Computers & Operations Research 36 (2009) 3281 – 3290

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

Iterated local search for the team orienteering problem with time windows

Pieter Vansteenwegen[a,*], Wouter Souffriau[a,b], Greet Vanden Berghe[b], Dirk Van Oudheusden[a]

[a]Centre for Industrial Management, Katholieke Universiteit Leuven, Celestijnenlaan 300A-bus 2422, 3001 Leuven, Belgium
[b]Information Technology, Katholieke Hogeschool Sint-Lieven, Gebroeders Desmetstraat 1, 9000 Gent, Belgium

ARTICLE INFO

Available online 24 March 2009

Keywords:
Iterated local search
Team orienteering problem with time windows
Meta-heuristics

ABSTRACT

A personalised electronic tourist guide assists tourists in planning and enjoying their trip. The planning problem that needs to be solved, in real-time, can be modelled as a team orienteering problem with time windows (TOPTW). In the TOPTW, a set of locations is given, each with a score, a service time and a time window. The goal is to maximise the sum of the collected scores by a fixed number of routes. The routes allow to visit locations at the right time and they are limited in length. The main contribution of this paper is a simple, fast and effective iterated local search meta-heuristic to solve the TOPTW. An insert step is combined with a shake step to escape from local optima. The specific shake step implementation and the fast evaluation of possible improvements, produces a heuristic that performs very well on a large and diverse set of instances. The average gap between the obtained results and the best-known solutions is only 1.8% and the average computation time is decreased with a factor of several hundreds. For 31 instances, new best solutions are computed.

1. Introduction

For tourists visiting a city or region during one or more days it is impossible to visit everything they are interested in. Therefore, tourists have to select what they believe to be the most valuable attractions. Making a feasible plan in order to visit the most interesting attractions in the available time span is often a difficult task. Besides, as soon as tourists deviate from their original plan and it becomes infeasible, they have to start all over again to find an attractive plan for the remaining part of their trip. We assume that many tourists might appreciate the assistance of a personalised electronic tourist guide (PET) when planning their trip. A PET is a hand-held device that presents a trip that maximises the satisfaction of the tourist, taking into account the location of the attractions, the opening and closing hours, the tourist interest "value", the available time and the traveling time. Typically, the PET has to solve tourist trip design problems (TTDP) [1]. The team orienteering problem with time windows (TOPTW) is a simplified version of the TTDP. In the TOPTW a set of locations is given, each with a score, a service time and a time window. The goal is to maximise the sum of the collected scores by a fixed number of routes. Each route may be interpreted as a day trip. Thus, a route visits locations at the right time and is limited in length.

The routing inside a PET is subject to an additional requirement: it has to be calculated in real-time in order to react to tourist actions and preferences or unexpected events. In order to obtain high quality results for these difficult planning problems, in only a few seconds, a meta-heuristic approach is required. For instance, when tourists spend longer than planned in a museum or, unexpectedly, a certain attraction appears to be closed, they do not want to wait minutes for a modified plan to become available.

The main contribution of this paper is an algorithm that obtains high quality results for the TOPTW in very limited computation time. This is achieved by speeding up the evaluation of possible improvements and by better exploring the whole solution space. As a consequence, the algorithm is suitable for the PET application. Furthermore, new best solutions are computed for many test instances.

In the next section a literature overview is presented and in Section 3 a rigorous problem definition is given. In Section 4 the heuristic is described in detail and in Section 5 experimental results are presented. Conclusions and further work are discussed in Section 6.

2. Literature review

The (team) orienteering problem (*without* time windows) is discussed extensively in the literature. The orienteering problem [2] is also known as the selective travelling salesperson problem [3], the maximum collection problem [4] and the bank robber problem [5].

* Corresponding author. Tel.: +32 16 32 25 67; fax: +32 16 32 29 86.
E-mail addresses: Pieter.Vansteenwegen@cib.kuleuven.be (P. Vansteenwegen), Wouter.Souffriau@kahosl.be (W. Souffriau), Greet.VandenBerghe@kahosl.be (G. Vanden Berghe), Dirk.VanOudheusden@cib.kuleuven.be (D. Van Oudheusden).

0305-0548/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2009.03.008

Furthermore, the OP can be formulated as a special case of the resource constrained TSP [6], a TSP with profits [7] or as a resource constrained elementary shortest path problem [8].

Many (T)OP applications are described in the literature: the sport game of orienteering [9], the home fuel delivery problem [10], athlete recruiting from high schools [4], routing technicians to service customers [11], etc. Usually, most of these applications also require time windows and, as a consequence, the TOPTW model can be applied to many real life situations.

The best performing heuristics for the TOP are described in Archetti et al. [12], Vansteenwegen et al. [13], Ke et al. [14] and Souffriau et al. [15]. These algorithms, however, cannot efficiently solve (T)OP with time windows. All of them apply local search moves that become useless when time windows have to be considered.

Boussier et al. [16] describe an exact solution method to solve small and medium size TOP instances to optimality, as well as the selective vehicle routing problem with time windows, which is a generalisation of the TOPTW. A detailed and extended review of many algorithms for the OP and the TOP is provided in Tang and Miller-Hooks [11].

Many articles have been published regarding vehicle routing with time windows. Yet, only five articles deal with the orienteering problem with time windows [8,17–20] and only two [16,21] consider (a generalisation of) the *team* orienteering problem with time windows.

Kantor and Rosenwein [17] were the first to solve the OPTW. They first describe a straightforward insertion heuristic. The location with the highest ratio "score over insertion time" is inserted into the tour, without violating any of the time windows. Secondly, a depth-first search tree algorithm is proposed that constructs partial tours, using the insertion heuristic and beginning in the start location. Partial routes are abandoned if they are infeasible or if they are unlikely to yield the best total score. Righini and Salani [8,20] apply bi-directional dynamic programming to solve OPTW to optimality. Starting forward from the start point and backwards from the end point, current states are extended by adding an extra location at the end. Forward and backward states are matched if feasible and dominance tests are applied to record only non-dominated states. Decremental state space relaxation [22] is used to reduce the number of states to be explored. Mansini et al. [19] developed a simple constructive heuristic and a granular variable neighbourhood search (GVNS) for a variant of the OPTW in which the starting and end point are the same. The granular VNS improves a VNS algorithm by reducing the size of the analysed neighbourhoods by preventing the insertion of non-promising arcs. Bar-Yehuda et al. [18] also mention the OPTW, but only for the special cases where all locations have the same score and are on a line or in the Euclidean plane. They do not consider time limits on the tour duration. Montemanni and Gambardella [21] based their algorithm on an ant colony system (ACS). The method is based on the solution of a hierarchic generalisation of the TOPTW. This algorithm clearly outperforms the algorithm of Mansini et al. [19] on all considered OPTW instances. The data sets of Montemanni and Gambardella will be used to validate the performance of the iterated local search (ILS) heuristic (Section 5). The quality of the ILS results and the computation time will be compared with the optimal results obtained by Righini and Salani [8,20] and the heuristic results of Montemanni and Gambardella [21].

3. Formulation as a mathematical problem

In the OPTW a set of n locations is given: each location $i = 1, \dots, n$ is assigned a score S_i, a service or visiting time T_i and a time window $[O_i, C_i]$. The starting point (location 1) and the end point (location n) of every tour are fixed. The time t_{ij} needed to travel from location i to j is known for all locations. Not all locations can be visited since the available time is limited to a given time budget T_{max}. The OPTW goal is to determine a single route, limited by T_{max}, that visits some of the locations during the corresponding time windows, and at the same time maximises the total collected score. Each location can be visited at most once and it is allowed to wait at a location before its time window starts. The TOPTW is an OPTW where the goal is to determine m routes, each limited by T_{max}, that maximises the total collected score.

