Erasmus University Rotterdam
Erasmus School of Economics
Bachelor Thesis Econometrie en Operationele Research

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

Job Voorma (573594)

Supervisor:	Dr. T.A.B. Dollevoet
Second assessor:	B.T.C. van Rossum
Date final version:	2nd July 2023

Abstract

In this paper, we introduce the Orienteering Problem with Hotel and Charging station Selection (OPHCS). This is a new variant of the Orienteering Problem with Hotel Selection (OPHS). In both problems, we consider a set of vertices and hotels, where vertices have a score assigned to them and hotels are assigned a score equal to zero. The goal of the problem is to find an optimal tour, which exists out of consecutive trips that start and end at a hotel, that maximises the sum of the scores of all vertices in the tour. In contrast to the OPHS, we consider a tour that is performed by an electric vehicle with a limited range. If the total distance driven exceeds the vehicle's range, it should first recharge at a charging station before it can continue the tour. We use a Skewed Variable Neighborhood Search (SVNS) algorithm which builds upon the SNVS algorithm proposed for the OPHS in Divsalar et al. (2013). We include extra local search moves to consider solutions with a variety of charging stations. In our study, we have created 1270 OPHCS instances which can serve as a valuable resource for future research. We test our algorithm on 231 instances and find an average gap to the optimal solution of 4.99% with an average computation time of 184.48 seconds.

1 Introduction

The Orienteering Problem with Hotel Selection (OPHS) is relatively new and has not been widely studied (Toledo et al., 2020; Divsalar et al., 2013). With this problem, we have a complete graph where each node represents either a hotel or a vertex. All vertices have a score associated to them. We assume the score of hotels to be equal to zero. The goal of the problem is to find the optimal tour, which has the highest total score. The tour must start and end at two predetermined hotels. The tour consists of consecutive trips, where trips start and end at a hotel as well. Between the first and last hotel of each trip, a subset of all given vertices is visited. Unlike hotels, vertices can only be visited once in the entire tour. Each trip must start at the hotel where the previous trip ended. The goal is to find a selection of intermediate hotels and vertices such that the total score of the tour is maximised. We extend the OPHS by introducing charging stations into the problem. We assume that the score of charging stations is equal to zero and that charging stations can be visited as often as necessary. Vehicles are now assumed to have a limited range and must recharge at a charging station whenever the distance traveled reaches the vehicle's range. We call this problem the Orienteering Problem with Hotel and Charging station Selection (OPHCS).

The OPHS knows many practical applications (Divsalar et al., 2013). For instance, trip planning for tourists, a submarine performing several consecutive missions and trip planning for truck drivers. By introducing electric vehicles with a limited range and charging stations, we aim to enhance practicality of the problem. The relevance of our extension is especially for this last application evident. Currently, still a large majority of freight truck registrations are non-electric vehicles (IEA, 2022). However, as we are moving towards a sustainable society, this number is expected to grow. An independent study of TNO predicts that by 2035, new electric trucks will be cheaper in their use than 99.8% of diesel trucks (Tol et al., 2022). The relevance of our extension is clearly not limited to that of freight trucks. Also for personal electric vehicles, navigation systems include recommended charging stations in the route. Since these applications require short computations times, the problem needs to be solved within a reasonable amount

of time. However, solving this problem exactly, with a solver like CPLEX, quickly results in infeasible computation times as the problem instance increases (Divsalar et al., 2013). For that reason, a fast heuristic with high quality solutions is needed to solve this problem.

For the OPHS several efficient meta-heuristics exist (Divsalar et al., 2013; Sohrabi et al., 2020). In this research, we introduce the OPHCS and propose a variable neighborhood search method for the OPHCS which extends the algorithm of (Divsalar et al., 2013). The performance of the algorithm is judged based on the average gap to the optimal solution and computation times. We analyse how the performance of the SVNS algorithm is affected by the vehicle's range and the number of charging stations.

Since there are no benchmark instances available for the OPHCS, we create our own instances to evaluate the performance of the presented algorithm. In total 1270 benchmark instances are created for the OPHCS. These instances are created based on a selection of available OPHS instances from www.mech.kuleaven.be/cib/op. We create OPHCS instances for SET1 and SET2.

To tackle the OPHCS we propose a skewed variable neighborhood search (SVNS) algorithm. This algorithm builds upon the algorithm proposed in Divsalar et al. (2013). This variant of variable neighborhood search has been developed to address the problem of exploring solution spaces far from the current solution with respect to ordinary variable neighborhood search (Hansen et al., 2019). It does so, by also accepting solutions that are slightly worse than the current solution. Usually, a distance function is used to decide whether a slightly worse solution is accepted. We prefer to use a SVNS method over other available algorithms to solve the OPHCS, due to its simplicity, high quality solutions and practical computation times for the OPHS (Divsalar et al., 2013).

From our results, we conclude that the OPHCS is more difficult to solve than the OPHS. Compared to the obtained results for the OPHS, we observe higher average gaps and significantly longer computation times for the OPHCS. We find that the performance of the algorithm is significantly affected by the vehicle's range and the number of charging stations. For instances with higher vehicle ranges, the average gaps approach those reported for the OPHS in Divsalar et al. (2013). All in all, with this research we introduce the OPHCS and provide a starting point for the development of efficient algorithms for this problem. Furthermore, we have generated a set of 1270 new benchmark instances for this problem. These instances serve as valuable resources for future research in this field.

In the next Section we give a general problem description for the OPHCS and introduce the necessary mathematical notation. We continue with a literature review in Section 3. In Section 4, we discuss the data that we use for our research and how we created the used benchmark instances. In Section 5, we present a mathematical formulation for the OPHCS and introduce our SVNS algorithm. After this, the obtained results are reported and analysed in Section 6. We finalise this paper with a conclusion, where we summarise our main findings and propose a number of extensions for future research.

2 Problem description

The OPHCS is closely related to the OPHS, which was introduced by Divsalar et al. (2013). In both problems, we are given a complete graph with a set of N vertices and a set of H hotels.

Among these hotels, a first hotel and a last hotel are specified. In contrast to the traveling salesman problem with hotel selection (Vansteenwegen et al., 2012), not all given vertices are required to be visited. We let P denote the set of all vertices and let S denote the set of hotels. All vertices $i \in P$ have a score s_i associated to them. We follow Divsalar et al. (2013) in their assumption that hotels have a score of zero assigned to them. In addition, we are given the number of trips we need to make (D). For each trip d, we are given a maximum length T_d for $d = 1, .., D$. The goal of the problem is to find the tour, that visits a subset of all given vertices, with the highest score associated to it. A tour consists of trips, where all trips must begin and end at a hotel. Each trip must start at the hotel where the previous trip, if any, ended. Between the hotels, a selection of all vertices can be visited. All vertices can only be visited once in the tour.

The OPHCS differs from the OPHS in the sense that now not only the length of each trip is limited, but the vehicle has a limited range, given by Q, as well. In addition, we are given a set of C charging stations denoted by R. If the distance traveled reaches the range of the vehicle, the vehicle must first recharge at a charging station before it can continue the tour. In our implementation, we do not assume that the vehicle can recharge at hotels. However, it is naturally possible that a charging station is located at the same location as one of the hotels. We allow for charging stations and hotels to be visited multiple times.

For the purpose of notation, we let $B = P \cup S \cup R$. We assume the score of all charging stations to be zero. All vertices, hotels and charging stations have an x- and y-coordinate associated to them and we assume that the distance between a pair of locations is given by the euclidean distance. The time it takes to travel between two points i and j is given by $t_{i,j}$, $i, j \in B$. We let $q_{i,j}$ denote the charge it takes to travel between i and j. Without loss of generality, we assume that $t_{i,j}$ and $q_{i,j}$ are equal to the euclidean distance between i and j. In Figure 1, we present an illustrative example of an OPHCS solution. In this example, the number of trips must be equal to two, which implies that one extra hotel has to be included in the tour.

