand, IterILS, SSA and I3CH were only executed once therefore only the best found solutions were reported. Take note that VNS, ACS* and GRILS were not tested on new instances, "OPT" category

Table 3: Estimation of single-thread performance (Hu and Lim, 2014)

Algorithm	Experimental environment	*SuperPi*	Estimate of single-thread performance
IterILS	Intel Core 2 with 2.5 gigahertz CPU, 3.45 gigabytes RAM	18.6	0.53
VNS	Intel Pentium 4 with 2.4 gigahertz CPU, 4 gigabytes RAM	unknown	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
GRILS	Intel Xeon with 2.5 gigahertz CPU, 4 gigabytes RAM	unknown	0.53
I3CH	Intel Xeon E5430 with 2.66 gigahertz CPU, 8 gigabytes RAM	14.7	0.67
ILS	Intel Core i7-4770 with 3.4 GHz processor, 16 gigabytes RAM	9.8	1

(Vansteenwegen et al., 2009). For comparison purpose, the solutions of our ILS were compared against the best known solutions (*BK*s) from the state-of-the-art algorithms.

Table 3 summarizes the experimental environment of each algorithm. We refer to the same rational proposed by Hu and Lim (2014) to compare the speed of the computers. *SuperPi* is a single-threaded program that computes the first 1 million digits of π of a particular processor. Labadie et al. (2011) showed the comparability of processors used by ACS* and GVNS since the *SuperPi* for ACS* is not available. The same conclusion is applied to VNS (Tricoire et al., 2010) and GRILS (Souffriau et al., 2013).

By setting the performance of our machine to be 1, we then estimate the single-thread performance of other processors by multiplying with the single-thread performance estimation (last column of Table 3). For the details, please refer to Hu and Lim (2014). Among all algorithms, only ACS* used one hour computational time for each instance, while the rest use the number of iterations. In this paper, we used ACS*'s computational time as our reference since we are more concerned with solution quality. The computational time was adjusted and only 35% of ACS*'s computational time (3600 seconds) is used for solving each instance by our ILS. In conclusion, the computational time for each instance is set to $35\% \times 0.22 \times 3600$ seconds = 272 seconds using our processor.

5.2. 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 followed the same scenario used by

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
f*Threshold1	-0.069	-0.035	0.03312	-1.05	0.298
f*Threshold2	-0.515	-0.258	0.03312	-7.78	0.000
f*Threshold3	-0.847	-0.424	0.03312	-12.79	0.000

.....

Figure 1: Design of Experiments result

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 $m = 4$. 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 Subsection 4.3. In our problem, the DOE result is shown in Figure 1. Two statistically significant parameters with p-value $< 5\%$ are f and THRESHOLD1. These two parameters are considered as important parameters. By referring to the effect values, the direction of adjustment for each parameter can be determined. For example, the most important parameters, f, has the highest effect value -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]. Similar idea is implemented to the second highest important parameter, THRESHOLD1. On the other hand, for non-significant parameters (e.g. THRESHOLD2 and THRESHOLD3), we set to a 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.

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

Our main focus now is on both important parameters, f and THRESHOLD1. We select some integer values and re-run ILS. For the remaining parameters, we

set THRESHOLD2 = 20 and THRESHOLD3 = 3. In our design, the experiment was repeated five 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. The results are summarized in Table 5 which presents the average gap for each possible combination of f and THRESHOLD1.

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**

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.

5.3. Experimental Results

The summary of results obtained are reported in the following sub-sections. The details of results can also be downloaded at `http://centres.smu.edu.sg/larc/Orienteering-Problem-Library`.

5.3.1. Benchmark Instances

In order to evaluate the benefit of implementing DOE in order to obtain a set of parameter values, we conduct two different scenarios, as shown in Table 6. The first scenario is to use the mid value of each initial range (Table 4), while the second one is to use the parameter values proposed by the DOE. Table 7 reports the overall results obtained by both scenarios. We compare the benchmark instances under "INST-M" category using the experimental setup explained in Section 5.1. For each particular value of m, the average of percentage gap between the best known solutions and ILS results is calculated. Improvement (%) is determined by calculating the percentage improvement from the results of Scenario 2 against the ones of Scenario 1.

Table 6: Two scenarios of parameter setting

Scenario	Description	f	THRESHOLD1	THRESHOLD2	THRESHOLD3
1	Mid value of each parameter interval	5	5	15	2
2	DOE	5	10	20	3

Table 7: Comparison between two parameter setting scenarios

m	Scenario 1 (%)	Scenario 2 (%)	Improvement (%)
1	0.73	0.63	14.05
2	1.34	1.27	5.56
3	1.33	1.26	5.42
4	1.65	1.54	6.97

We conclude that by implementing the DOE, the solutions are improved. The range of improvement is from 5.42% to 14.05%. We further explain the performance of ILS using the DOE approach and its comparison with other state-of-the-art algorithms below.

Table 8 reports the new best known solutions (new *BK*s) obtained by ILS. We discovered 17 new best known solution values for both Solomon's and Cordeau's instances. Most of new *BK*s (13 out 17) are from m = 1 and 2, while the rest are from m = 3 and 4.

