J Heuristics (2007) 13:49–76
DOI 10.1007/s10732-006-9004-0

Metaheuristics for the team orienteering problem

**Claudia Archetti · Alain Hertz ·
Maria Grazia Speranza**

Received: 3 October 2005 / Revised: 3 February 2006 / Accepted:
22 March 2006 / Published online: 21 December 2006

Abstract The Team Orienteering Problem (TOP) is the generalization to the case of multiple tours of the Orienteering Problem, known also as Selective Traveling Salesman Problem. A set of potential customers is available and a profit is collected from the visit to each customer. A fleet of vehicles is available to visit the customers, within a given time limit. The profit of a customer can be collected by one vehicle at most. The objective is to identify the customers which maximize the total collected profit while satisfying the given time limit for each vehicle. We propose two variants of a generalized tabu search algorithm and a variable neighborhood search algorithm for the solution of the TOP and show that each of these algorithms beats the already known heuristics. Computational experiments are made on standard instances.

Keywords Team orienteering problem · Selective traveling salesman problem · Tabu search heuristic · Variable neighborhood search heuristic

1 Introduction

A huge number of papers appear in the literature which study the well known Traveling Salesman Problem (TSP) and its generalizations to the case of multiple vehicles known as Vehicle Routing Problems (VRPs). While there exists one and only one TSP, many problems belong to the class of VRPs (see Toth and Vigo, 2002). In the TSP and in the

C. Archetti (✉) · M. G. Speranza
University of Brescia, Department of Quantitative Methods, Brescia, Italy
e-mail: archetti@eco.unibs.it

M. G. Speranza
e-mail: speranza@eco.unibs.it

A. Hertz
École Polytechnique and GERAD, Montréal, Canada
e-mail: alain.hertz@gerad.ca

VRPs all customers need to be visited. This means that in the situations modeled all customers are known at the time the optimization model is run. Moreover, the solution found by the model will not need to be modified later. While this is generally the case, there are many practical problems where are many other practical problems where some of these assumptions are not valid. For example, not all customers may need to be visited or the problem has a dynamic structure and the solution found may need to be modified while it is being implemented. This implies that the problems have a different structure and that this structure needs to be explicitly modeled.

Let us consider some situations where not all customers need to be visited when the optimization model is run. Consider the situation where all customers need to be visited but not necessarily in the same tour or set of tours, for instance in the cases where a customer has to be visited within a given time period, say three days. Then, when a tour or a set of tours has to be organized for a given day, there are customers that need to be visited but also customers that may be visited or whose visit may be postponed. In this case the lack of need to serve all customers in the same day comes from the dynamic nature of the problem. The customers to be served can be selected each day on the basis of their total value, assigning to each customer a value that measures the profit or the priority and taking into account restrictions on the vehicles available for the service. Another situation arises when customers have to be selected within a given set. Nowadays it is more and more frequent that demands for transportation service are posted on the web, usually in specific databases, and the carriers can pick up those demands and offer their service to some of these customers. Thus, the carrier has to select within the set of potential customers those who are most convenient for him. The carrier may need to take into account the sets of customers which, as traditionally, need to be served. Such customers will be assigned a very large value.

When a set of customers needs to be selected and a single tour organized, the optimization problems become variants of the TSP. A profit is associated with each customer. On the other hand the traveling cost or time needs to be taken into account. A recent survey by Feillet et al. (2004) defines those problems as the TSPs with profits. The objective function may be the maximization of the collected total profit (Orienteering Problem), the minimization of the total traveling cost (Prize-Collecting TSP) or the optimization of a combination of both (Profitable Tour Problem). While some of the TSPs with profit have been investigated by a number of researchers, such as the Orienteering Problem (OP), and for some others little can be found in the literature, very few papers are available for any of the extensions of the TSPs with profits to the case of multiple tours. We call this class of problems the VRPs with profits. The short list of papers in which multi-vehicle routing problems with profits are addressed is mentioned in the survey by Feillet et al. (2004) dedicated to the single vehicle case.

In this paper we investigate the VRP with profits which is the extension to the case of multiple tours of the most studied TSP with profits, namely the OP. In the OP, given a set of potential customers with associated profit and given the distances between pairs of customers, the objective is to find the subset of customers for which the collected profit is maximum, given a constraint on the total length of the tour. The OP is also called the Selective Traveling Salesman Problem (STSP). The name orienteering comes from an outdoor sport usually played on mountains or forest areas. Given a specified set of points, each competitor, with the help of a map and a compass, has to visit as many points as possible within a specified time limit. The competitor starts at a given point and has to return to the same point. Golden et al. (1984) proposed to apply the modeling as an OP of a vehicle routing problem with an inventory component. The extension of the Orienteering Problem to the case of multiple tours is known as the Team Orienteering Problem (TOP).

The TOP appeared in the literature in a paper by Butt and Cavalier (1994) under the name Multiple Tour Maximum Collection Problem, while the definition of TOP was introduced by Chao et al. (1996). In this paper a heuristic algorithm is presented together with an interesting variant of an algorithm proposed in Tsiligirides (1984). A tabu search heuristic has been recently proposed and compared with Chao, Golden and Wasil heuristic by Tang and Miller-Hooks (2005).

Among the metaheuristics proposed for the solution of combinatorial optimization problems, tabu search (see, for example, Gendreau et al., 1994) has been shown to be very effective for a variety of vehicle routing problems. Another interesting metaheuristic is the variable neighborhood search (see Mladenovic and Hansen, 1997). In this paper the effectiveness of these metaheuristics is confirmed. We propose two variants of a generalized tabu search algorithm and a variable neighborhood search algorithm for the solution of the TOP and show that such heuristics obtain very good results, in terms of solution quality, within a reasonable amount of time. The results have been compared with the results obtained by the heuristics proposed by Chao et al. (1996) and by Tang and Miller-Hooks (2005).

The paper is organized as follows. In Section 2 the TOP is defined, while in Section 3 the proposed heuristics are presented. The computational results on a large set of standard instances are presented and discussed in Section 4. Finally, some conclusions are drawn.

2 The team orienteering problem

We consider a complete undirected graph $G = (V, E)$, where $V = 1, \dots, n$ is the set of vertices and E is the set of edges. Vertex 1 is the starting and ending point of each tour and each vertex $i = 2, \dots, n$ represents a potential customer. An edge $(i, j) \in E$ represents the possibility to travel from customer i to customer j. A nonnegative profit s_i is associated with each vertex ($s_1 = 0$) and a symmetric travel time c_{ij} is associated with each edge $(i, j) \in E$. A set of m vehicles is available to visit the customers. Each vehicle can visit any subset of the customers V within a given time limit $T_{\max}$. The profit of each customer i can be collected by one vehicle at most.

The objective of the Team Orienteering Problem (TOP) is to maximize the total collected profit satisfying the time limit $T_{\max}$ for each vehicle.

As already mentioned in the introduction, the TOP has as special case the Orienteering Problem (OP), known also as Selective Traveling Salesman Problem. The OP has been shown to be NP-hard by Golden et al. (1987). Therefore, the TOP is NP-hard.

3 Solution algorithms

Let S be the set of solutions to a combinatorial optimisation problem. For a solution $s \in S$, let $N(s)$ denote a *neighborhood* of s which is defined as a set of *neighbor*

Choose an initial solution s; set $TL = \emptyset$ (tabu list); set $s^* = s$ (best solution)
Repeat the following until a stopping criterion is met

- Determine a best solution $s' \in N(s)$ such that either $s' \notin TL$ or s' is better than s^*
- If s' is better than s^* then set $s^* := s'$
- Set $s := s'$ and update TL

Fig. 1 Basic tabu search

Choose an initial solution s; set $k = 1$
Repeat the following until a stopping criterion is met

- *shaking*: Generate s' at random in $N^{(k)}(s)$
- *local search*: Apply a local search on s' using $N^{(0)}$. Let s'' be the resulting solution
- *update*: If s'' is better than s then set $s = s''$ and $k = 1$, else set $k = (k \bmod k_{max})+1$

Fig. 2 Basic VNS

solutions in S obtained from s by performing a *local change* on it. Local search techniques visit a sequence $s_0, \ldots, s_t$ of solutions, where s_0 is an initial solution and $s_{i+1} \in N(s_i)$ $(i = 1, \ldots, t-1)$. Tabu search is one of the most famous local search techniques. It was introduced by Glover (1986). A description of the method and its concepts can be found in Glover and Laguna (1997). A basic tabu search is described in Fig. 1.

A few years ago, Hansen and Mladenović proposed a new solution technique called *Variable Neighborhood Search* (VNS for short). The main idea of this new method is to use various neighborhoods during the search. Given an incumbent s, a neighbor solution s' is generated according to one of these neighborhoods, and a local search is then applied to s' in order to obtain a possibly better solution s''. If s'' is better than s, then s'' becomes the new incumbent; otherwise, a different neighborhood is considered in order to try to improve upon solution s. Let $N^{(k)}$ $(k = 0, \ldots, k_{\max})$ denote a finite set of neighborhoods, where $N^{(k)}(s)$ is the set of solutions in the k-th neighborhood of s. A basic VNS (Hansen, Mladenović, 1999) is described in Fig. 2.

Notice that neighborhood $N^{(0)}$ is used in the local search phase, but not in the shaking phase. Solutions in $N^{(0)}(s)$ are typically much closer to s than those in $N^{(k)}(s)$ with $k > 0$. For this reason, a move from s to a solution $s' \in N^{(k)}(s)$ $(k > 0)$ is often called a *jump*.

