European Journal of Operational Research 196 (2009) 118–127

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A guided local search metaheuristic for the team orienteering problem

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

Article history:
Received 2 April 2007
Accepted 22 February 2008
Available online 18 March 2008

Keywords:
Metaheuristics
Guided local search
Orienteering problem

ABSTRACT

In the team orienteering problem (TOP) a set of locations is given, each with a score. The goal is to determine a fixed number of routes, limited in length, that visit some locations and maximise the sum of the collected scores. This paper describes an algorithm that combines different local search heuristics to solve the TOP. Guided local search (GLS) is used to improve two of the proposed heuristics. An extra heuristic is added to regularly diversify the search in order to explore more areas of the solution space. The algorithm is compared with the best known heuristics of the literature and applied on a large problem set. The obtained results are almost of the same quality as the results of these heuristics but the computational time is reduced significantly. Applying GLS to solve the TOP appears to be a very promising technique. Furthermore, the usefulness of exploring more areas of the solution space is clearly demonstrated.

1. Introduction

In the orienteering problem (OP) a set of n locations i is given, each with a score S_i. The starting point (location 1) and the end point (location n) 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 goal of the OP is to determine a route, limited by T_{max}, that visits some of the locations, in order to maximise the total collected score. Each location can be visited at most once.

The OP is also known as the selective travelling salesperson problem (STSP) (Laporte and Martello, 1990), the maximum collection problem (MCP) (Butt and Cavalier, 1994) and the bank robber problem (Arkin et al., 1998). Furthermore the OP can be formulated as a special case of the resource-constrained *TSP* (RCTSP) (Pekny and Miller, 1990) or as a *TSP* with profits (Feillet et al., 2005).

The OP arises in many applications: the sport game of orienteering (Chao et al., 1996b), the home fuel delivery problem (Golden et al., 1987), athlete recruiting from high schools (Butt and Cavalier, 1994), routing technicians to service customers (Tang and Miller-Hooks, 2005), etc. The application we are most interested in is the personalised mobile tourist guide (MTG). The MTG, a handheld device, is aware of the user profile, attraction values and trip information. The device should at short notice suggest a (near) optimal combination of attractions in a city and a route between them, taking into account the weather, opening hours, crowded places, etc. The problem the MTG has to solve is called the tourist trip design problem (TTDP) (Vansteenwegen and Van Oudheusden, 2007). The OP is the simplest form of the TTDP.

Several researchers propose exact algorithms based on branch-and-bound (Laporte and Martello, 1990; Ramesh et al., 1992) and branch-and-cut (Fischetti et al., 1998; Gendreau et al., 1998b). With the branch-and-cut procedure instances up to 500 locations could be solved (Fischetti et al., 1998). Since Golden et al. (1987) prove that the OP is NP-hard, it is obvious that these exact algorithms are very time consuming and that most research has focused on heuristic approaches. Tsiligirides (1984) proposed deterministic and stochastic heuristics; Golden et al. (1987) developed a centre-of-gravity heuristic that was later improved (Golden et al., 1988); Ramesh and Brown (1991) introduced a four-phase heuristic and a five-step heuristic was proposed by Chao et al. (1996b). Gendreau et al. (1998a) proposed a tabu search heuristic.

The team orienteering problem (TOP) (Chao et al., 1996a; Tang and Miller-Hooks, 2005) or multiple tour maximum collection problem (MTMCP) (Butt and Cavalier, 1994) is an OP where the goal is to determine m routes, each limited to T_{max}, that maximises the total collected score. In the case of the mobile tourist guide, this would be the problem of a multiple day visit. The best published heuristics for the TOP are described in Archetti et al. (2007). A more detailed and extended review of published algorithms for the OP and the TOP can be found in Tang and Miller-Hooks (2005).

In the next section a mathematical formulation of the TOP is presented. In Section 3 the GLS algorithm to solve the TOP, and the OP as well, is explained. Section 4 introduces the test problems and the computational results are discussed in Section 5. Conclusions and further work are described in Section 6.

* 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), GreetVandenBerghe@kahosl.be (G.V. Berghe), Dirk.VanOudheusden@cib.kuleuven.be (D.V. Oudheusden).

0377-2217/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2008.02.037

2. Mathematical formulation

Based on the notation introduced above, the team orienteering problem (TOP) can be formulated as an integer program (T_i = time needed to visit location i; u_{id} = position of location i in tour d; 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):

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

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

$$\sum_{d=1}^{m} y_{kd} \leqslant 1; \quad \forall k = 2, \ldots, n-1, \tag{3}$$

$$\sum_{i=1}^{n-1} x_{ikd} = \sum_{j=2}^{n} x_{kjd} = y_{kd}; \quad \forall k = 2, \ldots, n-1; \quad \forall d = 1, \ldots, m, \tag{4}$$

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

$$2 \leqslant u_{id} \leqslant n; \quad \forall i = 2, \ldots, n; \quad \forall d = 1, \ldots, m, \tag{6}$$

$$u_{id} - u_{jd} + 1 \leqslant (n-1)(1 - x_{ijd}); \quad \forall i, j = 2, \ldots, n; \quad \forall d = 1, \ldots, m, \tag{7}$$

$$x_{ijd}, y_{id} \in \{0, 1\}; \quad \forall i, j = 1, \ldots, n; \quad \forall d = 1, \ldots, m. \tag{8}$$

The objective function (1) is to maximise the total collected score. Constraints (2) guarantee that each tour starts in location *1* and ends in location *n*. Constraints (3) ensure that every location is visited at most once. Constraints (4) guarantee that, if a location is visited in a given tour, it is preceded and followed by exact one other visit in the same tour. Constraints (5) ensure the limited time budget for each tour. Constraints (6) and (7) are necessary to prevent subtours. These subtour elimination constraints are formulated according to the Miller–Tucker–Zemlin (MTZ) formulation of the *TSP* (Miller et al., 1960). By setting $d = 1$, a correct formulation for the OP is obtained.

Other mathematical formulations of the TOP (including visiting times) are presented in Butt and Cavalier (1994) and Tang and Miller-Hooks (2005).

3. A composite algorithm

In this section a new approach to solve the OP and the TOP is described. The algorithm uses a combination of different heuristics. First, it is explained how the different heuristics are combined in the algorithm, next, the heuristics are explained in more detail and finally all the parameters are specified and discussed.