Based on the notation introduced above, the TOPTW can be formulated as an integer program ($x_{ijd} = 1$ if, in route d, a visit to location i is followed by a visit to location j, 0 otherwise; $y_{id} = 1$ if location i is visited in route d, 0 otherwise; s_{id} = the start of the service at location i in route d; M a large constant):

$$\text{Max} \sum_{d=1}^{m} \sum_{i=2}^{n-1} S_i y_{id} \tag{0}$$

$$\sum_{d=1}^{m} \sum_{j=2}^{n-1} x_{1jd} = \sum_{d=1}^{m} \sum_{i=2}^{n-1} x_{ind} = m \tag{1}$$

$$\sum_{i=1}^{n-1} x_{ikd} = \sum_{j=2}^{n} x_{kjd} = y_{kd} \quad (k = 2, \dots, n-1;\, d = 1, \dots, m) \tag{2}$$

$$s_{id} + T_i + c_{ij} - s_{jd} \leqslant M(1 - x_{ijd}) \quad (i, j = 1, \dots, n;\ d = 1, \dots, m) \tag{3}$$

$$\sum_{d=1}^{m} y_{kd} \leqslant 1 \quad (k = 2, \dots, n-1) \tag{4}$$

$$\sum_{i=1}^{n-1} \left(T_i y_{id} + \sum_{j=2}^{n} c_{ij} x_{ijd} \right) \leqslant T_{\max} \quad (d = 1, \dots, m) \tag{5}$$

$$O_i \leqslant s_{id} \quad (i = 1, \dots, n;\, d = 1, \dots, m) \tag{6}$$

$$s_{id} \leqslant C_i \quad (i = 1, \dots, n;\, d = 1, \dots, m) \tag{7}$$

$$x_{ijd}, y_{id} \in \{0, 1\} \quad (i, j = 1, \dots, n;\, d = 1, \dots, m) \tag{8}$$

The objective function (0) maximises the total collected score. Constraint (1) guarantees that all tours start from location 1 and end at location n. Constraints (2) and (3) determine the connectivity and timeline of each tour. Constraints (4) ensure that every location is visited at most once and constraints (5) limit the time budget. Constraints (6) and (7) restrict the start of the service to the time window.

Mathematical formulations of the TOP (without time windows) can be found in Butt and Cavalier [4], Tang and Miller-Hooks [11] and Boussier et al. [16].

4. Iterated local search heuristic

The TOPTW is a highly constrained problem and very difficult to solve. Since Golden et al. [10] prove that the OP is NP-hard, it is highly unlikely that the TOPTW can be solved to optimality within polynomial time. For the personalised electronic tourist guide, it is required to solve TOPTW with high quality in only a few seconds.

Gendreau et al. [23] discuss a few reasons why it is so difficult to design high-quality heuristics for the (T)OP. The score of a location and the distance to reach it are independent and often in opposition to each other. This makes it very difficult to select the locations that will be part of the optimal solution. Therefore, simple construction and improvement heuristics may direct the algorithm in undesirable directions. They do not sufficiently scrutinise large parts of the solution landscape and bad decisions cannot be corrected satisfactorily. The time windows further complicate the solution process.

Nevertheless, a straightforward and fast iterated local search heuristic, performing very well on the available data sets, has been developed. The heuristic combines an insertion step and a shaking step to escape from local optima.

4.1. Insertion step

The insertion step tries to add, one by one, new visits to a tour. Before an extra visit can be inserted in a tour, it should be verified that all visits scheduled after the insertion place still satisfy their time window. In order to develop a fast heuristic, a quick evaluation of each possible insert move is necessary. Checking all other visits on their feasibility would take much time. This can be avoided by recording *Wait* and *MaxShift* for each already included location. *Wait* is defined as the waiting time in case the arrival at a location (a_i) takes place before the time window. The service itself can only start when the time window opens. If the arrival takes place during the time window, *Wait* equals zero.

$$Wait_i = \max[0, O_i - a_i] \tag{10}$$

MaxShift is defined as the maximum time the service completion of a given visit can be delayed, without making any visit infeasible. *MaxShift* of location *i* is equal to the sum of *Wait* and *MaxShift* of the next location *i*+1, unless *MaxShift* is limited by its own time window (C_i):

$$MaxShift_i = \min[C_i - s_i, Wait_{i+1} + MaxShift_{i+1}] \tag{9}$$

Recording *MaxShift* enables the evaluation of a possible insert move in constant time instead of in linear time.

The total time consumption ($Shift_j$) to insert an extra visit *j* between visits *i* and *k*, is defined as

$$Shift_j = c_{ij} + Wait_j + T_j + c_{jk} - c_{ik} \tag{11}$$

For a feasible insertion of *j* between *i* and *k*, $Shift_j$ should be limited to the sum of $Wait_k$ and $MaxShift_k$ of visit *k*. This gives the following formula to check feasibility:

$$Shift_j = c_{ij} + Wait_j + T_j + c_{jk} - c_{ik} \leqslant Wait_k + MaxShift_k$$

Service *j* should also fit the time window of location *j*.

For each visit the lowest possible *Shift* is determined, i.e. the best possible insert position. Then, in order to determine the visit that will be selected for insertion, a ratio is calculated for each visit:

$$Ratio_i = (S_i)^2 / Shift_i$$

The visit with the highest ratio will be selected for insertion.

Due to the time windows, the time consumption of an insertion (*Shift*) becomes less relevant than the score when deciding which visit is the most promising to insert next. Therefore, the square of the score is applied in the ratio calculation. If the square is not applied, the obtained results are worse, as will be shown in Section 5.

Fig. 1 presents the pseudo code for the insertion step. After insertion, all other visits should be updated. Visits after the insertion

```
For each non included visit:
|       Determine the best possible insert position and Shift;
|       Calculate Ratio;
Insert visit with highest ratio (j);
Visit j: calculate Arrive, Start, Wait;
For each visit after j (until Shift == 0):
|       Update Arrive, Start, Wait, MaxShift, Shift;
Visit j: update MaxShift;
For each visit before j:
|       Update MaxShift;
```

Fig. 1. Pseudo code for the insertion step.

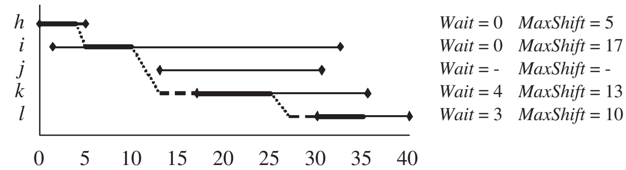

Fig. 2. Tour *h–i–k–l*.

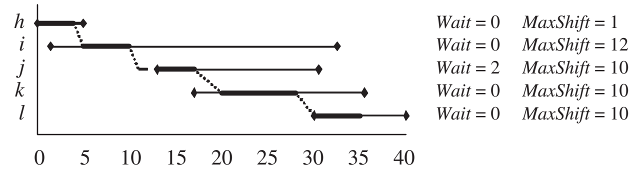

Fig. 3. Tour *h–i–j–k–l*.

require an update of the waiting time (*Wait*), the arrival time (*a*), the start of the service (*s*) and *MaxShift*. Every time a visit possesses a waiting time, the time shift for the start of that service and all following services will be reduced by this waiting time.