We describe in this section two generalized tabu search algorithms and one VNS for the solution of the TOP. The three proposed algorithms follow the general scheme illustrated in Fig. 3. Given an incumbent solution s, we make a jump to a solution s'. We then apply a tabu search on s' in order to try to improve it. The resulting solution s'' is then compared to s. If we follow a VNS strategy, then s'' becomes the new incumbent only if s'' is better than s. In the generalized tabu search strategy, we set $s = s''$ even if s'' is worse than s. This process is repeated until some stopping criterion is met. More details on this general scheme will be given in Section 3.5.

For comparison, Tang and Miller-Hooks (2005) have embedded their tabu search in an adaptive memory procedure (Rochat and Taillard, 1995). More precisely, their algorithm is an iterative process that uses a central memory containing vehicle routes. At each iteration, the information contained in the central memory is used for producing

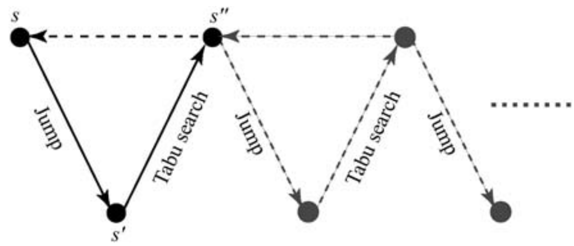


Fig. 3 General scheme of our solution methods

an offspring solution (i.e., a solution of the TOP) which is then possibly improved using tabu search. The so obtained solution is finally used to update the central memory.

3.1 Notations and basic concepts

The profit $P(C)$ of a set $C \subseteq V$ of customers is the total profit $\sum_{i \in C} s_i$ of the customers in C. The profit $P(r)$ of a route r is defined as the total profit of the customers on it, and its duration $T(r)$ is its total travel time. A route r is *feasible* if $T(r) \leq T_{\max}$. To measure the possible infeasibility of a route r, we define $I(r) = \max\{T(r) - T_{\max}, 0\}^2$. Hence, $I(r) = 0$ if and only if r is feasible. We define $I(r)$ as the square of the exceeding time, and not simply as the exceeding time, since we prefer to have a solution containing different routes with a small infeasibility (and thus easy to correct) rather than a solution having few routes with a great infeasibility.

For a set R of routes, $P(R) = \sum_{r \in R} P(r)$ denotes the total profit in R, $I(R) = \sum_{r \in R} I(r)$ is the total infeasibility in R, and $C(R)$ is the set of customers visited on the routes in R.

A *solution* is defined as a set of routes such that each route starts and ends at the depot, and each customer is visited exactly once by exactly one vehicle. We denote $R_{TOP}(s)$ the set of m most profitable routes in s, and $R_{NTOP}(s)$ the set of all remaining routes. A solution s is *feasible* if each route in s is feasible (i.e. $I(s) = 0$). A solution s is *admissible* if the routes in $R_{NTOP}(s)$ are feasible (i.e. $I(R_{NTOP}(s)) = 0$). Hence an admissible solution can have infeasible routes, but these are necessarily among the m most profitable ones. The aim of the TOP is to determine a feasible solution s that maximizes $P(R_{TOP}(s))$.

The customers in $C(R_{NTOP}(s))$ do not belong to the most profitable routes in s, but are already organized into routes. It is therefore much easier to get a new route with a high profit by inserting new customers in one of the routes in $R_{NTOP}(s)$ instead of creating a new route from scratch. Moreover, since our algorithms are based on moving customers between different routes, we can have routes which are on the border between $R_{TOP}(s)$ and $R_{NTOP}(s)$ and these moves can cause exchanges of routes between the two sets. Also, since there is no limit on the number of routes in $R_{NTOP}(s)$, while the profit of these routes is not taken into account in the objective value $P(R_{TOP}(s))$, there is no reason to accept infeasibility in $R_{NTOP}(s)$. Indeed, an infeasible route in $R_{NTOP}(s)$ can easily be split into a set of feasible routes $R_1, \ldots, R_r$ so that $C(R) = \cup_{i=1}^{r} C(R_i)$.

In a solution s, we denote $r_c(s)$ the route visiting customer c. For a customer c and a route $r \neq r_c(s)$, we denote $r + c$ the route obtained by adding c to r using the cheapest insertion technique. Similarly, given a route r and a customer c on r, we denote $r - c$

the route obtained from r by removing c and by linking its predecessor to its successor.

The tabu search algorithms we have implemented use two kinds of moves:

- *1-move*: In a 1-move, customer c is moved from its route to a route $r \neq r_c(s)$. Route r can be an empty route. Hence, $r_c(s)$ and r are replaced by $r_c(s) - c$ and $r + c$, respectively. A 1-move can be characterized by the pair (c, r). We denote $s \oplus (c, r)$ the resulting solution.
- *swap-move*: Let c and c' be two customers on two different routes. A swap-move consists in replacing $r_c(s)$ and $r_{c'}(s)$ by $(r_c(s) - c) + c'$ and $(r_{c'}(s) - c') + c$, respectively. A swap-move can be characterized by the ordered pair (c, c'). We denote $s \oplus (c, c')$ the resulting solution.

In the 1-moves, in general route r can be an empty route. In the cases where empty routes are not allowed this will be explicitly specified. Notice that if s is an admissible solution and (x, y) a move (i.e., a 1-move or a swap-move), then $s \oplus (x, y)$ is not necessarily admissible. We have therefore designed procedures that either reduce or totally remove the infeasibility in a subset of routes. These procedures are described in the next section.

3.2 Reducing and removing infeasibility

Let R be a set of routes such that $I(R) > 0$. The MAKE_FEASIBLE procedure described in Fig. 4 creates a new set of routes R' with $C(R') = C(R)$ and $I(R') = 0$. This is done by performing 1-moves that strictly reduce the infeasibility. Notice that such 1-moves always exist since it is always possible to remove a customer from a route r with $I(r) > 0$ and to insert it into a new route.

Notice that if s is an infeasible solution, then the output of MAKE_FEASIBLE(s) is a feasible solution. However, if s is a non-admissible solution, then the solution obtained by replacing $R_{NTOP}(s)$ by MAKE_FEASIBLE($R_{NTOP}(s)$) is not necessarily admissible. Indeed, some routes in MAKE_FEASIBLE($R_{NTOP}(s)$) are possibly obtained by adding customers from other routes in $R_{NTOP}(s)$, and these routes can therefore become more profitable than some infeasible routes in $R_{TOP}(s)$. Hence, several calls to MAKE_FEASIBLE can be necessary to transform a non-admissible solution into an admissible one. Procedure MAKE_ADMISSIBLE of Fig. 5 makes this transformation.

Procedure MAKE_FEASIBLE
Input: A set R of routes with $I(R) > 0$
Output: A set R' of routes with $C(R') = C(R)$ and $I(R') = 0$

Set $R' = R$
While $I(R') > 0$ do
- Choose a route $r \in R'$ with $I(r) > 0$ and a customer c in r (all choices are random)
- Choose a 1-move (c, r^*) such that $I(r^* + c) = 0$ and $T(r^* + c) - T(r^*)$ is minimum
- Replace r and r^* by $r - c$ and $r^* + c$ in R'

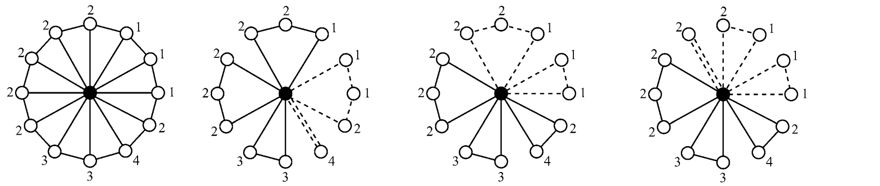
Fig. 4 Procedure that removes the infeasibility in a set of routes

Procedure MAKE_ADMISSIBLE
  Input: a non-admissible solution s
  Output: an admissible solution s'

Set $s' = s$;

While s' is not admissible do
  Replace the routes in $R_{NTOP}(s')$ by those in MAKE_FEASIBLE($R_{NTOP}(s')$)
  Update $R_{TOP}(s')$ and $R_{NTOP}(s')$

Fig. 5 Procedure that transforms a solution into an admissible one



(a) An instance | (b) A non admissible solution | (c) First call to MAKE_FEASIBLE | (d) Second call to MAKE_FEASIBLE

Fig. 6 Illustration of the MAKE_ADMISSIBLE procedure

The procedure is finite since the number of infeasible routes in s' strictly decreases at each call to MAKE_FEASIBLE. It allows to recover admissibility as soon as all infeasible routes of s' (if any) are in $R_{TOP}(s')$. This is illustrated on Fig. 6. The graph in Fig. 6(a) is the original network with the starting and ending point of each tour in the center (black vertex). All travel times on the edges are supposed equal to 1. When there is no edge between two vertices i and j, the travel time c_{ij} between these two vertices is equal to 2 since one can link i to j by going through the black vertex. The time limit $T_{\max}$ is equal to 3 and the numbers on the vertices are their profits. We suppose that m=3. A solution s is represented in Fig. 6(b). There are 5 routes, 3 in $R_{TOP}(s)$ and 2 in $R_{NTOP}(s)$. The routes in $R_{TOP}(s)$ are represented with solid lines while those in $R_{NTOP}(s)$ are represented with dashed lines. The solution is not admissible since one route in $R_{NTOP}(s)$ has a duration of $4 > 3 = T_{\max}$. The solution obtained by replacing $R_{NTOP}(s)$ by MAKE_FEASIBLE($R_{NTOP}(s)$) is represented in Fig. 6(c). It is not admissible since the route with a profit of 5 is not feasible and it now belongs to $R_{NTOP}(s)$. A second call to MAKE_FEASIBLE is necessary to obtain the admissible (but non feasible) solution of Fig. 6(d).