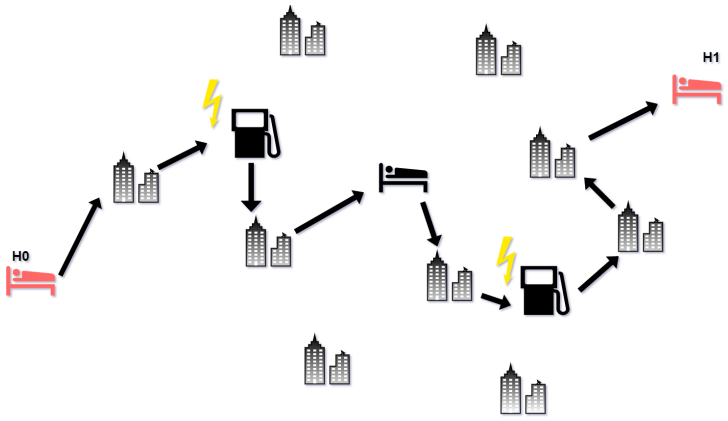

Figure 1: Illustrative example of an OPHCS tour with two trips

3 Literature review

In the literature, several problems have been considered that are closely related to the OPHCS. The OPHS problem can be seen as a variant of the OPHCS where battery capacities are infinite.

The OPHS is introduced by Divsalar et al. (2013) and is one of the most recent forms of the Orienteering Problem (OP). Several other variants of the OP exist, such as the Team OP, the Team OP with Time Windows and the Time Dependent OP (Gunawan et al., 2016). Because orienteering problems are NP hard (Keller, 1989), we rely on heuristics to solve these problems. In recent research, very efficient and high quality algorithms have been proposed for the OPHS (Divsalar et al., 2013; Sohrabi et al., 2020; Divsalar et al., 2014). We present a Skewed Variable Neighborhood algorithm to solve the OPHCS problem. We use the algorithm of Divsalar et al. (2013) as a starting point and extend their approach to make it suitable for the OPHCS.

In their paper, a SVNS algorithm to solve the OPHS is introduced. Skewed variable neighborhood search is a variant of general variable neighborhood search (VNS). In general, a VNS algorithm follows the following framework. We start with an initial solution and iteratively try to find better solutions by performing a shake step and improving this solution by means of local search. We accept the new solution if it is better than the solution used to start the current iteration. The shake step performs a neighborhood change and allows to escape from the current solution valley. Local search is applied on this solution to get to local optima within the neighborhood of the solution. Local search methods starts with an initial solution x. From this solution, we find a direction of descent within the neighborhood of x that improves the solution. Improvements are made by means of local search moves, such as the insertion of new vertices or relocating included vertices in the tour. We then move to this new solution and repeat this process until no direction of descent is found. Several descent heuristics exist, but most algorithms make use of the "best improvement" direction (Hansen et al., 2010). This heuristic moves in the direction with the steepest descent. After we complete the hotels-shake and improvement step, we decide if we move from the solution that was used to start the current iteration to the new solution. We move to this new solution if it is better than our current solution.

This last step is where SVNS differs from VNS. This method is designed to overcome the difficulty of exploring solutions that are distant from the incumbent solution (Hansen et al., 2010). If we have found the best solution in a solution valley, we need to move away from this solution and find an improved one. To allow for this, SVNS gives some compensation to solutions far from the incumbent solution by also accepting solution with a slightly lower score. By this, solutions that are in far apart valleys are also explored. This variant of VNS is often used for problems with many local optima or multiple far apart near optimal solutions (Hansen & Mladenovic, 2003).

To develop a SVNS algorithm suitable for the OPHCS, we extend the algorithm of Divsalar et al. (2013) by including additional local search steps that aim to find the optimal charging stations and their best positions in the tour. To decide on which extra local search moves should be included, we look for moves that are found to be effective for related problems. We include an insertion step for charging stations, since both Zhang et al. (2018) and Hiermann et al. (2016) include this move in their local search procedure. We choose to keep this move intuitive and simple. Our implementation of this move is described in Section 5.3.4. Besides from charging station insertion, a charging station removal step is often included for relevant problems (Keskin & Çatay, 2016; Zhang et al., 2020). For that reason, we include a charging station removal step in our local search procedure. This step is described in more detail in Section 5.3.4. To

maintain the advantage of very few parameters required as input for SVNS algorithms Divsalar et al. (2013), we choose for a more straightforward implementation compared to the approaches of Keskin & Çatay (2016) and Zhang et al. (2020).

4 Data

We use the paper of Divsalar et al. (2013) as a starting point for our research. The OPHS benchmark instances used in their paper can be found on www.mech.kuleaven.be/cib/op. We use these instances to replicate their results. Since there are no benchmark instances available for the OPHCS, we construct new cases for this problem. We make use of the instances in SET1 and SET2.

To create the OPHCS instances, we need the optimal OPHS tour for the cases from SET1 and SET2. We only create OPHCS instances for the cases for which the optimal OP solution can be found within half an hour by CPLEX. Due to how the instances in Divsalar et al. (2013) are created, obtaining the optimal OPHS tour from the OP tour is relatively straightforward. Given that the instances from Set1 and Set2 give the same optimal OP solution, we only need to find the optimal OP solutions for Set1. We find the optimal solution with CPLEX by solving the OP. We only use the instances for which we are able to find the optimal solution within half an hour of computation time. This is the case for 26 of the 35 OPHS instances. We start with creating three OPHCS instances for those cases. In these instances, we set the vehicle range equal to 20%, 40% and 60% of the length of the known optimal OPHS tour. For all three vehicle ranges, we traverse the optimal OPHS solution and as soon as the distance traveled exceeds the range of the vehicle, we insert a charging station at exactly the position of the last vertex that we could travel without exceeding the battery capacity. After insertion, the distance traveled is set to zero again and we continue traversing the optimal OPHS tour starting from the inserted charging station. We continue this process until we reach the final hotel of the tour.

We do not create these instances for the instances from SET3, since the computation time of the OPHS algorithm increases significantly for these instances (Divsalar et al., 2013). We do not consider SET4 either, because we need instances with a known optimal solution. If the battery capacity is lower than the maximum distance between two vertices in the optimal OP tour, no OPHCS instance can be created with this capacity. This is only the case for instance $100 - 30$ when the range equals 20% of the length of the optimal OPHS route. For that reason, instance $100 - 30$ is excluded from the created sets with a 20% range.

To make the benchmark instances more complex, we create two more instances for each OPHCS instance with respectively two and six extra charging stations. Adding the stations is done in a similar way to how the extra hotels are added for the OPHS instances (Divsalar et al., 2013). We let N denote the number of vertices in the instance, so not counting hotels and charging stations, and C denote the total number of charging stations in the new instance. Thus, C includes the charging stations to be added to the instance. The extra charging stations are placed at the locations of the the vertices with an index that is a multiple of $\frac{N}{C+1}$. The first hotel has index zero and the final hotel is assigned an index equal to one. We skip the other hotels when we compute the indices. As soon as the number of total charging stations equals C, we stop traversing the vertices. All in all, this gives us nine new OPHCS instances

for each OPHS instance that we consider. In accordance with how Divsalar et al. (2013) added the extra hotels to the OPHS instances, we allow an extra charging station to be equal to an already included charging station. We tried to solve these instances with CPLEX, but none of the OPHCS instances was solved to optimality within an hour of computation time. This stresses the importance of developing an efficient algorithm for this problem.

5 Methodology

In this section, we begin by presenting a mathematical formulation for the OPHCS. Following that, we describe the SVNS algorithm for the OPHCS. Finally, we introduce a benchmark algorithm that is employed to critically asses the performance of our SVNS approach.