The following formulas are used to update the visits after the insertion position, when visit *j* is inserted between *i* and *k*:

$$Shift_j = c_{ij} + Wait_j + T_j + c_{jk} - c_{ik}$$

$$Wait_{k^*} = \max[0, Wait_k - Shift_j]$$

$$a_{k^*} = a_k + Shift_j$$

$$Shift_k = \max[0, Shift_j - Wait_k]$$

$$s_{k^*} = s_k + Shift_k$$

$$MaxShift_{k^*} = MaxShift_k - Shift_k$$

$Shift_k$ and the same formulas are then used to update the visits after *k*, one after another, until *Shift* is reduced to zero.

Visits before the insertion may require an update of *MaxShift*, using formula (9) mentioned above.

By way of illustration, a small example is discussed (see Figs. 2 and 3). Visits *h* ($O_h = 0$, $C_h = 5$, $T_h = 4$, $a_h = s_h = 0$), *i* ($O_i = 1$, $C_i = 33$, $T_i = 5$, $a_i = s_i = 5$), *k* ($O_k = 17$, $C_k = 36$, $T_k = 8$, $a_k = 13$, $s_k = 17$) and *l* ($O_l = 30$, $C_l = 40$, $T_l = 5$, $a_l = 27$, $s_l = 30$) are included in the original tour in Fig. 2.

The feasibility check shows that visit *j* ($O_j = 13$, $C_j = 31$, $T_j = 5$) can be inserted between *i* and *k*:

$$Shift_j = c_{ij} + Wait_j + T_j + c_{jk} - c_{ik} = 1 + 2 + 4 + 3 - 3 \\ \leqslant 4 + 8 = Wait_k + MaxShift_k$$

Because visit *j* is inserted, *k* and *l* need to be updated. The arrival at location *k* ($a_k = 13$) is delayed by the total time consumption of the insertion ($Shift_j = 7$), thus $a_{k^*} = 20$. Due to $Wait_k$ (4) the time shift is reduced to 3 ($= Shift_k$) and the start of the service itself ($s_k = 17$) is only delayed by $Shift_k$ ($s_{k^*} = 20$). Furthermore, $MaxShift_k$ is reduced by $Shift_k$. Since $Wait_l$ (3) is equal to or bigger than $Shift_k$, s_l and $MaxShift_l$ are unchanged; only $Wait_{l^*}$ (0) and a_{l^*} (30) need to be updated. Any visit scheduled after *l* will not be affected by the insertion of *j*. Applying the abovementioned formulas:

$$Wait_{k^*} = \max[0, Wait_k - Shift_j] = \max[0, 4 - 7] = 0$$

$$a_{k^*} = a_k + Shift_j = 13 + 7 = 20$$

$$Shift_k = \max[0, Shift_j - Wait_k] = \max[0, 7 - 4] = 3$$

```
For each tour:
|       Delete the set of visits (i => j);
|       Calculate Shift;
|       For each visit after j (until Shift == 0):
|        |       Shift visit towards the beginning of the tour;
|        |       Update Arrive, Start, Wait, MaxShift, Shift;
|       For each visit before i:
|        |       Update MaxShift;
```

Fig. 4. Pseudo code for the shake step.

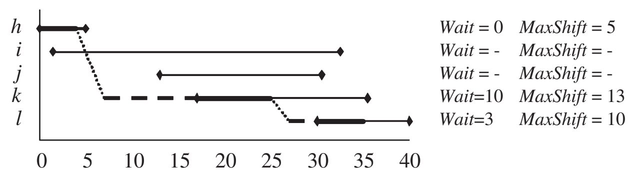

Fig. 5. Tour *h–k–l*.

$$s_{k^*} = s_k + Shift_k = 17 + 3 = 20$$

$$MaxShift_{k^*} = MaxShift_k - Shift_k = 13 - 3 = 10$$

$$Wait_{l^*} = \max[0, Wait_l - Shift_k] = \max[0, 3 - 3] = 0$$

$$a_{l^*} = a_l + Shift_k = 27 + 3 = 30$$

$$Shift_l = \max[0, Shift_k - Wait_l] = \max[0, 3 - 3] = 0$$

For visits *h* and *i*, *MaxShift* needs to be updated. For visit *i* in the example, $MaxShift_{i^*}$ is equal to the sum of $Wait_{j^*}$ and $MaxShift_{j^*}$. Due to the time window of location *h*, the start of service *h* can only be delayed by five time units. Thus $MaxShift_{h^*}$ remains equal to five and the insertion of visit *j* has no impact on a visit before visit *h*. The result can be seen in Fig. 3.

4.2. Shake step

The shake step is used to escape local optima. The pseudo code for the shake step is presented in Fig. 4. During this step, one or more visits will be removed in each tour. Each shake step uses two integers as input, the first integer indicates how many consecutive visits to remove in a tour (R_d), the second indicates the place in the tour to start the removing process (S_d). If during removal the end location is reached, it continues after the start location. Due to different tour lengths, the value of S_d will become different for different tours, during the execution of the algorithm. Different S_d values in different tours increase the possibility to escape from local optima.

After the removal, all visits following the removed visits are shifted towards the beginning of the tour, in order to avoid unnecessary waiting. If a visit cannot be shifted due to its time window, that visit and the visits scheduled after it remain unchanged. The shifted visits should be updated similarly to the process described for the insertion step. For the visits before the removed visits, only *MaxShift* should be updated.

If the shake step is applied on the previous example (Fig. 3), with *S* and *R* both equal to two, visits *i* and *j* are removed and the result is *h–k–l* (Fig. 5). Visit *k* can start three time units earlier after waiting ten units. Visit *l* cannot start earlier due to the time window; the arrival is thus earlier, but the start of the service will not change. $MaxShift_h$ is not altered due to the time window of visit *h*. Due to the time windows of visits *h* and *l*, no visits before visit *h* or after visit *l* change because of this removal. The large values $Wait_k$ and $MaxShift_k$ can now be used to visit another (combination of) location(s) characterised by a higher score than visits *i* and *j* combined.

```
S ← 1;
R ← 1;
NumberOfTimesNoImprovement ← 0;
while NumberOfTimesNoImprovement < 150 do
 |       while not local optimum do
 |        |       Insert;
 |       If Solution better than BestFound then
 |        |       BestFound ← Solution;
 |        |       R ← 1;
 |        |       NumberOfTimesNoImprovement ← 0;
 |       Else
 |        |       NumberOfTimesNoImprovement
 |        |                 ← NumberOfTimesNoImprovement+1;
 |       Shake Solution (R, S);
 |       S ← S + R;
 |       R ← R+1;
 |       If S>=Size of smallest Tour then
 |        |       S ← S - Size of smallest Tour;
 |       If R==n/(3*m) then
 |        |       R ← 1;
Return BestFound;
```

Fig. 6. Pseudo code for the ILS heuristic.

4.3. Heuristic

Fig. 6 presents the heuristic's pseudo code. The heuristic starts with a set of empty tours and initialises all the parameters of the shake step to one. The heuristic follows a loop until, during a fixed number of times, no improvements are identified for the best solution determined so far. Firstly, the insertion step is applied until a local optimum is reached. If this solution is better than the incumbent solution, the solution is recorded and *R* is reset to one for the next shake step. Secondly, the shake step is applied. After each shake step, *S* is increased by the value *R* and *R* is increased by one for the next shake step. If *S* is equal to or greater than the size of the smallest tour, this size is subtracted to determine the new position. If *R* equals $n/(3{*}m)$, it is reset to one.

This heuristic resembles what Lourenço et al. [24] call iterated local search: iteratively, a sequence of local search solutions is built up, instead of repeating random trials of the local search. In the past, ILS proved to work well on problems with time windows [25,26].