Table 8: New best known solution values found by ILS

Instance	m	Old *BK*	New *BK*	Instance	m	Old *BK*	New *BK*	Instance	m	Old *BK*	New *BK*
r203	1	1021	1026	rc206	1	895	899	rc206	2	1546	1552
r204	1	1086	1093	rc208	1	1053	1057	r104	3	777	778
r208	1	1112	1113	pr15	2	1219	1220	rc104	3	834	835
r209	1	950	956	r107	2	536	538	r104	4	972	973
r211	1	1046	1049	r205	2	1380	1385	rc107	4	980	985
rc202	1	936	938	r209	2	1405	1406				

Table 9 summarizes the results of "INST-M" instances solved by IterILS, VNS, ACS*, GVNS, SSA, GRILS, I3CH and our proposed ILS. The *numb* column provides the number of instances in a particular instance set. The table reports the average relative percentage deviation (*AG*), which refers to the average of percentage gap between *BK* and the average solutions obtained by one particular algorithm. However, IterILS, SSA and I3CH only reported their best known solution obtained; therefore, we calculate the best relative percentage deviation (*BG*) that represents the average of percentage gap between *BK* and the best solution obtained. Take note that the best known solutions (*BK*s) are collected from the results of all state-of-the-algorithms.

The reported computational times of ACS* and ILS represent the average of computational time to obtain the best found *BK* (in seconds). On the other hand, IterILS, VNS, SSA, GVNS, GRILS and I3CH report the average of computational time for solving each instance (in seconds). The computational times of

all approaches were adjusted according their computer's speed as summarized in Table 3.

It can be seen from Table 9 that ILS can produce a better Grand Mean against those of VNS, ACS*, GVNS and GRILS. The Grand Mean of *AG* for ILS is only 1.19%, whereas VNS, ACS*, GVNS and GRILS values are 1.42%, 2.33%, 1.74% and 3.87%, respectively. In general, ILS performs best for $m = 1$. In terms of the average of computational time, ILS requires less computational time than VNS and ACS do. On the other hand, ILS needs more computational time compared against the ones of GVNS and GRILS but ILS can improve the average of *AG* up to 0.42% in terms of the solution quality.

Since IterILS, SSA and I3CH only report their best solutions, we also include the best found relative percentage deviation (*BG*) of VNS, ACS*, GVNS, GRILS and ILS in Table 10. For $m = 1$ and 2, ILS is the best compared against other methods. For rc200 ($m = 1$) and r100 ($m = 2$), ILS is able to produce *BG* with values of -0.04% and -0.03%, respectively. It means that ILS is able to improve some *BK*s of those instance sets.

ILS is comparable to I3CH for greater values of m, except for Cordeau's instances. The variation of *BG* values of ILS is less than the one of I3CH. The *BG*s of ILS range from -0.04% to 2.91%, while the ones of I3CH range from 0.00% to 4.28%. On average, ILS provides a better Grand Mean compared with those of other methods, except VNS. The Grand Mean of *BG* is only 0.54%. Although the Grand Mean VNS is smaller, but it requires more computational time, as shown in Table 9.

Table 11 reports the results of difficult instances (the "OPT" category). The optimal solutions for these instances are known as the total score of all nodes / customers. In terms of *BG* solutions, ILS result is comparable to that of SSA, although it is worse than that of I3CH. ILS performs quite well in solving Solomon 200 instance set where the values of *BG* range from 0.00% to 0.09%.