5.1 Mathematical Formulation

Like the OPHS, the OPHCS can be formulated as an mixed-integer linear program (MIP). By making use to the notation introduced in Section 2, we can formulate the problem as shown by equation 1 to 13. We introduce the decision variables $x_{i,j,d}$ and y_i. The decision variable $x_{i,j,d}$ is equal to 1 if we travel from point i to point j in trip d and 0 otherwise for $i, j \in B$ and $d = 1, .., D$. We let y_i represent the range left in the battery when we arrive at point i, where $i \in B$. This formulation is based on the formulation for the OPHS of Divsalar et al. (2013). Extra constraints have been added to make it suitable for the OPHCS. Since we assume instantaneous and only full recharges, there is no need for us to make duplicates of the nodes that represent the charging stations that are visited multiple times. However, since we allow for hotels to be visited multiple times and do not assume that a vehicle recharges at a hotel, we are required to add duplicate nodes for the hotels. This formulation has also been used for the electric TSP (Roberti & Wen, 2016). We duplicate each hotel $d - 1$ times, such that the formulation still holds in the extreme case of a tour with d times the same hotel. We let S' denote the set of hotels with the duplicates included. This results in $|S'|$ to be equal to $d * H$. We let $B' = P \cup S' \cup R$. Index 0 denotes the first hotel and index 1 the final hotel. It is possible for the hotel at index 0 and index 1 to be the same.

$$\text{Max} \quad \sum_{d=1}^{D} \sum_{i \in B'} \sum_{j \in B'} s_i x_{i,j,d} \tag{1}$$

s.t.

$$\sum_{l \in B' \setminus \{0\}} x_{0,l,1} = 1 \tag{2}$$

$$\sum_{k \in B' \setminus \{1\}} x_{k,1,D} = 1 \tag{3}$$

$$\sum_{h \in S'} \sum_{l \in B'} x_{h,l,d} = 1, \quad \forall d = 1, ., , D \tag{4}$$

$$\sum_{h \in S'} \sum_{k \in B'} x_{k,h,d} = 1, \quad \forall d = 1, ., , D \tag{5}$$

$$\sum_{i \in B'} x_{i,k,d} = \sum_{j \in B'} x_{k,j,d}, \quad \forall k \in P \cup R, \quad \forall d = 1, ..., D \tag{6}$$

$$\sum_{d=1}^{D} \sum_{j \in B'} x_{i,j,d} \leq 1, \quad \forall i \in P \tag{7}$$

$$\sum_{i \in B'} \sum_{j \in B'} t_{i,j} x_{i,j,d} \leq T_d, \quad \forall d = 1, .., D \tag{8}$$

$$y_0 = Q \tag{9}$$

$$y_j + (q_{i,j} + Q) x_{i,j} \leq y_i + Q, \quad \forall i \in P \cup S', \quad \forall j \in B' \tag{10}$$

$$y_j + q_{i,j} x_{i,j} \leq Q, \quad \forall i \in R, \quad \forall j \in B' \tag{11}$$

$$x_{i,j,d} \in \{0, 1\}, \quad \forall i, j \in B' \text{ s.t } i \neq j, \quad d = 1, ..., D \tag{12}$$

$$y_i \in [0, Q], \quad \forall i \in B' \tag{13}$$

Equation 1 represents the objective of the problem. This equation calculates the total score of the tour, which we aim to maximise. With equation 2, we ensure that the tour starts at the specified first hotel. In a similar way constraint 3 ensures that the tour ends at the final hotel with index 1. Constraints 4 and 5 make sure that each trip starts and ends at a hotel. In addition constraint 6 guarantees the connectivity within each trip. To consider the restriction that each vertex, so excluding hotels and charging stations, can only be visited once, we include constraint 7. Constraint 8 ensures that the length of each trip does not exceed the maximum trip length. The remaining constraints are not included in the OPHS formulation and are incorporated to take into account the vehicle's range. To set the initial vehicle charge to full capacity, we include constraint 9. We include constraints 10 and 11 to monitor the battery charge of the vehicle, while also serving as subtour elimination constraints.

5.2 Skewed variable neighborhood search algorithm for the OPHS

The algorithm that we present for the OPHCS builds upon the algorithm for the OPHS of Divsalar et al. (2013). For that reason, we initially replicate their algorithm for the OPHS and compare our results with theirs to validate if our implementation is correct. The authors did not report all of their methods clearly, which forces us to make some decisions ourselves. Firstly, we set k_{max} equal to the maximum of $0.25 * NUFC$ and two. This is to ensure that

also for instances with $NUFC$ smaller than four, all shake moves are applied at least once. Unfortunately, the authors are not clear about which descent heuristic they use in their local search algorithm. They do specify for the sub-OP that they apply one local search move as long as it finds improvements to the solution. If no improvements are possible anymore, they move on to the next move. We choose to use this approach for the local search algorithm as well. We recognise that this approach is expected to lead to longer running times compared to a first-order descent heuristic (Hansen et al., 2010). This is due to the fact that the more time-consuming moves are positioned lower in the predetermined order of local search moves. The other parameter values are set equal to those reported in Divsalar et al. (2013).

5.3 Skewed variable neighborhood search algorithm for the OPHCS

We use a skewed variable neighborhood search (SVNS) framework to solve the OPHCS. We use the algorithm proposed in Divsalar et al. (2013) for the OPHS as a starting point for our approach. Modifications have been made to the feasibility checks for hotel combinations and routes to consider the limited range of the vehicle. In addition, extra local search steps have been included to create solutions with a diverse range of charging stations.

5.3.1 Parameter inputs

The framework of our SVNS algorithm is shown in Algorithm 1. We made adjustments to the parameter inputs of our algorithm in comparison to those presented in Divsalar et al. (2013). Since we observed that our SVNS algorithm for the OPHCS rarely finds improvements after 25 moves without improvement, we set $NoImprovementMax$ equal to 25 for time saving purposes. The "Number of Used Feasible Combinations of hotels" (NUFC) remains equal to 250. We set k_{max} equal to the maximum of $0.25 * NUFC$ and two. This is to ensure that also for instances with a $NUFC$ smaller than or equal to four, we perform at least one shake iteration. Similar to Divsalar et al. (2013), we set the value of $MaxPercentageWorse$ equal to 30%.

5.3.2 Initialization

To begin, we need to find a feasible OPHCS tour. This process can be broken down into three steps: computing the matrix of pairs of hotels, finding all feasible combinations of hotels and finally constructing a feasible initial solution. Firstly, we compute the matrix of pairs of hotels. This matrix contains the potential score for each pair of hotels. Naturally, this score is dependent on the maximum trip length T_d. For that reason, we compute and store this potential score for each maximum trip length T_d, with $d = 1, ..., D$. This results in a three dimensional matrix with dimensions H by H by d. The potential score for each pair of hotels is computed by solving a sub-OP. This process is described in Algorithm 2, where X_{init} is the empty pair of hotels. It is likely that the distance between a pair of hotels is longer than the vehicle's range. To give a feasible initial trip to the sub-OP algorithm, we first make the pair feasible by means of the "Insert Charging Stations" method. This method is used in Algorithm 3 and described in more detail further in this section. For pairs of hotels that have no feasible route between them, the sub-OP cannot be solved. We assign a score of zero to these pairs.

Algorithm 1: Skewed Variable Neighborhood Search

Initialization:
1) Construct matrix of pairs of hotels
2) Find all feasible hotel combinations
3) Construct a feasible initial solution: X
NoImprovement $\leftarrow 0$
while *NoImprovement* $<$ *NoImprovementMax* **do**
  $k \leftarrow 1$
  while $k < k_{max}$ **do**
    $X' \leftarrow$ Vertices-Shake(X)
    $X'' \leftarrow$ LocalSearch(X')
    $X' \leftarrow$ Hotels-Shake(X'')
    $X'' \leftarrow$ LocalSearch(X')
    Recenter:
    if X'' *is better than* X **then**
      $X \leftarrow X''$
      $k \leftarrow 1$
      NoImprovement $\leftarrow 0$
    end
    else
      NoImprovement++
      if X'' *is at most MaxPercentageWorse worse than* X **then**
        $X \leftarrow X''$
        $k \leftarrow 1$
      end
      else
        $k++$
      end
    end
  end
end

Algorithm 2: Sub-OP algorithm

Input: Feasible trip X_{init}
Output: Feasible trip X
while *Any of the moves improves solution* **do**
  X $\leftarrow$ Insert(X);
  X $\leftarrow$ InsertChargingStations(X);
  X $\leftarrow$ Replace(X);
  X $\leftarrow$ ReplaceChargingStations(X);
  X $\leftarrow$ TwoOpt(X);
  X $\leftarrow$ MoveBest(X);
end

Secondly, we compute all feasible combinations of hotels that start at the specified first hotel of the tour and end at the specified final hotel of the tour. Since each combination must result in D trips, all combinations must contain $D+1$ hotels. We consider a combination feasible, if it obeys both the trip length- and vehicle range constraints. It is likely that a hotel combination needs charging stations to be feasible with respect to the vehicle range constraints. If so, we add charging stations into the combination with the "Insert Charging Stations" move shown in Algorithm 3. This move is described in more detail later.