By using the shake parameters as described above, other visits are removed during every shake step and during the entire procedure, most likely, every visit is removed at least once. This proves to be an excellent technique to address the well known problems mentioned by Gendreau et al. [23] when using simple improvement heuristics. The entire solution space is now better explored and earlier taken wrong decisions are corrected. This property is enhanced by the fact that the heuristic always continues the search from the current solution, it never returns to the best found solution to continue. This procedure is called iterated local search with a random walk acceptance criterion [24].

The maximum number of locations to remove ($n/(3{*}m)$) and the maximum number of iterations without improvement (150) are the only parameters to predetermine in this heuristic. For the maximum number of locations to remove, a percentage of n/m (number of locations/number of routes) is used. Changing this percentage, from n/m to $n/(5{*}m)$, has no significant effect on the quality of the results or the computation time. It appeared that further increasing the maximum number of iterations without improvement does not significantly enhance the obtained results and only causes longer computation times.

5. Experimental results

5.1. Test instances

The heuristic was tested on the available test instances and compared with published results. Righini and Salani [20] designed 58 instances for the OPTW using Solomon's data set of vehicle routing problems with time windows (c*_100, r*_100 and rc*_100) [27] and 10 multi-depot vehicle routing problems of Cordeau et al. (pr1–pr10) [28]. Montemanni and Gambardella [21] added 27 extra instances based on Solomon (c*_200, r*_200 and rc*_200) and 10 instances based on Cordeau et al. (pr11–pr20). The number of possible visits of the Cordeau et al. instances varies between 48 and 288; all Solomon instances have 100 possible visits.

Since no TOPTW instances are available in the literature, Montemanni and Gambardella [21] designed new TOPTW instances. Instead of using only one tour, all aforementioned OPTW instances are also considered with two, three and four tours. However, no optimal solutions are available for these test instances. As a consequence, the performance of the ILS heuristic can only be compared with the best-known results calculated by the ant colony system of Montemanni and Gambardella [21]. The number of locations (48–288) and the number of tours (1–4) in these test instances, are good examples of the real-life problems that need to be solved by the PET.

However, to analyse the performance of the ILS heuristic more profoundly, a new data set with more difficult instances is constructed. Furthermore, for all these instances the optimal solution is known. This data set uses the original instances from Solomon [27] and Cordeau et al. [28], with the number of tours equal to the number of vehicles. Pr11–pr20 could not be used in this case, since these instances have multiple depots. With this number of tours it should be feasible to visit every location and hence the optimal result equals the sum of all scores. However, since these instances are now seen as orienteering problems, the algorithm is unaware of the possibility to visit all locations. It tries to select as many visits as possible and to design a feasible route between them. Since these instances have up to 20 tours they should be regarded as cases of intractable TOPTWs. Indeed, the difficulty of solving instances is not only related to the number of locations that can be visited in each tour, but also to the number of tours that can be generated.

It should be noticed that instances with time windows are very different from instances without time windows. As a consequence, TOP test sets cannot be used to verify the performance of the ILS heuristic. For instance, the well-known two-opt move [29] is indispensable to obtain high-quality results for the TOP, but due to the time windows, it cannot be applied to efficiently solve the TOPTW. The ILS algorithm, without a two-opt move, would not perform well in solving genuine TOP instances.

Boussier et al. [16] report on experimental results for the selective vehicle routing problem with time windows, a generalisation of the TOPTW. Compared to the TOPTW, two extra constraints are added, one related to the capacity of each vehicle and one related to the travel distance. The ILS heuristic of this paper cannot take these extra constraints into account. Therefore, the selective vehicle routing instances cannot be used to verify the performance of the ILS heuristic. Montemanni and Gambardella [21] also concluded that these possible alternative test instances cannot be used.

5.2. Results

All computations were carried out on a personal computer Intel Core 2 with 2.5 GHz processor and 3.45 GB Ram. Montemanni and

Table 1
Results for Solomon's test problems ($m = 1$).

Name	BK	ILS	Gap (%)	Visits	CPU (s)	Name	BK	ILS	Gap (%)	Visits	CPU (s)
c101	*320*	320	0.0	10	0.4	c201	870	840	3.4	27	1.1
c102	*360*	360	0.0	11	0.3	c202	930	910	2.2	31	2.8
c103	*400*	390	2.5	10	0.5	c203	960	940	2.1	31	1.7
c104	*420*	400	4.8	10	0.3	c204	970	950	2.1	31	1.6
c105	*340*	340	0.0	10	0.3	c205	910	900	1.1	30	1.2
c106	*340*	340	0.0	10	0.3	c206	920	910	1.1	30	1.6
c107	*370*	360	2.7	11	0.3	c207	920	910	1.1	30	2.1
c108	*370*	370	0.0	11	0.3	c208	950	930	2.1	31	1.6
c109	*380*	380	0.0	11	0.3						
r101	*198*	182	8.1	7	0.1	r201	797	788	1.1	37	1.2
r102	*286*	286	0.0	11	0.2	r202	903	880	2.5	47	1.4
r103	*293*	286	2.4	10	0.2	r203	993	980	1.3	50	1.6
r104	*303*	297	2.0	11	0.2	r204	1053	**1073**	−1.9	55	1.7
r105	*247*	247	0.0	11	0.1	r205	949	931	1.9	43	1.4
r106	*293*	293	0.0	11	0.2	r206	1008	996	1.2	48	1.5
r107	*299*	288	3.7	10	0.2	r207	1035	**1038**	−0.3	51	2.0
r108	*308*	297	3.6	11	0.2	r208	1071	1069	0.2	54	1.6
r109	*277*	276	0.4	11	0.2	r209	938	926	1.3	45	2.4
r110	*284*	281	1.1	11	0.3	r210	970	958	1.2	49	1.9
r111	*297*	295	0.7	11	0.2	r211	1016	**1023**	−0.7	49	1.6
r112	*298*	295	1.0	11	0.2						
rc101	*219*	219	0.0	9	0.2	rc201	795	780	1.9	34	1.0
rc102	*266*	259	2.6	9	0.2	rc202	932	882	5.4	38	1.3
rc103	*266*	265	0.4	11	0.3	rc203	979	960	1.9	45	2.7
rc104	*301*	297	1.3	11	0.3	rc204	1107	**1117**	−0.9	46	2.3
rc105	*244*	221	9.4	11	0.2	rc205	855	840	1.8	38	1.0
rc106	*252*	239	5.2	11	0.2	rc206	888	860	3.2	37	1.1
rc107	*277*	274	1.1	11	0.2	rc207	967	926	4.2	41	1.3
rc108	*298*	288	3.4	11	0.2	rc208	1040	1037	0.3	45	2.3
Average			1.9		0.2	Average			1.5		1.7
Max			9.4		0.5	Max			5.4		2.8

Table 2
Results for the test problems of Cordeau, Gendreau and Laporte ($m = 1$).