Table 9: Overall "Average" Comparison of ILS to the state-of-the-art methods

Instance Set	*Numb*	IterILS		VNS		ACS*		SSA		GVNS		GRILS		I3CH		ILS	
		BG(%)	Time	AG(%)	Time	AG(%)	Time‡	BG(%)	Time	AG(%)	Time	AG(%)	Time	BG(%)	Time	AG(%)	Time‡
$m = 1$																	
c100	9	1.11	0.2	0.11	51.8	**0.00**	1.4	**0.00**	11.1	1.22	36.8	0.32	0.9	**0.00**	16.8	**0.00**	1.0
r100	12	1.90	0.1	0.05	46.9	0.24	84.8	0.11	12.3	2.68	6.5	0.22	0.8	0.56	19.1	**0.00**	1.8
rc100	8	2.92	0.1	0.04	34.4	**0.00**	31.7	**0.00**	11.7	3.51	2.2	0.28	0.5	1.66	17.0	**0.00**	1.0
c200	8	2.28	0.9	0.21	295.1	0.58	75.8	**0.13**	19.8	1.11	42.6	1.04	3.2	0.40	56.3	0.14	90.0
r200	11	2.90	0.9	1.06	561.6	3.17	344.4	1.30	24.1	3.38	7.5	2.75	5.8	1.05	117.4	**0.93**	162.1
rc200	8	3.43	0.8	1.35	458.1	2.04	341.7	**0.96**	26.5	3.96	3.5	3.31	3.8	2.68	79.6	0.97	120.5
pr01-10	10	4.74	0.9	1.10	433.1	1.22	359.8	0.98	59.1	1.62	2.7	3.74	2.2	1.07	72.7	**0.74**	50.4
pr11-20	10	9.56	1.1	3.38	551.1	11.87	196.4	3.71	85.6	4.26	5.4	7.96	2.7	4.28	86.8	**2.12**	97.9
$m = 2$																	
c100	9	0.94	0.6	0.27	46.4	0.14	177.6	**0.00**	13.9	0.72	30.9	1.35	2.5	**0.00**	58.0	0.12	41.4
r100	12	2.36	0.5	1.40	33.4	0.55	305.0	0.23	19.3	1.80	13.3	1.64	2.1	0.58	42.0	**0.06**	55.1
rc100	8	2.47	0.4	1.46	29.1	1.27	288.9	0.19	21.3	2.80	4.5	2.31	1.7	0.90	39.3	**0.03**	53.4
c200	8	2.54	1.8	0.86	287.5	1.81	309.3	1.18	28.3	**0.58**	7.5	3.11	6.9	0.68	267.5	0.71	211.6
r200	11	2.74	1.2	0.75	534.8	3.71	605.1	0.58	48.2	1.30	3.3	3.13	7.9	**0.21**	351.2	1.50	194.9
rc200	8	4.14	1.2	1.49	424.1	3.83	518.3	1.25	42.2	2.57	2.8	5.21	6.7	**0.62**	293.1	1.92	186.0
pr01-10	10	6.22	2.5	3.88	276.5	3.57	418.0	2.45	91.6	1.79	8.6	6.76	6.2	**1.11**	164.7	2.30	126.5
pr11-20	10	7.86	2.7	3.63	326.0	6.15	527.6	3.88	106.2	**2.18**	18.2	11.28	7.1	2.70	203.1	3.38	150.2
$m = 3$																	
c100	9	2.55	0.8	0.73	45.0	0.79	230.8	0.33	18.6	0.95	36.5	3.22	4.0	**0.11**	126.8	0.69	124.6
r100	12	1.79	0.9	1.47	32.6	1.30	390.6	0.39	29.5	2.27	16.4	3.18	3.3	**0.21**	78.9	0.52	88.3
rc100	8	3.14	0.6	1.41	31.9	1.07	262.9	0.64	22.6	2.32	7.4	3.18	2.7	**0.27**	67.3	0.30	68.4
c200	8	1.93	1.2	0.01	127.3	1.63	292.1	1.24	31.5	0.19	1.7	4.03	7.9	**0.00**	8.2	0.95	222.6
r200	11	0.30	0.7	0.11	169.5	0.24	259.2	0.08	22.1	0.20	1.5	0.25	7.3	**0.01**	60.5	0.22	181.9
rc200	8	1.44	0.9	0.32	212.9	0.94	337.4	0.27	31.1	0.44	1.6	1.19	7.2	**0.04**	109.4	0.51	204.3
pr01-10	10	6.58	4.8	3.63	249.3	4.21	469.3	2.34	103.8	1.13	19.0	7.58	9.7	**0.36**	282.7	3.59	175.7
pr11-20	10	9.19	5.1	3.35	272.7	7.24	571.8	3.81	132.7	1.80	33.3	11.21	10.7	**1.11**	331.3	3.59	179.2
$m = 4$																	
c100	9	3.11	1.3	1.24	43.1	1.27	273.2	0.55	26.1	1.63	29.5	4.72	5.6	**0.10**	174.5	1.32	168.2
r100	12	3.31	1.4	1.54	32.2	1.92	392.2	0.73	30.8	2.28	18.7	5.23	4.6	**0.16**	122.9	1.19	107.5
rc100	8	3.18	1.1	2.32	30.8	2.18	376.1	0.37	35.9	1.79	8.2	5.19	3.8	**0.23**	101.6	0.83	105.8
c200	8	**0.00**	0.5	**0.00**	55.2	0.07	1.7	**0.00**	22.0	**0.00**	0.1	**0.00**	7.1	**0.00**	0.1	**0.00**	199.2
r200	11	**0.00**	0.5	**0.00**	79.4	**0.00**	28.0	**0.00**	20.9	**0.00**	0.1	**0.00**	7.9	**0.00**	0.1	**0.00**	147.9
rc200	8	**0.00**	0.6	**0.00**	86.7	0.01	143.1	**0.00**	21.2	0.01	0.2	0.01	7.5	**0.00**	0.1	**0.00**	183.6
pr01-10	10	7.08	7.4	3.57	212.4	3.42	554.0	2.23	134.7	1.60	28.2	8.10	13.4	**0.36**	377.7	4.00	190.0
pr11-20	10	8.47	7.2	3.49	215.0	6.34	571.5	3.95	149.6	2.81	51.5	10.34	14.4	**0.45**	485.7	4.42	201.3
Grand Mean		3.50	1.6	1.42	197.9	2.33	310.3	1.09	46.5	1.74	14.2	3.87	5.6	0.69	134.0	1.19	126.6