3.1. General structure

Fig. 1 presents the pseudo code of the algorithm. The algorithm starts with a construction phase. Afterwards the local search heuristics (described in Section 3.2) are implemented in two loops: the ALGorithm loop (*ALG_Loop*) and the Local Search Loop (*LS_Loop*). The *LS_Loop* is embedded in the *ALG_Loop*. The *ALG_Loop* starts with the *LS_Loop*, disturbs the constructed tours when necessary (see *Disturb* in Section 3.4) and ends the algorithm after *Max_ALG* iterations. The *LS_Loop* uses the operations *Swap*, *TSP*, *Move*, *Insert* and *Replace* in a fixed sequence and ends after *Max_LS* iterations or, to reduce the calculation time, if the current solution is not

```
Construct;
ALG_Loop=0;
while ALG_Loop < Max_ALG do
|    ALG_Loop+1;
|    LS_Loop=0;
|    while Solution improved and LS_Loop < Max_LS do
|    |    LS_Loop+1;
|    |    Swap;
|    |    TSP;
|    |    Swap;
|    |    Move;
|    |    Insert;
|    |    Replace;
|    end
|    if Solution better than BestFound then
|    |    BestFound=Solution;
|    else if Solution == BestFound then
|    |    if Not Disturbed before then
|    |    |    Disturb;
|    |    else
|    |    |    Return BestFound;
|    |    end
|    end
|    if ALG_Loop == Max_ALG/2 then
|    |    Disturb;
|    end
end
Return BestFound;
```

Fig. 1. The algorithm's pseudo code.

improved by *LS_Loop*. The obtained *LS_Loop* solution is then compared with the best found solution so far:

- if the *LS_Loop* solution is better than the best solution, the solution is saved and the *ALG_Loop* starts again with the *LS_Loop*;
- if the solution is not better, it is compared with the previously obtained *ALG_Loop* solution:
  - if these solutions are the same and *Disturb* was not yet used, the current solution will be disturbed and the *ALG_Loop* starts again;
  - if the solutions are the same and *Disturb* has been applied before, the algorithm ends;
  - if the solutions are different, the *ALG_Loop* starts again without using *Disturb*.

The solution is disturbed anyway after half of the *Max_ALG* iterations. Thus, *Disturb* will always be used in this algorithm, to guarantee a broader search in the solution space.

The way *Disturb* is applied ensures it is never used more than twice in the *ALG_Loop*. A trade-off is made here between possibly finding better solutions and using more calculation time.

3.2. Local search heuristics

The above-mentioned heuristics are explained here in more detail. All these heuristics are used until they reach a local optimum. First, the heuristic of the construction phase is explained. Next, two heuristics are discussed that try to increase the total score of the solution: *Insert* and *Replace*. Finally, three heuristics are described that reduce the travel time between the selected locations: *Swap*, *Move* and *TSP*. Finding the shortest route between the selected locations is not an explicit part of the OP objective. However, the shorter the tour, the more likely extra locations can be added to that tour later, by operations *Insert* or *Replace*.

3.2.1. Greedy construction heuristic: Construct

As a construction heuristic the *Initialisation* method described by Chao et al. (1996a,b) is used. First, all locations that cannot be reached are removed from the problem. Then, a fixed number of points, furthest from the start and end point, are chosen. Each of the m tours is started by assigning one of these chosen points to the tour (m is the number of tours). Next, all remaining points are inserted into these m tours using cheapest insertion, only looking at the distance, not at the score. If all m tours exhaust the time budget and unassigned points remain, new tours are constructed with these points (using cheapest insertion) until all points are assigned. Finally, the best m tours are selected to start the remaining part of the algorithm.

3.2.2. Insert an extra location: Insert

In short, *Insert* attempts to insert new locations in the tours in the position where the location consumes the least travel time.

For every location that is not yet included, this heuristic finds the least time consuming position in the tour to include that location. If the time consumption of the location in that position is lower than the available time budget, the location is inserted in the tour. The time consumption of inserting a location z between locations i and j equals:

$$T_z + t_{i,z} + t_{z,j} - t_{i,j}. \tag{9}$$

The order in which the locations are considered for insertion is not random, but for every tour, the locations are ranked based on their "appropriateness" (A_i) for the tour. To calculate the appropriateness, first the tour's centre of gravity (COG) is determined, based on the Cartesian coordinates (x_i,y_i) of the included locations weighted with the location score S_i:

$$x_{COG} = \left(\sum_{\forall i \in \text{tour}} x_i * S_i\right) \Big/ \sum_{\forall i \in \text{tour}} S_i, \tag{10}$$

$$y_{COG} = \left(\sum_{\forall i \in \text{tour}} y_i * S_i\right) \Big/ \sum_{\forall i \in \text{tour}} S_i. \tag{11}$$

Then the appropriateness is calculated, similar to the "desirability" defined by Tsiligirides (1984):

$$A_i = (S_i/(t_{i,\text{POG}} + T_i))^4. \tag{12}$$

The locations with the highest appropriateness are considered first.

The heuristic ends when all locations have been considered for each tour. To calculate the centre of gravity based on the coordinates, the Euclidean distance is used. (It appears that all published test problems for the TOP are defined in a two-dimensional Euclidean space.)

3.2.3. Replace a location by another one: Replace

Operation *Replace* seeks to replace an included location by a non-included location with a higher score.

The replace location heuristic considers one by one the non-included locations. The ranking of the locations is again based on their appropriateness (12). For every location it is checked what would be the least time consuming position in the tour to include that location. If enough time budget is left for insertion, the location is inserted and the heuristic starts again. If the remaining budget is insufficient, all included locations with a lower score are considered for exclusion. The time saving that becomes achievable by excluding location j, located in the tour between location i and k, equals:

$$T_j + t_{i,j} + t_{j,k} - t_{i,k}. \tag{13}$$

If excluding one of these locations from the tour creates enough time to insert the non-included location under consideration, a replacement will be carried out: the included node will be excluded, and the non-included node will be inserted at the least time consuming position. Afterwards, the heuristic starts again after adjusting the centre of gravity and the ranking of the non-included nodes. If no more replacements are possible, the heuristic reaches a local optimum.

3.2.4. Swap a location between two tours: Swap

In short, this heuristic endeavours to exchange two locations between two tours.

This heuristic is only used for TOP, not for OP, since multiple tours are required. For every two locations that do not belong to the same tour, the heuristic checks if time can be saved by swapping these locations between the tours. If the travel time can be reduced in each tour, or if the time saved in one tour is longer than the extra time needed in the other tour, the swap is carried out. Of course, the available time budget of both tours has to be respected. Swaps are carried out until a local optimum is reached.

3.2.5. Move a location from one tour to another: Move

Move tries to group together the available time left. For most problems, it is better to have a lot of time available in one tour and no time left in the other tours, than to have a little time available in each tour. In the former case, it is more likely that another location can be added to the solution. Based on this observation the *Move* heuristic is constructed.

