ERASMUS UNIVERSITEIT ROTTERDAM
ERASMUS SCHOOL OF ECONOMICS

Erasmus University Rotterdam

Tabu Search for the Orienteering Problem with Hotel Selection

Sigrid van Hoek
381505
Bachelor Thesis
Econometrics and Operations Research
Erasmus School of Economics
4 July 2016

Supervisor:
dr. T.A.B. Dollevoet
Second assessor:
dr. F. Frasincar

ABSTRACT Two metaheuristic solution approaches for the Orienteering Problem with Hotel Selection (OPHS) are compared in this thesis. The goal of the OPHS is to select the best vertices and the right sequence of hotels, such that the total score of the vertices is maximal and the trips along the selected vertices satisfy the time restrictions. We implement the Skewed Variable Neighborhood Search (SVNS) algorithm of Divsalar et al. (2013) and show that it is outperformed by a Tabu Search (TS) algorithm. Both algorithms have the same construction method for the initial solution. We try to improve this solution with two shakes in the SVNS algorithm and apply these two shakes with an additional Cross-over shake in the TS algorithm. Local Search or TS with nine moves is performed after each shake. Next, the algorithms decide on the solution that is used in the following iteration, which can be a solution with a better or a worse score. Concluding, this TS algorithm produced near-optimal solutions with an average gap of 0.675%.

Table of Contents

1. Introduction

Composing a good journey of attractions and hotels is important for tourist trip planning. The traveler wants to visit many attractions, but copes with a time restriction that makes it impossible to visit every attraction. Therefore, we need to choose the best attractions and find the shortest route along the chosen sites. The problem is known to be challenging, because there are many possible tours but more importantly, estimating a good sequence of hotels is difficult.

The attractions, which are called vertices, have an associated score that represents the desirability of that attraction. During the journey, the traveler wants to find the route with the optimal score. The tour consists of daily trips that start and end in a hotel. As most attractions are only opened during the day, each trip has a time restriction. In addition, the distance between a hotel and an attraction might be more than the available time for a trip, forcing us to select the best-located hotel each day. By the selection of a hotel each day, we are far more flexible than with fixed hotels, which can cause a higher score. We assume that the start hotel of the first trip, the end hotel of the last trip and the number of trips are fixed. Since there is a limited amount of hotels, the chosen hotels have a huge impact on the possible trips between the hotels.

Other applications of the Orienteering Problem with Hotel Selection (OPHS) are for example submarine surveillance activities for multiple missions and the visitation of clients during a multi-day tour that requires the selection of hotels (Divsalar et al. 2013).

As we aim to develop a fast and efficient algorithm for the OPHS, the research question can be formulated as follows: What is a good method to select hotels and vertices that maximize the total score within a reasonable amount of time, while the tour along the selected points satisfies the time constraints? Divsalar et al. (2013) designed a Skewed Variable Neighborhood Search (SVNS) algorithm that produces good solutions for this problem. We implement this algorithm and compare the results to a Tabu Search (TS) algorithm.

In the remainder of this thesis, we give an overview of the literature on the OPHS and related problems and a Mixed-Integer Linear Problem (MILP) formulation (Section 2 and 3). In Section 4 the SVNS algorithm is described. In Section 5, we present a TS algorithm that produces solutions with a smaller average gap with the optimal solutions. Next, the results of the SVNS and TS are compared in Section 6. Lastly, we present our discussion and conclusion in Section 7 and 8.

2. Literature

The OPHS is only considered recently in the literature. It is a generalization of the team orienteering problem (TOP), which is a generalization of the orienteering problem (OP). Moreover, it is closely related to the travelling salesperson problem with hotel selection (TSPHS). Although these problems differ, it might be useful to investigate some effective algorithms that are used for these problems. Therefore, we will briefly describe some solution approaches.

The OP aims to maximize the collected score, but has the restrictions of a single trip and a fixed start and end hotel. The problem is introduced by Tsiligirides (1984) and studied widely in the literature. It is as a combination of the knapsack problem that maximizes the collected score, and the travelling salesperson problem that minimizes the length of the route along the selected vertices. Exact algorithms like branch-and-cut are able to solve the OP for at most 500 vertices (Fischetti et al. 1998), but the problem is known as NP-hard and heuristics are necessary to solve larger problems. An overview of exact, heuristic algorithms as well as additional constraints is given in Feillet et al. (2005).

The variant of the OP with multiple tours is called TOP. As this problem extends the OP, exact solutions within reasonable time are only found for problems that contain up to 100 vertices (Boussier et al. 2007). While a broad range of heuristics is proposed, the variable neighborhood search (VNS) (Archetti et al. 2007), greedy randomized adaptive search procedure (GRASP) with path-relinking (Souffriau et al. 2010) and ant colony optimization (Ke et al. 2008) have the lowest optimality gap, as shown in an overview of Vansteenwegen et al. (2011). A TS algorithm also produces high-quality solutions (Tang & Miller-Hooks 2005). Some of the heuristics, like the SVNS of Vansteenwegen et al. (2009), focus on reducing the computation time with the cost of a higher average gap. An SVNS algorithm does also consider solutions with a worse score than the best found solution. This might be necessary to find an improvement in the next iterations.

Most of these solution approaches are metaheuristics, meaning that they guide an iterative search

procedure to find efficiently near-optimal solutions. Metaheuristics are especially used for the generalization of the TOP that requires the selection of a hotel each day: the OPHS.

A variant of the OPHS that includes visiting time for vertices is introduced by (Zhu et al. 2012). Their algorithm clusters the hotels, randomly picks a hotel from each cluster and performs Local Search (LS) to obtain trips between the hotels. A SVNS heuristic that selects sequences of hotels with a good estimated score provides good solutions for the OPHS (Divsalar et al. 2013). The algorithm consists of a shaking phase and LS. However, it is outperformed by a population-based memetic algorithm (Divsalar et al. 2014). Their sequences of hotels are based on a Cross-over procedures that combine two solutions and a Mutation procedure that transforms the current solution. The main difference is the randomness as this algorithm also uses LS to select vertices.

Although the TSPHS aims to minimize the length of the tour and must visit all the vertices, it contains vertices and hotels. It is introduced recently with a multi-start heuristic that uses VNS (Vansteenwegen et al. 2012). A GRASP with variable neighborhood descent gives good solutions (Castro et al. 2012), but a memetic algorithm with TS consistently finds the optimal solution for small problems (Castro et al. 2013). This search is based on a list of vertices that are not allowed to be inserted or deleted for a certain number of iterations.

An overview of other combinatorial problems regarding intermediate facilities is presented by Divsalar et al. (2014). Most algorithms use TS, but a comparison with these algorithms is difficult as none of those problems maximizes the collected score subjected to a time restriction.

3. Problem formulation

The OPHS is defined on a complete graph $G = (V, E)$ with edges E between all the vertices $(H \bigcup N) = V$ with $H = \{h_0, ..., h_H\}$ the set of hotels and $N = \{n_0, ..., n_N\}$ the set of vertices. The elements of H and N contain coordinates of their location and a non-negative score S_v with $v \in V$. Hotels have an associated score of zero. It is possible to stay in a hotel for multiple nights or return to a hotel, but vertices can be visited at most one time. In the remainder of this proposal, a possible tour is often called a solution for the OPHS and we refer to attractions as vertices. A time restriction is imposed on each trip, where a trip is defined as the route between two hotels. The total time of a trip is equivalent to the length of a trip, and is calculated according to the Euclidean distance. Thus, we assume no time for visiting the vertices. The maximal time for each trip is denoted by T_d, where $d \in \{d_0...d_D\}$ the day (number) of the tour.