Given a set R of routes with $I(R) > 0$, the next procedure, called REDUCE_INF, creates a new set of routes R' with $C(R') = C(R)$, $\mid R' \mid \leq \mid R \mid$ and $I(R') \leq I(R)$. The REDUCE_INF procedure is a local search that follows the general scheme of Fig. 7. Neighbors are obtained by making 1-moves and swap-moves, but without creating any new route.

The value of a set of routes is measured using two functions F_1 and F_2.

- $F_1(R)$ is the total infeasibility $I(R)$ in R;
- $F_2(R)$ is the total duration $\sum_{r \in R} T(r)$ of the routes in R.

Procedure REDUCE_INF
Input: A set R of routes with $I(R) > 0$
Output: A set R' of routes with $C(R') = C(R)$, $\mid R' \mid \leq \mid R \mid$, and $I(R') \leq I(R)$

Set $R' = R$
While no stopping criterion is met do
- Determine the F-best 1-move m_1 (which does not create new routes)
- If $R \oplus m_1 <_F R$ then set R equal to $R \oplus m_1$
- Else determine the F-best swap-move m_2 and set R equal to the F-best solution among $R \oplus m_1$ and $R \oplus m_2$
- If $R <_F R'$ then set $R' = R$

Fig. 7 Procedure that reduces the infeasibility in a set R of routes

A set R of routes is said *F-better* than a set R' of routes if $F_1(R) < F_1(R')$ or $F_1(R) = F_1(R')$ and $F_2(R) < F_2(R')$. We denote $R <_F R'$. Given a solution s, the set R of routes in a solution of $N^{(k)}(s)$ is F-best in $N^{(k)}(s)$ if $R <_F R'$ for any $R' \neq R$ in a solution of $N^{(k)}(s)$.

In practice, procedure REDUCE_INF tries to reduce the infeasibility of a set of routes without changing the set of customers visited in these routes (so that the profit collected remains the same) and without creating new routes. To speed-up the search, procedure REDUCE_INF first tries to detect a 1-move m_1 so that $R \oplus m_1$ is F-better than R. If no such move exists, then R is replaced by the best solution obtained by performing a 1-move or a swap-move on R. The stopping criterion is fixed at 100 iterations without improvements. The procedure is a simple local search without any tabu list. This choice is motivated by the need to make the REDUCE_INF procedure very fast since it can be called a large number of times during the entire algorithm.

3.3 Jumps

We have designed two procedures for generating jumps from a given admissible solution s. In the first procedure, a jump from s is obtained by performing a series of 1-moves from $R_{NTOP}(s)$ to $R_{TOP}(s)$. The *amplitude* of such a jump is defined as the number of customers that are moved. We denote $J_k^1(s)$ the set of neighbors which can be obtained from s with such jumps of amplitude k. When moving a customer to $R_{TOP}(s)$, we try to avoid creating infeasibility. Details are given in Fig. 8.

Procedure JUMP_1
Input: An admissible solution s and an amplitude $k \leq \mid C(R_{NTOP}(s)) \mid$
Output: An admissible solution $s' \in J_k^1(s)$

Set $s' = s$
For i=1 to k do
- Choose a customer c at random in $C(R_{NTOP}(s'))$
- Determine a 1-move (c, r) with $r \in R_{TOP}(s')$ that minimizes $I(r + c) - I(r)$
  Ties are broken by choosing a 1-move with minimum insertion cost $T(r + c) - T(r)$
- Replace r by $r + c$ in $R_{TOP}(s')$

Fig. 8 First kind of jump

Procedure JUMP_2
Input: An admissible solution s and an amplitude $k \leq \mid C(R_{TOP}(s)) \mid$
Output: An admissible solution $s' \in J_k^2(s)$

- Set $R = R_{TOP}(s)$ and $R' = R_{NTOP}(s)$
- For i=1 to k do
  Choose a customer c at random in $C(R)$, add it to U and replace r_c by $r_c - c$ in R
- If $P(R') < P(U)$ then set $W = C(R')$ and set $R' = \emptyset$
  Else set $W = \emptyset$ and repeat the following until $P(W) \geq P(U)$
  Choose $c \in C(R')$ at random, add c to W and replace r_c by $r_c - c$ in R'
- For all $c \in U$ do (sequentially)
  Let Q be the set of 1-moves (c, r) such that $r \in R'$ or is a new route, and $I(r+c) = 0$
  Choose a 1-move $(c, r) \in Q$ with minimum insertion cost $T(r + c) - T(r)$
  Replace r with $r + c$ in R'
- For all $c \in W$ do (sequentially)
  Let Q be the set of 1-moves (c, r) such that $r \in R$ and $I(r + c) - I(r)$ is minimum
  Choose a 1-move $(c, r) \in Q$ with minimum insertion cost $T(r + c) - T(r)$
  Replace r with $r + c$ in R
- Set $s' = R \cup R'$ (i.e., s' is the solution made by the union of the routes in R and R')
  If s' is not admissible then replace s' with MAKE_ADMISSIBLE(s')

Fig. 9 Second kind of jump

The second kind of jump is obtained by moving a set U of customers from $R_{TOP}(s)$ to $R_{NTOP}(s)$, and a set W of customers from $R_{NTOP}(s)$ to $R_{TOP}(s)$. In order to have a chance to increase the profit in $R_{TOP}(s)$, we try to determine sets U and W such that $P(U) \leq P(W)$. This is done as follows. We first choose U at random in $C(R_{TOP}(s))$. Then, if $P(U) > P(R_{NTOP}(s))$ we set $W = C(R_{NTOP}(s))$; otherwise, we build W by sequentially adding customers from $C(R_{NTOP}(s))$ as long as $P(W) < P(U)$. The customers in $U \cup W$ are moved as follows: they are first removed from their routes; the customers in U are then sequentially inserted into $R_{NTOP}(s)$ without creating infeasibility (new routes are created if necessary); finally, the customers in W are sequentially inserted into the existing routes in $R_{TOP}(s)$, with the smallest possible increase in infeasibility. The solution s' resulting from such an exchange is not necessarily admissible since infeasible routes can move from $R_{TOP}(s)$ to $R_{NTOP}(s')$. If needed, we therefore repair the resulting solution by using the MAKE_ADMISSIBLE procedure. The *amplitude* of this second kind of jump is defined as the number of customers in U. We denote $J_k^2(s)$ the set containing all solutions obtained from s with such jumps of amplitude k. Details are given in Fig. 9.

Procedure JUMP_2 is illustrated in Fig. 10. The example is constructed as in Fig. 6 with travel times equal to 1 on the edges and 2 on the non-edges. The time limit $T_{\max}$ is equal to 3 while the number m of available vehicles is here equal to 2. The solution s in Fig. 10(b) is admissible but not feasible since one route in $R_{TOP}(s)$ has a duration of $5 > 3 = T_{\max}$. A jump to a solution $s' \in J_1^2(s)$ is performed by moving v to $R_{NTOP}(s)$ and w to $R_{TOP}(s)$. These moves do not create any new infeasibility. The solution s' resulting from this exchange is represented in Fig. 10(c). It is not admissible since the infeasible route of $R_{TOP}(s)$ remains infeasible after the removal of v, while it is no longer one of the two most profitable routes. The MAKE_ADMISSIBLE procedure creates the admissible (and feasible) solution of Fig. 10(d).

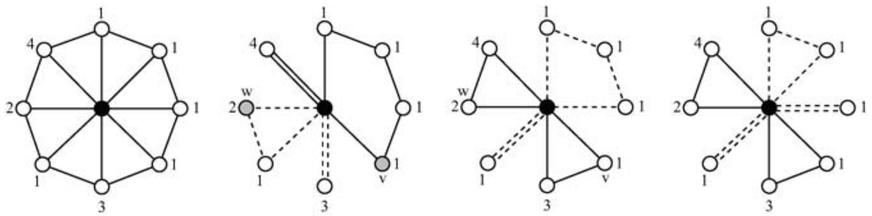


Fig. 10 Illustration of the second kind of jump

3.4 Tabu search

We have developed a tabu search algorithm with two possible different strategies. One explores the set of feasible solutions while the other one visits admissible but not necessarily feasible solutions. The algorithm uses 1-moves and swap-moves, and the tabu list contains pairs (c, r) with the meaning that it is forbidden to move customer c to route r. When performing a 1-move (c, r), the pair $(c, r_c(s))$ is introduced in the tabu list *TL*, while when performing a swap-move (c, c'), both pairs $(c, r_c(s))$ and $(c', r_{c'}(s))$ enter *TL*. A 1-move (c, r) is considered as tabu if $(c, r) \in TL$ while a swap-move (c, c') is considered as tabu if $(c, r_{c'}(s)) \in TL$ or (not exclusive) $(c', r_c(s)) \in TL$. Each time s^* is improved, we apply the classical 2-opt procedure (Lin, 1965) on each route in s^*.

We use five functions for measuring the quality of the solutions visited during the search.

- $f_1(s)$ is the total profit $P(R_{TOP}(s))$ of the routes in $R_{TOP}(s)$.
- $f_2(s)$ is the total duration $\sum_{r\in R_{TOP}(s)} T(r)$ of the routes in $R_{TOP}(s)$.
- $f_3(s)$ is defined as $P(R_{TOP}(s)) - \alpha I(R_{TOP}(s))$, where α is a parameter that gives more or less importance to the second component of this function. Notice that $f_3(s) = f_1(s)$ if s is feasible. Parameter α is initially set equal to 1 and is then adjusted every 10 iterations, as in Gendreau et al. (1994): if the ten previous solutions were feasible then α is divided by 2; if they were all infeasible, then α is multiplied by 2; otherwise α remains unchanged.
- $f_4(s)$ is the number of non empty routes in s.
- $f_5(s)$ is the total duration $\sum_{r\in R_{NTOP}(s)} T(r)$ of the routes in $R_{NTOP}(s)$.