For all of the hotel combinations that are found to be feasible, we compute a heuristic estimated score (HES) by summing all of the sub-OP values for each consecutive pair of hotels in the combination. These sub-OP values are taken from the matrix of pairs of hotels. We take the sub-OP value of the matrix that belongs to that pair of hotels and the respective trip in the combination. This value is used to get a first idea of which combinations are promising. The combinations are sorted based on their HES and we continue with the $NUFC$ highest combinations. If the Total Number of Feasible Combinations (TNFC) is smaller than $NUFC$, we set $NUFC$ equal to $TNFC$.

Finally, we apply three different search strategies to the chosen combinations. The solution with the highest score will be used as the initial solution. To start, we apply the full local search algorithm on each combination of hotels. It is possible that charging stations are included in this empty combination. To obtain another solution, we apply the sub-OP on each trip of the empty combination, starting from the first trip. After this, we improve the solution using local search. For one more solution, we again start by applying the sub-OP on the empty trips, but now starting at the last trip. Again the solution is improved by means of local search. With all three strategies, we do not allow for vertices to occur more than once in the tour. The last two approaches are now more complex than for the OPHS approach of Divsalar et al. (2013), because a change in one trip might cause a conflict with respect to the charge constraints in another trip. In our approach, we do not allow for the sub-OP to result in an infeasible tour. For that reason, when we apply the sub-OP on one of the trips of a tour, it is essential that we ensure that this does not cause conflicts with respect to the vehicle range among other trips. We save the solution with the highest associated score and continue to the next phase of the algorithm with this solution.

5.3.3 Shaking Phase

In the shaking phase, we apply three different shake moves on the current solution: two types of vertices-shakes and one hotels-shake. In the first shake move, we delete the first half of each trip of the current solution. If l denotes the length of the trip, we remove the first $\lfloor \frac{l}{2} \rfloor$ elements. These elements can be both vertices and charging stations. In order to try solutions with a high variety of charging station combinations, we also delete the charging stations in the first half of each trip. We improve the solution with local search.

The second shake move is comparable to the first. However, now we remove the second half of each trip. We again improve the solution by means of local search. If one of these moves results in a feasible tour with a higher score than the current tour, we update the current tour to the improved tour.

After we have made both shake moves, we continue with a hotels-shake on the current solution. In this move, the hotels in a solution are replaced by one of the NUFC combinations of hotels. We randomly select a combination out of the NUFC combinations. To prevent considering the same hotel combinations in different iterations without moving to a new solution, we store the NUFC combinations that we have considered. The combinations in this list can not be selected. As soon as we recenter to a new solution at the end of an iteration, we clear the list of tried combinations. All charging stations are removed. This hotel change is likely to result in an infeasible tour. In such cases we apply an iterative approach, where we sequentially remove the vertex with the lowest ratio of score over distance saved after removal. This process continues until the solution is feasible or no more vertices to remove are left. This new solution is improved by means of local search.

With all three shake steps, it is possible that the solution passed to the local search method is infeasible with respect to the vehicle's range. For that reason, we apply the "Make Feasible" method, which is shown in Algorithm 3, before starting the local search procedure. This method consists of an "Insert Charging Stations" move in combination with a "Remove Most Expensive" move. We initially apply the "Insert Charging Stations" move. If this does not result in a feasible solution, we apply the "Remove Most Expensive" move and try the "Insert Charging Stations" move again. This is done until we obtain a feasible solution. This procedure is shown in Algorithm 3. Both moves are described in more detail below.

Insert charging stations This method takes as input an infeasible solution with respect to the charging constraints. We iteratively insert the cheapest charging station into the route until the tour is feasible with respect to all constraints. If we can not insert more charging stations and the route is still infeasible, we start the make feasible method again, but start with inserting another charging station than we started with the previous times. We try this until we have obtained a feasible solution or we have tried all possible charging stations for the first insertion.

Remove most expensive This move takes as input an infeasible tour from the make feasible move. We remove the vertex from the tour, so no hotel or charging station, for which the removal results in the highest decrease in the distance traveled.

Algorithm 3: Make Feasible algorithm

```
Input: Infeasible solution X
while X is not feasible do
    X ← InsertChargingStations(X);
    if X is not feasible then
        X ← RemoveMostExpensive(X);
    end
end
```

5.3.4 Local-Search

Our local search algorithm utilises, among other moves, the nine local search moves of Divsalar et al. (2013). We include three extra moves, which contribute to choosing the optimal charging

stations and finding their best position in the tour. Except for the insert charging stations move, we apply one local search move as long as it finds improvements to the current solution. The local search moves are applied in the order below as long as one of the moves improves the current solution. We design the local search procedure in a way that limits the number of parameter inputs.

Insert This step inserts non-included vertices into the tour. For all non-included vertices we find the position for which insertion of the non-included vertex results in a minimum increase in the length of the tour. For these non-included vertices we compute a value that is calculated as the score of that vertex divided by the increase in trip length. We insert the vertex for which this value is the highest and insertion does not result in an infeasible solution. The vertex is inserted at the position which results in the lowest increase in trip length. We do not consider inserting charging stations in this step.

Insert charging stations This move is applied as soon as the insert move fails to insert new vertices. For each charging station we find the position of insertion that results in a minimum increase in trip-length while maintaining a feasible solution. We insert the charging station that can be added to the tour with the lowest increase in trip length. When a feasible insertion is found we go back to the insert step and try to insert new vertices to the route, as soon as this fail we go back to this move. If no feasible insertion is found for all charging stations, we continue to the next step.

Two-Opt For each trip in the tour, we find the two vertices for which reversing the order of the vertices between them results in the highest decrease of travel distance. If a profitable pair is found, the order of the vertices is reversed. Due to the limited vehicle range and the presence of charging stations, this step does not always result in a feasible solution. We perform the move that is most profitable and results in a feasible solution.

Move-best Starting at the first vertex of the tour, we evaluate whether moving this vertex to the position where insertion results in the minimal increase in trip length is feasible. If so, we complete this move and we move on to the next vertex in the tour. We also consider charging stations for relocation in this step.

Swap-trips For each pair of trips d_1 and d_2, we swap two vertices $i \in d_1, j \in d_2$ only if this swap results in a decrease in the distance traveled for at least one of the trips and whole tour remains feasible. The swap that results in the largest decrease in distance traveled is performed. Even though Divsalar et al. (2013) states that a swap-trips move is performed as long as the distance of at least one of the trips decreases, we follow other relevant literature and only perform a swap-trips move if this results in a decrease of the total tour length (Vansteenwegen et al., 2011).

Extract-Insert Starting form the first vertex in the tour, we check whether excluding this vertex and inserting as many non-included vertices as possible can increase the score of the trip.

In those cases, the change is made and we move on to the next vertex in the tour. The excluded vertex is not considered again for insertion in this iteration of the local search. However, inserted vertices are considered again for exclusion.

Extract2-Insert This move is similar to the previous move, but now we consider two consecutive vertices for exclusion.

Extract5-Insert This move resembles the Extract-Insert and Extract2-Insert moves, but now we consider five consecutive vertices for exclusion. If this change is found to be profitable, we perform the change and start a new local search iteration.

Extract-Move-Insert Starting from the first vertex in the first trip, we consider each vertex for removal from the tour. Subsequently, we examine if after this removal any other vertex can be moved to a different position in the tour. Charging stations are considered for both extraction and relocating. If this is possible, we check whether we can insert any of the non-included vertices in the trip the vertex was moved from. We only consider inserting vertices with a higher score than the removed vertex. If this move is found to be feasible, we make the changes and update the current solution. We move on to the next vertex in the trip, where the extracted vertex is considered again for insertion.