Similar to the *Swap* heuristic, *Move* is only used for TOP. For each included location the possibility of moving it to another tour is calculated. If possible, the location is moved from its original tour to the new tour. In this way, time is made available in the original tour and this time can be used by another heuristic to

insert a new location. Obviously, the algorithm guarantees that as soon a location has been removed from a tour, no other locations can be moved to this tour by this heuristic. This is necessary to avoid that the available time for this tour is first increased but later decreased. *Move* is used until no more locations can be moved.

3.2.6. A 2-opt heuristic: TSP

Operation *TSP* aims at reducing the travel time by finding the shortest route in each tour. In this heuristic 2-opt is used as local search mechanism. 2-opt removes two edges from the tour to replace them with two new edges not previously included in the tour. The tour has to remain closed and the total travel time should be reduced. With *greedy local search*, all possible edge exchanges are evaluated every iteration and the best one is selected and implemented. Local search terminates when no 2-opt move exists to further improve the tour.

3.3. Guided local search

Guided local search (GLS) was developed by Voudouris and Tsang (1999). In general, GLS heuristics obtain very good results for various vehicle routing problems (VRP) (Kilby et al., 1997; Beullens et al., 2003; Zhong and Cole, 2005), for the quadratic assignment problem (Mills et al., 2003) and for the *TSP* (Voudouris and Tsang, 1999). However, GLS was not used before to solve the (T)OP. GLS penalises, based on a utility function, unwanted solution features during each local search iteration. The penalty augments the objective function during every iteration. In this way the solution procedure may escape from local optima and allow the search to continue.

Before GLS was introduced, a test run showed that for all the test problems, on average, the obtained scores are 14% worse if *Replace* is not used in the algorithm. Leaving out *Insert* gives a difference of 6%. This indicates that *Replace* is the most contributing heuristic to increase the score. The same test run showed that, on average, the obtained scores are 7% worse if *TSP* is left out. Leaving out *Move* or *Swap* only gives a difference smaller than 1%. This indicates that, to reduce travel time, *TSP* is the most contributing heuristic. In order to further improve these heuristic, GLS is applied to *Replace* and *TSP*.

3.3.1. Replace with GLS

In order to escape the local optimum obtained by the replace heuristic two GLS penalties are used:

(1) Increase the augmented score of the non-included location with the highest score.

The utilities U_i of all non-included locations are calculated and the location with the highest utility is rewarded by increasing its score with p_R. In the next iteration the probability that this location will be included is increased.

$$U_i = S'_i/(1 + nr_i), \tag{14}$$

where S'_i is the increased score of location i: $S'_i = S_i + nr_i * p_R$, nr_i is the number of times this location was 'rewarded' and p_R is the magnitude of the reward used in the *Replace* heuristic.

The utility will be higher for locations with a higher increased score. The term $(1 + nr_i)$ is introduced to prevent the scheme from being completely biased towards rewarding high scoring features. If a feature is rewarded many times, this term will prevent other features from not receiving a reward.

(2) Decrease the augmented score of an included location with a low score and remove it from the tour.

The disutilities DU_i of all included locations are calculated and the location with the highest disutility is penalised by decreasing its score with p_R and removing it from the tour. In the next iteration the probability that the location will be included again is decreased and thus more time budget becomes available for non-included locations.

$$DU_i = 1/S'_i * (1 + np_i), \tag{15}$$

where S'_i is the increased score of location i: $S'_i = S_i + np_i * p_R$, np_i is the number of times this location was 'penalised' and p_R is the magnitude of the penalty used in the *Replace* heuristic.

In this case, since S'_i is part of the denominator, the disutility will be higher for locations with a lower score. The term $(1 + np_i)$ is used in the same way as above.

After applying these penalties and rewards, the replace location heuristic is used again during the next iteration step, but always with the increased scores. The GLS iteration steps are carried out a fixed number of times (*N_Replace*). At the end of each iteration, the real total score, without the penalties, is calculated and the best tour is saved.

The obtained results are very dependent on the parameters of this heuristic, both in efficiency and effectiveness. All parameters are discussed in Section 3.5.

3.3.2. TSP with GLS

Every time a local optimum is reached, the GLS framework will penalise the arc (used in the current solution to connect two locations) with the highest utility U_{ij}:

$$U_{ij} = t'_{ij}/(1 + np_{ij}), \tag{16}$$

where t'_{ij} is the increased distance between locations *i and j*: $t'_{ij} = t_{ij} + np_{ij} * p_{\mathrm{TSP}}$, np_{ij} is the number of times this distance was penalised before and p_{TSP} is the magnitude of the penalty used in the *TSP* heuristic.

After penalisation, 2-opt local search is used once more during the next iteration step, but always with the increased distances. The GLS iteration steps are carried out a fixed number of times (*N_TSP*). At the end of each iteration, the real total distance, without the penalties, is calculated and the best tour is saved.

3.4. Disturb

In order to better explore the whole solution space, many metaheuristics implement diversification strategies. In the presented approach *Disturb* will cause diversification.

The *Disturb* algorithm removes a part of each tour, to create an opportunity for other locations to enter the tour or to move from one tour to another. It removes a fixed percentage of locations between the starting location and the end location, either at the beginning or at the end of the tour, depending on the exact moment that this heuristic is used in the algorithm.

On the one hand, *Disturb* is used as one of the heuristics in the algorithm loop (see Section 3.1). *Disturb* will remove in this case *Disturb_Loop* percent of the locations at the end of each tour. On the other hand, *Disturb* is used inside the *Replace* heuristic. If no improvement is found after a fixed number of *Replace* iterations, *Disturb* will remove *Disturb_Replace* percent of the locations at the beginning of each tour. Afterwards, *Replace* will continue with inserting or replacing locations. In this way, almost all locations will be removed at least once during the search, guaranteeing a high degree of diversification.

Preliminary tests showed that using *Disturb* inside *Replace* (and not only in the algorithm loop), reduces the calculation time and improves the obtained results.

Together with GLS, *Disturb* reinforces the key contribution of the Replace heuristic in the algorithm.

3.5. Parameter tuning

To optimise the algorithm some parameters need to be set, they are outlined and discussed in this section. The parameter values that were used during testing are indicated in the last column of Table 1.