‡ Average computational time to obtain the best found

Table 10: Overall "Best" Comparison of ILS to the state-of-the-art methods

Instance Set	*Numb*	IterILS	VNS	ACS*	SSA	GVNS	GRILS	I3CH	ILS
		BG(%)	BG(%)	BG(%)	BG(%)	BG(%)	BG(%)	BG(%)	BG(%)
$m = 1$									
c100	9	1.11	**0.00**	**0.00**	**0.00**	0.56	**0.00**	**0.00**	**0.00**
r100	12	1.90	**0.00**	**0.00**	0.11	1.72	**0.00**	0.56	**0.00**
rc100	8	2.92	**0.00**	**0.00**	**0.00**	1.88	**0.00**	1.66	**0.00**
c200	8	2.28	**0.00**	0.40	0.13	0.55	0.26	0.40	**0.00**
r200	11	2.90	0.41	2.19	1.30	2.45	1.65	1.05	**0.11**
rc200	8	3.43	0.07	1.23	0.96	2.53	2.04	2.68	**-0.04**
pr01-10	10	4.74	**0.02**	1.06	0.98	0.56	1.90	1.07	0.34
pr11-20	10	9.56	1.40	11.13	3.71	3.17	5.39	4.28	**1.33**
$m = 2$									
c100	9	0.94	**0.00**	**0.00**	**0.00**	0.47	0.92	**0.00**	**0.00**
r100	12	2.36	0.15	0.20	0.23	1.19	0.95	0.58	**-0.03**
rc100	8	2.47	0.23	0.33	0.19	0.78	1.15	0.90	**0.00**
c200	8	2.54	0.51	1.27	1.18	0.25	2.71	0.68	**0.17**
r200	11	2.74	0.25	3.16	0.58	0.67	2.36	**0.21**	0.51
rc200	8	4.14	**0.49**	2.70	1.25	1.68	3.56	0.62	0.72
pr01-10	10	6.22	0.87	2.59	2.45	0.82	5.30	1.11	**0.77**
pr11-20	10	7.86	1.26	5.00	3.88	**1.21**	8.12	2.70	1.66
$m = 3$									
c100	9	2.55	**0.11**	0.22	0.33	0.45	2.09	**0.11**	**0.11**
r100	12	1.79	0.22	0.36	0.39	1.22	2.01	0.21	**0.05**
rc100	8	3.14	0.36	0.35	0.64	0.91	1.72	0.27	**0.00**
c200	8	1.93	**0.00**	1.10	1.24	0.07	3.04	**0.00**	0.35
r200	11	0.30	0.03	0.13	0.08	0.11	0.13	**0.01**	0.09
rc200	8	1.44	0.11	0.42	0.27	0.32	0.65	**0.04**	0.23
pr01-10	10	6.58	1.51	2.96	2.34	**0.36**	6.32	**0.36**	2.13
pr11-20	10	9.19	1.78	5.40	3.81	**1.02**	9.39	1.11	2.07
$m = 4$									
c100	9	3.11	0.29	0.36	0.55	1.04	3.73	**0.10**	0.39
r100	12	3.31	0.30	0.78	0.73	1.22	3.75	**0.16**	0.33
rc100	8	3.18	0.45	0.78	0.37	0.95	3.32	0.23	**0.11**
c200	8	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
r200	11	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
rc200	8	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
pr01-10	10	7.08	1.87	2.76	2.23	1.08	6.75	**0.36**	2.33
pr11-20	10	8.47	2.31	5.53	3.95	2.05	8.76	**0.45**	2.91
Grand Mean		3.50	0.48	1.69	1.09	1.00	2.81	0.69	0.54

Table 11: Comparison of ILS to the state-of-the-art methods of the new Solomon's instances

Instance Set	*Numb*	IterILS		SSA		GVNS		I3CH		ILS		
		BG(%)	Time (seconds)	BG(%)	Time (seconds)	AG(%)	Time (seconds)	BG(%)	Time (seconds)	BG(%)	AG(%)	Time‡(seconds)
c100	9	1.41	1.6	1.04	40.9	0.47	1.7	0.00	31.7	0.37	0.79	195.6
r100	12	1.93	1.6	0.42	55.1	1.55	8.7	0.07	585.1	0.61	1.26	192.4
rc100	8	2.06	2.0	0.35	44.6	1.29	8.7	0.00	38.2	0.68	1.26	179.6
c200	8	0.00	0.6	0.00	22.1	0.00	0.1	0.00	0.4	0.00	0.00	193.9
r200	11	0.62	0.9	0.16	30.7	0.17	1.2	0.07	115.6	0.09	0.34	187.2
rc200	8	0.47	0.9	0.07	21.8	0.16	0.6	0.04	126.8	0.07	0.14	186.3
pr01-pr10	10	2.32	16.1	1.04	298.2	1.25	11.4	0.78	217.8	1.22	1.68	185.6
Grand Mean		1.30	3.5	0.45	76.6	0.74	4.9	0.15	183.0	0.45	0.82	188.8

‡ Average computational time to obtain the best found

Table 12: New best known solution values found by ILS using I3CH computation time

Instance	*m*	Old *BK*	New *BK*	Instance	*m*	Old *BK*	New *BK*	Instance	*m*	Old *BK*	New *BK*
pr15	1	707	708	r206	2	1440	1442	pr13	2	832	841
r209	2	1405	1407	pr04	2	925	926	pr18	2	938	941

Table 13: The solution quality improvement by ILS (in %)