Replace For each trip, we check for all non-included vertices whether we can insert this vertex at the position that results in the lowest increase in travel distance. If this insertion is found to be feasible, we insert the vertex. If not, we check whether removing one of the vertices with a lower score than the vertex under consideration results in a feasible insertion. We also consider charging stations for removal. As soon as we find a removal that results in a feasible insertion, we perform this change.

Replace Charging Stations This move is similar to the Replace move. However, now we only consider charging stations for insertion and removal. In addition, we only consider replacing one charging station for another, so we do not insert a charging station without removing another one. This is to prevent excessive insertion of charging stations, as we prioritise solutions with less charging stations. We replace one charging station for another if this results in a decrease of the tour length and the tour remains feasible.

Remove Charging Stations This step is often included in relevant literature (Keskin & Çatay, 2016). To keep this move simple, we include one removal component for the charging stations. Starting with the first charging station in the first trip, we check if the tour remains feasible when we remove this charging station. If so, we apply all local search steps, except for this move and the insert charging stations move, to the tour with the removed charging station. If this results in an improvement of the solution with respect to either score or distance, we update the current solution to this new solution and move on the next trip. If this is not the case, we move on to the next charging station in the tour.

5.3.5 Recentering phase

Each time after we finish the local search after the hotels-shake, we decide if we update the current solution. If the solution after hotels-shake and local search is better than the one used to start this iteration of the algorithm with respect to the score of the tour, we start the next iteration of the algorithm with this new solution. Also if the new solution is only $MaxPercentageWorse$ percent worse than the solution used to start this iteration, we start the next iteration with the new solution. If the solution after hotels-shake and improvement is more than $MaxPercentageWorse$ percent worse, we start the next iteration with the same solution used to start this iteration.

5.4 Benchmark Algorithm

To critically evaluate the performance of the presented SVNS algorithm, we want to compare its results to those of a relatively simple benchmark algorithm. The benchmark algorithm starts with a feasible OPHS tour. This feasible solution is obtained via our implementation of the SVNS algorithm presented in Divsalar et al. (2013). To convert this tour into a feasible OPHCS tour, we apply the make feasible algorithm as described in Algorithm 3. This gives us a feasible OPHCS tour. To improve this solution, we perform the local search procedure of Section 5.3.4. Due to how the OPHCS instances are created, it is not representative for the true quality of the benchmark model if we use all instances to compare the algorithms. For the instances that our OPHS algorithm solved to optimality, we know that the optimal charging stations are located at exactly the locations of vertices in the tour. As a result, all optimal charging stations can be added to the optimal OPHS tour without increasing the distance of the tour. For that reason, we only compare the results of our OPHCS algorithm and the benchmark algorithm for the instances that our OPHS algorithm does not solve to optimality.

6 Results

In this section, we start by presenting the results of our algorithm for the OPHS instances and compare them to the results of Divsalar et al. (2013). After this, we continue with our results for the OPHCS. Finally, we compare the performance of our SVNS algorithm for the OPHCS with the results of a relatively easy benchmark model.

6.1 Results OPHS

The results of our implementation of the SVNS algorithm for the OPHS are shown in Table 1. We compare these with the results of Divsalar et al. (2013), which are shown in Table 2. The results for each instance separately can be found in Appendix A. Here, the results for SET4 are also included. In both tables, the rows represent the different sets of instances. The columns respectively give the average "Total Number of Feasible Hotel Combinations" (TNFC), the average gap when compared to the optimal solution given as a percentage value and the computation time of the algorithm in seconds. If we consider the average performance of both implementations, we observe that the average gap is slightly higher for our algorithm. Even

tough the average gaps are not too far apart, the gaps per set seem to differ significantly between both implementations. Our algorithm delivers lower gaps to the optimal solution for set1: 1-2, Set1: 2-3 and set2: 5-3. However, for set2: 6-4, set3: 10-4 and set3 12-5 our algorithm is outperformed.

Sets	TNFC	Gap (%)	CPU (s)
Set1: 1-2	2.89	0.83	11.39
Set1: 2-3	14.03	0.38	10.56
Set1: 3-4	92.80	0.93	15.53
Set2: 5-3	62.49	0.36	12.27
Set2: 6-4	379.31	1.97	24.41
Set3: 10-4	1417.18	2.76	365.73
Set3: 12-5	25029.18	5.26	252.71
Average		**1.78**	**98.94**

Table 1: SVNS result

Sets	TNFC	Gap (%)	CPU (s)
Set1: 1-2	2.89	1.22	0.18
Set1: 2-3	14.03	0.93	0.21
Set1: 3-4	92.80	0.92	0.65
Set2: 5-3	62.49	0.93	0.34
Set2: 6-4	379.31	1.21	1.39
Set3: 10-4	1417.18	2.61	8.20
Set3: 12-5	25029.18	3.58	6.77
Average		**1.63**	**2.53**

Table 2: SVNS results of Divsalar et al. (2013)

We can only speculate about the causes of these differences. It is likely that our implementation differs from that of the authors, since some of the steps of the SVNS algorithm are not described in detail. We suspect that Divsalar et al. (2013) uses a different descent heuristic, probably a "best improvement" approach, in their SVNS algorithm. If this is the case, this could provide a plausible explanation for the longer running times of our implementation and potentially also for the differences observed in the average gaps. In addition, it is likely that a part of the difference in running times can can be attributed to sub optimal programming on our part.

It is worth mentioning that our algorithm gives lower gaps than that of Divsalar et al. (2013) for the three sets with a lower average TNFC and performs worse for the remainder of the sets. An increase in the average gap as a result of a higher TNFC is expected (Divsalar et al., 2014), but the average gap in our implementation appears to exhibit a greater degree of responsiveness than the implementation of Divsalar et al. (2013).

Given that the average running time for set1: 1-2 is relatively low and the average gap is close to that of Divsalar et al. (2013), we decide to assess the effectiveness of our SVNS algorithm for the OPHCS based on the results for set1: 1-2. We choose not to evaluate the algorithm's performance on the results for set1: 2-3, because these results might exaggerate the quality of our algorithm.

6.2 Results OPHCS

We evaluate the performance of the presented algorithm for the OPHCS by using the instances created for Set1 with one extra hotel and two trips. We create these instances according to the procedure described in Section 4. As described in that section, we find the optimal OP solution within a time frame of thirty minutes for a total of 26 instances. Our OPHS algorithm achieved the optimal objective value in 16 out of 26 cases. The corresponding results are shown in Table 3. In Table 4, we present the baseline performance of our OPHCS algorithm. For

these instances, the capacity of the vehicle has been set equal to 100% of the optimal length of the optimal OP tour and no extra charging stations have been included. We observe that the average gap increases slightly and the average computation time increases from 12.03 to 20.34. Based on these results, we conclude that the baseline instances exhibit a slightly higher level of complexity compared to their respective OPHS instances. We use the results in Table 4 to evaluate how the vehicle's range and the number of extra charging stations influence the performance of the algorithm.