The tabu search with the *feasible strategy* visits only feasible solutions that are compared with functions f_1, f_2, f_4 and f_5. We say that a solution s is (1,2,4,5)-better than a solutions s' if $f_1(s) > f_1(s')$, or $f_1(s) = f_1(s')$ and $f_2(s) < f_2(s')$, or $f_1(s) = f_1(s')$, $f_2(s) = f_2(s')$ and $f_4(s) < f_4(s')$, or $f_1(s) = f_1(s')$, $f_2(s) = f_2(s')$, $f_4(s) = f_4(s')$ and $f_5(s) < f_5(s')$.

The tabu search with the *penalty strategy* can visit infeasible solutions but infeasibility is penalized. More precisely, we use in this case functions f_3, f_4 and f_5, and we say that s is (3,4,5)-better than s' if $f_3(s) > f_3(s')$, or $f_3(s) = f_3(s')$ and $f_4(s) < f_4(s')$, or $f_3(s) = f_3(s')$, $f_4(s) = f_4(s')$ and $f_5(s) < f_5(s')$.

To unify the description of the algorithms, we define $\nu = (1, 2, 4, 5)$ in the feasible strategy and $\nu = (3, 4, 5)$ in the penalty strategy, and we write about ν-better solutions.

Many 1-moves and swap-moves have no influence on the f_1-value of a solution. Indeed, for a 1-move (c, r) with $\{r_c(s), r\} \subseteq R_{NTOP}(s)$ we have $f_1(s) = f_1(s \oplus (c, r))$, unless $r + c \in R_{TOP}(s \oplus (c, r))$. Similarly, for a swap-move (c, c') with $\{r_c(s), r_{c'}(s)\} \subseteq R_{NTOP}(s)$ we have $f_1(s) = f_1(s \oplus (c, c'))$, unless $(r_c(s) - c) + c'$ or $(r_{c'}(s) - c') + c$ belongs to $R_{TOP}(s \oplus (c, c'))$. To better guide the search towards an optimal solution, we only use moves which can have an influence on the f_1-value of the current solution. More precisely, we only consider moves (x, y) such that $R_{TOP}(s) \neq R_{TOP}(s \oplus (x, y))$. Such moves are said *interesting*. This definition of *interesting* moves also includes 1-moves and swap-moves made only on routes in $R_{TOP}(s)$, even if the total profit on these routes is not changed. The reason is that a modified route in $R_{TOP}(s)$ can become less profitable than a non-modified route in $R_{NTOP}(s)$, and the f_1-value is therefore possibly modified.

The proposed tabu search is described in Fig. 11. It uses a subroutine called UPDATE that updates the current best neighbor each time a better neighbor is found. More precisely, let s' denote the current best neighbor of a solution s. For a move (x, y) and a vector $\nu = (1, 2, 4, 5)$ or $(3, 4, 5)$, the UPDATE subroutine replaces s' by $s \oplus (x, y)$ if $s \oplus (x, y)$ is interesting and ν-better than s', and if $(x, y) \notin TL$ or $s \oplus (x, y)$ is ν-better than the best solution s^* encountered so far.

Tabu search for the TOP

Choose an initial solution s; set $TL = \emptyset$ (tabu list); set $s^* = s$ (best solution)

Repeat the following until N_{max} iterations have been performed without improving s^*

- Set s'=s and ν=(1,2,4,5) for the feasible strategy or ν=(3,4,5) for the penalty strategy

  Feasible strategy
  For each 1-move (c, r) do
  If $I(r + c) = 0$ then UPDATE$(\nu, (c, r))$
  Else, for all customers $c' \neq c$ in $r + c$, do
  If both $(r_c(s) - c) + c'$ and $(r - c') + c$ are feasible routes then UPDATE$(\nu, (c, c'))$

  Penalty strategy
  For each 1-move (c, r) do
  If $I(r + c) = 0$ or $r + c \in R_{TOP}(s \oplus (c, r))$ then UPDATE$(\nu, (c, r))$
  Else, for all customers $c' \neq c$ in $r + c$ do
  If $I((r - c') + c) = 0$ then UPDATE$(\nu, (c, c'))$
  If s' is not admissible then replace s' with MAKE_ADMISSIBLE(s')

- If s' is better than s^* then
  improve each route of s' by means of the 2-opt procedure and set $s^* := s'$

- Set s:=s' and update TL

Subroutine UPDATE
Input A vector ν=(1,2,4,5) or (3,4,5) and a move (x, y)
Output A possible update of s'

If (x, y) is an interesting move and if $s \oplus (x, y)$ is ν-better than s' then
if $(x, y) \notin TL$ or $s \oplus (x, y)$ is ν-better than s^* then set $s' = s \oplus (x, y)$

Fig. 11 A tabu search algorithm for the TOP

General Scheme

Generate an initial feasible solution s as in Chao et al. (1996)

Repeat the following as long as a stopping criterion is met

- Choose an amplitude k and generate a solution $s' \in J^1_k(s) \cup J^2_k(s)$

  In case of a feasible strategy do

  - if $I(s') > 0$ then replace the routes in $R_{TOP}(s')$ by those in REDUCE_INF($R_{TOP}(s')$)

  - if $I(s') > 0$ then replace s' by MAKE_FEASIBLE(s')

- Apply the tabu search of Figure 11 on s', and let s'' denote the resulting solution
- Decide whether s is set equal to s'' or unchanged

Fig. 12 General scheme of our solution methods for the TOP

For moving to a neighbor solution, the feasible strategy considers all 1-moves (c, r); if $I(r + c) > 0$ then all swap moves (c, c') which induce a feasible solution $s \oplus (c, c')$ are also considered.

The penalty strategy also considers all 1-moves (c, r): if $r + c$ is an infeasible route in $R_{NTOP}(s \oplus (c, r))$ then all swap moves (c, c') which induce a feasible route $(r - c') + c$ are also considered. Notice that if s is admissible and $I(r + c) = 0$ for a 1-move (c, r), then $s \oplus (c, r)$ is possibly non-admissible. Indeed, if $r_c(s)$ is an infeasible route in $R_{TOP}(s)$, then $r_c(s) - c$ is possibly infeasible in $R_{NTOP}(s \oplus (c, r))$. Similarly, if s is admissible and $I((r_{c'}(s) - c') + c) = 0$ for a swap-move (c, c'), then $s \oplus (c, c')$ is possibly non-admissible since $(r_c(s) - c) + c'$ is possibly infeasible in $R_{NTOP}(s \oplus (c, c'))$. We therefore use the MAKE_ADMISSIBLE procedure to repair the best neighbor.

The tabu search is stopped when $N_{\max}$ iterations have been performed without improving s^*. We call LONG_TABU the version of the above algorithm where $N_{\max}$ is fixed equal to $400n$, while SHORT_TABU denotes the version with $N_{\max} = 25$.

3.5 Three algorithms for the TOP

The three algorithms that are tested and compared in the next section all follow the general scheme of Fig. 3 which we now detail in Fig. 12.

We start with an initial feasible solution generated using the initial solution proposed by Chao et al. (1996). We then perform a jump to a solution $s' \in J^1_k(s) \cup J^2_k(s)$. If we follow the feasible strategy, we first reduce the infeasibility in $R_{TOP}(s')$ by means of REDUCE_INF, and we then make s' feasible (if needed) by means of MAKE_FEASIBLE. We then apply the tabu search algorithm of Fig. 11 and finally decide whether the resulting solution s'' replaces s or not. This process is repeated until a stopping criterion is met.

Two of the proposed algorithms use LONG_TABU. We have observed in preliminary experiments that, when $s' \in J^1_k(s)$, LONG_TABU often turns back to s after a relatively short time. For this reason, we only consider the second kind of jump when using LONG_TABU. The amplitude of a jump k is a parameter of the algorithm. We call GENERALIZED_TABU_FEASIBLE the algorithm that uses the feasible strategy while GENERALIZED_TABU_PENALTY uses the penalty strategy. Both algorithms stop when a number $maxjumps$ of jumps has been performed. The algorithms are summarized in Figs. 13 and 14.

Algorithm GENERALIZED_TABU_FEASIBLE

Generate an initial feasible solution s as in Chao et al. (1996)

Repeat the following *maxjump* times

- Take a value k and determine $s' \in J_k^2(s)$ by means of JUMP_2
- if $I(s') > 0$ then replace the routes in $R_{TOP}(s')$ by those in REDUCE_INF($R_{TOP}(s')$)
- if $I(s') > 0$ then replace s' by MAKE_FEASIBLE(s')
- Apply LONG_TABU with the feasible strategy on s'; let s'' be the resulting solution
- Set $s = s''$

Fig. 13 The GENERALIZED_TABU_FEASIBLE algorithm

Algorithm GENERALIZED_TABU_PENALTY

Generate an initial feasible solution s as in Chao et al. (1996)

Repeat the following *maxjump* times

- Take a value k and determine $s' \in J_k^2(s)$ by means of JUMP_2
- Apply LONG_TABU with the penalty strategy on s'; let s'' be the resulting solution
- Set $s = s''$

Fig. 14 The GENERALIZED_TABU_PENALTY algorithm

The third algorithm is a Variable Neighborhood Search that uses SHORT_TABU as local search. In preliminary experiments, we have observed that SHORT_TABU is often not able to recover feasibility when it is lost. For this reason, we only use the feasible strategy with SHORT_TABU. Following the VNS scheme, we set $s = s''$ only if s'' is (1,2,4,5)-better than s. For each amplitude k of the jumps, a number $\overline{k}$ of jumps is made before changing k. We have implemented two rules for varying the amplitude k of the jumps. The *ascending* rule starts with $k = 1$ and augments k until the value $k_{\max}$ is reached. If s'' is (1,2,4,5)-better than s, k is reset equal to 1, otherwise k is augmented by one if the number of jumps with amplitude k is $\overline{k}$ and remains unchanged if this is not the case. When $\overline{k}$ jumps of amplitude $k_{\max}$ are made, the procedure is repeated making a new loop. The algorithm stops when a number of loops equal to $maxloops$ has been performed. We have also tested a *descending* rule which decreases k from $k_{\max}$ to 1. The results obtained are very similar to those with the ascending strategy. We therefore only report on results obtained with the descending rule. The algorithm, called VNS_FEASIBLE, is summarized in Fig. 15.