As the OPHS is a generalization of the OP, the problem is NP-hard. Even with a small number of attractions and hotels, there are many possible options to compose a tour. The main difficulty is the selection of hotels, since (i) it is not clear which sequences of hotels are promising and (ii) a different sequence of hotels changes the whole solution (Divsalar et al. 2013). Divsalar et al. (2014) even claim that it is impossible to predict the best sequence of hotels. Because the problem is challenging, heuristics are necessary to solve even small problems of the OPHS within a reasonable amount of time. However, some problems can be solved with the following MILP formulation:

$$\max \sum_{d \in D} \sum_{i \in V} \sum_{j \in V} S_i x_{i,j,d}$$

$$\text{s.t.} \sum_{j \in V} x_{0,j,1} = 1 \tag{1}$$

$$\sum_{i \in V} x_{i,1,D} = 1 \tag{2}$$

$$\sum_{h \in H} \sum_{j \in V} x_{h,j,d} = 1 \qquad \forall d \in D \tag{3}$$

$$\sum_{i \in V} \sum_{h \in H} x_{i,h,d} = 1 \qquad \forall d \in D \tag{4}$$

$$\sum_{i \in V} x_{i,h,d} = \sum_{j \in V} x_{h,j,d+1} \qquad \forall h \in H, d \in D \backslash \{d_D\} \tag{5}$$

$$\sum_{i \in V} x_{i,n,d} = \sum_{j \in V} x_{n,j,d} \qquad \forall n \in N, \forall d \in D \tag{6}$$

$$\sum_{d \in D} \sum_{j \in V} x_{n,j,d} \leq 1 \qquad \forall n \in N \tag{7}$$

$$\sum_{i \in V} \sum_{j \in V} t_{i,j} x_{i,j,d} \leq T_d \qquad \forall d \in D \tag{8}$$

$$u_i - u_j + 1 \leq ... \qquad \forall i, j \in N \tag{9}$$
$$...(N-1)(1 - \sum_{d \in D} x_{i,j,d})$$

$$x_{i,j,d} \in \{0, 1\} \qquad \forall i, j \in V, i \neq j, \forall d \in D \tag{10}$$

$$u_n \in \{1...N\} \qquad \forall n \in N \tag{11}$$

In this formulation, $x_{i,j,d} = 1$ if vertex i is visited before vertex j in trip d and 0 otherwise, and u_i denotes the position of vertex i in the tour. The goal of the OPHS is to maximize the total score of the selected vertices. The first constraint states that the first trip has a fixed start hotel, 0, and the second con-

straint states that the last trip has a fixed end hotel, 1. The next two constraints make sure that each trip starts and ends in a hotel. Constraint 5 ensures that the end hotel of a trip is equal to the start hotel of the next trip and constraint 6 ensures that when a trip visits a vertex it should also leave the vertex. The next constraint states that each vertex can be visited at most once. Constraint 8 is necessary to model the time restriction. Subtours are eliminated with the Miller-Tucker-Zemlin connectivity constraint (constraint 9). The last constraints ensure that the decision variable x is binary and u is integer.

4. Skewed Variable Neighborhood Search

We describe the SVNS algorithm in this section. First, we give an overview of the algorithm. Second, the initialization and main phase will be described. Third, we explain the nine LS moves that are used in both phases. Lastly, the parameters of the algorithm will be described.

The main idea of SVNS is that it examines solutions close and far from the incumbent (Hansen & Mladenović 2001). We can break down the algorithm in two phases: an initialization phase and an improvement and recentering phase as shown in Algorithm 1. The first phase tries to find good feasible solutions, while the second phase tries to improve the initial solution. The improvement phase consists of two Vertices- and a Hotels-shake and the recentering phase decides on the solution that will be used in the next iteration. The algorithm can recenter to a solution with a better score, but also to a solution with a worse score in order to avoid being trapped in a local optimum. After the shakes, a LS tries to find the best tour with the current sequence of hotels.

4.1. Initialization

The first part of the algorithm aims to find a feasible, good, initial solution in a small amount of time. The initialization consists of the following steps: (i) create a matrix with potential score between each pair of hotels, (ii) determine all feasible sequences of hotels and (iii) calculate a Heuristic Estimated Score (HES) for the sequences of the second step. Afterwards, the initial tour is one of the sequences of hotels that is improved with LS.

In the first step, a sequence of four moves is repeated to obtain a score of each trip between each pair of hotels. These moves (Insert, Replacement, Two-opt and Move-best) are described in Subsection 4.4 and called Sub-Op. In contrast to LS, the Sub-Op does not contain a level, but performs the moves in a fixed order until the solution converges. The second step investigates the time restriction for the trips. All possible empty sequences of hotels including repetition that satisfy the time restriction are placed in a list. The third step combines the results of the first and second step in a list with scores that is called the HES. The scores of the trips are summed up for each of the sequences of hotels, whereafter the solutions are sorted based on the HES. The parameter number of feasible combinations ($NUFC$) of hotels decides how many different sequences will be used in the remainder of the algorithm and have a chance to become the initial solution. Only the sequences of hotels with the highest HES are used.

Three strategies that make use of LS moves are applied to all of the $NUFC$ sequences of hotels to create an initial solution. The options that are performed are listed below:

1. LS with nine moves for an empty sequence of hotels;
2. Sub-Op for each trip between an empty sequence of hotels starting from the first trip and LS afterwards;
3. Sub-Op for each trip between an empty sequence of hotels starting from the last trip and LS afterwards.

After a Sub-Op is solved between two hotels in the second and third option, the selected vertices are placed in a list of vertices that cannot be inserted to the other trips. As a result, each vertex can be added at most once to the tour. The third step of the initialization does not contain such a list, so that the HES can be based on tours that visit some vertices multiple times. The tour with the highest score of the $NUFC * 3$ options is used as the initial solution.

4.2. Improvement

The improvement phase handles the vertices and hotels separately and investigates different tours to expand the solution space. The following two Vertices-shakes and Hotels-shake are examined:

1. delete the first half of the vertices of each trip;
2. delete the last half of the vertices of each trip;
3. change the hotels to the next HES sequence of hotels, while keeping the vertices fixed.

The Hotels-shake can give an unfeasible solution as the tour along the vertices and hotels might exceed the available time. Therefore, we remove the ver-

Algorithm 1 Outline of the SVNS heuristic

Input: List of $NUFC$ empty sequences of hotels; a feasible tour X with score S_X
Output: Feasible tour with a higher score

```
noImprovement = 0; k = 1
while noImprovement < maxNoImprovment do
    X1 = Remove first half of vertices and Local Search
    X2 = Remove second half of vertices and Local Search
    X3 = Hotels shake and Local Search with tour with score max{S_X1, S_X2}
    if S_X3 > S_Xbest then
        X and Xbest = tour hotels shake with score S_X3
        noImprovement = 0
    else
        noImprovement + 1
        if S_X3 > percentageWorse * S_Xbest or k = maxK then
            X = tour hotels shake with score S_X3
            k = 1
        else
            k + 1
```

tex with the lowest ratio of score over the decrease in trip length by excluding that vertex. This move is repeated until the solution is feasible and can be seen as the opposite of Insert. Computational results show that multiple vertices are removed in many cases, due to the time restrictions. When all the sequences of hotels are considered, the Hotels-shake starts again with the best HES sequence. After all the shakes are done, LS is performed to improve each solution and the solution with the highest score is saved.

4.3. Recentering

The goal of the recentering phase is to choose the solution that will be used in the next iteration. When the score of the shaking phase is higher than the score of the tour before the shaking phase, the tour is changed. As the algorithm uses the framework of SVNS and not VNS, tours with a score within a maximum percentage worse (*percentageWorse*) from the score of the tour before the shaking part are also accepted. We define solutions as different if the sequence of vertices and hotels is different, so the score or time of the tour is different. The definition affects the recentering, because computational results show that there are multiple tours with the same score in many problems.