Table 1
Parameters used in the algorithm

Name	Explanation	Value
Max_LS	Maximum number of iterations of *LS_Loop*	10
Max_ALG	Maximum number of iterations of *ALG_Loop*	10
N_TSP	Maximum number of GLS iterations in *TSP*	50
N_Replace	Maximum number of GLS iterations in *Replace*	1000
P_{TSP}	Magnitude of the penalty used in *TSP*	PD
P_R	Magnitude of the penalty or reward used in *Replace*	PD
N_Replace_Disturb	If no improvement is found after *N_Replace_Disturb* *Replace* iterations, *Disturb* is used within *Replace*	100
Max_Disturb	Maximum number of times *Disturb* is used within *Replace*	3
Disturb_Replace	Percent of locations that is removed by *Disturb* in *Replace*	75%
Disturb_Loop	Percent of locations that is removed by *Disturb* in *ALG_Loop*	75%

Table 2
Test sets overview

	Number of test problems	Number of locations (n)	Number of routes (m)
Tsiligirides	18	32	1
	11	21	1
	20	33	1
Fischetti et al.	11	32,21,33,31,52,101,101, 201, 300,319,401	1
Chao	26	66	1
	14	64	1
Tang and Miller-Hooks	8	32	2, 3, 4
	16	33	2, 3, 4
	56	100	2, 3, 4
	74	66	2, 3, 4
	24	64	2, 3, 4
	58	102	2, 3, 4

The values of P_{TSP} and P_R are made problem dependent (PD). The penalty used in *TSP* varies based on the average distance between all the locations and the starting location (t_{1i}) and the visit durations (T_i):

$$P_{TSP} = \mathrm{Max}\left(1; \left\lceil \sum_{i=2}^{n-1} (t_{1i} + T_i)/20^*(n-2) \right\rceil\right). \quad (17)$$

The penalty used in *Replace* depends on the average score of all the locations (S_i):

$$P_R = Max\left(1; \left\lceil \sum_{i=2}^{n-1} (S_i)/5^*(n-2) \right\rceil\right). \quad (18)$$

The maximum numbers *Max_LS*, *Max_ALG*, *N_TSP* and *N_Replace* are hardly required, due to the alternative stopping criteria that end the loops when no more improvements are found. *N_Replace_Disturb*, *Max_Disturb*, *Disturb_Replace* and *Disturb_Loop* determine the trade-off between converging to a high quality solution and broadening the search. This trade-off is necessary to limit the calculation time to obtain a high quality result. A low value for *N_Replace_Disturb* and a high value for *Max_Disturb*, *Disturb_Replace* and *Disturb_Loop* will result in a broader search in the solution space. The values of *N_Replace_Disturb*, *Max_Disturb*, *Disturb_Loop*, *Disturb_Replace* and the "20" and "5" values in formulas (17) and (18) are based on preliminary testing.

4. Test problems

The algorithm was tested on a large number of test problems and compared with published results. Test sets were used from Tsiligirides (1984), Chao (1993), Fischetti et al. (1998) and from Tang and Miller-Hooks (2005). The characteristics of the problems are presented in Table 2.

5. Computational experiments

All experiments were carried out on a personal computer Intel Pentium 4 with 2.80 GHz and 1 GB Ram. It has the same specifications as the computer used by Archetti et al. (2007).

Table 3
Results on Tsiligirides' test problems

$n = 32$; $m = 1$					$n = 21$; $m = 1$					$n = 33$; $m = 1$				
T_{max}	GLS	OPT	Gap	CPU	T_{max}	GLS	OPT	Gap	CPU	T_{max}	GLS	OPT	Gap	CPU
5	10	10	0	0.0	15	120	120	0	0.2	15	170	170	0	0.2
10	15	15	0	0.1	20	200	200	0	0.1	20	200	200	0	0.3
15	45	45	0	0.2	23	210	210	0	0.3	25	250	260	4	0.6
20	55	65	15	0.6	25	230	230	0	0.2	30	310	320	3	0.4
25	90	90	0	0.8	27	220	230	4	0.4	35	390	390	0	0.6
30	80	110	27	0.4	30	260	265	2	0.4	40	430	430	0	0.7
35	135	135	0	0.6	32	300	300	0	0.2	45	470	470	0	0.7
40	145	155	6	0.7	35	305	320	5	0.2	50	520	520	0	0.9
46	175	175	0	0.8	38	360	360	0	0.3	55	540	550	2	0.6
50	180	190	5	0.7	40	380	395	4	0.3	60	570	580	2	0.6
55	200	205	2	0.9	45	450	450	0	0.2	65	610	610	0	0.8
60	220	225	2	0.6						70	630	640	2	0.8
65	240	240	0	0.7						75	670	670	0	1.0
70	260	260	0	0.5						80	710	710	0	0.6
73	265	265	0	0.6						85	740	740	0	0.5
75	270	270	0	0.5						90	770	770	0	0.4
80	280	280	0	0.4						95	790	790	0	0.4
85	285	285	0	0.3						100	800	800	0	0.3
										105	800	800	0	0.3
										110	800	800	0	0.3

For all problem instances with fewer than 50 locations and $m = 1$, a near optimal solution is found in less than 1 second. In 38 of 49 problems published by Tsiligirides, the solution is equal to the optimal one. The average gap ((optimal_score-GLS_score)/optimal_score) with the optimal solution is 1.75%. The results are presented in Table 3. In all tables the gap is presented as a percentage and the CPU is given in seconds.

The CPU time in Table 3 clearly shows that problems with a time budget that allows to select almost no locations or (almost) all locations, are easier to solve than problems where around half of the locations can be selected. The 11 test problems of Fischetti et al. were all chosen with a budget that allows around half of the points to be selected. For three of the problems published by Fischetti et al. the GLS algorithm determines a better solution than the published "optimal" one. This is probably caused by the fact that Fischetti et al. multiply all the distances by 100 and then round them to the nearest integer, before starting the optimisation. Calculations with CPLEX confirm that the published solutions are not optimal, but that the results of the new algorithm are. For these test problems an average gap of 6.11% is obtained. These results are presented in Table 4.

The optimal routes obtained by the GLS algorithm (and CPLEX) are presented in Appendix. The time budget consumed by each route is indicated in the third column.

The rest of the test results are also presented in Appendix. The most important results are summarised in Tables 5–7. For the diamond-shaped and square-shaped test problems of Chao et al. ($n = 64, 66$; $m = 1$), the GLS algorithm performs almost as well as Chao's algorithm. The average gap with the scores obtained by Chao et al. is 0.81%, but the computational times are reduced substantially. The heuristic of Chao et al. was executed on a SUN 4/370 workstation.