Instance	TNFC	SVNS	Gap (%)	CPU (s)
100-30	2	173	**0.00**	0.68
100-35	1	241	**0.00**	1.27
100-40	2	299	**0.00**	1.42
100-45	3	367	**0.00**	3.86
64-45	3	816	**0.00**	8.49
64-50	3	876	2.67	10.46
64-55	3	966	1.83	17.58
64-60	3	1050	1.13	24.12
64-65	3	1116	**0.00**	26.80
64-70	3	1170	1.52	35.20
64-75	3	1230	0.49	35.37
64-80	3	1284	**0.00**	35.05
66-125	3	1655	0.90	51.06
T1-65	3	240	**0.00**	1.29
T1-70	3	260	**0.00**	1.57
T1-73	3	250	5.66	3.90
T1-75	3	270	**0.00**	3.96
T1-80	3	280	**0.00**	2.50
T1-85	3	285	**0.00**	3.06
T3-100	3	800	**0.00**	4.99
T3-65	3	610	**0.00**	3.43
T3-75	3	650	2.99	4.99
T3-80	3	710	**0.00**	4.37
T3-85	3	720	2.70	6.69
T3-90	3	770	**0.00**	4.26
T3-95	3	780	1.27	5.04
Average			**0.85**	**12.03**

Table 3: SVNS results for Set1: 1 extra hotel 2 trips (OPHS)

Instance	TNFC	SVNS	Gap (%)	CPU (s)
100-30	1	173	**0.00**	0.75
100-35	1	241	**0.00**	0.78
100-40	1	299	**0.00**	1.33
100-45	2	367	**0.00**	3.36
64-45	3	816	**0.00**	9.60
64-50	3	882	2.00	14.32
64-55	3	978	0.61	27.74
64-60	3	1062	**0.00**	26.31
64-65	3	1116	**0.00**	38.42
64-70	3	1170	1.52	60.62
64-75	3	1230	0.49	57.30
64-80	3	1284	**0.00**	85.35
66-125	3	1655	0.90	131.42
T1-65	3	240	**0.00**	2.31
T1-70	3	260	**0.00**	3.93
T1-73	3	245	7.55	4.01
T1-75	3	270	**0.00**	5.32
T1-80	3	280	**0.00**	5.25
T1-85	3	280	1.75	4.77
T3-100	3	800	**0.00**	7.13
T3-65	3	610	**0.00**	3.29
T3-75	3	650	2.99	7.73
T3-80	3	710	**0.00**	7.51
T3-85	3	710	4.05	6.80
T3-90	3	770	**0.00**	7.91
T3-95	3	760	3.80	5.53
Average			**0.99**	**20.34**

Table 4: 100% range, 0 extra charging stations

In Table 5, the results of our OPHCS algorithm are shown. In this table CS is used to abbreviate "Charging Stations". The results of each individual instance of the OPHCS can be found in Appendix B. For all instances created for Set 1 with one extra hotel and two trips, we report the average gap to the optimal OPHCS solution expressed as a percentage and the computation time in seconds. Starting with the instance with a vehicle range equal to 20% of the length of the known optimal OPHS tour and no extra charging stations, we observe that the average gap is significantly higher than the average gap reported in Table 4. The average gap rises from 0.99 to 15.97. The computation time increases from 20.34 to 189.29. We observe a downward trend in the average gap as the number of extra charging stations increases for the instances with 20% range. This reduction in the average gap is accompanied with an increase in the average computation times. From this pattern, we suspect that the vehicle's range is a prominent limiting factor for these instances.

For the instances with a vehicle range equal to both 40% and 60% of the length of the known optimal OPHS tour, a different pattern in the average gaps is observed. Increasing the number of extra charging stations from zero to two, results in a decrease in the average gaps. However, increasing the number of extra charging stations further to six extra stations leads to a increase in the average gaps. This response is stronger for the instances with a vehicle range equal to 60%. It is worth mentioning that for the instances with a range equal to 60% of the length of the optimal OPHS tour, the average gap is only 1.08%. This is not significantly higher than the average gap for the baseline instances, which equals 0.99%. The average gap for the instances with two extra hotels is with a value of 0.85% even lower than that of the baseline instances. It is likely, that this is due to how the instances were created. Since the optimal charging stations are placed at exactly the location of vertices in the optimal OPHS tour, the algorithm tends to include these points in the tour.

Capacity (%)	20		40		60		**Average**	
Extra CS's	Gap (%)	CPU(s)	Gap (%)	CPU(s)	Gap (%)	CPU(s)	Gap (%)	CPU (s)
0	15.97	189.29	3.32	85.41	1.07	80.24	**6.79**	**118.31**
2	10.95	219.13	2.00	121.93	0.85	125.58	**4.60**	**155.55**
6	7.39	355.02	2.02	276.13	1.31	207.61	**3.57**	**279.59**
Average	**11.44**	**254.48**	**2.45**	**161.16**	**1.08**	**137.81**	**4.99**	**184.48**

Table 5: SVNS results for the OPHCS

On average, it is evident that both changes in the vehicle range and the number of extra charging stations significantly affect the performance of the OPHCS algorithm. From Table 5, we conclude that increasing the vehicle's range leads to a significant decrease in the average gap and computation time. We conclude that decreasing the vehicle's range makes the OPHCS more difficult to solve. The effect of increasing the number of extra charging stations is less evident. The results in Table 5 suggest that increasing the number of extra charging stations leads to a reduction of the average gap. However, the true effect appears to depend on the vehicle's range. Only for the instances with a vehicle range equal to 20%, we observe that increasing the number of extra hotels from two to six results in a decrease in the average gap. This, in combination with the significantly higher average gap than for the other vehicle ranges, suggests that our SVNS algorithm struggles to find the optimal charging stations and their best position in the tour for these instances. This is not very surprising, since these instances require that a larger combination of charging stations is chosen correctly and that those charging stations are inserted at exactly the correct position in the tour. From this, we suspect that treating the charging stations in the same manner as hotels in the SVNS algorithm could result in more favorable results for low vehicle range instances.

To further analyse the performance of our SVNS algorithm for the OPHCS, we compare its performance to that of a simple benchmark algorithm. We compare both algorithms for the ten instances in Table 3 that were not solved to optimality. The choice for this subset of the instances is motivated in Section 5.4. The results of the SVNS algorithm for these instances are shown in Table 6. Table 7 presents the results for the benchmark algorithm. In Appendix C, the separate results for each instance can be found. We observe that the average gap of the benchmark algorithm has a value of 4.23%. This is lower than the average gap of the SVNS

algorithm, which has a value of 4.89%. However, we find that the SVNS algorithm outperforms the benchmark algorithm for all instances with a capacity equal to 40% and 60%. For the instances with a capacity equal to 20%, the benchmark algorithm delivers better results.

Capacity (%)	**20**		**40**		**60**		**Average**	
Extra CS's	Gap (%)	CPU(s)	Gap (%)	CPU(s)	Gap (%)	CPU(s)	Gap (%)	CPU (s)
0	17.38	290.36	2.31	125.20	2.31	141.06	**7.33**	**185.54**
2	8.47	319.95	1.61	184.52	1.45	206.94	**3.84**	**237.14**
6	6.47	510.44	1.87	461.32	2.14	335.45	**3.49**	**435.74**
Average	**10.77**	**373.58**	**1.93**	**257.01**	**1.97**	**227.82**	**4.89**	**286.14**

Table 6: SVNS results for the OPHCS (ten instances)

Capacity (%)	**20**		**40**		**60**		**Average**	
Extra CS's	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)
0	7.17	18.70	3.78	17.94	4.61	20.38	**5.19**	**19.01**
2	7.33	18.43	2.93	19.85	1.98	18.41	**4.08**	**18.90**
6	5.89	18.75	2.07	20.18	2.34	17.13	**3.43**	**18.69**
Average	**6.80**	**18.63**	**2.93**	**19.32**	**2.98**	**18.64**	**4.23**	**18.86**

Table 7: Benchmark algorithm results for the OPHCS (ten instances)

From the fact that the benchmark algorithm outperforms our SVNS approach for the 20% range instances, we suspect that for low vehicle ranges another implementation is likely to deliver better results. From Table 5, we already noticed that increasing the number of extra charging stations from two to six leads to a lower average gap for the 20% range instances. This indicates that the algorithm has more difficulty to find the optimal charging stations and their best positions for those instances than for the 40% and 60% range instances.

7 Conclusion

In this paper, we introduce the orienteering problem with hotel and charging station selection (OPHCS). We focus on presenting an algorithm for this problem. The OPHCS is closely related to the orienteering problem with hotel selection (OPHS), where the OPHS can be seen as a OPHCS with infinite vehicle range. Efficient algorithms for the OPHS exist. We build upon the SVNS algorithm of Divsalar et al. (2013) and apply this new algorithm to the OPHCS. We examine the performance of the algorithm and investigate how the performance is affected by the range of the vehicle and the number of extra charging stations in the instance.