Instance Set	"INST-M"				"OPT"
	$m = 1$	$m = 2$	$m = 3$	$m = 4$	
c100	2.05	3.99	3.92	4.28	3.06
r100	2.29	4.73	7.43	8.41	3.72
rc100	12.07	10.45	7.99	8.79	5.13
c200	4.54	4.74	4.50	0.14	0.14
r200	5.48	4.99	1.21	0.04	2.46
rc200	6.82	7.75	4.03	0.72	2.81
pr01-10	13.76	12.69	10.53	8.93	3.55
pr11-20	13.11	14.96	10.63	8.25	-
Grand Mean	7.40	7.99	6.34	5.12	3.02

As described in Section 4, the proposed algorithm consists of Greedy Construction Heuristic and Iterated Local Search. Table 13 summarizes the average of percentage improvement of the solution quality by implementing ILS to the initial solution generated by Greedy Construction Heuristic. It is observed that ILS is able to improve the initial solutions with the Grand Mean values range from 3.02% to 7.99%. Better improvements are achieved for $m = 1$ and 2 instances. For c*200, r*200 and rc*200 instances (with $m = 4$), the average improvement values are less than 1%. This due to the quality of solutions generated the Greedy Construction Heuristic are quite close to the optimal solutions. These instances are considered easier to solve. "OPT" instance sets are the most difficult to solve with the percentage improvement values are varied from 0.14% to 5.13%.

Hu and Lim (2014) concluded that I3CH outperforms SSA when using the same computational time that had been adjusted by their computers' speed. We also compare the performance of ILS against I3CH in solving the "INST-M" instances. By setting the computational time to the computational time required by I3CH, more new *BK*s especially for $m = 2$ instances have been found, as listed in Table 12. The best known of instance r209 ($m = 2$) has also been further improved to 1407. As shown in Table 14, we found that ILS's overall average performance in terms of *BG* is 0.16% better than that of I3CH.

In terms of *AG*, ILS outperforms for $m = 1$. The variation of *BG* values of I3CH is wider than the one of ILS. The *BG*s of ILS fluctuate from 0.01% to 3.77%, while the ones of I3CH range from 0.00% to 4.28%.

Table 14: Comparison with the same computational time with I3CH

m	Instance Set	I3CH	ILS		Time (seconds)
		BG(%)	BG(%)	AG(%)	
1	c100	0.00	0.00	0.00	16.9
	r100	0.56	0.00	0.00	19.2
	rc100	1.66	0.00	0.03	17.1
	c200	0.40	0.00	0.29	56.6
	r200	1.05	0.36	1.24	118.2
	rc200	2.68	0.20	1.30	80.0
	pr01-10	1.07	0.30	0.80	73.1
	pr11-20	4.28	1.29	2.30	87.3
2	c100	0.00	0.00	0.15	58.3
	r100	0.58	0.00	0.16	42.2
	rc100	0.90	0.02	0.20	39.5
	c200	0.68	0.42	0.78	268.9
	r200	0.21	0.44	1.50	353.1
	rc200	0.62	1.03	2.07	294.7
	pr01-10	1.11	0.51	2.33	165.6
	pr11-20	2.70	1.13	2.95	204.1
3	c100	0.11	0.11	0.77	127.5
	r100	0.21	0.13	0.70	79.3
	rc100	0.27	0.07	0.41	67.7
	c200	0.00	2.07	3.46	8.3
	r200	0.01	0.16	0.72	61.0
	rc200	0.04	0.65	1.47	110.0
	pr01-10	0.36	1.63	3.16	284.1
	pr11-20	1.11	1.33	3.31	333.0
4	c100	0.10	0.39	1.48	175.4
	r100	0.16	0.34	1.48	123.5
	rc100	0.23	0.17	0.87	102.1
	c200	0.00	0.00	0.03	0.1
	r200	0.00	0.02	0.18	0.2
	rc200	0.00	0.04	0.46	0.2
	pr01-10	0.36	2.05	3.77	379.6
	pr11-20	0.45	2.08	3.74	488.3
Grand Mean		0.69	0.53	1.33	134.7

We also carried out experiments on the "INST-M" instances by using the computational time required by VNS (Tricoire et al., 2010). Table 15 summarizes the comparison results. The computational time required by VNS is higher than the one of I3CH. It turns out that our ILS performs very well. The Grand Mean values of BG and AG are only 0.38% and 0.95%, respectively. The ILS also discovers 20 new best solutions which are summarized in Table 16. Eight instances have been further improved from the ones shown in Table 8.