On test sets 1 and 3 of Chao et al., the GLS algorithm performs almost as well as the Tabu Search heuristic of Tang and Miller-Hooks. The average gap is 0.33%, but the average computational time of the GLS is only 1.0 second compared to 2.1 seconds for Tang and Miller-Hooks. The heuristic of Tang and Miller-Hooks was executed on a DEC Alpha XP1000 computer with 1 GB RAM and 1.5 GB swap.

The results for test sets 4-7 ($n = 64, 66, 100, 102$; $m = 2, 3, 4$) are compared with the algorithm of Tang and Miller-Hooks (2005) and with the "z_min value" of the "FAST VNS_FEASIBLE" algorithm of Archetti et al. (2007). Compared to the other algorithms they developed, the FAST VNS_FEASIBLE algorithm is, according to the authors, "*an excellent compromise between solution quality and computational effort*". The z_min value is the lowest score obtained in three runs and is "*a guaranteed value, related to the robustness of the algorithm with respect to the random choices*" (Archetti et al., 2007). Since no randomness appears in the GLS algorithm the obtained results are always the same and one run is enough.

The obtained solutions of Tang and Miller-Hooks and Archetti et al. are slightly better than the GLS algorithm, on average 0.84% and 1.29%. However, the computational times of the GLS algorithm are lower. For three of the four sets the average CPU is shorter compared to the fastest algorithm of Archetti et al. (2007), and for all sets the maximal CPU is much shorter (see Table 7).

The outcomes of the GLS algorithm have also been compared with the Best Result (BR) of the four algorithms of Archetti et al. (2007). It appears that GLS did not produce a better outcome than BR, but, the gap is moderate and the computational time is decreased.

To show the necessity of exploring the whole solution space more intensively, all the test problems were also solved without

Table 4
Results on test problems of Fischetti et al., $m = 1$

n	T_{max}	GLS	OPT		Gap	CPU
			Fischetti	Cplex		
32	4127	160	160	160	0	0.7
21	2299	210	**205**	**210**	0	0.3
33	4878	510	**500**	**510**	0	0.7
31	191	7600	7600	7600	0	0.7
52	213	1707	**1674**	**1707**	0	3.2
101	3955	3265	3359		2.8	8.3
101	10641	3165	3212		1.5	11.6
201	14684	5428	6547		17	29.4
300	24096	8088	9161		12	64.9
319	21045	9145	10900		16	101.2
401	7641	11362	13648		17	215.5

Table 5
Score performance of GLS compared to other algorithms

Name	n	m	Average score gap			Maximal score gap		
			Chao	Tang	Archetti	Chao	Tang	Archetti
Diamond-shaped	64	1	0.68			4.4		
Square-shaped	66	1	0.88			10.0		
Set 1 and 3	32, 33	2, 3, 4		0.3			4.9	
Set 4	100	2, 3, 4		1.0	1.8		7.8	8.2
Set 5	66	2, 3, 4		0.5	1.2		6.1	6.8
Set 6	64	2, 3, 4		0.8	1.2		9.2	9.2
Set 7	102	2, 3, 4		1.1	1.0		11.0	11.5

Table 6
Number of times GLS performs better than, equal to or worse than other algorithms

Results of GLS	Chao	Tang	Archetti
Better	8	49	22
Equal	15	83	75
Worse	17	104	115
Total number of problems the comparison is based on	40	236	212

Table 7
Average and maximal CPU time compared to other algorithms

Name	n	m	Average CPU				Max CPU			
			GLS	Chao	Tang	Archetti	GLS	Chao	Tang	Archetti
Diamond-shaped	64	1	**2.2**	177.0			**4.5**	264.8		
Square-shaped	66	1	**2.3**	158.6			**4.4**	502.4		
Set 1 and 3	32,33	2,3,4	**1.0**		2.1		**2.5**		6.6	
Set 4	100	2,3,4	**12.7**		193.2	22.5	**24.1**		796.7	121.0
Set 5	66	2,3,4	**5.0**		23.6	34.2	**12.2**		71.3	30.0
Set 6	64	2,3,4	**5.8**		18.9	8.7	**7.9**		45.7	20.0
Set 7	102	2,3,4	16.7		104.8	**10.3**	**42.7**		432.6	90.0

using the *Disturb* heuristic. On average, this procedure led to scores which are 3.8% worse than the results with *Disturb*. For 181 of the 336 test problems a higher score is found due to the *Disturb* heuristic, for 11 problems a lower score is found. If guided local search is not applied, the scores are 6.2% worse than the current results. For 263 test problems a better score is determined thanks to GLS, in only two cases a lower score is obtained. This clearly demonstrates the usefulness of applying *Disturb* and the guided local search technique.

6. Conclusions and further work

The presented algorithm uses a combination of local search heuristics to solve the (Team) orienteering problem (TOP). Guided local search (GLS) is used to improve the two most contributing local search heuristics. The obtained results are almost of the same quality as the results of the best known heuristics. However, the computational time is reduced significantly. Applying GLS to solve the TOP is clearly a very promising technique. Furthermore, the necessity of "diversifying" during the search algorithm is made explicit in this paper. For some published test problems, the algorithm even determines a better solution than the published "optimal" one.

To further improve the algorithm, the presented local search algorithms could be used as moves in a variable neighbourhood search framework (Hansen and Mladenovic, 2001). This framework can then be used within GLS to obtain higher scores.

Just like most metaheuristics, the presented GLS algorithm requires accurate parameter setting. Instead of determining these parameters based on preliminary testing it might be useful to

Table A.1
New optimal routes for test set of Fischetti et al.

n	T_{max}	Time used	Score	Route
32	4127	4057.74	160	1 28 27 31 26 25 23 22 21 12 11 10 9 8 2 3 7 6 32
21	2299	2264.78	210	1 7 6 5 4 3 2 8 9 10 11 14 21
33	4878	4814.49	510	1 24 22 7 5 14 4 20 17 16 15 13 3 6 2 8 31 12 29 26 33
31	191	189.91	7600	1 20 30 26 25 2 3 22 31
52	213	210.81	1707	1 32 27 51 46 12 17 4 41 19 42 44 15 37 5 38 49 10 9 50 34 21 29 20 3 22 52