4 Computational experiments

The computational experiments have been made on the set of 320 benchmark instances published in Chao et al. (1996). The results produced by our algorithms have been compared with those produced by the algorithm of Chao et al. (1996), from now on CGW algorithm, and with those produced by the algorithm of Tang and Miller-Hooks (2005), from now on TMH algorithm. We did not compare the algorithms with Tsiligirides algorithm (Tsiligirides, 1984), because it has been shown to be dominated by the CGW or TMH in Tang and Miller-Hooks (2005). The experiments were run on

Algorithm VNS_FEASIBLE

Generate an initial feasible solution s as in Chao et al. (1996)

Set $k = k_{max}$, $counter\_jumps = 1$ and $counter\_loops = 1$

Repeat the following until $counter\_loops > maxloops$

- Choose i at random in {1,2} and determine a solution $s' \in J_k^i(s)$ by means of JUMP_i
- if $I(s') > 0$ then replace the routes in $R_{TOP}(s')$ by those in REDUCE_INF($R_{TOP}(s')$)
- if $I(s') > 0$ then replace s' by MAKE_FEASIBLE(s')
- Apply SHORT_TABU with the feasible strategy on s'; let s'' be the resulting solution
- If s'' is (1,2,4,5)-better than s then set $s = s''$, $k = k_{max}$, $counter\_jumps = 1$ and $counter\_loops = 1$. Otherwise if $counter\_jumps = \overline{k}$ set $k = k - 1$. If $k = 0$ set $k = k_{max}$, $counter\_jumps = 1$, $counter\_loops = counter\_loops + 1$

Fig. 15 The VNS_FEASIBLE algorithm

a personal computer Intel Pentium 4 with 2.80 GHz and 1.048 GB Ram. The values of the solutions obtained by CGW and TMH algorithms have been taken from Tang and Miller-Hooks (2005).

The stopping criterion we have chosen for GENERALIZED_TABU_FEASIBLE and for GENERALIZED_TABU_PENALTY is a total number of 3 jumps. In other words, the sequence of operations described in Figs. 13 and 14 is repeated three times. We tested two versions of VNS_FEASIBLE, a SLOW VNS_FEASIBLE and a FAST VNS_FEASIBLE. In SLOW VNS_FEASIBLE the parameter $k_{\max}$ has been set to $\frac{2}{3}(n-2)$, the parameter $\overline{k}$ has been set to 3 and the maximum number of loops $maxloops$ to 10. In FAST VNS_FEASIBLE we set $k_{\max} = \frac{n-2}{3}$, $\overline{k} = 1$ and $maxloops = 3$.

In all the test instances, the starting point of each tour is different from the ending point. The total number of vertices n includes the starting and ending points. The 320 instances include seven sets. The number of vertices is $n = 32$ in set 1, $n = 21$ in set 2, $n = 33$ in set 3, $n = 100$ in set 4, $n = 66$ in set 5, $n = 64$ in set 6 and $n = 102$ in set 7. Customer location and score is identical in all instances of the same set. In each set, an instance is characterized by a number of vehicles m, which varies between 2 and 4, and a different value of the time limit $T_{\max}$. On 121 of the 320 instances all the tested algorithms have obtained the same solution. For some of these instances the solution is the same because all customers with distance from the starting and ending point less than $T_{\max}$ can be visited within the time limit. Therefore, the solution is obviously optimal. In the case that not all such customers are visited, we believe that the solution found by all algorithms is the optimal one. We report in the following tables only the results we obtained on the set of 199 remaining instances. The 199 instances are distributed among the sets as follows: 9 in set 1, 3 in set 2, 29 in set 3, 53 in set 4, 48 in set 5, 11 in set 6 and 46 in set 7. A detailed table of results for all the test instances, together with the instances themselves, can be found at the web site www-c.eco.unibs.it/∼ archetti/TOP.zip.

We have made some preliminary tests to determine the length of the tabu list TL. On the basis of the results of these tests, we have determined

$$TL = \left\lfloor \frac{\sqrt{\beta * random}}{4} + n\lceil\sqrt{m}\rceil\theta \right\rfloor$$

where:

- $\beta = n$ multiplied by the number of routes created in the initial solution;
- $random$ = a random number in (0, 1];
- θ = multiplier which takes value $\frac{1}{8}$ in GENERALIZED_TABU_FEASIBLE and GENERALIZED_TABU_PENALTY, and value $\frac{1}{16}$ in VNS_FEASIBLE.

In Tables 1–4 the value of the solution obtained by the different tested algorithms is shown for sets 1–3, for set 4, for set 5 and for sets 6 and 7, respectively. The best results are indicated with bold numbers. Each instance is identified by the notation $p\alpha.\beta.\gamma$, where α indicates the set to which the instance belongs, β represents the number of vehicles, and γ is a label that differentiates between instances from the same set and with the same number of vehicles. On each instance, three runs have been executed for each of the algorithms that include random choices. With *zmin* and *zmax* we denoted the minimum and the maximum value of the objective function obtained. The value *zmin* can be seen as a sort of guaranteed value, related to the robustness of the algorithm with respect to the random choices. The value *zmax* is a value related to the ability of the algorithm to reach good solutions. It is obtained by running the algorithm more than once (three times in our experiments) and taking advantage of the randomness, at the expense of an increase of the computational time. We will see later than the computational time of a run is rarely larger than 10 minutes and, thus, the increase of the computational time due to multiple runs is acceptable.

A summarized view of the results is provided in Table 5. In the first row the number of best solutions found by each algorithm over all the 199 instances is shown. The average and the maximum error over all the instances are shown in the second and third row. In the fourth and fifth row each algorithm is compared to CGW and TMH algorithms, respectively. Finally, the last row shows the number of times each algorithm has obtained a solution strictly better than both CGW and TMH algorithms. From this table it can be seen that each of the proposed algorithms improves on the average the performance of CGW and TMH algorithms. The best of the proposed algorithms turns out to be SLOW VNS_FEASIBLE.

Finally, Table 6 shows the average and maximum computational time required by each algorithm for a run over the instances of the various sets. The CGW algorithm was run on a SUN 4/730 Workstation 25 MHz, while TMH was run on a DEC Alpha XP1000 computer 667 MHz.

In conclusion, SLOW VNS_FEASIBLE requires in the worst case less than 20 minutes. GENERALIZED_TABU_FEASIBLE and GENERALIZED_TABU_PENALTY require a similar computational time, in most cases less than 10 minutes. The time required by FAST VNS_FEASIBLE is only in one case slightly above 2 minutes. This latter algorithm is an excellent compromise between solution quality and computational effort.

From Tables 1–4 it can be seen that we obtained a strictly better solution than the best known solution for 128 benchmark test instances. Some insight as to why our metaheuristics outperform the algorithm proposed by Tang and Miller-Hooks (2005) can be obtained by comparing the tabu search implementations. Tang and Miller-Hooks use a tabu search based on the penalty strategy (see Section 3.4). They define a solution s as a set of m routes which are possibly infeasible. The customers that are not included in one of these m routes are however not combined for creating routes in

Table 1 Results for sets 1–3

		GEN_TABU_PENALTY		GEN_TABU_FEASIBLE		FAST VNS_FEASIBLE		SLOW VNS_FEASIBLE			
Instance	T max	z min	z max	z min	z max	z min	z max	z min	z max	TMH	CGW
p1.2.i	23.0	**135**	**135**	**135**	**135**	**135**	**135**	**135**	**135**	**135**	130
p1.2.l	30.0	190	**195**	**195**	**195**	**195**	**195**	**195**	**195**	190	190
p1.3.h	13.3	70	70	70	70	70	70	70	70	70	**75**
p1.3.m	21.7	**175**	**175**	**175**	**175**	**175**	**175**	**175**	**175**	170	**175**
p1.3.o	24.3	205	205	205	205	205	205	205	205	205	**215**
p1.3.p	25.0	**220**	**220**	**220**	**220**	**220**	**220**	**220**	**220**	**220**	215
p1.4.j	12.5	**75**	**75**	**75**	**75**	**75**	**75**	**75**	**75**	**75**	70
p1.4.o	18.2	**165**	**165**	**165**	**165**	**165**	**165**	**165**	**165**	**165**	160
p1.4.p	18.8	**175**	**175**	**175**	**175**	**175**	**175**	**175**	**175**	**175**	160
p2.2.k	22.5	**275**	**275**	**275**	**275**	**275**	**275**	**275**	**275**	270	270
p2.3.g	10.7	**145**	**145**	**145**	**145**	**145**	**145**	**145**	**145**	140	140
p2.3.h	11.7	165	**170**	165	165	165	165	165	165	165	165
p3.2.c	12.5	**180**	**180**	**180**	**180**	**180**	**180**	**180**	**180**	**180**	170
p3.2.e	17.5	**260**	**260**	**260**	**260**	**260**	**260**	**260**	**260**	250	**260**
p3.2.f	20.0	**300**	**300**	**300**	**300**	**300**	**300**	**300**	**300**	290	**300**
p3.2.g	22.5	**360**	**360**	**360**	**360**	**360**	**360**	**360**	**360**	350	350
p3.2.h	25.0	400	**410**	400	**410**	400	**410**	**410**	**410**	**410**	390
p3.2.i	27.5	450	**460**	**460**	**460**	**460**	**460**	**460**	**460**	**460**	440
p3.2.j	30.0	**510**	**510**	**510**	**510**	**510**	**510**	**510**	**510**	490	470
p3.2.k	32.5	**550**	**550**	**550**	**550**	**550**	**550**	**550**	**550**	540	540
p3.2.m	37.5	610	**620**	**620**	**620**	**620**	**620**	**620**	**620**	**620**	**620**
p3.2.n	40.0	650	**660**	650	**660**	**660**	**660**	**660**	**660**	**660**	**660**
p3.2.o	42.5	680	**690**	**690**	**690**	**690**	**690**	**690**	**690**	**690**	680
p3.2.p	45.0	710	**720**	**720**	**720**	**720**	**720**	**720**	**720**	710	710
p3.2.q	47.5	750	750	**760**	**760**	**760**	**760**	**760**	**760**	**760**	750
p3.2.r	50.0	770	770	780	**790**	780	**790**	**790**	**790**	780	780