All in all, our SVNS algorithm for the OPHCS achieves promising results. We found an average gap to the optimal solutions of 4.99% and an average computation time of 184.48 seconds. The proposed algorithm demonstrates great efficiency by solving all the considered OPHCS instances within a maximum of fifteen minutes. In contrast, CPLEX failed to achieve optimality for any of those instances within an hour of computation time. We find that for an increasing vehicle range the average gap to the optimal solution decrease significantly. For high vehicle ranges, the average gaps approach those of our implementation of the SVNS algorithm for the OPHS. We observe that for the instances with 60% range and two extra hotels, the

average gap is even lower than that for the instances with 100% range and no included charging stations.

However, we do believe that improvements can be made to our algorithm. To ensure that our algorithm is applicable for practical scenarios, such as trip planning for electric trucks, improvements with respect to both the average gaps and the computation times are essential. Compared to the performance of our OPHS algorithm, we observed a significant increase in all computation times. From this, we conclude that the OPHCS is more difficult to solve than the OPHS. Even for the set that is solved within the shortest time, computation times are almost four times higher than for the OPHS. We found that the algorithm's performance is significantly affected by the vehicle range. A lower vehicle range, and thus more charging stations to be included in the route, results in higher gaps and increased computation times. These results implies that the OPHCS becomes more difficult with a lower vehicle range. In addition, we found that for the instances with vehicle ranges equal to 0.20% of the optimal OPHS solution, increasing the number of extra charging stations leads to a lower average gap. This implies that the algorithm struggles to select the tour with the optimal charging stations. One more difficulty with the presented algorithm are the long computation times.

It is worth mentioning that the computation time of our algorithm can most likely be improved with more efficient programming. This is likely to be one of the reasons for why our implementation of the SVNS for the OPHS has a longer average running time that that of Divsalar et al. (2013). Another reason for these higher running times could be the descent heuristic used in the local search method. We use a relatively time consuming descent heuristic, so it is possible that Divsalar et al. (2013) uses a less time consuming heuristic like the best-improvement or first-descent heuristic. Since the proposed algorithm for the OPHCS builds upon that for the OPHS, it is most likely that the running times for the OPHCS are also dependent on the running times for the OPHS.

We have contributed to research in this area by introducing a new variant of the OPHS, called the OPHCS, and creating 1270 new benchmark instances for this problem. We provide an SVNS algorithm for this problem that obtains high quality solutions within acceptable computation times, especially for instances with a higher vehicle range. Furthermore, by critically evaluating the proposed SVNS algorithm, we have provided interesting suggestions for future research. We found that for instances with low vehicle ranges, our algorithm struggles to find the optimal charging stations and their optimal positions in the tour. Since charging stations need to be at exactly the correct position in the tour to find the optimal tour, treating them in a fashion comparable to how hotels are treated in the SVNS algorithm might deliver more favorable results. In addition, future research could be on improving the local search moves used in the descent phase and on what effect the chosen descent heuristic has on the performance of the algorithm. Another interesting extension would be to assign scores to charging stations, which is relevant for many practical purposes. Since people have to spend more time at a charging stations than at ordinary gas stations, they assign more value to the available facilities such as restaurants. In addition, some car manufacturers offer their customers so called superchargers that allow for faster recharges. Those customers are likely to value these charging stations more than ordinary charging stations.

References

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.

Gunawan, A., Lau, H. C. & Vansteenwegen, P. (2016). Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, *255*(2), 315–332.

Hansen, P. & Mladenovic, N. (2003). A tutorial on variable neighborhood search. *Les Cahiers du GERAD ISSN*, *711*, 2440.

Hansen, P., Mladenović, N., Brimberg, J. & Pérez, J. A. M. (2019). *Variable neighborhood search*. Springer.

Hansen, P., Mladenović, N. & Moreno Pérez, J. A. (2010). Variable neighbourhood search: methods and applications. *Annals of Operations Research*, *175*, 367–407.

Hiermann, G., Puchinger, J., Ropke, S. & Hartl, R. F. (2016). The electric fleet size and mix vehicle routing problem with time windows and recharging stations. *European Journal of Operational Research*, *252*(3), 995–1018.

IEA. (2022). *Trends in electric heavy-duty vehicles – global ev outlook 2022 – analysis*. Retrieved from https://www.iea.org/reports/global-ev-outlook-2022/trends-in-electric-heavy-duty-vehicles

Keller, C. P. (1989). Algorithms to solve the orienteering problem: A comparison. *European Journal of Operational Research*, *41*(2), 224–231.

Keskin, M. & Çatay, B. (2016). Partial recharge strategies for the electric vehicle routing problem with time windows. *Transportation Research part C: Emerging Technologies*, *65*, 111–127.

Roberti, R. & Wen, M. (2016). The electric traveling salesman problem with time windows. *Transportation Research Part E: Logistics and Transportation Review*, *89*, 32–52.

Sohrabi, S., Ziarati, K. & Keshtkaran, M. (2020). A greedy randomized adaptive search procedure for the orienteering problem with hotel selection. *European Journal of Operational Research*, *283*(2), 426–440.

Tol, D., Frateur, T., Verbeek, M., Riemersma, I. & Mulder, H. (2022). *Techno-economic uptake potential of zero-emission trucks in europe* (Tech. Rep.). Tech. rep.

Toledo, A., Riff, M.-C. & Neveu, B. (2020). A hyper-heuristic for the orienteering problem with hotel selection. *IEEE Access*, *8*, 1303-1313. doi: 10.1109/ACCESS.2019.2960492

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.

Zhang, S., Chen, M., Zhang, W. & Zhuang, X. (2020). Fuzzy optimization model for electric vehicle routing problem with time windows and recharging stations. *Expert systems with applications*, *145*, 113123.

Zhang, S., Gajpal, Y., Appadoo, S. & Abdulkader, M. (2018). Electric vehicle routing problem with recharging stations for minimizing energy consumption. *International Journal of Production Economics*, *203*, 404–413.

A OPHS results

Instances	SVNS	TNFC	Gap (%)	CPU (s)
100-30	2	173	0.00	0.68
100-35	1	241	0.00	1.27
100-40	2	299	0.00	1.42
100-45	3	367	0.00	3.86
102-50	3	181	0.00	0.34
102-60	3	243	0.00	1.39
64-45	3	816	0.00	8.49
64-50	3	876	2.67	10.46
64-55	3	966	1.83	17.58
64-60	3	1050	1.13	24.12
64-65	3	1116	0.00	26.80
64-70	3	1170	1.52	35.20
64-75	3	1230	0.49	35.37
64-80	3	1284	0.00	35.05
66-125	3	1655	0.90	51.06
66-130	3	1680	0.00	60.63
66-40	3	570	0.87	2.25
66-45	3	645	0.77	4.00
66-50	3	705	3.42	6.21
66-55	3	825	0.00	9.12
66-60	3	890	2.73	8.95
T1-65	3	240	0.00	1.29
T1-70	3	260	0.00	1.57
T1-73	3	250	5.66	3.90
T1-75	3	270	0.00	3.96
T1-80	3	280	0.00	2.50
T1-85	3	285	0.00	3.06
T3-100	3	800	0.00	4.99
T3-105	3	800	0.00	4.35
T3-65	3	610	0.00	3.43
T3-75	3	650	2.99	4.99
T3-80	3	710	0.00	4.37
T3-85	3	720	2.70	6.69
T3-90	3	770	0.00	4.26
T3-95	3	780	1.27	5.04
Average			**0.83**	**11.39**