Table A.3
Results on test sets 1 and 3 of Chao et al. compared to Tang and Miller-Hooks

n	m	T_{max}	GLS		Tang		Gap score
			Score	CPU	Score	CPU	
32	2	23.0	130	2.5	135	1.3	3.7
	3	13.3	70	0.8	70	0.8	0.0
		21.7	175	0.8	170	1.4	−2.9
		24.3	205	1.6	205	2.6	0.0
		25.0	210	1.1	220	1.5	4.5
	4	12.5	75	0.7	75	0.8	0.0
		18.2	165	1.0	165	1.3	0.0
		18.8	175	0.9	175	1.5	0.0
33	2	12.5	180	0.4	180	1.2	0.0
		17.5	260	0.8	250	0.8	−4.0
		20.0	300	1.9	290	1.2	−3.4
		25.0	390	1.3	410	3.1	4.9
		27.5	440	0.7	460	3.8	4.3
		30.0	510	1.2	490	1.5	−4.1
		42.5	680	1.4	690	6.6	1.4
		47.5	750	0.6	760	5.4	1.3
	3	25.0	520	1.3	510	2.0	−2.0
		28.3	590	1.1	590	2.0	0.0
		30.0	640	0.6	640	2.1	0.0
		31.7	680	1.1	680	3.1	0.0
		36.7	720	0.9	750	3.3	4.0
	4	10.0	190	0.3	190	0.6	0.0
		15.0	310	0.6	310	0.8	0.0
		22.5	560	0.9	560	0.7	0.0
Average				*1.0*		*2.1*	*0.33*

Table A.2
Results on diamond-shaped and square-shaped test problems of Chao et al.

Diamond-shaped test problems $n = 64$; $m = 1$						Square-shaped test problems $n = 66$; $m = 1$					
T_{max}	GLS		Chao		Gap score	T_{max}	GLS		Chao		Gap score
	Score	CPU	Score	CPU			Score	CPU	Score	CPU	
15	96	0.8	96	13.0	0.0	5	10	0.0	10	1.1	0.0
20	294	2.0	294	27.9	0.0	10	40	0.2	40	0.5	0.0
25	390	2.5	390	238.9	0.0	15	120	1.3	120	4.3	0.0
30	474	4.5	474	74.5	0.0	20	175	1.7	195	6.2	10.0
35	552	2.3	570	139.8	3.2	25	290	1.6	290	73.4	0.0
40	702	2.5	714	137.9	1.7	30	400	2.0	400	54.8	0.0
45	780	3.2	816	205.0	4.4	35	465	2.8	460	32.4	−1.1
50	888	2.1	900	231.6	1.3	40	575	3.2	575	98.9	0.0
55	972	1.4	984	246.2	1.2	45	640	2.9	650	58.1	1.5
60	1062	2.3	1044	264.8	−1.7	50	710	3.0	730	68.1	2.7
65	1110	1.8	1116	232.6	0.5	55	825	3.7	825	65.2	0.0
70	1188	1.5	1176	231.0	−1.0	60	905	4.1	915	84.6	1.1
75	1236	2.1	1224	223.1	−1.0	65	935	3.3	980	82.2	4.6
80	1260	1.5	1272	212.3	0.9	70	1070	3.5	1070	119.0	0.0
						75	1140	4.4	1140	116.7	0.0
						80	1195	3.1	1215	108.9	1.6
						85	1265	2.3	1270	132.5	0.4
						90	1300	2.3	1340	502.4	3.0
						95	1385	2.2	1380	467.1	−0.4
						100	1445	2.1	1435	128.6	−0.7
						105	1505	2.2	1510	316.3	0.3
						110	1560	2.0	1550	469.9	−0.6
						115	1580	1.7	1595	474.6	0.9
						120	1635	1.2	1635	358.0	0.0
						125	1665	1.3	1655	268.9	−0.6
						130	1680	0.7	1680	32.1	0.0
Average	*2.2*		*177.0*		*0.68*			*2.3*		*158.6*	*0.88*

Table A.4
Results on test set 4 compared to Tang and Miller-Hooks and Archetti et al.

m	T_{max}	Score			Gap with		m	T_{max}	Score			Gap with	
		GLS	Tang	Archetti	Tang	Archetti			GLS	Tang	Archetti	Tang	Archetti
2	25.0	206	202	206	−2.0	0.0		50.0	881	906	912	2.8	3.4
	30.0	341	341	341	0.0	0.0		53.3	948	951	955	0.3	0.7
	35.0	452	438	452	−3.2	0.0		56.7	1014	1005	1033	−0.9	1.8
	40.0	528	517	527	−2.1	−0.2		60.0	1067	1119	1092	4.6	2.3
	45.0	613	593	612	−3.4	−0.2		63.3	1121	1151	1168	2.6	4.0
	50.0	670	666	672	−0.6	0.3		66.7	1153	1218	1184	5.3	2.6
	55.0	733	749	745	2.1	1.6		70.0	1236	1249	1227	1.0	−0.7
	60.0	791	827	818	4.4	3.3		73.3	1239	1265	1262	2.1	1.8
	65.0	845	915	857	7.7	1.4		76.7	1261	1282	1266	1.6	0.4
	70.0	956	914	954	−4.6	−0.2		80.0	1297	1288	1298	−0.7	0.1
	75.0	945	963	1001	1.9	5.6	4	20.0	38	38	38	0.0	0.0
	80.0	1040	1022	1052	−1.8	1.1		22.5	183	182	183	−0.5	0.0
	85.0	1040	1089	1098	4.5	5.3		25.0	324	315	324	−2.9	0.0
	90.0	1111	1150	1134	3.4	2.0		27.5	461	453	461	−1.8	0.0
	95.0	1140	1175	1194	3.0	4.5		30.0	555	554	571	−0.2	2.8
	100.0	1220	1208	1227	−1.0	0.6		32.5	622	627	653	0.8	4.7
	105.0	1253	1255	1258	0.2	0.4		35.0	694	732	723	5.2	4.0
	110.0	1281	1277	1275	−0.3	−0.5		37.5	815	819	819	0.5	0.5
	115.0	1286	1294	1298	0.6	0.9		40.0	848	875	876	3.1	3.2
	120.0	1298	1306	1306	0.6	0.6		42.5	839	910	914	7.8	8.2
3	20.0	38	38	38	0.0	0.0		45.0	932	977	961	4.6	3.0
	23.3	193	192	193	−0.5	0.0		47.5	969	1014	1050	4.4	7.7
	26.7	333	333	333	0.0	0.0		50.0	1073	1056	1118	−1.6	4.0
	30.0	468	465	461	−0.6	−1.5		52.5	1120	1124	1156	0.4	3.1
	33.3	553	579	579	4.5	4.5		55.0	1143	1165	1198	1.9	4.6
	36.7	645	646	647	0.2	0.3		57.5	1215	1243	1249	2.3	2.7
	40.0	707	709	715	0.3	1.1		60.0	1275	1255	1251	−1.6	−1.9
	43.3	768	785	799	2.2	3.9							
	46.7	848	860	855	1.4	0.8					*Average*	*1.0*	*1.8*