Name	BK	ILS	Gap (%)	Visits	CPU (s)	Name	BK	ILS	Gap (%)	Visits	CPU (s)
pr01	*308*	304	1.3	20	0.5	pr11	328	**330**	−0.6	20	0.3
pr02	*404*	385	4.7	20	0.6	pr12	437	431	1.4	25	0.9
pr03	*394*	384	2.5	21	1.0	pr13	442	**450**	−1.8	25	1.9
pr04	*489*	447	8.6	26	1.9	pr14	501	482	3.8	26	1.1
pr05	*595*	576	3.2	32	4.6	pr15	528	**638**	−20.8	34	5.3
pr06	567	538	5.1	26	2.5	pr16	525	**559**	−6.5	30	4.1
pr07	*298*	291	2.3	16	0.4	pr17	358	346	3.4	19	0.2
pr08	*463*	463	0.0	25	1.0	pr18	504	479	5.0	25	0.8
pr09	*493*	461	6.5	24	1.4	pr19	480	**499**	−4.0	30	2.7
pr10	591	539	8.8	28	3.6	pr20	556	**570**	−2.5	31	2.5
Average			4.3		1.8	Average			−2.3		2.0
Max			8.8		4.6	Max			5.0		5.3

Table 3
Results for Solomon's test problems ($m = 2$).

Name	BK	ILS	Gap (%)	Visits	CPU (s)	Name	BK	ILS	Gap (%)	Visits	CPU (s)
c101	590	590	0.0	21	1.4	c201	1460	1400	4.1	59	2.7
c102	660	650	1.5	22	0.9	c202	1460	1430	2.1	63	5.1
c103	710	700	1.4	22	1.2	c203	1460	1430	2.1	63	3.8
c104	760	750	1.3	22	1.5	c204	1440	**1460**	−1.4	65	4.2
c105	640	640	0.0	21	0.8	c205	1460	1450	0.7	64	3.1
c106	620	620	0.0	20	0.8	c206	1460	1440	1.4	63	2.8
c107	670	670	0.0	22	1.4	c207	1460	1450	0.7	64	3.2
c108	680	670	1.5	22	0.8	c208	1470	1460	0.7	65	2.8
c109	720	710	1.4	22	0.9						
r101	349	330	5.4	13	0.4	r201	1239	1231	0.6	70	2.1
r102	508	508	0.0	21	0.9	r202	1310	1270	3.1	77	2.3
r103	520	513	1.3	20	0.9	r203	1358	**1377**	−1.4	85	1.9
r104	544	539	0.9	22	1.5	r204	1404	**1440**	−2.6	97	3.4
r105	453	430	5.1	18	0.8	r205	1346	1338	0.6	85	2.8
r106	529	529	0.0	21	0.9	r206	1381	**1401**	−1.4	88	2.8
r107	529	529	0.0	21	1.0	r207	1400	**1428**	−2.0	91	1.7
r108	556	549	1.3	24	1.4	r208	1433	**1458**	−1.7	100	1.6
r109	506	498	1.6	22	0.5	r209	1361	1345	1.2	83	2.6
r110	525	515	1.9	22	1.0	r210	1360	**1365**	−0.4	83	1.9
r111	538	535	0.6	23	0.6	r211	1411	**1422**	−0.8	92	1.9
r112	543	515	5.2	21	0.5						
rc101	427	427	0.0	19	0.6	rc201	1376	1305	5.2	62	1.9
rc102	505	494	2.2	20	0.8	rc202	1472	1461	0.7	75	2.1
rc103	516	**519**	−0.6	20	1.1	rc203	1573	1573	0.0	85	2.0
rc104	575	565	1.7	22	0.7	rc204	1622	**1656**	−2.1	93	2.1
rc105	480	459	4.4	22	0.8	rc205	1428	1381	3.3	70	3.2
rc106	481	458	4.8	20	0.6	rc206	1514	1495	1.3	77	1.9
rc107	534	515	3.6	21	0.5	rc207	1544	1531	0.8	78	2.7
rc108	550	546	0.7	23	0.6	rc208	1646	1606	2.4	84	1.7
Average			1.6		0.9	Average			0.6		2.6
Max			5.4		1.5	Max			5.2		5.1

Gambardella [21] did five runs of their ACS on a comparable computer with a Dual AMD Opteron 250 2.4 GHz processor with 4 GB Ram. In order to illustrate the difficulty of solving the instances, it is worthwhile to mention that a commercial solver (CPLEX 11) could not solve to optimality any of the instances with two or more tours, within a computational time limit of 6 h.

Tables 1–8 compare the scores obtained by the ILS heuristic with the best-known solutions for the TOPTW instances. The best-known result for each instance is either the optimal solution or the best result out of five runs of the ant colony system [21]. Column one and two give the instance's name and the best-known solution (BK). If the best known solution was proven to be the optimal solution, it is indicated in italic. The third column presents the score obtained by the ILS heuristic. If the score is a new best-known solution, it is indicated in bold. The gap between the best-known solution and the ILS solution, in column four, is stated as a percentage. In column five the number of visited locations of the solution is presented and the computation time in the last column is expressed in seconds.

The average gap between the ILS result and the best-known solution for all these instances is only 1.8%. In the worst case the gap is 9.4% and in 31 cases a new best-known solution is obtained. For 78 instances the optimal solution is known, for 49 of those the ILS heuristic also obtains the optimal solution. These results clearly illustrate that high quality results are obtained for instances that correspond to personalised electronic tourist guide planning problems.

It appears that the ILS heuristic performs slightly better on instances with wider time windows (r/c/rc_200 and pr11–pr21), compared to instances with tighter time windows (r/c/rc_100 and pr1–pr10). The difference is small because another parameter of the instances has an opposite effect. It is clear that instances with more

Table 4
Results for test problems of Cordeau, Gendreau and Laporte ($m = 2$).

Name	BK	ILS	Gap (%)	Visits	CPU (s)	Name	BK	ILS	Gap (%)	Visits	CPU (s)
pr01	502	471	6.2	31	0.5	pr11	547	542	0.9	36	0.7
pr02	714	660	7.6	37	1.2	pr12	768	727	5.3	40	1.3
pr03	740	714	3.5	38	3.3	pr13	816	757	7.2	47	2.4
pr04	899	863	4.0	49	4.1	pr14	952	925	2.8	52	8.1
pr05	1034	1011	2.2	57	7.1	pr15	1120	**1126**	−0.5	62	8.2
pr06	995	**997**	−0.2	52	9.8	pr16	1119	1110	0.8	59	11.0
pr07	566	552	2.5	34	1.0	pr17	652	624	4.3	37	1.3
pr08	819	796	2.8	43	5.1	pr18	907	877	3.3	48	2.9
pr09	880	867	1.5	48	5.2	pr19	950	**955**	−0.5	56	5.5
pr10	1078	1004	6.9	59	10.3	pr20	1122	1056	5.9	61	10.7
Average			3.7		4.8	Average			3.0		5.2
Max			7.6		10.3	Max			7.2		11.0

Table 5
Results for Solomon's test problems ($m = 3$).