Tables 17 and 18 summarize the comparison between I3CH vs ILS and VNS vs ILS in terms of the best found solutions. Out of 304 "INST-M" instances, ILS

Table 15: Comparison with the same computational time with VNS

m	Instance Set	VNS		ILS		Time (seconds)
		BG(%)	AG(%)	BG(%)	AG(%)	
1	c100	0.00	0.11	0.00	0.00	98.4
	r100	0.00	0.05	0.00	0.00	89.1
	rc100	0.00	0.04	0.00	0.00	65.2
	c200	0.00	0.21	0.00	0.05	560.3
	r200	0.41	1.06	-0.31	0.57	1066.2
	rc200	0.07	1.35	0.08	0.74	869.6
	pr01-10	0.02	1.10	0.07	0.57	822.1
	pr11-20	1.40	3.38	1.12	1.70	1046.0
2	c100	0.00	0.27	0.00	0.02	88.0
	r100	0.15	1.40	-0.03	-0.01	63.5
	rc100	0.23	1.46	0.00	0.02	55.2
	c200	0.51	0.86	0.08	0.52	545.8
	r200	0.25	0.75	0.11	1.03	1015.6
	rc200	0.49	1.49	0.56	1.83	805.1
	pr01-10	0.87	3.88	0.50	1.90	524.9
	pr11-20	1.26	3.63	0.99	2.73	619.0
3	c100	0.11	0.73	0.11	0.42	85.5
	r100	0.22	1.47	0.08	0.33	61.9
	rc100	0.36	1.41	-0.01	0.06	60.6
	c200	0.00	0.01	0.21	0.39	241.7
	r200	0.03	0.11	0.03	0.15	322.1
	rc200	0.11	0.32	0.30	0.51	404.3
	pr01-10	1.51	3.63	1.27	2.78	473.3
	pr11-20	1.78	3.35	1.77	3.20	517.8
4	c100	0.29	1.24	0.28	0.97	81.9
	r100	0.30	1.54	0.10	0.94	61.2
	rc100	0.45	2.32	-0.09	0.59	58.5
	c200	0.00	0.00	0.00	0.00	104.9
	r200	0.00	0.00	0.00	0.00	151.2
	rc200	0.00	0.00	0.00	0.00	164.8
	pr01-10	1.87	3.57	2.13	3.61	403.3
	pr11-20	2.31	3.49	2.50	3.88	408.4
Grand Mean		0.48	1.42	0.38	0.95	375.8

Table 16: New best known solution values found by ILS using VNS computational time

Instance	m	Old BK	New BK	Instance	m	Old BK	New BK	Instance	m	Old BK	New BK
r203	1	1021	1028	r211	1	1046	1051	pr18	2	938	953
r206	1	1029	1031	c204	2	1480	1490	pr20	2	1232	1237
r207	1	1072	1075	r206	2	1440	1443	r108	4	994	995
r208	1	1112	1117	r209	2	1405	1414	rc103	4	974	975
r209	1	950	958	rc207	2	1587	1589	rc107	4	980	987
r210	1	987	991	pr09	2	905	908				

Table 17: I3CH and ILS

Methods	*Numb*	%
I3CH $\sim$ ILS	120	39.5
I3CH $\succ$ ILS	90	29.6
I3CH $\prec$ ILS	94	30.9

Table 18: VNS and ILS

Methods	*Numb*	%
VNS $\sim$ ILS	170	55.9
VNS $\succ$ ILS	46	15.1
VNS $\prec$ ILS	88	28.9

is able to outperform I3CH in 94 instances (30.9%) while I3CH performs better in 90 instances (29.6%). For the rest of the "INST-M" instances (120 instances), both are able to provide the same best found solutions. When comparing with VNS, ILS can outperform VNS in 88 instances (28.9%). Both obtain the same best found solutions for 170 instances (55.9%).

Table 19 presents the updated best known values obtained by three different scenarios: ILS using 272 seconds, I3CH and VNS computational times, respectively. For example, instance r209 was initially improved from 1405 to 1406 by using 272 seconds. This solution is further improved to 1407 and 1414 by using I3CH and VNS computational times, respectively. Finally, we can conclude that ILS is able to improve 31 best known solution values of benchmark instances.

Table 19: New best known solution values found by three different scenarios

Instance	m	Old *BK*	ILS (272 seconds)	ILS (I3CH computational time)	ILS (VNS computational time)
r203	1	1021	1026	-	1028
r208	1	1112	1113	-	1117
r209	1	950	956	-	958
r211	1	1046	1049	-	1051
r206	2	1440	-	1442	1443
r209	2	1405	1406	1407	1414
pr18	2	938	-	941	953
rc107	4	980	985	-	987

5.3.2. Real-world Instances

The last set of instances in Table 2 summarizes a set of real-world instances based on Singapore's tourism attractions (Point Of Interests / POI). This real-world problem represents the Tourist Trip Design Problem (TTDP) as an application of the extended TOPTW. It is assumed that the user preference on a particular POI has been translated to the score between 0 to 7 (the higher the value the more preferable a POI is). We generate different instances to simulate different user behaviours by varying three different inputs:

- Number of days (routes) of each trip, $|M|$.
- Start and end nodes of each day (route) m, N^{start} and N^{end}.
- Length of each day (route) m, T_m^{max}.