Table A.5
Results on test set 5 compared to Tang and Miller-Hooks and Archetti et al.

m	T_{max}	Score			Gap with		m	T_{max}	Score			Gap with	
		GLS	Tang	Archetti	Tang	Archetti			GLS	Tang	Archetti	Tang	Archetti
2	5.0	20	20	20	0.0	0.0		25.0	870	835	870	−4.2	0.0
	7.5	50	50	50	0.0	0.0		26.7	960	990	990	3.0	3.0
	10.0	80	80	80	0.0	0.0		28.3	1025	1065	1070	3.8	4.2
	12.5	180	180	180	0.0	0.0		30.0	1080	1115	1125	3.1	4.0
	15.0	240	240	240	0.0	0.0		31.7	1130	1175	1190	3.8	5.0
	17.5	320	320	315	0.0	−1.6		33.3	1245	1240	1255	−0.4	0.8
	20.0	385	410	410	6.1	6.1		35.1	1330	1330	1345	0.0	1.1
	22.5	465	480	480	3.1	3.1		36.7	1420	1410	1415	−0.7	−0.4
	25.0	580	560	580	−3.6	0.0		38.3	1460	1465	1475	0.3	1.0
	27.5	665	670	670	0.7	0.7		40.1	1525	1530	1540	0.3	1.0
	30.0	760	770	800	1.3	5.0		41.7	1585	1580	1590	−0.3	0.3
	32.5	835	860	860	2.9	2.9		43.3	1635	1635	1635	0.0	0.0
	35.0	920	920	925	0.0	0.5	4	3.8	20	20	20	0.0	0.0
	37.5	1010	975	1020	−3.6	1.0		5.0	20	20	20	0.0	0.0
	40.0	1080	1090	1150	0.9	6.1		6.2	20	20	20	0.0	0.0
	42.5	1155	1185	1190	2.5	2.9		7.5	80	80	80	0.0	0.0
	45.0	1245	1260	1260	1.2	1.2		8.8	140	140	140	0.0	0.0
	47.5	1295	1310	1340	1.1	3.4		10.0	140	140	140	0.0	0.0
	50.0	1360	1380	1400	1.4	2.9		11.2	240	240	240	0.0	0.0
	52.5	1445	1445	1460	0.0	1.0		12.5	340	340	340	0.0	0.0
	55.0	1500	1500	1500	0.0	0.0		13.8	340	340	340	0.0	0.0
	57.5	1560	1560	1560	0.0	0.0		15.0	430	430	430	0.0	0.0
	60.0	1610	1610	1590	0.0	−1.3		16.2	550	555	550	0.9	0.0
	62.5	1620	1630	1635	0.6	0.9		17.5	620	620	620	0.0	0.0
	65.0	1680	1665	1670	−0.9	−0.6		18.8	690	680	690	−1.5	0.0
3	3.3	15	15	15	0.0	0.0		20.0	760	760	760	0.0	0.0
	5.0	20	20	20	0.0	0.0		21.2	830	860	860	3.5	3.5
	6.7	60	60	60	0.0	0.0		22.5	920	960	960	4.2	4.2
	8.3	95	95	95	0.0	0.0		23.8	1005	1000	1025	−0.5	2.0
	10.0	110	110	110	0.0	0.0		25.0	1110	1100	1160	−0.9	4.3
	11.7	185	185	185	0.0	0.0		26.2	1300	1275	1300	−2.0	0.0
	13.3	260	260	260	0.0	0.0		27.5	1275	1310	1320	2.7	3.4
	15.0	335	335	335	0.0	0.0		28.8	1375	1380	1380	0.4	0.4
	16.7	470	470	470	0.0	0.0		30.0	1410	1410	1450	0.0	2.8
	18.3	485	495	495	2.0	2.0		31.2	1520	1520	1520	0.0	0.0
	20.0	545	575	585	5.2	6.8		32.5	1575	1575	1620	0.0	2.8
	21.7	645	650	650	0.8	0.8							
	23.3	755	755	745	0.0	−1.3					*Average*	*0.5*	*1.2*

apply a genetic algorithm as an upper level heuristic to determine the best parameter set. Very promising results were obtained with this technique and are described in Souffriau et al. (2006). Furthermore, the problems solved in this paper are standard (T)OP. The algorithm should be extended in the future to solve the tourist trip design problem, i.e. a TOP with visiting durations, time windows, extra benefits for certain combinations of locations, etc.

Appendix. Test results

See Tables A.1–A.7.

Table A.6
Results on test set 6 compared to Tang and Miller-Hooks and Archetti et al.

m	T_{max}	Score			Gap with	
		GLS	Tang	Archetti	Tang	Archetti
2	15.0	192	192	192	0.0	0.0
	17.5	360	360	360	0.0	0.0
	20.0	534	588	588	9.2	9.2
	22.5	660	660	660	0.0	0.0
	25.0	780	780	780	0.0	0.0
	27.5	888	888	888	0.0	0.0
	30.0	912	936	948	2.6	3.8
	32.5	1032	1032	1032	0.0	0.0
	35.0	1104	1116	1116	1.1	1.1
	37.5	1158	1188	1170	2.5	1.0
	40.0	1230	1260	1242	2.4	1.0
3	15.0	276	282	282	2.1	2.1
	16.7	438	444	444	1.4	1.4
	18.3	642	612	642	−4.9	0.0
	20.0	828	828	828	0.0	0.0
	21.7	888	876	894	−1.4	0.7
	23.3	996	990	1002	−0.6	0.6
	25.0	1074	1080	1080	0.6	0.6
	26.7	1122	1152	1170	2.6	4.1
4	15.0	360	366	366	1.6	1.6
	16.2	528	522	528	−1.1	0.0
	17.5	684	696	696	1.7	1.7
	18.8	912	912	912	0.0	0.0
	20.0	1068	1068	1068	0.0	0.0
				Average	*0.8*	*1.2*

Table A.7
Results on test set 7 compared to Tang and Miller-Hooks and Archetti et al.