(*Continued on next page.*)

Table 1 (*Continued*).

Instance	T max	GEN_TABU_PENALTY		GEN_TABU_FEASIBLE		FAST VNS_FEASIBLE		SLOW VNS_FEASIBLE		TMH	CGW
		z min	z max	z min	z max	z min	z max	z min	z max		
p3.3.k	21.7	**440**	**440**	**440**	**440**	**440**	**440**	**440**	**440**	430	430
p3.3.l	23.3	**480**	**480**	**480**	**480**	**480**	**480**	**480**	**480**	470	470
p3.3.m	25.0	**520**	**520**	**520**	**520**	**520**	**520**	**520**	**520**	510	**520**
p3.3.n	26.7	**570**	**570**	**570**	**570**	**570**	**570**	**570**	**570**	550	550
p3.3.o	28.3	**590**	**590**	**590**	**590**	**590**	**590**	**590**	**590**	**590**	580
p3.3.p	30.0	**640**	**640**	**640**	**640**	**640**	**640**	**640**	**640**	**640**	620
p3.3.q	31.7	**680**	**680**	**680**	**680**	**680**	**680**	**680**	**680**	**680**	630
p3.3.s	35.0	**720**	**720**	**700**	**720**	**720**	**720**	**720**	**720**	710	710
p3.3.t	36.7	**760**	**760**	**760**	**760**	**760**	**760**	**760**	**760**	750	720
p3.4.f	10.0	**190**	**190**	**190**	**190**	**190**	**190**	**190**	**190**	**190**	180
p3.4.i	13.8	**270**	**270**	**270**	**270**	**270**	**270**	**270**	**270**	260	260
p3.4.j	15.0	**310**	**310**	**310**	**310**	**310**	**310**	**310**	**310**	**310**	300
p3.4.m	18.8	**390**	**390**	**390**	**390**	**390**	**390**	**390**	**390**	380	380
p3.4.o	21.2	490	**500**	**500**	**500**	490	**500**	**500**	**500**	490	490
p3.4.p	22.5	**560**	**560**	**560**	**560**	**560**	**560**	**560**	**560**	**560**	530

Table 2 Results for set 4

Instance	T max	GEN_TABU_PENALTY z min	GEN_TABU_PENALTY z max	GEN_TABU_FEASIBLE z min	GEN_TABU_FEASIBLE z max	FAST VNS_FEASIBLE z min	FAST VNS_FEASIBLE z max	SLOW VNS_FEASIBLE z min	SLOW VNS_FEASIBLE z max	TMH	CGW
p4.2.a	25.0	**206**	**206**	**206**	**206**	**206**	**206**	**206**	206	202	194
p4.2.c	35.0	**452**	**452**	**452**	**452**	**452**	**452**	**452**	**452**	438	440
p4.2.d	40.0	528	530	**531**	**531**	527	**531**	**531**	**531**	517	**531**
p4.2.e	45.0	599	**618**	613	613	612	**618**	**618**	**618**	593	580
p4.2.f	50.0	676	**687**	672	676	672	684	677	**687**	666	669
p4.2.g	55.0	747	751	751	**756**	745	750	750	753	749	737
p4.2.h	60.0	793	795	804	820	818	827	827	**835**	827	807
p4.2.i	65.0	882	882	886	899	857	916	**918**	**918**	915	858
p4.2.j	70.0	933	946	937	962	954	**962**	**962**	**962**	914	899
p4.2.k	75.0	1008	1013	986	1013	1001	1019	1019	**1022**	963	932
p4.2.l	80.0	1058	1061	1054	1058	1052	1073	**1074**	**1074**	1022	1003
p4.2.m	85.0	1095	1106	1048	1098	1098	**1132**	**1132**	**1132**	1089	1039
p4.2.n	90.0	1053	1169	1155	**1171**	1134	1159	1167	**1171**	1150	1112
p4.2.o	95.0	1149	1180	1162	1192	1194	1216	1207	**1218**	1175	1147
p4.2.p	100.0	1194	1226	1225	1239	1227	1239	1236	**1241**	1208	1199
p4.2.q	105.0	1252	1252	1250	1255	1258	**1265**	1263	1263	1255	1242
p4.2.r	110.0	1280	1281	1281	1283	1275	1283	**1286**	**1286**	1277	1199
p4.2.s	115.0	1296	1296	1299	1299	1298	1300	1300	**1301**	1294	1286
p4.2.t	120.0	**1306**	**1306**	**1306**	**1306**	**1306**	**1306**	**1306**	**1306**	**1306**	1299
p4.3.c	23.3	**193**	**193**	**193**	**193**	**193**	**193**	**193**	**193**	192	191
p4.3.d	26.7	334	**335**	**335**	**335**	**333**	**335**	**335**	**335**	333	333
p4.3.e	30.0	**468**	**468**	**468**	**468**	461	**468**	**468**	**468**	465	432
p4.3.f	33.3	**579**	**579**	**579**	**579**	**579**	**579**	**579**	**579**	**579**	552
p4.3.g	36.7	649	651	652	652	647	**653**	**653**	**653**	646	623
p4.3.h	40.0	722	722	727	727	715	724	728	**729**	709	717
p4.3.i	43.3	799	806	806	806	799	806	806	**807**	785	798

(*Continued on next page.*)

Table 2 (*Continued*).

Instance	T max	GEN_TABU_PENALTY		GEN_TABU_FEASIBLE		FAST VNS_FEASIBLE		SLOW VNS_FEASIBLE		TMH	CGW
		z min	z max	z min	z max	z min	z max	z min	z max		
p4.3.j	46.7	855	858	844	858	855	861	859	**861**	860	829
p4.3.k	50.0	908	**919**	904	918	912	**919**	**919**	**919**	906	889
p4.3.l	53.3	972	976	970	973	955	975	975	**978**	951	946
p4.3.m	56.7	1020	1034	1037	1049	1033	1056	1034	**1063**	1005	956
p4.3.n	60.0	1080	1108	1093	1115	1092	1111	**1121**	**1121**	1119	1018
p4.3.o	63.3	1134	1156	1142	1157	1168	**1172**	1149	1170	1151	1078
p4.3.p	66.7	1164	1207	1200	1221	1184	1208	1199	**1222**	1218	1115
p4.3.q	70.0	1183	1237	1237	1241	1227	1250	1240	**1251**	1249	1222
p4.3.r	73.3	1222	1224	1261	1269	1262	**1272**	1262	**1272**	1265	1225
p4.3.s	76.7	1250	1250	1285	**1294**	1266	1289	1280	1293	1282	1239
p4.3.t	80.0	1297	1303	1297	1304	1298	1298	1298	**1304**	1288	1285
p4.4.e	22.5	**183**	**183**	**183**	**183**	**183**	**183**	**183**	**183**	182	182
p4.4.f	25.0	**324**	**324**	**324**	**324**	**324**	**324**	**324**	**324**	315	304
p4.4.g	27.5	**461**	**461**	**461**	**461**	**461**	**461**	**461**	**461**	453	460
p4.4.h	30.0	**571**	**571**	**571**	**571**	**571**	**571**	**571**	**571**	554	545
p4.4.i	32.5	654	655	656	**657**	653	**657**	**657**	**657**	627	641
p4.4.j	35.0	729	731	728	731	723	**732**	**732**	**732**	**732**	697
p4.4.k	37.5	815	**821**	816	816	819	**821**	**821**	**821**	819	770
p4.4.l	40.0	877	878	876	878	876	879	879	**880**	875	847
p4.4.m	42.5	913	916	914	918	914	916	916	**919**	910	895
p4.4.n	45.0	966	972	962	976	961	968	968	968	**977**	932
p4.4.o	47.5	1049	1057	1029	1057	1050	1051	1051	**1061**	1014	995
p4.4.p	50.0	1095	**1120**	1087	**1120**	1118	**1120**	**1120**	**1120**	1056	996
p4.4.q	52.5	1140	1148	1144	1157	1156	1160	1160	**1161**	1124	1084
p4.4.r	55.0	1188	1203	1202	**1211**	1198	1207	1203	1203	1165	1155
p4.4.s	57.5	1243	1245	1239	1256	1249	**1259**	1246	1255	1243	1230
p4.4.t	60.0	1275	1279	1279	**1285**	1251	1282	1275	1279	1255	1253