Name	BK	ILS	Gap (%)	Visits	CPU (s)	Name	BK	ILS	Gap (%)	Visits	CPU (s)
c101	810	790	2.5	29	1.1	c201	*1810*	1750	3.3	94	2.2
c102	920	890	3.3	32	2.1	c202	1790	1750	2.2	94	2.0
c103	980	960	2.0	33	2.2	c203	1760	1760	0.0	95	2.0
c104	1020	1010	1.0	34	1.3	c204	1770	**1780**	−0.6	97	1.5
c105	870	840	3.4	30	1.0	c205	1800	1770	1.7	96	2.5
c106	870	840	3.4	30	1.1	c206	1800	1770	1.7	96	1.5
c107	910	900	1.1	33	1.5	c207	1790	**1810**	−1.1	100	3.4
c108	920	900	2.2	33	1.2	c208	1800	**1810**	−0.6	100	2.4
c109	970	950	2.1	33	2.0						
r101	481	481	0.0	21	0.8	r201	1432	1408	1.7	92	2.4
r102	691	685	0.9	31	1.0	r202	1449	1443	0.4	98	2.7
r103	736	720	2.2	31	2.0	r203	1456	**1458**	−0.1	100	1.6
r104	773	765	1.0	34	1.5	r204	*1458*	1458	0.0	100	1.0
r105	620	609	1.8	27	2.3	r205	*1458*	1458	0.0	100	1.1
r106	722	719	0.4	32	2.1	r206	*1458*	1458	0.0	100	1.1
r107	757	747	1.3	33	1.1	r207	*1458*	1458	0.0	100	1.0
r108	790	790	0.0	36	3.1	r208	*1458*	1458	0.0	100	0.8
r109	710	699	1.5	31	1.8	r209	*1458*	1458	0.0	100	1.1
r110	737	711	3.5	32	1.4	r210	*1458*	1458	0.0	100	1.2
r111	770	764	0.8	34	1.8	r211	*1458*	1458	0.0	100	1.0
r112	769	758	1.4	34	1.1						
rc101	621	604	2.7	29	1.4	rc201	1681	1625	3.3	87	1.9
rc102	710	698	1.7	30	1.3	rc202	1706	1686	1.2	95	1.7
rc103	747	747	0.0	30	1.1	rc203	*1724*	1724	0.0	100	2.9
rc104	823	822	0.1	33	1.3	rc204	*1724*	1724	0.0	100	1.0
rc105	682	654	4.1	28	0.8	rc205	1698	1659	2.3	93	2.4
rc106	695	678	2.4	31	1.0	rc206	1722	1708	0.8	99	1.3
rc107	755	745	1.3	31	0.9	rc207	1722	1713	0.5	98	1.5
rc108	783	757	3.3	29	1.1	rc208	*1724*	1724	0.0	100	1.1
Average			1.8		1.5	Average			0.6		1.7
Max			4.1		3.1	Max			3.3		3.4

Table 6
Results for test problems of Cordeau, Gendreau and Laporte ($m = 3$).

Name	BK	ILS	Gap (%)	Visits	CPU (s)	Name	BK	ILS	Gap (%)	Visits	CPU (s)
pr01	619	598	3.4	42	0.4	pr11	649	632	2.6	43	0.5
pr02	942	899	4.6	57	3.9	pr12	985	902	8.4	60	1.8
pr03	999	946	5.3	55	3.9	pr13	1101	1046	5.0	64	8.2
pr04	1243	1195	3.9	66	9.0	pr14	1263	1197	5.2	69	8.3
pr05	1417	1356	4.3	79	12.5	pr15	1509	1488	1.4	86	14.6
pr06	1370	**1376**	−0.4	74	19.2	pr16	1516	1478	2.5	83	28.2
pr07	744	713	4.2	45	1.0	pr17	832	808	2.9	52	0.9
pr08	1118	1082	3.2	59	4.3	pr18	1229	1165	5.2	68	6.0
pr09	1227	1144	6.8	68	10.3	pr19	1320	1238	6.2	77	10.2
pr10	1492	1473	1.3	85	27.9	pr20	1505	**1514**	−0.6	87	18.2
Average			3.6		9.2	Average			3.9		9.7
Max			6.8		27.9	Max			8.4		28.2

Table 7
Results for Solomon's test problems ($m = 4$).

Name	BK	ILS	Gap (%)	Visits	CPU (s)	Name	BK	ILS	Gap (%)	Visits	CPU(s)
c101	1020	1000	2.0	39	3.8	c201	*1810*	1810	0.0	100	1.1
c102	1150	1090	5.2	43	1.8	c202	*1810*	1810	0.0	100	1.1
c103	1190	1150	3.4	44	2.5	c203	*1810*	1810	0.0	100	1.0
c104	1240	1220	1.6	45	3.0	c204	*1810*	1810	0.0	100	1.0
c105	1060	1030	2.8	40	1.8	c205	*1810*	1810	0.0	100	1.0
c106	1070	1040	2.8	40	2.1	c206	*1810*	1810	0.0	100	1.0
c107	1120	1100	1.8	43	2.0	c207	*1810*	1810	0.0	100	1.0
c108	1120	1100	1.8	44	3.6	c208	*1810*	1810	0.0	100	0.8
c109	1190	1180	0.8	45	2.5						
r101	608	601	1.2	28	1.4	r201	*1458*	1458	0.0	100	1.3
r102	836	807	3.5	39	1.7	r202	*1458*	1458	0.0	100	1.1
r103	909	878	3.4	42	2.2	r203	*1458*	1458	0.0	100	0.9
r104	957	941	1.7	45	3.8	r204	*1458*	1458	0.0	100	0.6
r105	771	735	4.7	35	2.9	r205	*1458*	1458	0.0	100	0.9
r106	893	870	2.6	41	3.5	r206	*1458*	1458	0.0	100	0.9
r107	937	927	1.1	44	3.3	r207	*1458*	1458	0.0	100	0.8
r108	994	982	1.2	47	3.2	r208	*1458*	1458	0.0	100	0.5
r109	879	866	1.5	40	2.1	r209	*1458*	1458	0.0	100	1.0
r110	908	870	4.2	42	2.0	r210	*1458*	1458	0.0	100	0.9
r111	944	935	1.0	45	2.0	r211	*1458*	1458	0.0	100	0.7
r112	954	939	1.6	44	3.1						
rc101	808	794	1.7	37	1.9	rc201	*1724*	1724	0.0	100	2.1
rc102	903	881	2.4	42	2.3	rc202	*1724*	1724	0.0	100	1.1
rc103	948	947	0.1	42	2.0	rc203	*1724*	1724	0.0	100	0.9
rc104	1052	1019	3.1	43	1.7	rc204	*1724*	1724	0.0	100	0.8
rc105	875	841	3.9	37	1.5	rc205	*1724*	1724	0.0	100	2.1
rc106	908	874	3.7	37	2.5	rc206	*1724*	1724	0.0	100	1.0
rc107	964	951	1.3	42	1.9	rc207	*1724*	1724	0.0	100	1.0
rc108	1007	998	0.9	43	2.0	rc208	*1724*	1724	0.0	100	0.9
Average			2.3		2.4	Average			0.0		1.0
Max			5.2		3.8	Max			0.0		2.1

Table 8
Results for test problems of Cordeau, Gendreau and Laporte ($m = 4$).

Name	BK	ILS	Gap (%)	Visits	CPU (s)	Name	BK	ILS	Gap (%)	Visits	CPU (s)
pr01	657	644	2.0	45	0.2	pr11	657	654	0.5	47	0.2
pr02	1072	1014	5.4	71	2.4	pr12	1118	1041	6.9	66	1.9
pr03	1222	1162	4.9	76	10.5	pr13	1329	1263	5.0	84	6.6
pr04	1515	1452	4.2	86	11.6	pr14	1568	1528	2.6	89	16.6
pr05	1740	1665	4.3	96	19.6	pr15	1854	1818	1.9	102	19.5
pr06	1740	1696	2.5	91	35.4	pr16	1887	**1889**	−0.1	109	35.9
pr07	872	840	3.7	57	1.6	pr17	925	889	3.9	63	1.9
pr08	1376	1267	7.9	69	6.9	pr18	1470	1352	8.0	80	5.7
pr09	1561	1460	6.5	84	13.8	pr19	1596	1560	2.3	97	22.2
pr10	1827	1782	2.5	102	38.7	pr20	1841	**1846**	−0.3	110	26.9
Average			4.4		14.1	Average			3.1		13.7
Max			7.9		38.7	Max			8.0		35.9

possible visits (r/c/rc_200 and pr11–pr21) are more difficult. This also explains why the average gap for the Cordeau et al. instances is bigger than the gap for the Solomon instances. Another explanation for differences in the average gap with best-known solutions could be the performance of the ACS of Montemanni and Gambardella [21] on particular instances.