4.4. Local Search

Nine moves are considered in the LS that have shown to be effective for the TOP (Vansteenwegen et al. 2011). As the hotels are fixed during this procedure and the OPHS without the selection of hotels is equivalent to the TOP. Some of the moves aim to reduce the tour time, while others focus on increasing the total score. The search involves intra-trip as well as inter-trip moves. LS starts with insertion as the level is predefined as 1 and the maximum level is equal to the number of possible moves (Algorithm 2).

The first moves, Insert and Move-Best, effect a single vertex every time. In Insert, the non-included vertex with the highest ratio of score over increase in trip length by including that vertex is included, when the solution remains feasible. In Move-Best each included vertex is moved to the best feasible position in the tour, such that the tour length is minimal.

Next, Two-Opt and Swap-Trips have an impact on two vertices or the part of the trip between two vertices. In Two-Opt, two crossing edges within each trip are exchanged. When there are multiple feasible possibilities, the move that leads to the smallest trip length is executed. Swap-Trips considers the exchange of two vertices from different trips, that minimizes the trip lengths. The move is executed if the total length of the tour decreases.

The following moves are a combination of multiple moves. In Extract-Insert one vertex is considered for exclusion, whereafter non-included vertices are

Algorithm 2 Local Search heuristic

Input: X = feasible tour with score S_X
Output: Feasible tour with a higher score

```
Neighborhoods: 1 = Insert; 2 = Move-Best; 3 = Two-Opt; 4 = Swap-Trips; 5 = Extract-Insert; 6 = Extract2Insert; 7 = Extract5Insert; 8 = Extract-Move-Insert; 9 = Replacement; level = 1; maxLevel = 9
while level < maxLevel do
    X1 = tour after performing move of that level
    if S_X1 > S_X then
        X = X1
        level = 1
    else
        level + 1
```

added by Insert in the current trip as long as the tour is feasible. The move is executed for each vertex of the tour, that leads to an increase of the total score. Extract2Insert and Extract5Insert are similar moves, but consider the removal of respectively two and five consecutive vertices.

Extract-Move-Insert is a combination of Extract-Insert and Move-Best. First, we look at the removal of a vertex. Second, we try to exchange two vertices, not necessarily of a different trip, that minimize the length of the tour. Lastly, if there is feasible exchange, non-included vertices are considered for insertion in the trip of the removed vertex if the tour remains feasible. When the score of the inserted vertex is higher than the score before the removal, the move is executed.

The last move, Replacement, is roughly the opposite of Extract-Insert. All non-included vertices are considered for insertion to the place in the tour with the smallest increase in the trip length. If the solution is still feasible after the insertion of a vertex, the move is executed. Otherwise, if deleting a vertex with a lower score makes the tour feasible, it is deleted and the move is executed.

4.5. Stopping criterion

The algorithm keeps track of two variables that determine the stopping criterion: k and noImprovement (Algorithm 1). If the score of the shaking part is worse than the score of the best-found tour, the variable noImprovement is increased until the maximum number of consecutive iterations without improvement ($maxNoImprovement$) is met. If the score is worse than $percentageWorse$ times the score of the previous solution, k is increased. Therefore, a recentering will always happen within $maxK$ iterations.

Our implementation of the algorithm differs from the algorithm of Divsalar et al. (2013) as we use one while-loop instead of two. This choice was based on computational results, which showed that the solutions after the Hotels-shake are often within $percentageWorse$ (as an indication, the HES of T1_65 is 200, 235 and 205). As a result, k does not increase much and the main phase gets trapped inside the second while-loop that terminates if k is equal to the parameter $maxK$.

4.6. Parameters

Four parameters are necessary for the algorithm. One by one Divsalar et al. (2013) tested three different values for all of these parameters and concluded that the values $NUFC$=250, $maxNoImprovement$=50, $maxK \approx 0.25 * NUFC$ and $percentageWorse$=0.3 give the best results. Our implementation uses a different value for $maxK$, because $maxK$ can only affect the solution if it is lower than $maxNoImprovement$. $MaxK = \lfloor 0.25 * maxNoImprovement \rfloor$ is used and gives a lowerbound on the number of times that the algorithm recenters.

5. Tabu Search

In this section, we propose a simple and fast TS algorithm for the OPHS, that uses TS instead of LS to select vertices after the shakes. The algorithm uses random numbers and an additional Cross-over shake. First, we will describe the outline of the algorithm. Thereafter, we describe the main phase, the TS, the stopping criterion and the involved parameters.

The TS uses the same outline as the SVNS as shown in Algorithm 3. Therefore, we can break down the algorithm in two phases. The first phase is the initialization. We use the same initialization as the

Algorithm 3 Outline of the TS heuristic

Input: List of $NUFC$ empty sequences of hotels; a feasible tour X with score S_X
Output: Feasible tour with a higher score

```
noImprovement = 0; k = 1; history = empty;
while noImprovement < maxNoImprovment do
    X1 = Remove first half of vertices and Local Search
    X2 = Remove second half of vertices and Local Search
    X3 = Hotels shake and Local Search with tour with score max{S_X1, S_X2}
    X4 = Cross-over shake and Local Search between X3 and best found tour
    for Xi = X1 to X4 do
        if Xi ∈ history then
            S_Xi = 0
    if max{ S_X1 S_X2 S_X3 S_X4 } > S_Xbest then
        X and Xbest = tour hotels shake with score max { S_X1 S_X2 S_X3 S_X4 }
        Add X to history
        noImprovement = 0
    else
        noImprovement + 1
        if k = maxK then
            X = tour hotels shake with score { S_X1 S_X2 S_X3 S_X4 }
            Add X to history
            k = 1
        else
            k + 1
```

SVNS, because it leads to solutions with a low average optimality gap. The second phase, or main phase, works iteratively and consists of two similar Vertices-shakes, a Hotels-shake and a Cross-over. After each shake, a TS that embeds the nine moves is performed to improve the solution. Next to that, we decide on the solution that is used in the next iteration. Just as the SVNS, the algorithm can recenter to a solution that is worse. The stopping criterion is based on the number of consecutive iterations without improvement.

5.1. Improvement

The TS algorithm contains three kinds of shakes that focus on changing and improving the tour. The first aspect is necessary in order to recenter and possibly improve the tour in the next iteration. The shakes consist of the same characteristics:

1. change the tour by removing or replacing vertices, hotels or both;
2. make the trips feasible;
3. perform TS to improve the solution.

The shakes start with destructing the solution to escape from a local optimum and to diversify the search. The TS is used to intensify the search and improve the solution with fixed hotels.

Next to the shakes of the SVNS algorithm, we added the Cross-over shake for three reasons. First, Cross-over has the capability to go beyond the HES sequences of hotels, due to combinations of tours. This feature is especially advantageous if the total, feasible number of sequences of hotels is larger than the parameter $NUFC$. As a result of the shake, the main phase is less dependent on the initialization. Second, while the Hotels-shake produces completely different solutions, this shake is based on the idea that a better solution might be close to the best-found solution. This idea seems promising, since a genetic algorithm that includes this shake produces near-optimal solutions (Divsalar et al. 2014). Third, the shake has the advantage that the relation between the vertices and hotels is not damaged. At most a few vertices are removed to make the solution feasible and the trips between the hotels have a high score.

The Vertices-shake is based on the idea that removing vertices and performing a TS can lead to a better solution. While the SVNS algorithm always removes half of the vertices in the improvement phase, the

TS algorithm uses a random number to split up the current solution. All trips are split according to the same number and at least one of the vertices remains in each trips. After the shakes, the removed vertices are added to the tabu list. TS is performed for the two tours: one with the vertices before and the other with vertices after the split. Adding these vertices to the tabu list is very important for the success of the TS. It helps to get a different and possibly better solution after the shake.

The difference of the Hotels-shake between the TS and SVNS is that TS is used to improve the solution instead of LS. The vertices that are removed due to the time restriction, are placed in the tabu list.