Springer

Table 3 Results for set 5

		GEN_TABU_PENALTY		GEN_TABU_FEASIBLE		FAST VNS_FEASIBLE		SLOW VNS_FEASIBLE			
Instance	T max	z min	z max	z min	z max	z min	z max	z min	z max	TMH	CGW
p5.2.e	12.5	**180**	**180**	**180**	**180**	**180**	**180**	**180**	**180**	**180**	175
p5.2.g	17.5	**320**	**320**	**320**	**320**	315	**320**	**320**	**320**	**320**	315
p5.2.h	20.0	**410**	**410**	**410**	**410**	**410**	**410**	**410**	**410**	**410**	395
p5.2.j	25.0	**580**	**580**	**580**	**580**	**580**	**580**	**580**	**580**	560	**580**
p5.2.l	30.0	**800**	**800**	**800**	**800**	**800**	**800**	**800**	**800**	770	790
p5.2.m	32.5	855	**860**	**860**	**860**	**860**	**860**	**860**	**860**	**860**	855
p5.2.n	35.0	920	**925**	**925**	**925**	**925**	**925**	**925**	**925**	920	920
p5.2.o	37.5	**1020**	**1020**	**1020**	**1020**	**1020**	**1020**	**1020**	**1020**	975	1010
p5.2.p	40.0	1100	1130	**1150**	**1150**	**1150**	**1150**	**1150**	**1150**	1090	**1150**
p5.2.q	42.5	1165	**1195**	**1195**	**1195**	1190	**1195**	**1195**	**1195**	1185	**1195**
p5.2.r	45.0	1255	**1260**	**1260**	**1260**	**1260**	**1260**	**1260**	**1260**	**1260**	1250
p5.2.s	47.5	1300	1330	1320	**1340**	**1340**	**1340**	**1340**	**1340**	1310	1310
p5.2.t	50.0	1360	1380	**1400**	**1400**	**1400**	**1400**	**1400**	**1400**	1380	1380
p5.2.u	52.5	1405	1440	1450	**1460**	**1460**	**1460**	**1460**	**1460**	1445	1450
p5.2.v	55.0	1465	1490	**1505**	1505	1500	1500	**1505**	**1505**	1500	1490
p5.2.w	57.5	1525	1555	1560	**1565**	1560	1560	1560	1560	1560	1545
p5.2.x	60.0	1590	1595	1600	**1610**	1590	1590	1595	**1610**	**1610**	1600
p5.2.y	62.5	1600	**1635**	**1635**	**1635**	**1635**	**1635**	**1635**	**1635**	1630	**1635**
p5.2.z	65.0	1655	1670	1670	**1680**	1670	1670	1670	1670	1665	**1680**
p5.3.e	8.3	95	95	95	95	95	95	95	95	95	**110**
p5.3.h	13.3	**260**	**260**	**260**	**260**	**260**	**260**	**260**	**260**	**260**	255
p5.3.k	18.3	**495**	**495**	**495**	**495**	**495**	**495**	**495**	**495**	**495**	480
p5.3.l	20.0	**595**	**595**	**595**	**595**	585	**595**	**595**	**595**	575	**595**
p5.3.n	23.3	750	**755**	**755**	**755**	745	**755**	**755**	**755**	**755**	**755**
p5.3.o	25.0	**870**	**870**	**870**	**870**	**870**	**870**	**870**	**870**	835	**870**
p5.3.q	28.3	1065	**1070**	**1070**	**1070**	**1070**	**1070**	**1070**	**1070**	1065	1060

(*Continued on next page.*)

Table 3 (*Continued*).

		GEN_TABU_PENALTY		GEN_TABU_FEASIBLE		FAST VNS_FEASIBLE		SLOW VNS_FEASIBLE			
Instance	T max	z min	z max	z min	z max	z min	z max	z min	z max	TMH	CGW
p5.3.r	30.0	1105	1110	**1125**	**1125**	**1125**	**1125**	**1125**	**1125**	1115	1105
p5.3.s	31.7	1185	1185	**1190**	**1190**	**1190**	**1190**	**1190**	**1190**	1175	1175
p5.3.t	33.3	1245	1250	**1260**	**1260**	1255	**1260**	**1260**	**1260**	1240	1250
p5.3.u	35.0	1340	1340	**1345**	**1345**	**1345**	**1345**	**1345**	**1345**	1330	1330
p5.3.v	36.7	1410	1420	1420	**1425**	1415	**1425**	**1425**	**1425**	1410	1400
p5.3.w	38.3	1475	**1485**	**1485**	**1485**	1475	**1485**	**1485**	**1485**	1465	1450
p5.3.x	40.0	1530	**1555**	1540	**1555**	1540	**1555**	1550	**1555**	1530	1530
p5.3.y	41.7	1575	1590	1590	**1595**	1590	**1595**	**1595**	**1595**	1580	1580
p5.3.z	43.3	1615	1625	**1635**	**1635**	**1635**	**1635**	**1635**	**1635**	**1635**	**1635**
p5.4.m	16.2	550	**555**	**555**	**555**	**550**	**555**	**555**	**555**	**555**	495
p5.4.o	18.8	685	**690**	**690**	**690**	**690**	**690**	**690**	**690**	680	675
p5.4.p	20.0	**765**	**765**	**765**	**765**	760	**765**	**765**	**765**	760	750
p5.4.q	21.2	840	**860**	**860**	**860**	**860**	**860**	**860**	**860**	**860**	**860**
p5.4.r	22.5	955	**960**	**960**	**960**	**960**	**960**	**960**	**960**	**960**	950
p5.4.s	23.8	1025	1025	**1030**	**1030**	1025	**1030**	**1030**	**1030**	1000	1020
p5.4.t	25.0	**1160**	**1160**	**1160**	**1160**	**1160**	**1160**	**1160**	**1160**	1100	**1160**
p5.4.u	26.2	**1300**	**1300**	**1300**	**1300**	**1300**	**1300**	**1300**	**1300**	1275	1260
p5.4.v	27.5	**1320**	**1320**	**1320**	**1320**	**1320**	**1320**	**1320**	**1320**	1310	1310
p5.4.w	28.8	1370	1375	1385	**1390**	1380	**1390**	1385	**1390**	1380	1380
p5.4.x	30.0	1435	1440	**1450**	**1450**	**1450**	**1450**	**1450**	**1450**	1410	1420
p5.4.y	31.2	1510	**1520**	**1520**	**1520**	**1520**	**1520**	**1520**	**1520**	**1520**	1490
p5.4.z	32.5	1595	**1620**	**1620**	**1620**	**1620**	**1620**	**1620**	**1620**	1575	1545

Table 4 Results for sets 6–7

Instance	T max	GEN_TABU_PENALTY		GEN_TABU_FEASIBLE		FAST VNS_FEASIBLE		SLOW VNS_FEASIBLE		TMH	CGW
		z min	z max	z min	z max	z min	z max	z min	z max		
p6.2.j	30.0	936	**948**	**948**	**948**	**948**	**948**	**948**	**948**	936	942
p6.2.l	35.0	1092	1098	1104	1110	**1116**	**1116**	**1116**	**1116**	**1116**	1104
p6.2.m	37.5	1146	1164	**1188**	**1188**	1170	**1188**	**1188**	**1188**	**1188**	1176
p6.2.n	40.0	1224	1242	**1260**	**1260**	1242	**1260**	1242	**1260**	**1260**	1242
p6.3.i	18.3	**642**	**642**	**642**	**642**	**642**	**642**	**642**	**642**	612	**642**
p6.3.k	21.7	**894**	**894**	**894**	**894**	**894**	**894**	**894**	**894**	876	**894**
p6.3.l	23.3	984	**1002**	**1002**	**1002**	**1002**	**1002**	**1002**	**1002**	990	972
p6.3.m	25.0	1074	**1080**	**1080**	**1080**	**1080**	**1080**	**1080**	**1080**	**1080**	**1080**
p6.3.n	26.7	1152	**1170**	**1170**	**1170**	**1170**	**1170**	**1170**	**1170**	1152	1158
p6.4.k	16.2	528	528	528	528	528	528	528	528	522	**546**
p6.4.l	17.5	684	**696**	**696**	**696**	**696**	**696**	**696**	**696**	**696**	690
p7.2.e	50.0	**290**	**290**	**290**	**290**	**289**	**289**	**290**	**290**	**290**	275
p7.2.f	60.0	**387**	**387**	**387**	**387**	**384**	**387**	**387**	**387**	382	379
p7.2.g	70.0	456	456	457	**459**	457	**459**	**459**	**459**	**459**	453
p7.2.h	80.0	519	520	519	520	518	**521**	**521**	**521**	**521**	517
p7.2.i	90.0	578	**579**	578	**579**	574	575	**579**	**579**	578	576
p7.2.j	100.0	641	643	**644**	**644**	636	643	**644**	**644**	638	633
p7.2.k	110.0	702	702	704	**705**	695	704	702	**705**	702	693
p7.2.l	120.0	758	758	759	**767**	758	759	**767**	**767**	**767**	758
p7.2.m	130.0	818	**827**	818	824	821	824	821	**827**	817	811
p7.2.n	140.0	884	884	**888**	**888**	863	883	884	**888**	864	864
p7.2.o	150.0	925	933	941	**945**	922	**945**	**945**	**945**	914	934
p7.2.p	160.0	992	1000	994	**1002**	**1002**	**1002**	1000	**1002**	987	987
p7.2.q	170.0	1040	1041	1042	1043	1021	1038	1043	**1044**	1017	1031
p7.2.r	180.0	1081	1091	1080	1088	1080	1094	**1094**	**1094**	1067	1082
p7.2.s	190.0	1117	1123	1124	1128	1127	**1136**	**1136**	**1136**	1116	1127

(*Continued on next page.*)

Table 4 (*Continued*).