Table 9 compares the average computation time of the ILS heuristic with the ACS. For both heuristics the computation times are averaged per set of instances and per number of tours. The ILS heuristic is many times faster than the ACS algorithm. The ILS heuristic is also many times faster than the fastest method of Righini and Salani [20] that required more than 400 s for problems with only one tour. Furthermore, based on Tables 1–8, it can be concluded that the computation time is correlated with the number of visited locations.

Table 9
Average CPU time for the ILS heuristic and the ant colony system (s).

m	ILS				ACS			
	1	2	3	4	1	2	3	4
Solomon 100	0.2	0.9	1.5	2.4	200	1317	1422	1523
Solomon 200	1.7	2.6	1.7	1.0	1193	2142	1345	245
Cordeau 1–10	1.8	4.8	9.2	14.1	1627	1890	2164	2448
Cordeau 11–20	2.0	5.2	9.7	13.7	888	2385	2350	2583

In Tables 10 and 11 scores are shown that are obtained for the new data sets of TOPTW with up to 20 tours. A column is added with the number of tours for each instance. The optimal score is the sum of the score of all possible visits.

Table 10
Results for the new data set (Solomon).

Name	m	Opt	ILS	GAP (%)	Visits	CPU (s)	Name	m	Opt	ILS	GAP (%)	Visits	CPU (s)
c101	10	*1810*	1720	5.0	91	4.1	c201	4	*1810*	1810	0.0	100	1.5
c102	10	*1810*	1790	1.1	98	4.2	c202	4	*1810*	1810	0.0	100	1.1
c103	10	*1810*	1810	0.0	100	3.0	c203	4	*1810*	1810	0.0	100	1.0
c104	10	*1810*	1810	0.0	100	1.8	c204	4	*1810*	1810	0.0	100	1.0
c105	10	*1810*	1770	2.2	96	2.8	c205	4	*1810*	1810	0.0	100	1.0
c106	10	*1810*	1750	3.3	94	3.8	c206	4	*1810*	1810	0.0	100	1.0
c107	10	*1810*	1790	1.1	98	3.1	c207	4	*1810*	1810	0.0	100	1.0
c108	10	*1810*	1810	0.0	100	2.5	c208	4	*1810*	1810	0.0	100	0.9
c109	10	*1810*	1810	0.0	100	2.0							
r101	19	*1458*	1441	1.2	97	2.5	r201	4	*1458*	1458	0.0	100	1.3
r102	17	*1458*	1450	0.5	98	3.1	r202	3	*1458*	1443	1.0	98	2.7
r103	13	*1458*	1450	0.5	98	2.0	r203	3	*1458*	1458	0.0	100	1.5
r104	9	*1458*	1402	3.8	90	2.3	r204	2	*1458*	1440	1.2	97	3.3
r105	14	*1458*	1435	1.6	97	4.1	r205	3	*1458*	1458	0.0	100	1.1
r106	12	*1458*	1441	1.2	96	3.1	r206	3	*1458*	1458	0.0	100	1.0
r107	10	*1458*	1431	1.9	94	3.3	r207	2	*1458*	1428	2.1	91	1.6
r108	9	*1458*	1430	1.9	93	2.7	r208	2	*1458*	1458	0.0	100	1.6
r109	11	*1458*	1432	1.8	95	2.5	r209	3	*1458*	1458	0.0	100	1.1
r110	10	*1458*	1419	2.7	92	4.4	r210	3	*1458*	1458	0.0	100	1.2
r111	10	*1458*	1410	3.3	93	3.0	r211	2	*1458*	1422	2.5	92	1.9
r112	9	*1458*	1418	2.7	91	2.4							
rc101	14	*1724*	1686	2.2	95	4.3	rc201	4	*1724*	1724	0.0	100	2.2
rc102	12	*1724*	1659	3.8	92	2.9	rc202	3	*1724*	1686	2.2	95	1.7
rc103	11	*1724*	1689	2.0	97	3.4	rc203	3	*1724*	1724	0.0	100	2.8
rc104	10	*1724*	1719	0.3	99	3.2	rc204	3	*1724*	1724	0.0	100	1.0
rc105	13	*1724*	1691	1.9	95	3.9	rc205	4	*1724*	1724	0.0	100	2.1
rc106	11	*1724*	1665	3.4	93	4.8	rc206	3	*1724*	1708	0.9	99	1.3
rc107	11	*1724*	1701	1.3	96	2.4	rc207	3	*1724*	1713	0.6	98	1.4
rc108	10	*1724*	1698	1.5	97	5.6	rc208	3	*1724*	1724	0.0	100	1.0
Average				1.8		3.2	Average				0.4		1.5
Max				5.0		5.6	Max				2.5		3.3

Table 11
Results for new data set (Cordeau, Gendreau and Laporte).

Name	m	Opt	ILS	GAP (%)	Visits	CPU (s)
pr01	3	*657*	608	7.5	41	0.7
pr02	6	*1220*	1180	3.3	88	4.5
pr03	9	*1788*	1738	2.8	132	11.3
pr04	12	*2477*	2428	2.0	183	45.4
pr05	15	*3351*	3297	1.6	232	37.3
pr06	18	*3671*	3650	0.6	279	106.1
pr07	5	*948*	909	4.1	66	1.5
pr08	10	*2006*	1984	1.1	139	12.0
pr09	15	*2736*	2729	0.3	214	33.0
pr10	20	*3850*	3850	0.0	288	52.3
Average				2.3		30.4
Max				7.5		106.1

The gap between the ILS outcome and the optimal solution is, on average, only 1.3%. The maximal score gap is 7.5%. For 25 instances the ILS heuristic computes the optimal solution. These results confirm that the ILS heuristic obtains high quality results for TOPTW instances. Of course, the computation time for these very complicated instances is longer, due to the many tours that have to be constructed and improved.

Finally, in order to illustrate the sensitivity of some heuristic design decisions, Table 12 presents the average gap (%) with the best-known solutions and the average CPU time (s) for all instances of Montemanni and Gambardella. The first row summarises the performance of the ILS heuristic. Rows two and three present the performance when the maximum number of iterations without improvement is altered from 150 to 100 or 200. Unsurprisingly, the maximum number has a positive correlation with the quality as well as with computation time. The differences, however, remain small. If other values are used for the maximum number of locations to remove, the difference in performance also remains limited. These results are shown in row four and five. In row six, the impact of the randomisation in the shake step is illustrated. If the position to start the removal is equal over all tours instead of different per tour, the performance of the algorithm stays almost the same. This type of randomisation has almost no influence on the performance of the ILS heuristic. The only design decisions with a significant influence in performance are presented in row seven and eight. The ILS heuristic applies $Score^2$ when the ratio is calculated to determine the most promising visit to insert next (Section 4.1). If instead *Score* is used, the average gap increases significantly (row seven). To prove the key contribution of the shake step, the instances are also solved without using the shake step. In this case, the average gap increases to 8.5% (row eight). In the last row, the quality of the ILS results is presented for computation times limited to 1 s. The resulting gap of 2.5% is probably the best proof that this ILS heuristic is suited to obtain high quality PET trips in real-time.