m	T_{max}	Score			Gap with		m	T_{max}	Score			Gap with	
		GLS	Tang	Archetti	Tang	Archetti			GLS	Tang	Archetti	Tang	Archetti
2	10.0	30	30	30	0.0	0.0		80.0	675	681	666	0.9	−1.4
	20.0	64	64	64	0.0	0.0		86.7	745	756	727	1.5	−2.5
	30.0	101	101	101	0.0	0.0		93.3	789	789	808	0.0	2.4
	40.0	190	190	190	0.0	0.0		10.0	833	874	859	4.7	3.0
	50.0	279	290	289	3.8	3.5		106.7	915	922	906	0.8	−1.0
	60.0	340	382	384	11.0	11.5		113.3	957	966	969	0.9	1.2
	70.0	440	459	457	4.1	3.7		120.0	1013	1011	1022	−0.2	0.9
	80.0	521	521	518	0.0	−0.6		126.7	1066	1061	1046	−0.5	−1.9
	90.0	579	578	574	−0.2	−0.9		133.3	1102	1098	1110	−0.4	0.7
	100.0	633	638	636	0.8	0.5	4	10.0	30	30	30	0.0	0.0
	110.0	697	702	695	0.7	−0.3		15.0	46	46	46	0.0	0.0
	120.0	758	767	758	1.2	0.0		20.0	79	79	79	0.0	0.0
	130.0	821	817	821	−0.5	0.0		25.0	123	123	123	0.0	0.0
	140.0	873	864	863	−1.0	−1.2		30.0	164	164	164	0.0	0.0
	150.0	937	914	922	−2.5	−1.6		35.0	209	217	217	3.7	3.7
	160.0	984	987	1002	0.3	1.8		40.0	285	285	285	0.0	0.0
	170.0	1037	1017	1021	−2.0	−1.6		45.0	359	359	366	0.0	1.9
	180.0	1076	1067	1080	−0.8	0.4		50.0	462	462	462	0.0	0.0
	190.0	1108	1116	1127	0.7	1.7		55.0	499	503	514	0.8	2.9
	200.0	1155	1165	1161	0.9	0.5		60.0	573	576	575	0.5	0.3
3	13.3	46	46	46	0.0	0.0		65.0	638	643	639	0.8	0.2
	20.0	79	79	79	0.0	0.0		70.0	698	726	699	3.9	0.1
	26.7	117	117	117	0.0	0.0		75.0	739	776	770	4.8	4.0
	33.3	175	175	175	0.0	0.0		80.0	813	832	846	2.3	3.9
	40.0	247	247	247	0.0	0.0		85.0	888	905	899	1.9	1.2
	46.7	344	344	344	0.0	0.0		90.0	938	966	970	2.9	3.3
	53.3	418	416	425	−0.5	1.6		95.0	994	1019	1021	2.5	2.6
	60.0	480	481	487	0.2	1.4		100.0	1020	1067	1077	4.4	5.3
	66.7	539	563	556	4.3	3.1							
	73.3	586	632	619	7.3	5.3					*Average*	*1.1*	*1.0*

References

Archetti, C., Hertz, A., Speranza, M., 2007. Metaheuristics for the team orienteering problem. Journal of Heuristics 13, 49–76.

Arkin, E., Mitchell, J., Narasimhan, G., 1998. Resource-constrained geometric network optimization. In: Proceedings of the 14th ACM Symposium on Computational Geometry, pp. 307–316.

Beullens, P., Muyldermans, L., Cattrysse, D., Van Oudheusden, D., 2003. A guided local search heuristic for the capacitated arc routing problem. European Journal of Operational Research 147, 629–643.

Butt, S., Cavalier, T., 1994. A heuristic for the multiple tour maximum collection problem. Computers and Operations Research 21, 101–111.

Chao, I., 1993. Algorithms and solutions to multi-level vehicle routing problems. Ph.D. Dissertation, Applied Mathematics Program, University of Maryland, College Park, USA.

Chao, I., Golden, B., Wasil, E., 1996a. The team orienteering problem. European Journal of Operational Research 88, 464–474.

Chao, I., Golden, B., Wasil, E., 1996b. A fast and effective heuristic for the orienteering problem. European Journal of Operational Research 88, 475–489.

Feillet, D., Dejax, P., Gendreau, M., 2005. Travelling salesman problems with profits. Transportation Science 39, 188–205.

Fischetti, M., Salazar, J., Toth, P., 1998. Solving the orienteering problem through branch-and-cut. INFORMS Journal on Computing 10, 133–148.

Gendreau, M., Laporte, G., Semet, F., 1998a. A tabu search heuristic for the undirected selective travelling salesman problem. European Journal of Operational Research 106, 539–545.

Gendreau, M., Laporte, G., Semet, F., 1998b. A branch-and-cut algorithm for the undirected selective travelling salesman problem. Networks 32, 263–273.

Golden, B., Levy, L., Vohra, R., 1987. The orienteering problem. Naval Research Logistics 34, 307–318.

Golden, B., Wang, Q., Liu, L., 1988. A multifaceted heuristic for the orienteering problem. Naval Research Logistics 35, 359–366.

Hansen, P., Mladenovic, N., 2001. Variable neighbourhood search: Principles and applications. European Journal of Operational Research 130, 449–467.

Kilby, P., Prosser, P., Shaw, P., 1997. Guided local search for the vehicle routing problem. In: Proceedings of the Second International Conference on Metaheuristics.

Laporte, G., Martello, S., 1990. The selective travelling salesman problem. Discrete Applied Mathematics 26, 193–207.

Miller, C., Tucker, A., Zemlin, R., 1960. Integer programming formulations and travelling salesman problems. Journal of the ACM 7, 326–329.

Mills, P., Tsang, E., Ford, J., 2003. Applying an extended guided local search to the quadratic assignment problem. Annals of Operations Research 118, 121–135.

Pekny, J., Miller, D., 1990. 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.

Ramesh, R., Brown, K., 1991. An efficient four-phase heuristic for the generalised orienteering problem. Computers and Operations Research 18, 151–165.

Ramesh, R., Yoon, Y., Karwan, M., 1992. An optimal algorithm for the orienteering tour problem. ORSA Journal on Computing 4, 155–165.

Souffriau, W., Vansteenwegen, P., Vanden Berghe, G., Van Oudheusden, D., 2006. Multilevel metaheuristics for the orienteering problem. In: Seventh EU/Meeting on Adaptive, Self-Adaptive and Multilevel Metaheuristics, Malaga.

Tang, H., Miller-Hooks, E., 2005. A tabu search heuristic for the team orienteering problem. Computer and Operations Research 32, 1379–1407.

Tsiligirides, T., 1984. Heuristic methods applied to orienteering. Journal of the Operational Research Society 35, 797–809.

Vansteenwegen, P., Van Oudheusden, D., 2007. The mobile tourist guide: An OR opportunity. OR Insights 20, 21–27.

Voudouris, C., Tsang, E., 1999. Guided local search and its application to the travelling salesman problem. European Journal of Operational Research 113, 469–499.

Zhong, Y., Cole, M., 2005. A vehicle routing problem with backhauls and time windows: A guided local search solution. Transportation Research Part E 41, 131–144.