The fourth shake, Cross-over, combines the best-found solution with the current solution after the Hotels-shake. The procedure makes uses of two random numbers: to split up the tours and to determine the solution that is used as the start of the new tour. We consider only one tour for improvement, as TS is the most computationally expensive part of the Cross-over. The move is only considered if the distance between the two hotels that form the new trip is less than the time that is available for the trip. When the time between the hotels exceeds the time restriction, other splits are investigated. In the unusual occasion that all splits lead to unfeasible trips, the Cross-over shake is not applied. It is possible that the best-found and the current solution have the same sequence of hotels. In this case, the shake combines the vertices of the tours. Thereafter, duplicated vertices are removed and the vertices are removed until the trips satisfy the time restriction. Afterwards, the removed vertices are added to the tabu list and TS is performed.

An example of the shake is shown in Figure 1, where the left graphs represent the best found and current tour and the right graphs the two possible tours when the split is after the second trip. The second random number decides whether we use the upper-right or bottom-right tour. The shake is applied is the distance from from hotel two to four and hotel four to three do not violate the time restriction of the trip.

5.2. Recentering

In the recentering part the algorithm recenters to a solution of the shaking phase. In more detail, the solution that is used in the next iteration can be the result of the Vertices-, Hotels-shake and Cross-over

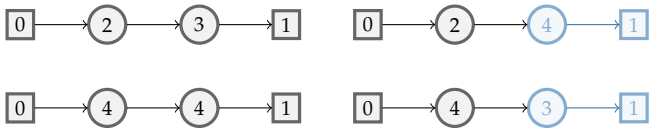

Figure 1 Cross-over with split after the second trip.

shake. Despite the tabu list, the Vertices-shake finds the same solution in many iterations. As a result, the algorithm recenters to a solution that is already considered. To diversify the search space, we allow the algorithm to recenter at most once to a specific solution. Only in the unlikely case when all shakes produce a solution that is already used, the algorithm can recenter to a solution that is already used. The idea behind the history is based on the same argument as the tabu list: although the score of some solutions might decrease, the probability of finding the global instead of a local optimum increases.

To simplify the algorithm, the parameter *MaxPercentageWorse* in the recentering phase is removed. Although improvements are likely to be found within solutions that are close to the current solution, there is also the risk of getting trapped in a local optimum. Recentering to another worse solution can be necessary for such a situation to finally reach the optimal solution. Moreover, most solutions of the improvement phase are within *MaxPercentageWorse*, such that the influence of this parameter is small. We choose to keep the algorithm simple and recenter after a fixed number of iterations: *maxK*. This gives the possibility to perform multiple Vertices-shakes for the same solution. In order to consider many solutions, we decrease the value of *maxK*.

5.3. Tabu Search

Many algorithms for problems that cope with intermediate facilities apply TS (Divsalar et al. 2014). For example, Castro et al. (2013) show that it can provide a good solution for many instances of the related TSPHS and Tang & Miller-Hooks (2005) apply TS to the TOP. It is designed to find the global optimum. The main idea is to explore a large part of the search space and avoid premature convergence. As the OPHS is a challenging problem it might be worthwhile to use TS and consider a lot of different solutions.

The structure of our TS is different than the preceding LS algorithm that uses levels (Algorithm 4). The LS goes to the next move if the solution is the

Algorithm 4 Tabu Search heuristic

Input: X = feasible initial solution with score S_X; tabu list
Output: Feasible tour with a higher score

```
1: iteration = 0;
2: while noImprovement < maxNoImprovement do
3:     while Insertion is possible do
4:         Insert a vertex, so change X and S_X
5:         Add move to tabu list
6:     for Move-Best; Two-Opt; Swap-Trips; Extract-Insert; Extract2Insert; Extract5Insert; Extract-Move-Insert;
   Replacement do
7:         Execute the move
8:         if There is an improvement then
9:             Change X and S_X
10:            Add move to tabu list
11:    Update the tabu list
12:    if The solution is not improved then
13:        noImprovement + 1
```

same and does not consider the move again when the following levels also do not lead to a better solution. The TS always investigates every move in every iteration. When a move does not result in improvements in the current iteration, it can lead to an improved solution in the next iteration due to a tabu list. We will show in Section 6 that Insert is the most important move. Therefore, TS repeats the move Insert in each iteration until a feasible insertion is not possible anymore.

The key element of the TS is a tabu list of moves that are not possible for a specific duration. In most algorithms for problems that cope with intermediate facilities the tabu list is defined as a set of vertices in a specific trip (Castro et al. 2013). However, we define the tabu list as a set of vertices that cannot be moved, inserted or deleted. Hence, more moves are tabu and the probability of achieving a different solution increases. When a move is executed, the involved vertex or vertices are added to the tabu list. At the start of the TS, the list contains vertices that are removed in the Vertices- or Hotels-shake.

The number of iterations that a move is unfeasible, the length of the tabu, is a random number of TS iterations. For example, if a vertex is inserted in iteration i, then changing the vertex is impossible until iteration $i + j$, where j is a randomly chosen from the pre-specified tabu length interval. This variable tabu length leads to good solutions and helps to explore a larger part of the search space (Tang & Miller-Hooks 2005; Castro et al. 2013).

5.4. Stopping criterion

The simple stopping criterion of the main phase is defined as $maxNoImprovement$ successive iterations of the main phase with the same best solution. The value of the parameter $maxNoImprovement$ is low, because most improvements are found in the first iterations and the algorithm should be fast. The TS also uses a stopping criterion based on the number of successive iterations without improvement: always one more than the maximum tabu length. As a result, each vertex has the chance to be moved and the TS cannot improve the solution anymore.

5.5. Parameters

The TS algorithm contains five parameters. Different values of the parameters minimum and maximum tabu length as well as $maxK$, the maximum number of iterations without recentering, will be tested in Section 6. Moreover, the algorithm uses $maxNoImprovement$ as a stopping criterion and $NUFC$ for the number of sequences of hotels that are considered in the main phase.

6. Results

In this section, four data set with test problems are described. Next, the implications of different parameter values for the TS algorithm is described and the results of the SVNS and TS algorithm for these instances are compared. A few solutions are compared with the exact solutions and the impact of the nine moves and the shakes are described.

6.1. Data

The test instances of set I, II, III and IV are available at www.mech.kuleuven.be/en/cib/op. These instances contain minimal 32 and maximal 102 vertices and 1 to 15 hotels next to the fixed start and end hotel. The instances with the least number of vertices and most strict time restrictions are much easier to solve, but the instances with a name that begins with 64 and 66 need more computation time. Set I and II require few computation time, as the number of hotels and trips is low. Set III is the most computationally intensive, due to the relatively high number of hotels. As a consequence, we were not able to obtain results for the instances with 12 hotels and 5 trips within the available time for this thesis. The scores of both algorithms are compared with the optimal scores of the instances that are solved by Divsalar et al. (2013). For the instances of set IV with three trips, there were no optimal values available. However, it was possible to find the optimal value for each trip between two hotels. They summed up these score for each sequence of hotels. As a result, some vertices might be included multiple times in the tour and the obtained score is an upper bound rather than the optimal score for the problems with duplicate vertices.

The results for the set I with three hotels and two trips are shown in Table 1. The second column gives the optimal score and the third column the score of the initialization. TNFC means total number of feasible combinations of hotels and gives a first impression of the difficulty of the instances. The maximum value of TNFC can be calculated by $h^{(D-1)}$, where h denotes the number of hotels and D the number of trips. The actual value of TNFC is sometimes lower, as not all sequences of hotels are feasible. The gap shows the difference between the score of SVNS (and TS) and the optimal score and can be calculated by (optimal score - SVNS score)/optimal score * 100. Unless states differently, these 35 instance of set I with two hotels and three trips are used for the results in the remainder of this section.

6.2. Parameter tuning

The TS uses the parameters minimum and maximum length of the tabu. Different values for these parameters and the maximal number of iterations without recentering, $maxK$, are given in Table 2. The algorithm also contains the parameters $NUFC$ for the initialization and $maxNoImprovement$ for stopping criterion. We use the same values for these parameters as the SVNS algorithm, because Divsalar et al. (2013) concluded that these parameters do not significantly affect the quality of the solutions.

The average gap is not very sensitive to changes in the tabu length. The computation time, however, increases with average tabu length. The tabu length between 3 and 6 is chosen, because the speed of the algorithm is important for applications of the OPHS. The influence of the parameter $maxK$ appears to be small compared to the tabu length. The value 10 is used, as the computation time is a little lower and the gaps are similar. The solutions of all parameter values except tabu length in [3, 6] and $maxK$=5 are based on performing the algorithm once.

6.3. TS and SVNS

The reproduction of the SVNS algorithm finds the optimal score for 17 out of the 35 instances. The score of the best-found solution differs from the score of the initialization phase for 7 out of these 17 instances. The average score of the TS algorithm is optimal for 24 out of the 35 instances and leads to an average gap with the optimal solution of 0.394% (Table 1). It is remarkable that the intermediate hotels of the TS and SNVS solutions are similar to the hotels of the initial solution for all instances except one. This can be due to a good initial solution, but can also indicate that the Hotel-shake and Cross-over shake are not capable to find good solutions with other sequences of hotels.

Vansteenwegen et al. (2009) show that TS produces good solutions for small instances of the TOP. The average scores of the TS algorithm, based on performing the algorithm three times, are better than the SVNS for all sets (Table 5, 6, 7 and 8 in Appendix A). A summary of the results is given in Table 3. The third column represents the number of problems and the third column gives the number of problems that are solved optimally. This number in not restricted to be an integer, as the algorithm is performed multiple times. The TS algorithm consistently finds higher scores with a lower maximal gap for the sets. The average gap for set I to IV is respectively 0.498%, 0.397%, 0.876% and 1.469% better.

To understand the impact of the randomness in the algorithm, we performed the TS three times for the first 35 instances of set I. The difference between the average gaps for the 35 instances is at most 0.143% with a maximal gap of 3.750% that occurred for

Table 1 Score, minimal and average gap of TS and gap of SVNS for set I with three hotels and two trips.

Instance	Opt.	Init.	TNFC	SVNS Gap (%)	TS Av. (%)	TS Min. (%)
T1_65	240	235	3	2.083	0	0
T1_70	260	260	3	0	0	0
T1_73	265	260	3	0	0	0
T1_75	270	270	3	0	0	0
T1_80	280	275	3	0	0	0
T1_85	285	280	3	0	0	0
T3_65	610	610	3	0	0	0
T3_75	670	650	3	2.985	0	0
T3_80	710	660	3	4.225	0	0
T3_85	740	720	3	2.703	0	0
T3_90	770	740	3	3.896	0	0
T3_95	790	750	3	3.797	0.844	0
T3_100	800	770	3	3.750	1.250	0
T3_105	800	800	3	0	0	0
64_45	816	804	3	1.471	0	0
64_50	900	852	3	2.667	2.667	2.667
64_55	984	960	3	1.220	1.829	1.220
64_60	1062	1020	3	1.130	1.130	1.130
64_65	1116	1080	3	0	0	0
64_70	1188	1140	3	1.515	1.151	1.515
64_75	1236	1224	3	0	0	0
64_80	1284	1260	3	0	0	0
66_40	575	570	3	0.870	0.870	0.870
66_45	650	625	3	0.769	0.769	0.769
66_50	730	690	3	2.055	2.055	2.055
66_55	825	805	3	2.424	0	0
66_60	915	890	3	0.546	0.546	0.546
66_125	1670	1655	3	0.599	0.299	0
66_130	1680	1675	3	0	0	0
100_30	173	173	2	0	0	0
100_35	241	241	1	0	0	0
100_40	299	299	2	0	0	0
100_45	367	367	3	0	0	0
102_50	181	181	3	0	0	0
102_60	243	243	3	0	0	0
Average				1.106	0.394	0.308

Table 2 Average gap with optimal solution and computation time of set I with three hotels and two trips for different parameter values.

Tabu length	*maxK*	Gap (%)	Time (min.)
[3, 6]	5	0.394	2.006
[5, 10]	5	0.482	2.733
[3, 6]	10	0.360	2.008
[5, 10]	10	0.521	4.601
[10, 20]	10	0.414	5.838

Table 3 Summary of the results with known optimal solutions.

Set	Alg.	# Prob.	# Opt.	Max. (%)	Av. (%)
I	SVNS	105	55	6.494	0.937
	TS	105	71.666	3.750	0.440
II	SVNS	70	38	6.494	0.936
	TS	70	46.000	5.000	0.539
III	SVNS	22	4	8.259	3.089
	TS	22	5.333	8.259	2.213
IV	SVNS	5	2	4.511	2.233
	TS	5	4.000	3.822	0.764

T3_100. Although the scores of the TS algorithm differ quite much, the average score is still higher than the SVNS algorithm.

The computation times of the two algorithms are similar. The SVNS algorithm takes on average 2.354 minutes, compared to on average 2.006 minutes for the TS algorithm. Even with larger instances the computation times is similar (respectively 7.811 and 7.498 minutes for set II with eight hotels and four trips). These higher computation times are mainly due to the initialization. For example, the initialization of 100_160 from set III takes 5 hours and 30.069 minutes, while the main phase needs less than 16.234 minutes. The computation time of the main phase is dependent on two things. First, the time is dependent on the number of times that the algorithm recenters as shown in Algorithm 1 and the number of improvements that are found in the TS. Second, it is dependent on the speed of the convergence. The tabu list prevents premature convergence and, hence, increases the computation time. On the other side, the TS has a different structure and stopping criterion than the LS.

6.4. Exact solutions

Solving some instances exact by means of the MILP formulation shows the difficulty of the OPHS. The instances of set I with three hotels and two trips are considered with the CPLEX implementation for Java. While instance T1_70 was solved in 8.138 seconds, instance T3_105 took 38.253 minutes. 20 of the 35 instances can be solved within one hour. This enables us to compare the optimal tour and the tours of the SVNS algorithm.

The SVNS algorithm was not able to solve 6 out of these 20 instances. Only T3_100 has the wrong sequence of intermediate hotels. This indicates that most improvements in the algorithm are possible on the second level of selecting the vertices for small instances. Parts of the trips should be reversed, vertices of the trips have to be switched completely or just a few vertices have to be removed or inserted. Moreover, the number of vertices is not equal for 12 of the 14 trips. The optimal and SVNS tour of T3_75 are shown in Figure 2 and 3 respectively. The order of the trips is different: the optimal route starts with vertex 26, while the SVNS tour starts with vertex number 25.

6.5. Selection of moves

An example of a test instance where the SVNS algorithm is not able to find the right trips is T1_65 with score 235. In order to reach the optimal score, 240, two vertices have to be removed and one vertex has to be inserted. Note that the sum of the two vertices that should be removed is lower than the score of the inserted vertex. Hence, a Replacement move which considers removing multiple vertices would lead to the optimal solution.

A version of Replacement that considers removing up to three vertices with a lower sum of score indeed gave a score of 240 for T1_65. The SVNS algorithm with this move is able to find an improved score for the instances T1_65 and T3_85 out of the 35 instances, but does also lead to 10 worse solutions. The worse solutions indicate that the SVNS algorithm is sensitive to changes in the search path. The move did not result in any changed scores for the TS. Therefore, we choose not to adjust Replacement and keep the move simple.

The downside of the extended Replacement is the additional computation time, as each combination of vertices has to be considered until an improvement is found. However, by (i) only considering combina-

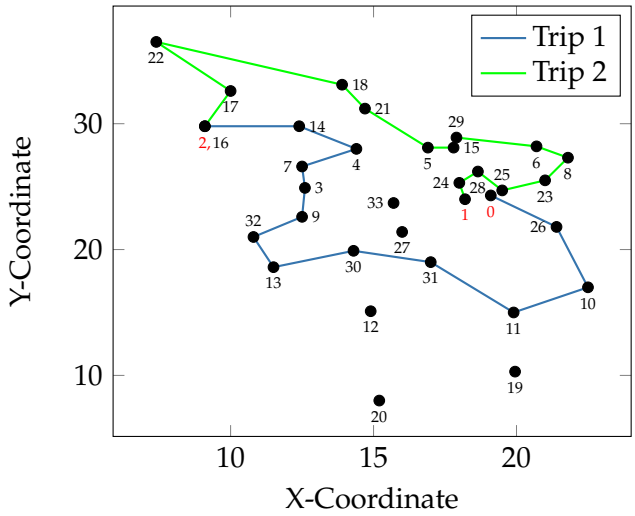

Figure 2 Optimal route of instance T3_75 with score 670. The hotels are red and vertices black.

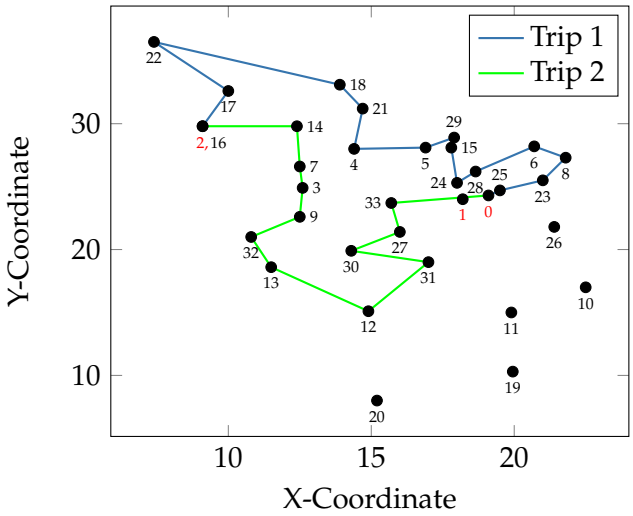

Figure 3 Route from SVNS algorithm of instance T3_75 with score 650.

tions of vertices with a lower score than the inserted vertex and (ii) directly executing the move if a combination of vertices is found that no longer violates the restrictions, the average computation time does not differ a lot. The average running time of the SVNS algorithm was 2.354 minutes, which is similar to the running time of the original SVNS algorithm (2.350 minutes).

An analysis of the nine moves showed that Insert is the most important move, as it is used in 49.677% of the moves. The next moves (Move-Best, Two-Opt, Swap-Trips, Extract-Insert, Extract2Insert, Exact5Insert, Extract-Move-Insert and Replacement) are used 33.367%, 8.187%, 4.097%, 2.573%, 2.093%, 0.005% 0.000% and 0.000% respectively. These decreasing percentages give an indication that the order of the moves is correct, as it converges quickly. However, this is difficult to state as the percentages can be dependent of the order. On average 674 moves are performed during the main phase. The main phase needed a little more than 50 iterations, so that the average number of moves per LS is equal to 4.451. As the instances with more trips and hotels might need the last two moves and the computation time of the TS algorithm is quite low, we decide to keep the nine moves.

6.6. Performance of shakes

The effect of the Cross-over in the TS algorithm has two sides. On the one hand, it increases the flexibility. On the other hand, hotels that are not present in the *NUFC* sequences of hotels are still not considered and the shake increases the computation time. The shake results in a higher average gap for the instances with a small number of trips and a smaller gap for more difficult instances (Table 4). As many practical problems (like the planning of a holiday) contain more trips, we have chosen to include Cross-over. Note that the average differences for set I and II, 0.015% and 0.027% respectively, are relatively small compared to the influence of the randomness (Table 1). Therefore, these differences can also be due to the effect of random numbers.

Table 4 Gap with the optimal solution for TS and TS without Cross-over (TS-C) for set I and II.

Set	(# hot., # tr.)	TS-C (%)	TS (%)
I	(3, 2)	0.357	0.394
I	(4, 3)	0.329	0.442
I	(5, 4)	0.677	0.483
II	(7, 3)	0.415	0.370
II	(8, 4)	0.717	0.709

Most genetic algorithms contain two procedures: Cross-over and Mutation. The last procedure ensures that the solutions are diverse enough. The genetic algorithm of Divsalar et al. (2014) produces very good solutions for the OPHS. Therefore, we implemented the Mutation procedure that uses a random number to select the hotel that will be replaced. If changing the tour into a sequence of hotels results in one or more of the *NUFC* sequences of hotels, the hotel is changed to one of these new hotels with a probability proportional to the HES. Otherwise, a random num-

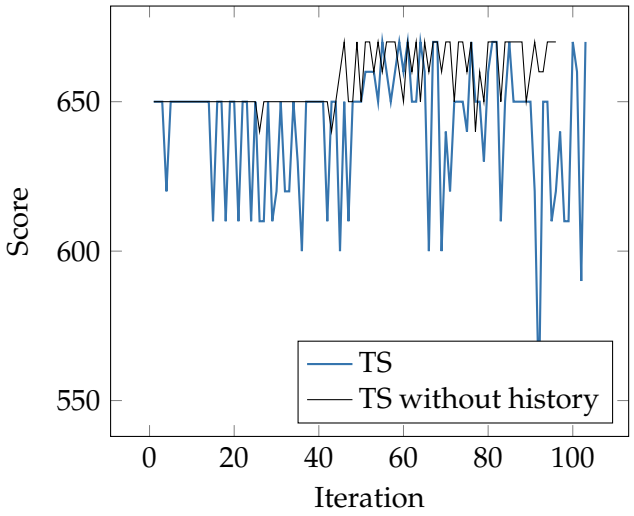

Figure 4 Search path of the TS algorithm with and without the search history for instance T3_75 of set I with two trips and three hotels.

ber is used to determine the new hotel. Thereafter, vertices are removed to make the tour feasible to the time restrictions and TS is performed. Due to computation time, the Hotels-Shake is removed during this analysis.

The algorithm with the Mutation shake leads to an average gap of 0.514%, based on performing the algorithm three times. As the gap is higher than gap of the algorithm with the Hotels-shake, we decided to keep the Hotels-shake.

6.7. Search path diversification

The recentering phase of the SVNS and TS algorithm differ as the TS algorithm uses the search history. The algorithm can recenter at most $maxK$ times to a specific solution. A comparison between the score in each iteration of the TS algorithm with and without this history of instance T3_75 is shown in Figure 4. The TS algorithm finds solutions with 10 unique scores, while the same TS algorithm without the history only finds solutions with four distinct scores. Although the parameter $maxPercentageWorse$ is removed in both versions of the TS algorithm, the TS without history cannot find solutions with a lower score than 640. The search history of other instances follows a similar pattern, confirming that the TS algorithm with history is indeed able to diversify the search. It is important to note that multiple solutions can give the same score.

7. Discussion

This section compares the results of the two algorithm with the results of Divsalar et al. (2013) and Divsalar et al. (2014). Next, some implications of the SVNS and TS algorithm are described.

Although we use the same SVNS algorithm as Divsalar et al. (2013), the results differ a little bit. On average our implementation of the SVNS algorithm performs 0.086% better for set I and 0.134% better for set II. The results for the instances of set III were 0.479% worse and set IV 1.473% worse. The total number of feasible sequences of hotels is the same, but the score of the initial solution is different for some of the instances. Therefore, our implementation of the LS or Greedy Sub-Op is different.

The different results can be due to a other implementation of the moves, for example, Extract-Insert. We save the tour at the beginning of the move and only consider the vertices of this tour for insertion. A different implementation would also consider the move for vertices that are inserted during the move. Also, our results differ as we only have one while-loop and we recenter when the score is the same, but the total time of the tour is reduced.

To the knowledge of the author, the population-based memetic algorithm of Divsalar et al. (2014) produces the best results for the OPHS. The computation time of their algorithm is lower than our computation time and they reach solutions with a lower gap for the difficult instances of set III. However, our results for set I and II show a lower gap and an equal gap for set IV. Their algorithm also uses random numbers and a Cross-over operator, but contains a LS of six instead of nine moves, a Mutation procedure and a different construction phase. Therefore, it is difficult to compare the effectiveness of the main phases.

When comparing the TS and SVNS algorithm, the tabu list is the main advantage of the TS algorithm. Also the ability to recenter to a solution with a worse score is an advantage. The tabu list guides the algorithm towards new solutions to explore a wider part of the search space. The additional Cross-over shake leads to a higher score. Although determining promising sequences of hotels is difficult, this result indicates that an improved solution might be close to the best-found solution.

A disadvantage of the algorithms is the high computation time of the initialization phase for instances with many possible, feasible sequences of hotels. A

Sub-Op need to be solved for each sequence, in order to obtain the HES. However, it is difficult to rank the sequences without performing time-consuming algorithms like Sub-Op, LS or TS. We performed the TS for the first 35 instances of set I with the sequence of hotels that has the second highest has to investigate robustness of our algorithm. These initial solutions resulted in an average gap of 0.955%, which is still low. Therefore, a faster construction heuristic for the initial solution might be useful for difficult problems, as it requires less computation time and results in good solutions. The analysis of the moves showed that we can possibly reduce the computation time by deleting some moves like Extract-Move-Insert and Replacement. Moreover, it takes only a little additional time to perform the TS algorithm multiple times, due to the absence of randomness in the initialization.

Unlike many other TS algorithms with intermediate facilities for the TSPHS and TOP, our algorithm does not include an aspiration condition (Tang & Miller-Hooks 2005; Castro et al. 2013). Such a condition considers whether moves that are present in tabu list but lead to a substantial increase in the score, can nonetheless be executed. Tang & Miller-Hooks (2005) claim that it helps to find the global instead of a local optimum. Our algorithm recenters at least once in the $maxK$ iterations and has three kinds of shakes, so the probability of being trapped in a local optimum is already small. Therefore, we expect that an aspiration condition will not have a big effect on the score.

8. Conclusion

In this thesis, we compared two algorithms for the OPHS. The presence of intermediate facility leads to an algorithm that consists of two levels: the level of selecting the correct sequence of hotels and the level of selecting the best vertices in each trip. The chosen sequence of hotels is of great importance for the quality of the solutions. The MILP formulation allowed us to solve only a few instances within a hour, indicating indeed that this problem is challenging.

The algorithm of Divsalar et al. (2013) contains a SVNS framework and uses a short version of LS, called Sub-op, to obtain a ranking for the each trips between each pair of hotels and LS to find a feasible initial solution. Next, the improvement and recentering phase changes the vertices and hotels and performs an extensive LS with nine moves. To diversify the search, the algorithm recenters to solutions with a worse score.

The second algorithm is based on TS to find tours with a good score. The key element of this algorithm is a tabu list of vertices that cannot be moved during a certain number of iterations. Random numbers are used for the tabu length and three kinds of shakes are applied to find more and better solutions.

While our implementation of the SVNS has an average gap of 1.203%, the TS algorithm can find tours with an average gap of 0.675% and the optimal score is found for 62.871% of the test instances.

Although the TS algorithm produces near-optimal solutions, more research would be useful. It would be interesting to see whether an aspiration criterion does affect the quality of the solutions. More research is also necessary for the construction of initial solutions with low computation times. Studying extensions of the OPHS like scores for beautiful routes, time windows or hotel costs does also add value.

All in all, the proposed TS algorithm has a simple structure with five parameters. The algorithm produces high-quality solutions for a low computation time and clearly outperforms the SVNS algorithm.

References

Archetti, C., Hertz, A., & Speranza, M. G. (2007). Metaheuristics for the team orienteering problem. *Journal of Heuristics*, *13*(1), 49–76.

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

Castro, M., Sörensen, K., Vansteenwegen, P., & Goos, P. (2012). A simple grasp + vnd for the travelling salesperson problem with hotel selection. Technical report.

Castro, M., Sörensen, K., Vansteenwegen, P., & Goos, P. (2013). A memetic algorithm for the travelling salesperson problem with hotel selection. *Computers & Operations Research*, *40*(7), 1716–1728.

Divsalar, A., Vansteenwegen, P., & Cattrysse, D. (2013). A variable neighborhood search method for the orienteering problem with hotel selection. *International Journal of Production Economics*, *145*(1), 150–160.

Divsalar, A., Vansteenwegen, P., Sörensen, K., & Cattrysse, D. (2014). A memetic algorithm for the orienteering problem with hotel selection. *European Journal of Operational Research*, *237*(1), 29–49.

Feillet, D., Dejax, P., & Gendreau, M. (2005). Traveling salesman problems with profits. *Transportation science*, *39*(2), 188–205.

Fischetti, M., Gonzalez, J. J. S., & Toth, P. (1998). Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, *10*(2), 133–148.

Hansen, P. & Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, *130*(3), 449–467.

Ke, L., Archetti, C., & Feng, Z. (2008). Ants can solve the team orienteering problem. *Computers & Industrial Engineering*, *54*(3), 648–665.

Souffriau, W., Vansteenwegen, P., Berghe, G. V., & Van Oudheusden, D. (2010). A path relinking approach for the team orienteering problem. *Computers & Operations Research*, *37*(11), 1853–1859.

Tang, H. & Miller-Hooks, E. (2005). A tabu search heuristic for the team orienteering problem. *Computers & Operations Research*, *32*(6), 1379–1407.

Tsiligirides, T. (1984). Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, *35*(9), 797–809.

Vansteenwegen, P., Souffriau, W., Berghe, G. V., & Van Oudheusden, D. (2009). Metaheuristics for tourist trip planning. In *Metaheuristics in the Service Industry* (pp. 15–31). Springer.

Vansteenwegen, P., Souffriau, W., & Sörensen, K. (2012). The travelling salesperson problem with hotel selection. *Journal of the Operational Research Society*, *63*(2), 207–217.

Vansteenwegen, P., Souffriau, W., & Van Oudheusden, D. (2011). The orienteering problem: A survey. *European Journal of Operational Research*, *209*(1), 1–10.

Zhu, C., Hu, J., Wang, F., Xu, Y., & Cao, R. (2012). On the tour planning problem. *Annals of Operations Research*, *192*(1), 67–86.

A. Appendix

Table 5 Optimal value and gap of the SVNS and TS of set I, with 4 and 5 hotels and 3 and 4 trips.

		3 trips, 4 hotels					4 trips, 5 hotels				
				SVNS	TS	TS			SVNS	TS	TS
Instances	Opt.	Init.	TNFC	Gap (%)	Av. (%)	Min (%)	Init.	TNFC	Gap (%)	Av. (%)	Min (%)
T1_65	240	235	16	2.083	0	0	240	107	0	0	0
T1_70	260	260	16	0	0	0	260	107	0	0	0
T1_73	265	265	16	0	0	0	265	107	0	0	0
T1_75	270	270	16	0	0	0	270	107	0	0	0
T1_80	280	280	16	0	0	0	280	125	0	0	0
T1_85	285	280	16	0	0	0	285	125	0	0	0
T3_65	610	610	16	0	0	0	610	100	0	0	0
T3_75	670	650	16	2.985	0	0	650	125	2.985	0	0
T3_80	710	710	16	0	0	0	710	107	0	0	0
T3_85	740	710	16	0	0	0	730	107	0	0	0
T3_90	770	770	16	0	0	0	720	100	6.494	0	0
T3_95	790	790	16	0	0	0	760	100	0	0	0
T3_100	800	800	16	0	1.667	0	770	125	3.750	2.500	2.500
T3_105	800	790	16	1.250	0.833	0	750	125	1.250	0	0
64_45	816	816	12	0	0	0	816	44	0	0	0
64_50	900	870	16	2.667	2.667	2.667	870	72	3.333	3.333	3.333
64_55	984	966	16	1.829	1.423	1.220	978	100	0.610	0.610	0.610
64_60	1062	990	16	1.695	0	0	1044	100	1.695	0	0
64_65	1116	1104	16	0	0	0	1116	125	0	0	0
64_70	1188	1164	16	2.020	1.515	1.515	1140	125	4.040	1.515	1.515
64_75	1236	1194	16	0.971	0.971	0.971	1206	125	1.942	1.456	1.456
64_80	1284	1260	16	1.402	1.869	1.869	1254	125	0.467	0.156	0
66_40	575	570	16	0.870	0.870	0.870	570	107	0.870	0.870	0.870
66_45	650	645	16	0.769	0.769	0.769	645	88	0.769	0.769	0.769
66_50	730	690	16	4.110	2.055	2.055	715	107	2.055	2.055	2.055
66_55	825	805	16	0	0	0	805	107	2.424	0	0
66_60	915	890	16	0.546	0.546	0.546	890	107	0.546	0.546	0.546
66_125	1670	1655	16	0.898	0	0	1635	125	0.898	0.898	0.898
66_130	1680	1675	16	0.298	0.298	0.298	1660	125	1.190	1.190	1.190
100_30	173	173	1	0	0	0	173	2	0	0	0
100_35	241	241	2	0	0	0	241	1	0	0	0
100_40	299	299	2	0	0	0	299	1	0	0	0
100_45	367	367	2	0	0	0	367	2	0	0	0
102_50	181	181	12	0	0	0	181	33	0	0	0
102_60	243	243	12	0	0	0	243	60	0	0	0
Average				0.697	0.442	0.365			1.009	0.483	0.450

Table 6 Optimal value and gap of the SVNS and TS of set II.

		3 trips, 7 hotels					4 trips, 8 hotels				
				SVNS	TS	TS			SVNS	TS	TS
Instances	Opt.	Init.	TNFC	Gap (%)	Av. (%)	Min. (%)	Init.	TNFC	Gap (%)	Av. (%)	Min. (%)
T1_65	240	240	49	0	0	0	240	482	0	0	0
T1_70	260	260	49	0	0	0	260	482	0	0	0
T1_73	265	265	49	0	0	0	265	482	0	0	0
T1_75	270	270	49	0	0	0	270	482	0	0	0
T1_80	280	280	49	0	0	0	280	512	0	0	0
T1_85	285	280	49	0	0	0	285	512	0	0	0
T3_65	610	610	49	0	0	0	610	448	0	0	0
T3_75	670	650	49	2.985	0	0	650	512	2.985	0	0
T3_80	710	710	49	0	0	0	710	482	0	0	0
T3_85	740	710	49	0	0	0	730	482	0	0	0
T3_90	770	740	49	3.896	0	0	720	448	6.494	0	0
T3_95	790	790	49	0	0	0	760	448	3.797	3.797	3.797
T3_100	800	800	49	0	0	0	770	512	3.750	2.500	2.500
T3_105	800	800	49	0	0	0	760	512	5.000	5.000	5.000
64_45	816	816	42	0	0	0	816	252	0	0	0
64_50	900	870	49	2.667	2.667	2.667	870	378	3.333	2.667	2.667
64_55	984	966	49	1.829	1.219	1.219	978	448	0.610	0.610	0.610
64_60	1062	1020	49	3.390	0	0	1044	448	1.695	0.188	0
64_65	1116	1104	49	0	0	0	1116	512	0	0	0
64_70	1188	1164	49	2.020	1.515	1.515	1152	512	2.525	2.694	2.020
64_75	1236	1206	49	1.456	1.133	0.971	1212	512	1.942	1.295	0.971
64_80	1284	1194	49	0.935	1.869	1.869	1260	512	0.935	0.623	0
66_40	575	570	45	0.870	0.870	0.870	570	302	0.870	0.870	0.870
66_45	650	645	47	0.769	0.769	0.769	645	228	0.769	0.769	0.769
66_50	730	690	49	2.055	2.055	2.055	715	357	2.055	2.055	2.055
66_55	825	805	49	0	0	0	805	400	2.424	0	0
66_60	915	890	49	0.546	0.546	0.546	890	424	0.546	0.546	0.546
66_125	1670	1655	49	0.898	0	0	1655	512	0.898	0.898	0.898
66_130	1680	1675	49	0.298	0.298	0.298	1675	512	0.298	0.298	0.298
100_30	173	173	1	0	0	0	173	2	0	0	0
100_35	241	241	2	0	0	0	241	1	0	0	0
100_40	299	299	2	0	0	0	299	1	0	0	0
100_45	367	367	2	0	0	0	367	2	0	0	0
102_50	181	181	14	0	0	0	181	60	0	0	0
102_60	243	243	17	0	0	0	243	105	0	0	0
Average				0.703	0.370	0.365			1.169	0.709	0.657

Table 7 Optimal value and gap of the SVNS and TS of set III with 4 trips and 12 hotels.

Instances	Opt	Init	TNFC	SVNS Gap (%)	TS Av. (%)	TS Min. (%)
64_75	1236	1224	1728	0.971	0.971	0.971
64_80	1284	1272	1728	0.935	0.312	0
66_125	1670	1670	1728	0	0	0
66_130	1680	1675	1728	0.298	0.298	0.298
100_50	412	408	39	0.971	0.971	0.971
100_60	504	504	161	0	0	0
100_70	590	575	276	2.542	0	0
100_80	652	652	892	0	0	0
100_90	725	706	1220	0	0	0
100_100	782	774	1296	1.023	1.023	1.023
100_110	835	798	1728	4.072	4.072	4.072
100_120	894	845	1728	5.369	4.064	3.691
100_130	956	884	1551	7.531	2.510	0.314
100_140	1013	928	1551	7.009	3.356	3.356
100_150	1057	1027	1728	2.838	1.987	1.987
100_160	1114	1022	1728	8.259	8.259	8.259
100_170	1164	1082	1728	7.045	5.441	4.983
100_180	1201	1103	1728	6.911	6.106	5.995
100_190	1234	1159	1728	3.890	3.485	2.836
100_200	1261	1201	1728	3.013	2.776	2.538
100_210	1284	1222	1728	4.595	2.882	2.648
100_240	1306	1296	1728	0.689	0.179	0.077
Average				3.089	2.213	2.001

Table 8 Optimal value and gap of the SVNS and TS of set IV.

	2 trips, 5 hotels						**3 trips, 5 hotels**					
Instance	Opt.	Init.	TNFC	SVNS Gap (%)	TS Av. (%)	TS Min. (%)	UB	Init.	TNFC	SVNS Gap (%)	TS Av. (%)	TS Min. (%)
100_20	247	240	4	2.834	0	0	376	368	19	2.128	2.128	2.128
100_25	385	385	5	0	0	0	568	524	25	7.746	7.746	7.746
102_35	157	151	2	3.822	3.822	3.822	380	324	7	14.737	14.737	14.737
102_40	210	210	2	0	0	0	493	385	7	21.907	21.095	21.095
102_45	266	254	2	4.511	0	0	579	430	7	25.734	23.143	22.798
Average				2.233	0.764	0.764				14.450	13.629	13.560