Table 20: Real-world instances

Instance Set	$\|M\|$	Route m	Start node	End node	Start time	End time	T_m^{max} (mins)
$R1$	1	1	Hotel A (Marina Bay area)	Hotel B (China Town area)	9.00	18.00	540
$R2$	2	1	Hotel A (Marina Bay area)	Hotel C (Bugis Area)	9.00	18.00	540
		2	Hotel A (Marina Bay area)	Hotel D (Orchard area)	9.00	18.00	540
$R3$	3	1	Hotel A (Marina Bay area)	Hotel B (China Town area)	9.00	18.00	540
		2	Hotel A (Marina Bay area)	Hotel D (Orchard area)	9.00	18.00	540
		3	Hotel D (Orchard area)	Hotel D (Orchard area)	9.00	18.00	540
$R4$	4	1	Hotel A (Marina Bay area)	Hotel B (China Town area)	9.00	18.00	540
		2	Hotel A (Marina Bay area)	Hotel C (Bugis Area)	9.00	18.00	540
		3	Hotel A (Marina Bay area)	Hotel E (East area)	9.00	18.00	540
		4	Hotel E (East area)	Hotel E (East area)	9.00	15.00	360
$R5$	5	1	Hotel A (Marina Bay area)	Hotel B (China Town area)	9.00	18.00	540
		2	Hotel A (Marina Bay area)	Hotel C (Bugis Area)	9.00	18.00	540
		3	Hotel A (Marina Bay area)	Hotel F (Sentosa area)	9.00	18.00	540
		4	Hotel F (Sentosa area)	Hotel E (East area)	9.00	18.00	540
		5	Hotel E (East area)	Hotel E (East area)	9.00	15.00	360

Table 21: Real-world solution values by ILS

Instance	$\|M\|$	Total Score	Route m	Number of POIs visited
$R1$	1	51	1	11
$R2$	2	87.5	1	10
			2	10
$R3$	3	117	1	8
			2	9
			3	9
$R4$	4	112	1	10
			2	8
			3	6
			4	1
$R5$	5	133.5	1	10
			2	7
			3	8
			4	3
			5	1

For illustration purpose, Table 20 shows the details of instances. Each instance has a different value of $|M|$. The start and end nodes may vary, depends on the request of the user. The user may have an appointment after the trip or he / she may want to stay in another hotel which is closer to a certain location, e.g. the

Table 22: The detailed routes of $R3$

Route	POIs visited
1	(1) Hotel A (Marina Bay area) → (2) Former Empress Place Building → (3) Sri Mariamman Temple → (4) Chinatown Heritage Centre → (5) Former Singapore Conference Hall and Trade Union House → (6) Former St James Power Station → (7) Capella Singapore → (8) Church of St Teresa → (9) WANGZ Hotel → (10) Hotel B (China Town area)
2	(1) Hotel A (Marina Bay area) → (2) Esplanade Park Memorials → (3) Thian Hock Kheng Temple → (4) Keng Teck Whay (5) Tanjong Pagar Railway Station → (6) Bowyer Block → (7) Tan Teck Guan Building → (8) College of Medicine Building (9) Singapore Tyler Print Institute → (10) MacDonald House → (11) Hotel D (Orchard area)
3	(1) Hotel D (Orchard area) → (2) Former Raffles College → (3) Command House → (4) St Joseph's Church (5) Malay Heritage Centre → (6) Church of St Peter and St Paul → (7) Church of Our Lady of Lourdes → (8) Civilian War Memorial (9) Civil Defence Heritage Gallery → (10) The Cathay → (11) Hotel D (Orchard area)

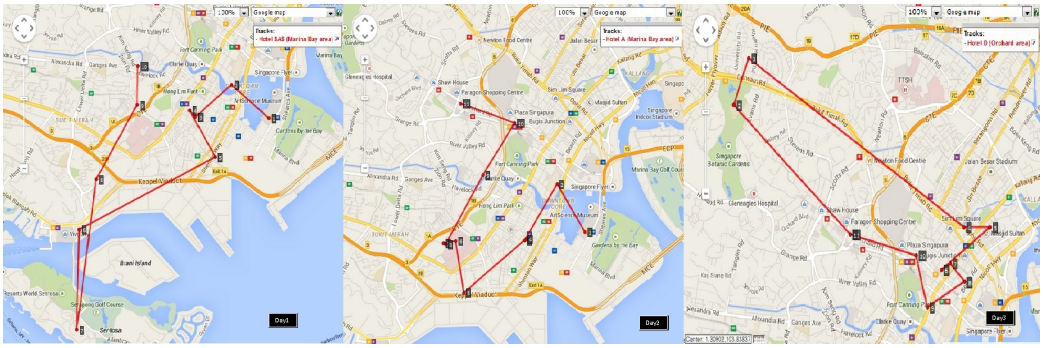

Figure 2: Example Routes of $R3$

airport. The start and end times are also different. The user may want to catch his / her flight therefore the last day should be shorter. Take note that T_m^{max} values are discretized by minutes. We assume that the user is planning the trips for visiting some historical sites, including museum, landmarks and memorials.

Table 21 summarizes the results obtained for five different instances. We only run the proposed algorithm once for each instance. The main purpose is to generate the trip for each route m within reasonable computational time (e.g. 5 seconds). The second column gives the value of $|M|$ for each instance. The third column shows the total score obtained. The last two columns summarize the number of POIs visited for each route m. As we observe for m = 1 to 3, the higher the value of m, the more POIs are visited. The number of POIs visited is slightly decreased for m = 4 since the user is moved to a hotel which is closer to the airport and far from most of POIs. We conclude that ILS is able to solve different scenarios within reasonable computational time.