Instance	T max	GEN_TABU_PENALTY z min	GEN_TABU_PENALTY z max	GEN_TABU_FEASIBLE z min	GEN_TABU_FEASIBLE z max	FAST VNS_FEASIBLE z min	FAST VNS_FEASIBLE z max	SLOW VNS_FEASIBLE z min	SLOW VNS_FEASIBLE z max	TMH	CGW
p7.2.t	200.0	1149	1172	1165	1174	1161	1168	**1179**	**1179**	1165	1173
p7.3.e	33.3	**175**	**175**	**175**	**175**	**175**	**175**	**175**	**175**	**175**	163
p7.3.f	40.0	**247**	**247**	**247**	**247**	**247**	**247**	**247**	**247**	**247**	235
p7.3.g	46.7	**344**	**344**	**344**	**344**	**344**	**344**	**344**	**344**	**344**	338
p7.3.h	53.3	**425**	**425**	**425**	**425**	**425**	**425**	**425**	**425**	416	419
p7.3.i	60.0	484	**487**	**487**	**487**	**487**	**487**	**487**	**487**	481	466
p7.3.j	66.7	557	**564**	560	**564**	556	562	562	**564**	563	539
p7.3.k	73.3	626	**633**	632	**633**	619	632	632	**633**	632	602
p7.3.l	80.0	678	**683**	673	679	666	681	681	681	681	676
p7.3.m	86.7	737	749	741	755	727	745	744	**762**	756	754
p7.3.n	93.3	798	810	805	811	808	814	813	**820**	789	813
p7.3.o	100.0	857	873	862	865	859	871	873	**874**	**874**	848
p7.3.p	106.7	910	917	916	923	906	926	923	**927**	922	919
p7.3.q	113.3	965	976	971	**987**	969	978	987	**987**	966	943
p7.3.r	120.0	1016	1018	1012	1022	1022	**1024**	1022	1022	1011	1008
p7.3.s	126.7	1070	**1081**	1068	1081	1046	1079	1068	1079	1061	1064
p7.3.t	133.3	1106	1114	1112	**1116**	1110	1112	1110	1115	1098	1095
p7.4.f	30.0	**164**	**164**	**164**	**164**	**164**	**164**	**164**	**164**	**164**	156
p7.4.g	35.0	**217**	**217**	**217**	**217**	**217**	**217**	**217**	**217**	**217**	209
p7.4.h	40.0	**285**	**285**	**285**	**285**	**285**	**285**	**285**	**285**	**285**	283
p7.4.i	45.0	**366**	**366**	**366**	**366**	**366**	**366**	**366**	**366**	359	338
p7.4.k	55.0	517	**520**	518	**520**	514	518	518	**520**	503	516
p7.4.l	60.0	585	**590**	588	588	575	588	**590**	**590**	576	562
p7.4.m	65.0	639	644	645	**646**	639	**646**	**646**	**646**	643	610
p7.4.n	70.0	717	723	712	721	699	715	715	**730**	726	683

(*Continued on next page.*)

Table 4 (*Continued*).

Instance	T max	GEN_TABU_PENALTY		GEN_TABU_FEASIBLE		FAST VNS_FEASIBLE		SLOW VNS_FEASIBLE		TMH	CGW
		z min	z max	z min	z max	z min	z max	z min	z max		
p7.4.o	75.0	765	772	770	778	757	770	760	**781**	776	728
p7.4.p	80.0	829	841	833	839	828	**846**	842	**846**	832	801
p7.4.q	85.0	891	902	895	898	896	899	905	**906**	905	882
p7.4.r	90.0	957	**970**	969	969	959	**970**	970	**970**	966	886
p7.4.s	95.0	1012	1021	1014	1020	1010	1021	1014	**1022**	1019	990
p7.4.t	100.0	1068	1071	1069	1071	1048	**1077**	1071	**1077**	1067	1066

Table 5 Summary of results over 199 ins tances

	GEN_TABU_PENALTY		GEN_TABU_FEASIBLE		FAST VNS_FEASIBLE		SLOW VNS_FEASIBLE			
	z min	z max	z min	z max	z min	z max	z min	z max	TMH	CGW
# best solution found	64	109	106	141	92	138	138	180	52	25
Average error with respect to best	1.27	0.57	0.72	0.33	0.90	0.31	0.36	0.18	1.54	2.80
Maximum error with respect to best	13.64	13.64	13.64	13.64	13.64	13.64	13.64	13.64	13.64	11.07
# better than or equal to CGW	156	177	184	194	172	190	192	194	157	–
# better than or equal toTMH	131	167	166	184	152	186	184	198	–	76
# better than CGW and TMH	71	101	95	115	79	113	109	125	–	–

Table 6 Computational times

	GEN_TABU_PENALTY		GEN_TABU_FEASIBLE		FAST VNS_FEASIBLE		SLOW VNS_FEASIBLE			
	Average CPU	Max CPU	Average CPU	Max CPU	Average CPU	Max CPU	Average CPU	Max CPU	TMH	CGW
Set 1	4.67	10.00	1.63	5.00	0.13	1.00	7.78	22.00	N.A.	15.41
Set 2	0.00	0.00	0.00	0.00	0.00	0.00	0.03	1.00	N.A.	0.85
Set 3	6.03	10.00	1.59	9.00	0.15	1.00	10.19	19.00	N.A.	15.37
Set 4	105.29	612.00	282.92	324.00	22.52	121.00	457.89	1118.00	796.70	934.80
Set 5	69.45	147.00	26.55	105.00	34.17	30.00	158.93	394.00	71.30	193.70
Set 6	66.29	96.00	20.19	48.00	8.74	20.00	147.88	310.00	45.70	150.10
Set 7	158.97	582.00	256.76	514.00	10.34	90.00	309.87	911.00	432.60	841.40

$R_{NTOP}(s)$. A neighbor solution is obtained by exchanging customers of the m routes with customers that are not included in a route, exchanging customers between routes, changing the sequence of customers in a route, or adding customers in a route. We think that the use of the set $R_{NTOP}(s)$ of routes in our tabu search algorithm is the main reason that explains its success. More precisely, when a customer is removed from a route r in $R_{TOP}(s)$, the modified route possibly becomes less profitable than a route r' in $R_{NTOP}(s)$, and we can therefore exchange r with r' in s. To make such a change, the tabu search proposed by Tang and Miller-Hooks needs to build r' from scratch.

5 Conclusions

The Team Orienteering Problem (TOP) is the problem where a set of customers may be visited, with a profit guaranteed for each visit. A team of people/vehicles is available and each member of the team can visit any set of customers within a given time limit. The profit of each customer can be collected by one person/vehicle at most. The problem combines the decision of which customers to select with the decision of how to plan the routes. Such decisions might be taken separately, by first selecting the subset of customers to serve and then solving a Vehicle Routing Problem. The sequential solution of the two sub-problems would provide the TOP with a feasible but typically suboptimal solution.

In this paper we presented effective meta-heuristics for the TOP. A variable neighborhood search algorithm turned out to be more efficient and effective for this problem than two tabu search algorithms. With the proposed algorithms we improved the best known solution of 128 benchmark test instances.

Future research will be devoted to extend the proposed meta-heuristics to other VRPs and TSPs with profits.

Acknowledgments We wish to acknowledge the contribution of anonymous referees that helped us to improve a previous version of this paper.

References

Butt, S.E. and T.M. Cavalier. (1994). "A Heuristic for the Multiple Tour Maximum Collection Problem." *Computers and Operations Research* 21, 101–111.

Chao, I-M., B. Golden, and E.A. Wasil. (1996). "The Team Orienteering Problem." *European Journal of Operational Research* 88, 464–474.

Feillet, D., P. Dejax, and M. Gendreau. (2005). "Traveling Salesman Problems with Profits." *Transportation Science* 39, 188–205.

Gendreau, M., A. Hertz, and G. Laporte. (1994). "A Tabu Search Heuristic for the Vehicle Routing Problem." *Management Science* 40, 1276–1290.

Golden, B., A. Assad, and R. Dahl. (1984). "Analysis of a Large Scale Vehicle Routing Problem with an Inventory Component." *Large Scale Systems* 7, 181–190.

Golden, B., L. Levy, and R. Vohra. (1987). "The Orienteering Problem." *Naval Research Logistics* 34, 307–318.

Glover F. (1986). "Future Paths for Integer Programming and Links to Artificial Intelligence." *Computers and Operations Research* 5, 533–549.

Glover, F. and M. Laguna. (eds.) (1997). *Tabu Search*. Kluwer Academic Publishers.

Hansen, P. and N. Mladenović. (1999). "An Introduction to Variable Neighborhood Search." In S. Voss et al. (eds.), *Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Dordrecht, PP. 433–458.

Lin, S. (1965). "Computer Solutions of the Traveling Salesman Problem." *Bell System Technical Journal* 44, 2245–2269.

Mladenović, N. and P. Hansen. (1997). "Variable Neighborhood Search." *Computers and Operations Research* 24, 1097–1100.

Rochat, Y. and E. Taillard. (1995). "Probabilistic Diversification and Intensification in Local Search for Vehicle routing." *Journal of Heuristics* 1, 147–167.

Tang, H. and E. Miller-Hooks. (2005). "A Tabu Search Heuristic for the Team Orienteering Problem." *Computers and Operations Research* 32, 1379–1407.

Tsiligirides, T. (1984). "Heuristic Methods Applied to Orienteering." *Journal of the Operational Research Society* 35, 797–809.

Toth, P. and D. Vigo. (eds.) (2002). *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia.