Int. J. Production Economics 145 (2013) 150–160

Contents lists available at SciVerse ScienceDirect

Int. J. Production Economics

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

A variable neighborhood search method for the orienteering problem with hotel selection

A. Divsalar [a,b,*], P. Vansteenwegen [a,c], D. Cattrysse [a]

[a] KU Leuven, Centre for Industrial Management/Traffic and Infrastructure, Celestijnenlaan 300A, 3001 Leuven, Belgium
[b] Faculty of Mechanical Engineering, Babol University of Technology, Babol, Mazandaran, Iran
[c] Department of Industrial Management, Ghent University, Ghent, Belgium

ARTICLE INFO

Article history:
Received 7 February 2012
Accepted 9 January 2013
Available online 21 January 2013

Keywords:
Orienteering problem
Hotel selection
Variable neighborhood search

ABSTRACT

In this paper, we present the orienteering problem with hotel selection (OPHS), an extension of the orienteering problem (OP). In the OPHS, a set of vertices with a score and a set of hotels are given. The goal is to determine a fixed number of connected trips that visits some vertices and maximizes the sum of the collected scores. Each trip is limited in length and should start and end in one of the hotels. We formulate the problem mathematically, explain the differences with related optimization problems and indicate what makes this problem inherently more difficult.

We use a skewed variable neighborhood search that consists of a constructive initialization procedure and an improvement procedure. The algorithm is based on a neighborhood search operator designed specifically for the hotel selection part of this problem, as well as some typical neighborhoods for the regular OP.

We generate a set of 224 benchmark instances of varying sizes with known optimal solutions. For 102 of these instances our algorithm finds the optimal solution. The average gap with the optimal solution over all these instances is only 1.44% and the average computation time is 1.91 s.

1. Introduction

The orienteering problem with hotel selection (OPHS) is a new variant of the orienteering problem. Although many studies have focused on variants of the orienteering problem, this problem has not yet been discussed in the literature. In the OPHS, a set of $H+1$ hotels is given ($i=0,\ldots,H$). Also, N vertices are given and each vertex $i=H+1,\ldots,H+N$ is assigned a score S_i. Hotels have no score. The time t_{ij} needed to travel from vertex i to j is known for all pairs. The time available for each trip $d=1,\ldots,D$ is limited to a given time budget T_d which can be different for each trip. The goal is to determine a tour that maximizes the total collected score. The tour is composed of D connected trips and visits each vertex at most once. In this formulation, the number of trips, D, is a given parameter of the problem. Every trip should start and end in one of the available hotels. The initial starting and final arriving hotel of the tour are given ($i=0$ and 1, respectively). They can also be used as a hotel during the tour.

To avoid any confusion regarding the terminology used in this paper, the term "trip" refers to an ordered set of vertices with a specific starting and ending hotel. The term "tour" is used for the ordered set of trips that connect the initial departure hotel to the final arrival hotel.

As the above mentioned definition shows, the OPHS is a new and challenging problem. It is a generalization of the OP and is therefore also NP-hard. Moreover, the OPHS is much more complex than the OP, since the starting and ending location for each trip need to be optimized as well.

In order to explain the nature of the orienteering problem with hotel selection, it is helpful to consider an example. Imagine a tourist who is planning to visit a certain region with various attractions. The tourist wants to select the combination of attractions that maximizes his pleasure. The visit will last several days and only the initial departure and the final arrival location are fixed. The departure and arrival locations (hotels) during the visit should be selected in an optimal way based on the attractions that are selected. Obviously, the hotel in which the tourist ends a given day has to be the same as the starting hotel of the next day. The hotels can be selected from all suitable hotels available in the region.

A large number of practical applications can be modeled by the OPHS. For instance, a submarine performing a surveillance activity (tour) composed of consecutive missions (trips). It requires

* Corresponding author at: Celestijnenlaan 300A Box 2422, 3001 Heverlee, Belgium. Tel.: +32 16 32 25 67; fax: +32 16 32 29 86.
E-mail addresses: ali.divsalar@cib.kuleuven.be, alidivsalar@gmail.com (A. Divsalar), Pieter.Vansteenwegen@cib.kuleuven.be (P. Vansteenwegen), Dirk.Cattrysse@cib.kuleuven.be (D. Cattrysse).

0925-5273/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ijpe.2013.01.010

save zones (hotels) for provisioning between two missions. There are several possible points (vertices) to survey, but not all of them can be visited due to the limited available time. The submarine wants to maximize its benefit by selecting the most interesting combination of points. Only the initial departure and final arrival location of the whole surveillance activity are fixed. The departure and arrival save zone of each mission during the activity should be selected in an optimal way, considering the points that are selected for a visit. It is clear that when the submarine ends its current mission (trip) in a certain save zone, the next mission has to start in the same save zone.

Actually, depending on the exact practical circumstances, many of the applications for the orienteering problem (Vansteenwegen et al., 2011) or for the traveling salesperson problem with hotel selection (Vansteenwegen et al., 2012) can be modeled more appropriately by the OPHS. For example, the well-known traveling salesperson problem turns into an OPHS under (realistic) circumstances: if the traveling salesperson needs to select which of his possible clients he will actually visit during his multiple day tour and he also needs to select the most appropriate hotels to stay every night. Other examples are truck drivers with limited driving hours wanting to reach an appropriate parking space, routing maintenance technicians with several depots to pick up spare parts, etc.

In Section 2, the related problems are briefly reviewed. Then, the mathematical formulation for this new problem is presented in Section 3. In Section 4, the proposed algorithm is discussed in detail. How benchmark instances are generated is discussed in Section 5. Experimental results are presented in Section 6 and the paper is concluded in Section 7.

2. Literature review

The OPHS is closely related to the regular orienteering problem (OP), the team orienteering problem (TOP), and the traveling salesperson problem with hotel selection (TSPHS). Since the OPHS is a new problem, we will explain the similarities and differences with each of these problems. In the regular OP (Tsiligirides, 1984) a set of N vertices i is given, each with a score S_i. The initial and final vertices are fixed. The time t_{ij} needed to travel from vertex i to j is given for all vertices. Because of the time limitation (T_{max}), not all vertices can be visited. The goal is to find a single route with maximum score, respecting the time limitation T_{max}. Each vertex can be visited at most once. The OP is also known as the selective traveling salesperson problem (Gendreau et al., 1998), the maximum collection problem (Butt and Cavalier,1994) and the bank robber problem (Arkin et al.,1998). Moreover, the OP can be formulated as a special case of the resource constrained TSP, a TSP with profits or as a resource constrained elementary shortest path problem (Vansteenwegen et al., 2009a). A number of challenging practical applications were modeled as orienteering problems and many exact and heuristic solution approaches were developed to solve this problem (Vansteenwegen et al., 2011).

When there are several routes allowed to visit vertices and all of them start and end at the same vertex, we are dealing with a team orienteering problem (TOP) (Chao et al.,1996a). A comprehensive survey about the (T)OP can also be found in Vansteenwegen et al. (2011).

Although the OP and the TOP have been studied in numerous papers, the OPHS has not been considered before. The OPHS has more than one trip and from this point of view it is comparable to the TOP. The main difference between the TOP and the OPHS is that in the TOP all trips have to start and end in the same vertex and no hotels need to be selected.

A recent publication by Vansteenwegen et al. (2012) discusses the traveling salesperson problem with hotel selection (TSPHS). In the TSPHS, a number of hotels are available as well as a number of vertices. Each vertex is assigned a visiting time, and the required times to travel between all pairs of vertices are known. The available time that each trip takes is limited to a time budget, and the goal is to first minimize the number of connected trips and then minimize the total length of the tour (Vansteenwegen et al., 2012), while visiting all vertices. Two obvious differences with the OPHS are that in the TSPHS all the vertices have to be visited and that the objective is to minimize the number of trips and the total travel time. For a discussion about the similarities and differences between the TSPHS and other routing problems (multiple TSP, multi-depot VRP, location routing problems, etc.), we refer to the literature review in Vansteenwegen et al. (2012).

3. Mathematical formulation

Making use of the notation in the first section, the OPHS can be formulated as a mixed-integer linear problem ($x_{i,j,d}=1$ if, in trip d, a visit to vertex i is followed by a visit to vertex j, 0 otherwise; $u_i=$ the position of vertex i in the tour).

$$\text{Max} \sum_{d=1}^{D}\sum_{i=0}^{H+N}\sum_{j=0}^{H+N} S_i x_{i,j,d}$$

$$s.t. \tag{0}$$

$$\sum_{l=1}^{H+N} x_{0,l,1} = 1 \tag{1}$$

$$\sum_{k=0}^{H+N} x_{k,1,D} = 1 \tag{2}$$

$$\sum_{h=0}^{H}\sum_{l=0}^{H+N} x_{h,l,d} = 1 \quad d=1,\ldots,D \tag{3}$$

$$\sum_{h=0}^{H}\sum_{k=0}^{H+N} x_{k,h,d} = 1 \quad d=1,\ldots,D \tag{4}$$

$$\sum_{k=0}^{H+N} x_{k,h,d} = \sum_{l=0}^{H+N} x_{h,l,d+1} \quad d=1,\ldots,D-1 \quad h=0,\ldots,H \tag{5}$$

$$\sum_{i=0}^{H+N} x_{i,k,d} = \sum_{j=0}^{H+N} x_{k,j,d} \quad k=H+1,\ldots,H+N \quad d=1,\ldots,D \tag{6}$$

$$\sum_{d=1}^{D}\sum_{j=0}^{H+N} x_{i,j,d} \le 1 \quad i=H+1,\ldots,H+N \tag{7}$$

$$\sum_{i=0}^{H+N}\sum_{j=0}^{H+N} t_{i,j} x_{i,j,d} \le T_d \quad d=1,\ldots,D \tag{8}$$

$$u_i - u_j + 1 \le (N-1)\left(1-\sum_{d=1}^{D} x_{i,j,d}\right) \quad i=H+1,\ldots,H+N \quad j=H+1,\ldots,H+N \tag{9}$$

$$u_i \in 1,\ldots,N \quad i=H+1,\ldots,H+N \tag{10}$$

$$x_{i,j,d} \in \{0,1\} \quad \forall i,j=1,\ldots,H+N \,|\, i\neq j \quad d=1,\ldots,D \tag{11}$$

The objective function (0) maximizes the total collected score. Constraint (1) guarantees that the tour starts from the initial starting hotel ($i=0$). Constraint (2) guarantees that the tour ends at the final ending hotel ($i=1$). Constraints (3) and (4) ensure that each trip starts and ends in one of the available hotels. Constraints (5) verify that if a trip ends in a given hotel, the next trip starts in the same one. Constraints (6) determine the connectivity inside each trip. Constraints (7) ensure that every vertex is visited at most once. Constraints (8) limit each trip with a time budget. If visiting a vertex requires a certain service time, this time can easily be modeled as a part of all travel times to and/or from the vertex. Constraints (9) are sub-tour elimination constraints which are formulated based on the Miller–Tucker–Zemlin (MTZ) formulation of the TSP (Miller et al., 1960).

4. Proposed algorithm

In this section, a metaheuristic algorithm to solve the OPHS is described. The algorithm combines different moves using a modified variable neighborhood search framework (VNS).

The hotel selection is the most challenging part of the OPHS, because choosing a different hotel significantly affects the whole solution (tour). At the same time, it is very difficult to determine which are the promising (combinations of) hotels leading to a high scoring selection of vertices. This is also pointed out in the paper about the TSPHS (Vansteenwegen et al., 2012). Therefore, a lot of computational effort is assigned to considering alternative combinations of hotels. Also the initialization phase focuses mainly on the feasible combination of hotels.

4.1. General structure of the algorithm

The framework we use is called Skewed VNS (SVNS) (Hansen and Mladenović, 2001) and it is a variant of VNS that is often used for problems which have several separated and possibly far apart near-optimal solutions. This is the case for all variants of the OP.

VNS has been successfully applied for the OP (Sevkli and Sevilgen, 2006), the TOP (Archetti et al., 2006), the bi-objective OP (Schilde et al., 2009), the multi-period OP with multiple time windows (Tricoire et al., 2010) and for the TOP with time windows (Labadie et al., 2012). SVNS has been used successfully to solve TOP instances (Vansteenwegen et al., 2009b). Moreover, the SVNS has a simple scheme and only needs a few parameters. The modification compared to a regular VNS is that some of the solutions with a slightly worse score are also accepted. In order to better explore the search space far away from the incumbent, a solution could be accepted that is not necessarily as good as the best one known, provided that it is far from this solution. However, in our algorithm slightly worse solutions are accepted without considering their distance from the incumbent. An overview of the algorithm is presented in Fig. 1.

The algorithm starts with an initialization phase, explained in Section 4.2. Then, an improvement phase is implemented. The improvement phase starts with "shaking" the vertices in the solution (Vertices-shake), followed by applying *Local Search* (Section 4.4). Then, the hotels are "shaken" (Hotels-shake) and the same *Local Search* is applied. In the recentering phase (Section 4.5), it is decided which solution should be used to start the next iteration. *Kmax* limits the number of subsequent iterations without recentering. The number of subsequent iterations without an improvement is used as the stopping criterion of the algorithm (*NoImprovementMax*).

4.2. Initialization

This phase includes three successive steps resulting in an initial solution. In the first step, a matrix containing the potential score between every pairs of hotels is created. Then, the list of all feasible combinations of hotels is determined. After that, a number of feasible combinations of hotels is selected to construct the initial solution as well as to be used in the improvement phase. The number of combinations that is selected is set to the "Number of Used Feasible Combinations of hotels" (NUFC), a parameter of the algorithm that will be discussed in detail in Section 6.2. Finally, the initial solution is created.

```
Initialization:
    • Make the matrix of pairs of hotels
    • Construct the list of feasible combintions of hotels
    • Construct the  initial solution, X

Improvement:
While NoImprovement < NoImprovementMax do
        K <--- 1;
        While K<Kmax do
                X'<---Vertices-Shake(X);
                X"<---Local Search(X');
                X'<---Hotels-Shake(X");
                X"<---Local Search (X');
                Recenter or not:
                If X" is better than X then
                        Recenter X <--- X"; K <--- 1; NoImprovement <--- 0;
                Else
                        NoImprovement +1;
                        If X" is slightly worse than X then
                                Recenter : X <--- X"; K <--- 1;
                        Else
                                No Recenter:K+1;
                        End
                End
        End
End
```

Fig. 1. General structure of the SVNS algorithm.

```
While (Any of the moves can improve the solution) do
        Insert ;
        Replacement ;
        Two-Opt;
        Move-best;
End
```

Fig. 2. Greedy sub-OP heuristic.

Matrix of Pairs of Hotels: in this preprocessing function, using the time limitation of each trip, a sub-OP (an orienteering problem between two hotels) is solved heuristically between every pairs of hotels, including the situations where the starting and ending hotel of a trip are the same. As a result, a three dimensional matrix with potential scores between each pair of hotels and for each trip is calculated. Since the size of this matrix increases quadratically with the number of possible hotels, it is important that each sub-OP is solved very fast. Moreover, it is not worthwhile to invest much computation time in solving each sub-OP, since the results will only be used to get a first idea about the possible score of a certain pair of hotels for a certain trip. Every time we refer to solving a sub-OP heuristically in this paper, only a straightforward local search method is used. The structure of this greedy sub-OP heuristic is presented in Fig. 2.

Four different local search moves are implemented: *Insert*, *Replacement*, *Two-Opt* and *Move-best*. A single move is applied as long as improvements are found. Then, the next move is considered. These moves are applied, in a fixed order, as long as any of the moves can improve the solution.

List of feasible combinations of hotels: in this pre-processing function, all the feasible combinations of hotels for the whole tour are determined. To do this, a pruning rule considers the travel time to the final hotel and compares it with the remaining time of

the whole tour. If by selecting a hotel, it is not possible to reach the final hotel in the remaining available time of the tour, then this combination is infeasible. Obviously, if in the previous step (Matrix of pairs of hotels) a pair of hotels appears infeasible for a certain trip, all combinations of hotels for the whole tour including this pair are infeasible as well.

Next, a heuristic estimated score (HES) is calculated for each feasible combination, by adding the sub-OP score of each pair in the feasible combination for the whole tour. We call this a *heuristic estimated score*, since the whole tour might include some vertices more than once, if these vertices appear in more than one sub-OP solution used in the whole tour; and the sub-OPs were only solved heuristically and possibly better sub-OP solutions can be found. Finally, all feasible combinations of hotels are sorted in a list based on their HES.

The initial solution: a number of feasible combinations of hotels with the highest HES are selected to be used in the rest of the algorithm. The Number of Used Feasible Combinations of hotels (NUFC) is a parameter of the algorithm and will be discussed in detail in Section 6.2. To make the initial solution, the three following strategies are applied to this selected set of feasible combinations of hotels.

1. *Local Search* (Section 4.4) is applied on the empty combination of hotels (without any vertices between the hotels) to create a complete solution.
2. Starting from the first trip, a sub-OP is solved for each trip to create a feasible solution. Then, *Local Search* is applied to further improve the solution.
3. Starting from the last trip, a sub-OP is solved for each trip to create a feasible solution. Then, *Local Search* is applied to further improve the solution.

The difference between solving the sub-OPs in these strategies 2 and 3 and when making the Matrix of Pairs of Hotels is that here vertices included in previous trips are no longer considered for insertion. Therefore, in this case, the combination of sub-OP solutions for each trip results in a feasible solution for the whole tour.

For each combination of hotels, the total scores of these three achieved solutions are compared and the best solution is saved. Then, all saved solutions are sorted based on their total score. In the end, the solution with the highest score is selected as the initial solution.

We will show in the experimental results section that this initialization phase goes very fast and is crucial in order to obtain high quality results.

4.3. Shaking phase

The algorithm tries to find an optimal combination of hotels and, at the same time, looks for the best selection of vertices for a given combination of hotels. Therefore, the shaking phase also focuses on both aspects separately.

Vertices-shake: this step is about shaking the vertices in order to diversify the exploration for a given combination of hotels. Two different shakes are applied and the best result is used further in the algorithm:

1. The first half of the vertices in each trip are deleted and then the solution is improved again using *Local Search*.
2. The last half of the vertices in each trip are deleted and then the solution is improved again using *Local Search*.

Obviously, if the best found solution is improved by this shake (and improvement) step, the new best found solution is saved.

Hotels-shake: the only fixed hotels in an OPHS problem are the initial starting and the final ending hotel. So, if the tour consists of more than one trip, the remaining hotels can be chosen among all available hotels. In the hotels-shake part, the hotels in the current solution are replaced by one of the *NUFC* combinations of hotels with the highest HES. The vertices in each trip are not changed yet. In most cases, this step will lead to an infeasible solution. Then, one by one, the infeasibilities are removed by deleting vertices from the infeasible trips. The vertices are selected based on the least ratio of score over the time saved by a removal. After that, the algorithm tries to improve the solution using *Local Search*. *Kmax* (Fig. 1) determines the maximum number of alternative hotel combinations that are considered in the hotels-shake, for a given solution, without recentering.

As a result of the hotels-shake and the vertices-shake, each combination of hotels is considered for improvement with three different sets of vertices: the one resulting from the previous combination of hotels (Hotels-shake) and the two resulting from deleting in every trip the first or last half of the vertices (Vertices-shake).

4.4. Local Search

The *Local Search* within the proposed heuristic refers to a typical variable neighborhood structure with nine local search moves, presented in Fig. 3.

The local search moves are all developed to improve the selection of vertices for a given combination of hotels. Many different local search moves are applied in the literature to different variants of the orienteering problem (Vansteenwegen et al., 2011). Once the hotels are fixed, the OPHS reduces to a TOP (with a different start and end hotel for each trip). Therefore, the most relevant local search moves which have proven to be successful for the TOP (Vansteenwegen et al., 2011) are selected, modified and applied to OPHS. A brief explanation of each move is presented in this section.

Insert: For each non-included vertex, the position of insertion in the tour with minimum increase in trip length is found. Among the vertices for which the insertion does not make the trip infeasible, the vertex with maximum ratio of score over this increase is inserted.

Move-best: For each vertex in each trip, it is checked that if removing this vertex from its current position and putting it in its best possible position within the whole tour is feasible. Then this move is completed and the next vertex in the tour is considered for the move.

Two-Opt: It reduces the travel time in each trip. For each trip, two vertices are determined for which the inversion of the order of vertices between them, leads to the highest travel time saving in the trip. Then, the order of the vertices between them is reversed.

Swap-trips: It exchanges two vertices between two different trips to reduce the total travel time. For each pair of trips (d_1 and d_2) two vertices (one from each trip) are swapped only if the length of at least one of the trips decreases and both trips remain feasible. In order to determine this, four travel time changes are calculated: the amount of time saved in trip d_1 by excluding vertex i; the amount of time saved in trip d_2 by excluding vertex j; the best position in trip d_1 to include vertex j, and the best position in trip d_2 to include vertex i (least time consuming positions).

Extract-Insert: It starts from the first vertex in each trip and checks if by excluding this vertex and inserting as many vertices as possible in the trip, the score can be increased. If that is the case, this exchange is applied and the algorithm goes to the next vertex in the trip. Inserting the vertices is done by *Insert*,

```
Local Search
Set of local search moves (N_Level):
Insert,Move-best,Two-Opt,Swap-trips,Extract-Insert, Extract2-Insert
Extract5-Insert, Extract-Move-Insert
Level <--- 1;
While Level <Level_Max do
      Find best neighbour in N_Level(X')====> X"
      If X" is better than X' then
        X' <---X"; Level <---1 ;
      Else
        Level+1;
      End
End
```

Fig. 3. *Local Search* structure.

described above, but the excluded vertex is not considered again for insertion.

Extract2-Insert: It is very similar to the previous move. The only difference is that every time two consecutive vertices are considered for exclusion.

Extract5-Insert: It is similar to the previous moves, but five consecutive vertices in a trip are removed. This move is only considered for trips with at least five vertices. The new neighbor is only accepted if the total score is improved. As soon as a new neighbor is accepted, the neighborhood search is terminated. In the survey paper of Vansteenwegen et al. (2011) this move is called "Change".

Extract-Move-Insert: It is a combination of three different moves. Starting from the first vertex in the first trip, each vertex ($v1$) is excluded from the trip one by one. Each time, it is checked then if any of the other vertices in the tour can be moved to a different position ($v2$). Every time this is possible, all non-included vertices ($v3$) are considered, one by one, for insertion in the trip vertex $v2$ was moved from. The whole move is applied only if the inserted vertex ($v3$) has a higher score than the excluded vertex ($v1$).

Replacement: For each trip, the following steps are performed. All non-included vertices are considered for insertion. For every vertex, it is checked what would be the least time consuming position in the trip to include that vertex. If this insertion is feasible, the vertex is inserted. If the remaining budget is insufficient, all included vertices with a lower score are considered for exclusion. If excluding one of these vertices from the trip creates enough time to insert the non-included vertex under consideration, a replacement will be made.

4.5. Recentering phase

Each time a different combination of hotels is selected and *Local Search* is finished, the algorithm decides which solution will be used to start the next iteration: the current one (after Hotels-shake+improvement) or the one that was used to start the current iteration (before Vertices-shake). In this case, we decided that even if the current solution is slightly worse than the previous one, it is still used as the new solution to start the next iteration. This helps to diversify the search and to explore other parts of the solution space. We defined that a solution is only slightly worse than another one if the relative ratio of their scores is less than *MaxPercentageWorse*. *MaxPercentageWorse* is a parameter of the algorithm and is discussed in Section 6.2.

5. Test instances

Since no test instances are available for the OPHS, four different sets of benchmark instances are designed to test the performance of the new algorithm. These instances are available at ⟨www.mech.kuleuven.be/cib/op⟩.

For the first two sets, we generate instances of varying size based on benchmark instances for the orienteering problem with known optimal solution (Tsiligirides, 1984; Chao et al., 1996b). The instances are generated in such a way that the optimal OP solutions are also optimal solutions for the OPHS. First of all, the total tour length of the optimal solution is divided by the number of trips, D, we plan to have. The result (T_f) is a first indication of the time budget T_d of each trip and will be used to split up the optimal OP tour (Fig. 4a). Now, based on the optimal tour, extra hotels are added to the OP instance in the following way. The starting vertex of the optimal tour is used as the initial starting hotel and the ending vertex as the final ending hotel of the OPHS instance. Then, we go through the optimal OP tour and sum over the time between the vertices one by one. As soon as the cumulative time exceeds T_f, the first extra hotel is put exactly on the same location as the vertex under consideration (Fig. 4b). In this way, both the original vertex and the hotel are present in the OPHS instance. The cumulative time to reach this hotel is used as the time budget (T_1) of the first trip. This process is repeated until the final ending hotel is reached. As a result, T_1 until T_{D-1} will be larger than T_f, T_D will be smaller than T_f and the sum of all T_d will be equal to the tour length of the optimal OP solution (Fig. 4c).

Starting from 35 different OP instances, 105 OPHS instances have been designed, using the above mentioned methodology with the number of trips equal to 2, 3 and 4. In these instances the number of extra hotels is exactly one number less than the number of trips. This set of 105 instances is called SET1. The number of vertices is between 32 and 102 and the tour time budget varies from 30 to 130.

For SET2, a number of extra hotels are added at locations of other vertices of the instances of SET1, in order to make these instances more complicated, although the same optimal solution is still valid. These extra locations are selected among vertices with a multiple of $\lfloor N/h+1 \rfloor$ where N is the number of vertices and h is the number of extra hotels in total. As a result, for the instances of SET2, the number of extra hotels is higher than the number of trips. So, SET2 includes 35 instances with 3 trips and 3 more extra hotels (in total 5 extra hotels) and 35 instances with 4 trips and 3 more extra hotels (in total 6 extra hotels).

For SET3, the same technique as for SET1 and SET2 is applied to create more difficult OPHS instances based on longer optimal solutions for OP instances with 64 and 100 vertices and a tour time budget between 50 and 240. 22 OPHS instances with 4 trips and 10 extra hotels and 22 instances with 5 trips and 12 extra hotels are designed.

The key point in making these three sets of OPHS instances based on optimal OP solutions is that it is impossible to come up with a tour with a higher score than the optimal OP tour. Since all the hotels are on the same location of existing vertices, an OPHS solution with a higher score than the optimal OP tour (but a tour length limited to the tour length of the optimal OP tour) would also be a better solution for the original OP instance.

We tried to solve the OPHS instances with six extra hotels and four trips of SET2 with CPLEX. 16 of the 35 instances were solved to optimality within 1 hour of computation per instance (these instances are indicated in Table 7 using a "*"). This illustrates that MIP solvers as CPLEX cannot solve OPHS instances of larger (or realistic) size. It is also an indication of the complexity of the problem and it was the reason we decided to modify OP-benchmark instances in such a way that the optimal OPHS solution is known (based on the optimal OP solution).

SET4 contains large OPHS instances where no optimal solution is known in advance. Starting from large OP instances, 3 extra

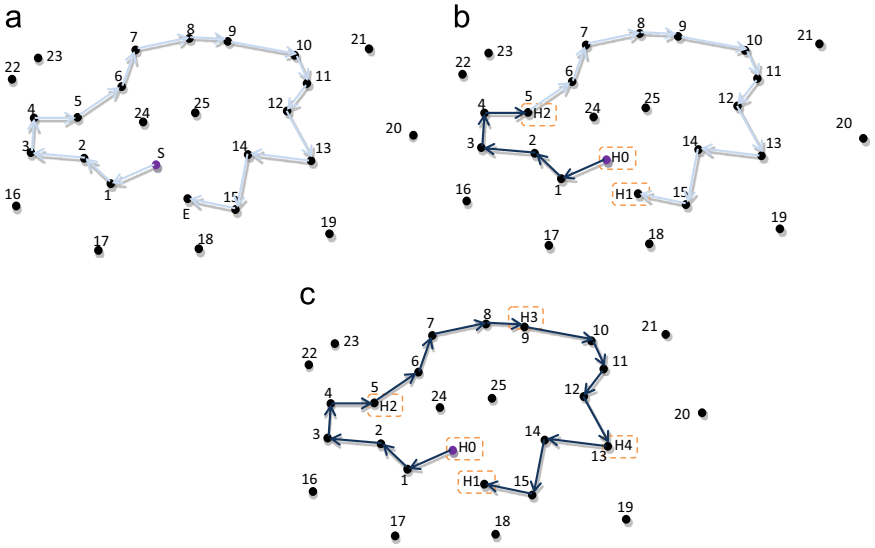


Fig. 4. (a) OP optimal tour. Total tour length=40; $D=4$; $T_f=10$; (b) by reaching the 5th node, the cumulative time has exceeded the T_f; $T_1=10.7>T_f=10$; (c) OPHS optimal tour; $T_1=10.7$; $T_2=11.2$; $T_3=10.3$; $T_4=7.8$.

hotels are added on the location of some randomly selected vertices. In SET4, the original vertices on these locations are removed. An equal time limit T_d is imposed on every trip. For this set of instances, CPLEX is used to optimally solve the sub-OP for each possible pair of hotels. (This is not a part of the proposed algorithm to solve the OPHS, but only to make new benchmark instances.) Obviously, the optimal sub-OP solution between, for instance, the first and the second extra hotel, might include one or more of the vertices included in the sub-OP solution between the second and the third extra hotel. This implies that different sub-OP solutions cannot just be added to obtain an OPHS solution. However, by just adding up the ideal combination of sub-OP solutions, an upper bound for the OPHS solution can be determined. If no vertices are included twice in this OPHS solution, it is also the optimal OPHS solution. If some vertices are included more than once, these are simply removed until only one visit to each of these vertices is left. This results in a feasible OPHS solution with the highest known value, but this is not necessarily the optimal solution.

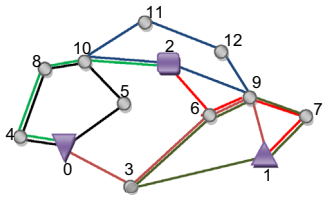
A simplified example of this method is presented in Fig. 5 and Tables 1 and 2. The coefficient of each vertex is also used as its score and all optimal sub-OP solutions are presented with their score. Table 1 shows the instances for $D=2$. In this case, the highest upper bound does not contain any duplicated node. So, the upper bound is the optimal solution for this instance. Table 2 presents the details on making an instance for $D=3$ from the same example of Fig. 5. In this case, the highest upper bound does contain repeated nodes. Then, we can only have a best known feasible solution.

For SET4, OP instances with 100 and 102 vertices are used (Chao et al., 1996b). For the instances with 100 vertices, time budgets equal to 20 and 25 (for each trip) are applied. For the instances with 102 vertices, trip time budgets of 35, 40 and 45 are applied. By considering these instances with 2 and 3 trips, 10 large OPHS instances were created.

6. Experiments and results

6.1. Discussion

The algorithm is coded in Visual C++ 2010. All computations were carried out on a personal computer Intel Core 2 with 3.00 GHz processor and 4.00 GB Ram.



			Final hotel		
			0	1	2
Initial hotel	0	Optimal Sub-OP solution	0-4-8-10-5-0	0-3-6-9-1	0-4-8-10-2
		Optimal Sub-OP score	**27**	**18**	**22**
	1	Optimal Sub-OP solution	1-9-6-3-0	1-3-6-9-7-1	1-7-9-6-2
		Optimal Sub-OP score	**18**	**25**	**22**
	2	Optimal Sub-OP solution	2-10-8-4-0	2-6-9-7-1	2-10-11-12-9-2
		Optimal Sub-OP score	**22**	**22**	**42**

Fig. 5. A simplified example on how SET4 instances have been made.

Table 1
Making OPHS instance with $H=3$ and $D=2$.

Hotels' order in 2-days tour	Upper bound	Duplicated nodes	Tour after removing duplicated nodes
0-0-1	45	–	**0**-4-8-10-5-**0**-3-6-9-**1**
0-1-1	43	3-6-9	**0**-6-9-**1**-3-7-**1**
0-2-1	44	–	**0**-4-8-10-**2**-6-9-7-**1**

Optimal solution: **0**-4-8-10-5-**0**-3-6-9-**1**; **Optimal value**=45.

The results for SET1 are summarized in Tables 3–5, depending on the number of trips in a tour. The results for SET2 and SET3 are presented in Tables 6–9 in the same way. The first column in each of these tables gives the OP instance's name which includes the time budget for the whole tour. Then, the optimal solution of each instance is given in the second column. In the third column, TNFC, the total number of feasible combinations of hotels is presented. This number is an indication of the difficulty of finding a good or optimal combination of hotels. For our SVNS results, the gap ((*Optimal result—SVNS result*)/*Optimal result*) is presented as a percentage and the CPU shows the computation time in seconds.

For SET1, SET2 and SET3 all results are compared with the optimal solutions. For SET4, in case that there is no optimal

Table 2
Making OPHS instance with $H=3$ and $D=3$.

Hotels' order in 3-days tour	Upper bound	Tour after removing duplicated nodes	
		Tour	Score
0-0-0-1	72	**0**-4-8-10-5-**0**-**0**-3-6-9-**1**	45
0-0-1-1	70	**0**-4-8-10-5-**0**-3-6-9-**1**-7-**1**	52
0-0-2-1	71	**0**-4-8-10-5-**0**-**2**-6-9-7-**1**	49
0-1-0-1	54	**0**-3-6-9-**1**-**0**-**1**	18
0-1-1-1	68	**0**-3-6-9-**1**-7-**1**-**1**	25
0-1-2-1	62	**0**-3-6-9-**1**-7-**2**-**1**	25
0-2-0-1	62	**0**-4-8-10-**2**-**0**-3-6-9-**1**	40
0-2-1-1	69	**0**-4-8-10-**2**-6-9-7-**1**-3-**1**	47
0-2-2-1	86	**0**-4-8-10-**2**-11-12-9-**2**-6-7-**1**	67

Upper bound: **0**-4-8-10-**2**-10-11-12-9-**2**-6-9-7-**1**; **Value**=86;
Best known feasible: **0**-4-8-10-**2**-11-12-9-**2**-6-7-**1**; **Value**=67.

Table 3
SET1: 1 extra hotel – 2 trips.

Instances	Opt	TNFC[a]	SVNS	Gap (%)	CPU (s)
T1_65	240	3	**240**	**0.00**	0.05
T1_70	260	3	**260**	**0.00**	0.06
T1_73	265	3	245	7.55	0.04
T1_75	270	3	**270**	**0.00**	0.04
T1_80	280	3	270	3.57	0.05
T1_85	285	3	280	1.75	0.05
T3_65	610	3	**610**	**0.00**	0.05
T3_75	670	3	650	2.99	0.05
T3_80	710	3	690	2.82	0.06
T3_85	740	3	**740**	**0.00**	0.05
T3_90	770	3	**770**	**0.00**	0.06
T3_95	790	3	780	1.27	0.04
T3_100	800	3	770	3.75	0.05
T3_105	800	3	**800**	**0.00**	0.04
64_45	816	3	**816**	**0.00**	0.30
64_50	900	3	876	2.67	0.43
64_55	984	3	972	1.22	0.38
64_60	1062	3	1050	1.13	0.43
64_65	1116	3	**1116**	**0.00**	0.47
64_70	1188	3	1170	1.52	0.37
64_75	1236	3	**1236**	**0.00**	0.44
64_80	1284	3	**1284**	**0.00**	0.36
66_40	575	3	570	0.87	0.15
66_45	650	3	645	0.77	0.19
66_50	730	3	715	2.05	0.20
66_55	825	3	805	2.42	0.23
66_60	915	3	860	6.01	0.33
66_125	1670	3	1665	0.30	0.32
66_130	1680	3	**1680**	**0.00**	0.36
100_30	173	2	**173**	**0.00**	0.06
100_35	241	1	**241**	**0.00**	0.10
100_40	299	2	**299**	**0.00**	0.19
100_45	367	3	**367**	**0.00**	0.21
102_50	181	3	**181**	**0.00**	0.03
102_60	243	3	**243**	**0.00**	0.08
Average				**1.22**	**0.18**

[a] TNFC: Total number of feasible combinations.

Table 4
SET1: 2 extra hotels – 3 trips.

Instances	Opt	TNFC	SVNS	Gap (%)	CPU (s)
T1_65	240	16	**240**	**0.00**	0.08
T1_70	260	16	**260**	**0.00**	0.07
T1_73	265	16	**265**	**0.00**	0.07
T1_75	270	16	**270**	**0.00**	0.09
T1_80	280	16	**280**	**0.00**	0.07
T1_85	285	16	**285**	**0.00**	0.08
T3_65	610	16	**610**	**0.00**	0.07
T3_75	670	16	650	2.99	0.07
T3_80	710	16	**710**	**0.00**	0.07
T3_85	740	16	**740**	**0.00**	0.11
T3_90	770	16	730	5.19	0.08
T3_95	790	16	**790**	**0.00**	0.08
T3_100	800	16	**800**	**0.00**	0.08
T3_105	800	16	790	1.25	0.08
64_45	816	12	**816**	**0.00**	0.30
64_50	900	16	876	2.67	0.37
64_55	984	16	954	3.05	0.41
64_60	1062	16	1038	2.26	0.68
64_65	1116	16	1092	2.15	0.49
64_70	1188	16	1170	1.52	0.56
64_75	1236	16	1212	1.94	0.53
64_80	1284	16	1260	1.87	0.53
66_40	575	16	570	0.87	0.12
66_45	650	16	645	0.77	0.15
66_50	730	16	715	2.05	0.24
66_55	825	16	805	2.42	0.26
66_60	915	16	910	0.55	0.31
66_125	1670	16	**1670**	**0.00**	0.55
66_130	1680	16	1665	0.89	0.46
100_30	173	1	**173**	**0.00**	0.04
100_35	241	2	**241**	**0.00**	0.08
100_40	299	2	**299**	**0.00**	0.07
100_45	367	2	**367**	**0.00**	0.11
102_50	181	12	**181**	**0.00**	0.03
102_60	243	12	**243**	**0.00**	0.08
Average				**0.93**	**0.21**

solution, the results are compared with both the best known feasible solution and the (infeasible) upper bound. The results for SET4 are presented in Table 10. The first column gives the name of the instance, the time limitation for each trip, and the number of trips. Then, the best known feasible solution and the upper bound for the instances without known optimal solution are given in the second and third columns, respectively. In the next column, the optimal solution is presented, if known. The sixth column (SVNS) shows the results obtained by our algorithm. The next three columns are presenting the gap in percentage with the best known feasible solution, with the upper bound and with the optimal value, if applicable. The last column presents the computation time in seconds. The last row in all these tables presents the average gap and computation time.

In total, for SET1, in 55 instances out of 105 the optimal solution is found in less than three seconds. The average gap is

Table 5
SET1: 3 extra hotels – 4 trips.

Instances	Opt	TNFC	SVNS	Gap (%)	CPU (s)
T1_65	240	107	**240**	**0.00**	0.16
T1_70	260	107	**260**	**0.00**	0.19
T1_73	265	107	**265**	**0.00**	0.18
T1_75	270	107	**270**	**0.00**	0.20
T1_80	280	125	275	1.79	0.27
T1_85	285	125	**285**	**0.00**	0.25
T3_65	610	100	**610**	**0.00**	0.18
T3_75	670	125	650	2.99	0.25
T3_80	710	107	**710**	**0.00**	0.23
T3_85	740	107	**740**	**0.00**	0.23
T3_90	770	100	**770**	**0.00**	0.33
T3_95	790	100	**790**	**0.00**	0.24
T3_100	800	125	760	5.00	0.31
T3_105	800	125	**800**	**0.00**	0.30
64_45	816	44	**816**	**0.00**	0.29
64_50	900	72	870	3.33	0.96
64_55	984	100	978	0.61	1.54
64_60	1062	100	1038	2.26	1.20
64_65	1116	125	1104	1.08	1.65
64_70	1188	125	1170	1.52	2.69
64_75	1236	125	1200	2.91	1.94
64_80	1284	125	1266	1.40	2.16
66_40	575	107	570	0.87	0.26
66_45	650	88	645	0.77	0.30
66_50	730	107	715	2.05	0.50
66_55	825	107	805	2.42	0.59
66_60	915	107	910	0.55	0.70
66_125	1670	125	1635	2.10	2.19
66_130	1680	125	1670	0.60	2.14
100_30	173	2	**173**	**0.00**	0.04
100_35	241	1	**241**	**0.00**	0.04
100_40	299	1	**299**	**0.00**	0.06
100_45	367	2	**367**	**0.00**	0.08
102_50	181	33	**181**	**0.00**	0.05
102_60	243	60	**243**	**0.00**	0.08
Average				**0.92**	**0.65**

Table 6
SET2: 5 extra hotels – 3 trips.

Instances	Opt	TNFC	SVNS	Gap (%)	CPU (s)
T1_65	240	49	**240**	**0.00**	0.10
T1_70	260	49	**260**	**0.00**	0.11
T1_73	265	49	**265**	**0.00**	0.11
T1_75	270	49	**270**	**0.00**	0.16
T1_80	280	49	**280**	**0.00**	0.11
T1_85	285	49	**285**	**0.00**	0.11
T3_65	610	49	**610**	**0.00**	0.10
T3_75	670	49	650	2.99	0.11
T3_80	710	49	**710**	**0.00**	0.12
T3_85	740	49	710	4.05	0.12
T3_90	770	49	740	3.90	0.12
T3_95	790	49	**790**	**0.00**	0.12
T3_100	800	49	**800**	**0.00**	0.12
T3_105	800	49	**800**	**0.00**	0.12
64_45	816	42	**816**	**0.00**	0.42
64_50	900	49	870	3.33	0.53
64_55	984	49	954	3.05	0.64
64_60	1062	49	1056	0.56	0.73
64_65	1116	49	1092	2.15	0.82
64_70	1188	49	1170	1.52	0.91
64_75	1236	49	1218	1.46	0.92
64_80	1284	49	1260	1.87	0.90
66_40	575	45	570	0.87	0.18
66_45	650	47	645	0.77	0.28
66_50	730	49	715	2.05	0.35
66_55	825	49	**825**	**0.00**	0.66
66_60	915	49	890	2.73	0.80
66_125	1670	49	1655	0.90	0.98
66_130	1680	49	1675	0.30	0.80
100_30	173	1	**173**	**0.00**	0.04
100_35	241	2	**241**	**0.00**	0.09
100_40	299	2	**299**	**0.00**	0.08
100_45	367	2	**367**	**0.00**	0.13
102_50	181	14	**181**	**0.00**	0.03
102_60	243	17	**243**	**0.00**	0.08
Average				**0.93**	**0.34**

only 1.02% and the average computation time is 0.35 s. For SET2, this is 36 out of 70, an average gap of 1.07% and 0.87 s, for SET3, 7 out of 44, an average gap of 3.10% and 7.49 s and for the instances of SET4 with known optimal solutions, 4 out of 5, an average gap of 0.76% and 0.17 s. For all other instances of SET4, the best found feasible solution is improved. Over all 224 instances with known optimal solutions, 102 optimal solutions are found with an average gap of 1.44% and an average computation time of 1.91 s.

In Table 3, one somewhat larger gap appears of 7.55%. This is surprising since in Tables 4 and 5 the same instance with more extra hotels is solved to optimality. In order to explain the difference in these results, it is useful to look at how the instances were constructed. The same optimal solution of the OP instance (T1_73) is used to create different OPHS instances. As a result, the time budget per trip for the instance with two trips is higher than for instance with three or four trips. This implies that the sub-OPs of the instance with two trips is more difficult to solve than with three or four trips, since more combinations of vertices should be considered and evaluated.

In the three last columns of Table 9, we also present the results obtained by the algorithm when all (instead of only the *NUFC* best) feasible combinations of hotels are considered. In these columns, the obtained total score is presented in the column named AFC. Then, the gap with the optimal solution and the CPU time are presented in the next columns. Comparing the achieved SVNS results with the AFC illustrates the importance of selecting a good combination of hotels on obtaining high quality results. For almost all instances in Table 9, the optimal or a very good solution is found. For two instances, however, the SVNS gap with the optimal solution remains around 8%. When looking into these instances, it appears that for both instances the optimal combination of hotels (required to obtain the optimal solution) is not one of the *NUFC* combinations selected based on the heuristic estimated score. This is confirmed by the AFC results, indicating that for the same instances high quality results are found when all combinations of hotels are considered in the algorithm. This clearly illustrates the importance (and difficulty) of selecting a good combination of hotels. This will hold for any OPHS algorithm: if the optimal combination of hotels is not somehow considered during the search, the quality of the obtained results decreases significantly. For our algorithm, this means that if the algorithm is able to find the optimal combination of hotels, high quality solutions are obtained, with an average gap of only 1.84%. If the optimal combination of hotels is not considered, still the gap is always smaller than 8.68% and the gap is only 3.58% on average.

The strength of our algorithm is that it is able to determine the optimal combination of hotels for 163 of the 224 instances with known optimal solutions (including five instances of SET4). Of all 97 instances with at least 100 feasible combinations of hotels and known optimal solutions, the optimal combination of hotels is found 53 times. This is 28 out of 69 instances with at least 250 feasible combinations of hotels. 206 times a very good combination of hotels is found, leading to solutions with a gap of less than 5%. This is important since, for larger instances, it is not reasonable to check all the feasible combinations of hotels due to the limited computation time in practical applications.

We can conclude that two instances in Table 3 (T1_73 and 66_60) are more difficult to solve from the sub-OP point of view

Table 7
SET2: 6 extra hotels – 4 trips.

Instances	Opt	TNFC	SVNS	Gap (%)	CPU (s)
T1_65*	240	482	**240**	**0.00**	0.43
T1_70*	260	482	**260**	**0.00**	0.45
T1_73*	265	482	**265**	**0.00**	0.46
T1_75*	270	482	**270**	**0.00**	0.48
T1_80*	280	512	**280**	**0.00**	0.52
T1_85	285	512	**285**	**0.00**	0.51
T3_65	610	448	**610**	**0.00**	0.48
T3_75*	670	512	650	2.99	0.51
T3_80	710	482	**710**	**0.00**	0.53
T3_85	740	482	**740**	**0.00**	0.56
T3_90*	770	448	730	5.19	0.56
T3_95*	790	448	750	5.06	0.59
T3_100	800	512	760	5.00	0.59
T3_105*	800	512	**800**	**0.00**	0.61
64_45*	816	252	792	2.94	1.71
64_50	900	378	870	3.33	2.28
64_55	984	448	972	1.22	2.94
64_60	1062	448	1044	1.69	3.08
64_65	1116	512	**1116**	**0.00**	3.57
64_70	1188	512	1164	2.02	3.84
64_75	1236	512	1200	2.91	4.03
64_80	1284	512	1266	1.40	4.10
66_40	575	302	570	0.87	0.65
66_45	650	228	645	0.77	0.83
66_50	730	357	715	2.05	1.21
66_55	825	400	805	2.42	1.42
166_60	915	424	910	0.55	1.77
66_125	1670	512	1645	1.50	5.01
66_130	1680	512	1670	0.60	4.49
100_30*	173	2	**173**	**0.00**	0.04
100_35*	241	1	**241**	**0.00**	0.04
100_40*	299	1	**299**	**0.00**	0.07
100_45*	367	2	**367**	**0.00**	0.08
102_50*	181	60	**181**	**0.00**	0.06
102_60*	243	105	**243**	**0.00**	0.15
Average				**1.21**	**1.39**

Table 8
SET3: 10 extra hotels – 4 trips.

Instances	Opt	TNFC	SVNS	Gap (%)	CPU (s)
64_75	1236	1728	1224	0.97	3.65
64_80	1284	1728	1278	0.47	4.02
66_125	1670	1728	**1670**	**0.00**	4.91
66_130	1680	1728	1675	0.30	4.66
100_50	412	39	408	0.97	0.32
100_60	504	161	**504**	**0.00**	1.25
100_70	590	276	575	2.54	2.90
100_80	652	892	641	1.69	4.03
100_90	725	1220	706	2.62	5.02
100_100	782	1296	766	2.05	5.95
100_110	835	1728	**835**	**0.00**	7.36
100_120	894	1728	886	0.89	7.95
100_130	956	1551	909	4.92	9.23
100_140	1013	1551	954	5.82	10.11
100_150	1057	1728	1042	1.42	10.73
100_160	1114	1728	1023	8.17	12.30
100_170	1164	1728	1077	7.47	13.08
100_180	1201	1728	1142	4.91	14.12
100_190	1234	1728	1176	4.70	14.13
100_200	1261	1728	1220	3.25	14.70
100_210	1284	1728	1235	3.82	15.31
100_240	1306	1728	1299	0.54	14.77
Average				**2.61**	**8.20**

Table 9
SET3: 12 extra hotels – 5 trips.

Instances	Opt	TNFC	SVNS	Gap (%)	CPU (s)	AFC	Gap (%)	CPU (s)
64_75	1236	32928	1212	1.94	3.13	1218	1.46	451.94
64_80	1284	32928	1260	1.87	3.63	1278	0.47	480.71
66_125	1670	38416	1645	1.50	4.71	1655	0.90	714.92
66_130	1680	38416	1670	0.60	4.36	1675	0.30	4958.54
100_50	412	26	393	4.61	0.15	393	4.61	1.33
100_60	504	442	**504**	**0.00**	1.58	**504**	**0.00**	22.09
100_70	590	1273	**590**	**0.00**	2.35	**590**	**0.00**	51.41
100_80	652	3053	**652**	**0.00**	2.92	**652**	**0.00**	202.84
100_90	725	5240	**725**	**0.00**	4.99	**725**	**0.00**	536.37
100_100	782	15272	766	2.05	4.89	766	2.05	1779.13
100_110	835	10991	807	3.35	5.71	828	0.84	620.85
100_120	894	15423	832	6.94	6.70	886	0.89	376.03
100_130	956	24210	897	6.17	6.75	943	1.36	684.42
100_140	1013	27440	954	5.82	8.32	958	5.43	869.45
100_150	1057	38416	970	8.23	9.09	1042	1.42	1362.15
100_160	1114	35672	1044	6.28	9.38	1052	5.57	1391.28
100_170	1164	38416	1063	8.68	10.18	1119	3.87	1626.33
100_180	1201	38416	1117	6.99	11.30	1156	3.75	1922.50
100_190	1234	38416	1170	5.19	11.51	1182	4.21	2486.98
100_200	1261	38416	1225	2.85	11.39	1248	1.03	2582.64
100_210	1284	38416	1242	3.27	12.34	1258	2.02	2664.83
100_240	1306	38416	1275	2.37	13.54	1303	0.23	2120.87
Average				**3.58**	**6.77**		**1.84**	**1268.53**

and some of the instances in Table 9 are more difficult to solve from the point of view of finding a good or optimal combination of hotels. With a maximum gap of 8.68% and an average gap smaller than 2% over 224 instances with known optimal solutions, the algorithm clearly performs outstandingly, certainly when the short calculation time is considered: a maximum of 15.31 s and 1.91 s on average.

6.2. Parameter sensitivity

For each parameter, a limited number of values is considered: *NoImprovementMax*=50, 100 or 200; *NUFC*=50, 100, 150, 200, 250 or 300; *MaxPercentageWorse*=0, 0.1, 0.2 or 0.3; and *Kmax*=0.25, 0.5, 1 or 2 times *NUFC*. It should be noted that when the Total Number of Feasible Combinations (TNFC) is smaller than *NUFC*, *NUFC* is set equal to TNFC.

Based on a set of 28 instances, selected from all sets, all possible combinations of these parameter values were compared, looking for a good trade-off between high quality results and small computational times. The following parameter values gave the best result and were selected for the computational experiments: *NoImprovementMax*=50, *NUFC*=250, *MaxPercentageWorse*=0.3, and *Kmax*=0.25 times the number of considered combinations of hotels ($0.25 \times NUFC$).

In our opinion, it is necessary to evaluate whether it is important that these exact parameter values are used or that small changes to these values do not significantly change the performance of the algorithm. In this section, the influence of the different parameters on the performance of the algorithm is examined, based on the instances of Table 9.

Four parameters are changed one by one, to verify the influence on the performance. The results of this sensitivity analysis are reported in Table 11. For each parameter, the SVNS column indicates the values used in the experimental results above and the obtained average gap and CPU time. In the state 1 and state 2 columns the average gap and CPU time are given when only this parameter is changed to an alternative value.

This analysis indicates satisfying results for the parameters we used. Obviously, using higher amounts for *NoImprovementMax*, and *NUFC* increases the computation time. The most important conclusion from this sensitivity analysis is that small changes in the parameter settings do not cause significant changes in the performance of the algorithm.

Table 10
SET4: 3 extra hotels – 2 trips/3 trips.

Name	Best feasible	Upper bound	Optimal value	TNFC	SVNS	Gap (%) w BF	Gap (%) w UB	Gap (%) w OV	CPU (s)
100_20_2	–	–	247	4	**247**	–	–	**0.00**	0.25
100_25_2	–	–	385	5	**385**	–	–	**0.00**	0.31
102_35_2	–	–	157	2	151	–	–	3.82	0.06
102_40_2	–	–	210	2	**210**	–	–	**0.00**	0.10
102_45_2	–	–	266	2	**266**	–	–	**0.00**	0.16
100_20_3	357	376	–	19	368	−3.08	2.13	–	0.33
100_25_3	495	568	–	25	524	−5.86	7.75	–	0.57
102_35_3	230	380	–	7	324	−40.87	14.74	–	0.13
102_40_3	299	493	–	7	383	−28.09	22.31	–	0.22
102_45_3	356	579	–	7	442	−24.16	23.66	–	0.28
Average						**−20.41**	**14.12**	**0.76**	**0.24**

Table 11
Sensitivity analysis.

Parameters	SVNS			State 1			State 2		
	Parameter value	Average Gap (%)	Average CPU time	**Parameter value**	Average Gap (%)	Average CPU time	**Parameter value**	Average Gap (%)	Average CPU time
NoImprovementMax	**50**	3.58	6.31	**25**	3.58	6.24	**100**	3.58	6.52
NUFC[a]	**250**	3.58	6.31	**150**	4.13	3.83	**350**	3.58	9.54
Kmax	***NUFC***/4	3.58	6.31	***NUFC***/8	3.58	7.00	***NUFC***/2	3.58	8.33
MaxPercentageWorse	**30**%	3.58	6.31	**20**%	3.58	6.93	**40**%	3.58	6.38

[a] *NUFC*: Number of used feasible combinations of hotels.

Table 12
Effectiveness of the initialization phase.

		Average gap (%)	Maximal gap (%)	Average CPU (s)
SET1	**SVNS**	1.02	7.55	0.35
	Initial	1.86	7.55	0.12
	No HES	2.19	14.36	0.25
SET2	**SVNS**	1.07	5.19	0.87
	Initial	1.45	6.49	0.43
	No HES	2.18	9.09	0.51
SET3	**SVNS**	3.10	8.68	7.49
	Initial	3.82	9.97	3.56
	No HES	5.48	13.23	4.34
SET4[a]	**SVNS**	0.76	3.82	0.17
	Initial	1.33	3.82	0.01
	No HES	0.76	3.82	0.14
All sets	**SVNS**	1.44	8.68	1.91
	Initial	2.11	9.97	0.89
	No HES	2.80	14.36	1.13

[a] Only instances with known optimal solution are considered.

Next, we illustrate the quality of the initialization phase and the effectiveness of calculating the heuristic estimated score (HES) for each feasible combination of hotels, explained in Section 4.2. Therefore, we create two new versions of the algorithm. For the first, we interrupt the algorithm at the end of the initialization phase (“Initial”). This gives an idea about the computation time required by the initialization phase, the quality of the initial solution and the quality improvement by the improvement phase. The second algorithm runs without using HES (“No HES”). In the original SVNS, HES is used to sort the feasible combinations of hotels in the initialisation phase and to select new feasible combinations of hotels in the Hotels-Shake. In this version of the algorithm, the initial solutions in the initialisation phase and the hotel combinations in the Hotels-Shake are selected randomly. Table 12 presents for each set of instances, the average gap, the maximal gap and the average CPU time for these three versions of the SVNS algorithm. The last three rows summarize these results over all sets.

Based on the results presented in Table 12, it can be concluded that the initialization phase takes less than half of the solution time on average, obtaining already high quality results, in general. Nevertheless, the improvement phase significantly improves the average and maximal gap, in an acceptable computational effort. What is even more appealing is the importance of calculating HES. Without HES, the average and maximal gap are almost twice as large compared to the original SVNS. Compared to the Initial variant, both gaps and the computational effort are larger.

7. Conclusion and future work

In this paper, the hotel selection variant of the orienteering problem is introduced. The goal of the OPHS is to determine a tour of maximal score that is composed of connected trips with limited length and every trip should start and end in one of the available hotels. The OPHS has a number of practical applications but is a challenging problem to solve. It is difficult to find a good combination of hotels which leads to a high-quality solution. Each time the combination of hotels is changed, the whole solution is affected.

We propose an SVNS solution approach to deal with this problem. Based on a careful tradeoff between trying to improve the combination of hotels and the selection of vertices, high quality results are obtained in small computation times. On 224 benchmark instances with known optimal solutions, 102 are solved to optimality, the average gap with the known optimal

solutions is only 1.44% and the average computation time is 1.91. Both the quality and the required computation time are good enough to be applied in practical applications. Some additional experiments indicate that the maximal (and average) gaps can be further reduced by increasing the number of used feasible combinations of hotels (*NUFC*). However, increasing *NUFC* also increases the computational times significantly. In the results we report in this paper, we made a tradeoff between the quality of the results and the computational time required to obtain these results. A sensitivity analysis shows that small changes in the parameter settings do not influence the performance of the algorithm. The results also indicate that it is challenging, and at the same time crucial for a good performing algorithm, to find an appropriate combination of hotels. Additional experimental results illustrate the effectiveness of the initialization phase of the SVNS algorithm.

Another contribution of this paper is the fact that we were able to create these 224 benchmark instances with known optimal solutions for this new problem. These instances vary in size, number of extra hotels, number of trips and degree of difficulty.

A number of future extensions can be considered: scores for hotels (based on their facilities, place and price), time windows, arc profits (beautiful routes), etc. A major challenge would be to deal with problems with more than one tour: the Team Orienteering Problem with Hotel Selection. Finally, a profound analysis on how to determine promising or good combinations of hotels would certainly be worthwhile for the development of future algorithms.

Acknowledgment

We would like to thank Dominique Feillet for providing some of the optimal OP solutions.

References

Archetti, C., Hertz, A., Speranza, M.G., 2006. Metaheuristics for the team orienteering problem. Journal of Heuristics 13 (1), 49–76.

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

Butt, S., Cavalier, T., 1994. A heuristic for the multiple tour maximum collection problem. Computers and Operations Research 21 (1), 101–111.

Chao, I.-M., Golden, B.L., Wasil, E.A., 1996a. Theory and methodology—the team orienteering problem. European Journal of Operational Research 88 (3), 464–474.

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

Gendreau, M., Laporte, G., Semet, F., 1998. A branch and cut algorithm for the undirected selective traveling salesman problem. Networks 32 (4), 263–273.

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

Labadie, N., Mansini, R., Melechovský, J., Wolfler Calvo, R., 2012. The team orienteering problem with time windows: an LP-based granular variable neighborhood search. European Journal of Operational Research 220 (1), 15–27.

Miller, C.E., Tucker, W., Zemlin, R. a., 1960. Integer programming formulation of traveling salesman problems. Journal of the ACM 7 (4), 326–329.

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

Sevkli, Z., Sevilgen, F.E., 2006. Variable neighborhood search for the orienteering. Lecture Notes in Computer Science 4263/2006, 134–143.

Tricoire, F., Romauch, M., Doerner, K.F., Hartl, R.F., 2010. Heuristics for the multi-period orienteering problem with multiple time windows. Computers & Operations Research 37 (2), 351–367.

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

Vansteenwegen, P., Souffriau, W., Berghe, G., Oudheusden, D., 2009a. A guided local search metaheuristic for the team orienteering problem. European Journal of Operational Research 196 (1), 118–127.

Vansteenwegen, P., Souffriau, W., Berghe, G., Oudheusden, D., 2009b. Metaheuristics for tourist trip planning. Lecture Notes in Economics and Mathematical Systems 624, 15–31.

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

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