Table 22 shows one particular route ($R3$) for a three-days trip generated by ILS. Figure 2 visualizes the proposed routes. As mentioned earlier, the inputs can

always be adjusted according to the user preference and information given. We conclude that ILS is effective for solving real-world problems related to the TTDP within a few seconds.

6. CONCLUSION

In this paper, we studied the extended version of the Team Orienteering Problem with Time Windows (TOPTW) by incorporating some additional real-world constraints, such as different maximum total travel times and different start and end nodes.

A simple algorithm based on Iterated Local Search (ILS) is proposed. We implement a factorial experimental design to tune the parameter values of ILS. By doing so, the parameter search space is reduced so our search focus on the promising regions of the important parameter search space. Computational results have shown that the proposed ILS algorithm is an effective algorithm. It can improve 31 best known solution values among benchmark instances of the TOPTW ($\approx 9\%$). These new results can serve as benchmarks for future studies. Our algorithm has also been implemented to real-world problems which are related to the Tourist Trip Design Problems (TTDP), the extension of the TOPTW. We conclude that the proposed algorithm is able to provide effective solutions within a few seconds.

The future focus is to improve the performance of ILS by modifying it in order to solve larger values of m, e.g. focus on allocating nodes with tight time windows that are more difficult to allocate. Since the TOPTW is highly constrained, the oscillation strategy that allows infeasible solutions during the search process can be considered.

It would be interesting to consider applying the proposed algorithm to other variants of the Orienteering Problem, e.g. the Time Dependent Orienteering Problem (TDOP). In the TDOP, the major different is the travel time between two nodes depends on the departure time at the first node. On the other hand, there is no time window constraints. In the real-world problems, more constraints, such as certain POIs have to be visited in certain time periods, can also be considered for future work. Last but not least, future work would also consider the uncertainty or stochastic aspects, such as stochastic profits and travel times between two nodes.

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

Boussier, S., Feillet, D., Gendreau, M., 2007. An exact algorithm for the team orienteering problem. 4OR 5 (3), 211–230.

Chao, I.-M., Golden, B. L., Wasil, E. A., 1996. A fast and effective heuristic for the orienteering problem. European Journal of Operational Research 88 (3), 475–489.

Cordeau, J. F., Grendreau, M., 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., Medaglia, A. L., 2015. Solving the orienteering problem with time windows via the pulse framework. Computers and Operations Research 54, 168–176.

Garcia, A., Vansteenwegen, P., Arbelaitz, O., Souffriau, W., Linaza, M. T., 2013. Integrating public transportation in personalised electronic tourist guides. Computers and Operations Research 40 (3), 758–774.

Garcia, A., Vansteenwegen, P., Souffriau, W., Arbelaitz, O., Linaza, M. T., 2009. Solving multiconstrained team orienteering problems to generate tourist routes. Tech. rep., Centre for Industrial Management, Katholieke Universiteit Leuven, Leuven, Belgium.

Goldberg, D. E., 1989. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, Massachusetts.

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

Hu, Q., 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, H. H., Leyton-Brown, K., Stützle, T., 2009. ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36, 267–306.

Johnson, S. M., 1963. Generation of permutations by adjacent transposition. Mathematics of Computation 17 (83), 282–285.

Labadie, N., Mansini, R., Melechovskỳ, J., Calvo, R. W., 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, J., Calvo, R. W., 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., Yu, V. F., 2012. A simulated annealing heuristic for the team orienteering problem with time windows. European Journal of Operational Research 217 (1), 94–107.

Lozano, L., Duque, D., Medaglia, A. L., 2014. An exact algorithm for the elementary shortest path problem with resource constraints. Transportation Science. URL http://hdl.handle.net/1992/1181

Montemanni, R., Gambardella, L. M., 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., Gambardella, L. M., 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). pp. 381–384.

Ridge, E., Kudenko, D., 2007. Tuning the performance of the MMAS heuristic. In: St*ü*tzle, T., Birattari, M., Hoos, H. H. (Eds.), proceedings of the workshop of Stochastic local search algorithms (SLS2007). Vol. 4638 of Lecture Notes in Computer Science. Springer, pp. 46–60.

Righini, G., Salani, M., 2008. New dynamic programing algorithms for the resource constrained elementary shortest path problem. Networks 51 (3), 155–170.

Righini, G., 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.

Schilde, M., Doerner, K. F., Hartl, R. F., Kiechle, G., 2009. Metaheuristics for the biobjective orienteering problem. Swarm Intelligence 3 (3), 179–201.

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., Vanden Berghe, G., Van Oudheusden, D., 2013. The multiconstraint team orienteering problem with multiple time windows. Transportation Science 47 (1), 1–11.

Tricoire, F., Romauch, M., Doerner, K. F., Hartl, R. F., 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., Oudheusden, D. V., 2011a. The orienteering problem: A survey. European Journal of Operational Research 209 (1), 1–10.

Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., 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., Vanden Berghe, G., Van Oudheusden, D., 2011b. The city planner: An expert system for tourists. Expert Systems with Applications 38 (6), 6540–6546.