At first glance, the results in rows five and six suggest to change some parameter settings in order to obtain a slightly better performance. The difference in performance is, however, always related to the test instances that are used. Fine-tuning the parameters based on the test set of 304 problems bears the risk of "overfitting" the algorithm to the instances. Thus, the most important conclusion based on Table 12 remains that the results are not sensitive to changes in the parameter settings.

Distances between locations were rounded down in the same way as Righini and Salani [20] and Montemanni and Gambardella

Table 12
Sensitivity analysis of the design decisions.

	Gap with best-known (%)			CPU (s)		
	Solomon	Cordeau	All	Solomon	Cordeau	All
ILS	1.3	3.0	1.8	1.7	7.6	3.1
MaxTimesNoImprovement = 100	1.5	3.4	2.0	1.0	5.2	2.1
MaxTimesNoImprovement = 200	1.2	2.8	1.6	2.0	10.0	4.1
MaxNumberToRemove = n/m	1.3	2.8	1.7	2.1	7.6	3.5
MaxNumberToRemove = $n/5m$	1.2	2.8	1.7	1.1	7.3	2.8
No randomisation in shake step	1.3	2.9	1.7	1.4	7.3	3.0
Ratio = score/shift	1.8	4.4	2.5	1.5	7.6	3.1
No shake step	6.9	13.0	8.5	0.1	0.3	0.1
Limit CPU time to 1 s	1.5	5.1	2.5	–	–	–

[21]: to the first decimal for the Solomon [27] instances and to the second decimal for the instances of Cordeau et al. [28].

6. Conclusions and further work

The main contribution of this paper is an algorithm that solves the team orienteering problem with time windows (TOPTW) fast and effectively. On a large set of test instances, the average gap with the best-known solutions is only 1.8% and the computation time is decreased with a factor of several hundreds compared to other algorithms. Even when the computation time is limited to 1 s, high quality results are obtained. This is achieved by speeding up the evaluation of possible improvements and the specific implementation of the shake step to better explore the whole solution space. All this makes the algorithm appropriate for the personalised tourist guide application.

Furthermore, new best solutions are calculated for many test instances and new instances are designed. Since the optimal solutions for these new instances are known, they can be used as a benchmark for further research.

Doubtlessly, this heuristic can be tailored to solve realistic tourist trip design problems. Some improvements may be possible due to the specificity of time windows in tourist applications. Typically, most tourist attractions will be open all day and only the opening and closing times will be slightly different. As a result, time windows will largely overlap. In this situation, adding two-opt moves [29] to the heuristic could be beneficial and significantly reduce the travel time. Other promising moves are the insertion of two or more activities simultaneously or explicitly moving visits between tours. Such moves can be embedded in a variable neighbourhood search framework [30].

Certain parameters, such as the number of possible visits, the number of tours, the width of the time windows and the number of visits per tour, determine the difficulty of an instance. In order to get more insight in the significance of each of these parameters, specific test instances should be designed.

Acknowledgement

Dr. P. Vansteenwegen is a post-doctoral research fellow of the "Fonds Wetenschappelijk Onderzoek-Vlaanderen (FWO)".

References

[1] Vansteenwegen P, Van Oudheusden D. The mobile tourist guide: an OR opportunity. OR Insights 2007;20(3):21–7.
[2] Tsiligirides T. Heuristic methods applied to orienteering. Journal of the Operational Research Society 1984;35:797–809.
[3] Laporte G, Martello S. The selective travelling salesman problem. Discrete Applied Mathematics 1990;26:193–207.
[4] Butt S, Cavalier T. A heuristic for the multiple tour maximum collection problem. Computers & Operations Research 1994;21:101–11.
[5] Arkin E, Mitchell J, Narasimhan G. Resource-constrained geometric network optimization. In: Proceedings of the 14th ACM symposium on computational geometry; 1998. p. 307–16.
[6] Pekny J, Miller D. An exact parallel algorithm for the resource constrained travelling salesman problem with application to scheduling with an aggregate deadline. In: Proceedings of the 1990 ACM annual conference on cooperation; 1990.
[7] Feillet D, Dejax P, Gendreau M. Travelling salesman problems with profits. Transportation Science 2005;39:188–205.
[8] Righini G, Salani M. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. Computers & Operations Research 2009;4: 1191–203.
[9] Chao I, Golden B, Wasil E. A fast and effective heuristic for the orienteering problem. European Journal of Operational Research 1996;88:475–89.
[10] Golden B, Levy L, Vohra R. The orienteering problem. Naval Research Logistics 1987;34:307–18.
[11] Tang H, Miller-Hooks E. A tabu search heuristic for the team orienteering problem. Computers & Operations Research 2005;32:1379–407.
[12] Archetti C, Hertz A, Speranza M. Metaheuristics for the team orienteering problem. Journal of Heuristics 2007;13:49–76.
[13] Vansteenwegen P, Souffriau W, Vanden Berghe G, Van Oudheusden D. A guided local search metaheuristic for the team orienteering problem. European Journal of Operational Research 2009;196(1):118–27.
[14] Ke L, Archetti C, Feng Z. Ants can solve the team orienteering problem. Computers & Industrial Engineering 2008;54:648–65.
[15] Souffriau W, Vansteenwegen P, Vanden Berghe G, Van Oudheusden D. A path relinking approach for the team orienteering problem. Computers & Operations Research, special issue, Logistics 2009, under review.
[16] Boussier S, Feillet D, Gendreau M. An exact algorithm for the team orienteering problem. 4OR 2007;5:211–30.
[17] Kantor M, Rosenwein M. The orienteering problem with time windows. The Journal of the Operational Research Society 1992;43(6):629–35.
[18] Bar-Yehuda R, Even G, Shahar S. On approximating a geometric prize-collecting travelling salesman problem with time windows. Journal of Algorithms 2005;55:76–92.
[19] Mansini R, Pelizzari M, Wolfer R. A granular variable neighbourhood search heuristic for the tour orienteering problem with time windows. Technical Report R.T 2006-02-52, University of Brescia, Italy.
[20] Righini G, Salani M. Dynamic programming for the orienteering problem with time windows. Technical Report 91, Dipartimento di Tecnologie dell'Informazione, Universita degli Studi Milano, Crema, Italy, 2006.
[21] Montemanni R, Gambardella L. Ant colony system for team orienteering problems with time windows. European Journal of Operational Research; 2009, under review.
[22] Righini G, Salani M. New dynamic programming algorithms for the resource constrained elementary shortest path. Networks 2008;51(3):155–70 [accepted for publication].
[23] Gendreau M, Laporte G, Semet F. A tabu search heuristic for the undirected selective travelling salesman problem. European Journal of Operational Research 1998;106:539–45.
[24] Lourenço H, Martin O, Stützle T. Iterated local search. In: Glover F, Kochenberger G, editors. Handbook of metaheuristics. Dordrecht: Kluwer Academic Publishers; 2003. p. 321–53.
[25] Ibaraki T, Imahori S, Kubo M, Masuda T, Uno T, Yagiura M. Effective local search algorithms for routing and scheduling with general time-window constraints. Transportation Science 2005;39(2):206–32.
[26] Hashimoto H, Yagiura M, Ibaraki T. An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. Discrete Optimization 2008;5:434–56.
[27] Solomon M. Algorithms for the vehicle routing and scheduling problem with time window constraints. Operations Research 1987;35:254–65.
[28] Cordeau J-F, Gendreau M, Laporte G. A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks 1997;30:105–19.
[29] Lin S. Computer solutions of the traveling salesman problem. Bell System Technical Journal 1965;44:2245–69.
[30] Hansen P, Mladenovic N. Variable neighbourhood search: principles and applications. European Journal of Operational Research 2001;130:449–67.