Table 8: Set1: 1 extra hotel - 2 trips

Instances	SVNS	TNFC	Gap (%)	CPU (s)
100-30	1	173	0.00	0.18
100-35	2	241	0.00	1.92
100-40	2	299	0.00	0.87
100-45	2	367	0.00	2.61
102-50	12	181	0.00	0.66
102-60	12	243	0.00	0.56
64-45	12	816	0.00	10.19
64-50	16	876	2.67	7.38
64-55	16	972	1.22	12.37
64-60	16	1062	0.00	15.37
64-65	16	1116	0.00	21.37
64-70	16	1170	1.52	37.02
64-75	16	1218	1.46	29.55
64-80	16	1260	1.87	30.85
66-125	16	1670	0.00	51.78
66-130	16	1675	0.30	64.67
66-40	16	570	0.87	2.73
66-45	16	645	0.77	2.09
66-50	16	715	2.05	3.51
66-55	16	825	0.00	4.60
66-60	16	910	0.55	6.28
T1-65	16	240	0.00	1.85
T1-70	16	260	0.00	2.31
T1-73	16	265	0.00	1.85
T1-75	16	270	0.00	2.26
T1-80	16	280	0.00	2.59
T1-85	16	285	0.00	3.80
T3-100	16	800	0.00	5.12
T3-105	16	800	0.00	8.16
T3-65	16	610	0.00	3.66
T3-75	16	670	0.00	5.94
T3-80	16	710	0.00	4.79
T3-85	16	740	0.00	4.44
T3-90	16	770	0.00	11.41
T3-95	16	790	0.00	4.71
Average			**0.38**	**10.56**

Table 9: Set1: 2 extra hotels - 3 trips

Instances	SVNS	TNFC	Gap (%)	CPU (s)
100-30	2	173	0.00	0.92
100-35	1	241	0.00	0.32
100-40	1	299	0.00	0.58
100-45	2	367	0.00	0.95
102-50	33	181	0.00	0.22
102-60	60	243	0.00	1.30
64-45	44	810	0.74	2.66
64-50	72	870	3.33	9.51
64-55	100	978	0.61	12.02
64-60	100	1044	1.69	16.38
64-65	125	1116	0.00	26.53
64-70	125	1152	3.03	30.08
64-75	125	1206	2.43	65.08
64-80	125	1272	0.93	51.29
66-125	125	1650	1.20	71.25
66-130	125	1670	0.60	75.94
66-40	107	570	0.87	1.84
66-45	88	645	0.77	3.51
66-50	107	715	2.05	4.82
66-55	107	825	0.00	7.65
66-60	107	910	0.55	8.65
T1-65	107	240	0.00	1.03
T1-70	107	260	0.00	1.41
T1-73	107	265	0.00	1.70
T1-75	107	270	0.00	3.27
T1-80	125	280	0.00	3.36
T1-85	125	285	0.00	3.16
T3-100	125	770	3.75	6.75
T3-105	125	790	1.25	9.10
T3-65	100	610	0.00	3.40
T3-75	125	650	2.99	5.52
T3-80	107	710	0.00	3.53
T3-85	107	740	0.00	5.80
T3-90	100	720	6.49	5.35
T3-95	100	790	0.00	9.50
Average			**0.95**	**12.98**

Table 10: Set1: 3 extra hotel - 4 trips

Instances	SVNS	TNFC	Gap (%)	CPU (s)
100-30	1	173	0.00	0.16
100-35	2	241	0.00	0.82
100-40	2	299	0.00	0.97
100-45	2	367	0.00	2.12
102-50	14	181	0.00	0.32
102-60	17	243	0.00	1.99
64-45	42	816	0.00	7.51
64-50	49	876	2.67	9.54
64-55	49	978	0.61	16.06
64-60	49	1038	2.26	20.61
64-65	49	1116	0.00	26.85
64-70	49	1170	1.52	30.02
64-75	49	1224	0.97	38.46
64-80	49	1284	0.00	50.72
66-125	49	1670	0.00	64.92
66-130	49	1675	0.30	71.96
66-40	45	570	0.87	2.14
66-45	47	645	0.77	3.22
66-50	49	715	2.05	5.18
66-55	49	825	0.00	6.36
66-60	49	910	0.55	7.56
T1-65	49	240	0.00	1.95
T1-70	49	260	0.00	2.19
T1-73	49	265	0.00	2.49
T1-75	49	270	0.00	3.51
T1-80	49	280	0.00	3.19
T1-85	49	285	0.00	3.28
T3-100	49	800	0.00	5.73
T3-105	49	800	0.00	6.75
T3-65	49	610	0.00	3.10
T3-75	49	670	0.00	4.22
T3-80	49	710	0.00	4.99
T3-85	49	740	0.00	5.31
T3-90	49	770	0.00	9.26
T3-95	790	790	0.00	5.85
Average			**0.36**	**12.27**

Table 11: Set2: 5 extra hotels - 3 trips

Instances	TNFC	SVNS	Gap (%)	CPU (s)
100-30	2	173	0.00	0.16
100-35	1	241	0.00	0.37
100-40	1	299	0.00	0.63
100-45	2	367	0.00	1.24
102-50	60	181	0.00	0.30
102-60	105	243	0.00	1.24
64-45	424	810	0.74	12.29
64-50	252	870	3.33	25.52
64-55	378	978	0.61	32.66
64-60	448	1056	0.56	44.97
64-65	448	1116	0.00	56.32
64-70	512	1164	2.02	54.77
64-75	512	1224	0.97	68.98
64-80	512	1266	1.40	107.54
66-125	512	1650	1.20	111.55
66-130	512	1670	0.60	149.08
66-40	512	570	0.87	3.41
66-45	302	645	0.77	4.92
66-50	228	715	2.05	10.86
66-55	357	825	0.00	12.25
66-60	400	910	0.55	18.65
T1-65	482	220	8.33	3.33
T1-70	482	240	7.69	3.80
T1-73	482	245	7.55	5.66
T1-75	482	270	0.00	7.11
T1-80	512	280	0.00	8.46
T1-85	512	285	0.00	7.66
T3-100	512	770	3.75	15.38
T3-105	512	790	1.25	16.29
T3-65	448	610	0.00	7.85
T3-75	512	620	7.46	11.01
T3-80	482	710	0.00	9.78
T3-85	482	660	10.81	13.73
T3-90	448	720	6.49	13.99
T3-95	448	790	0.00	12.76
Average			**1.97**	**24.41**

Table 12: Set2: 6 extra hotels - 4 trips

Instances	SVNS	TNFC	Gap (%)	CPU (s)
100-100	1296	774	1.02	123.59
100-110	1728	815	2.40	150.28
100-120	1728	878	1.79	174.53
100-130	1551	903	5.54	309.71
100-140	1551	935	7.70	352.37
100-150	1728	1002	5.20	362.29
100-160	1728	1044	6.28	465.56
100-170	1728	1093	6.10	689.80
100-180	1728	1148	4.41	675.93
100-190	1728	1186	3.89	749.38
100-200	1728	1214	3.73	868.19
100-210	1728	1243	3.19	1307.61
100-240	1728	1298	0.61	1276.91
100-50	39	408	0.97	2.21
100-60	161	504	0.00	7.75
100-70	276	587	0.51	26.07
100-80	892	652	0.00	43.49
100-90	1220	706	2.62	75.64
64-75	1728	1218	1.46	58.49
64-80	1728	1260	1.87	77.90
66-125	1728	1650	1.20	128.46
66-130	1728	1675	0.30	119.82
Average			**2.76**	**365.73**

Table 13: Set3: 10 extra hotel - 4 trips

Instances	SVNS	TNFC	Gap (%)	CPU (s)
100-100	15272	697	10.87	76.80
100-110	10991	794	4.91	97.58
100-120	15423	812	9.17	123.53
100-130	24210	850	11.09	167.17
100-140	27440	930	8.19	249.64
100-150	38416	1008	4.64	242.86
100-160	35672	992	10.95	339.74
100-170	38416	1075	7.65	375.70
100-180	38416	1146	4.58	552.54
100-190	38416	1161	5.92	615.23
100-200	38416	1211	3.97	579.67
100-210	38416	1235	3.82	724.82
100-240	38416	1304	0.15	962.03
100-50	26	393	4.61	1.06
100-60	442	504	0.00	9.26
100-70	1273	590	0.00	18.71
100-80	3053	604	7.36	31.40
100-90	5240	663	8.55	61.02
64-75	32928	1194	3.40	51.35
64-80	32928	1254	2.34	65.12
66-125	38416	1630	2.40	108.00
66-130	38416	1660	1.19	106.49
Average			**5.26**	**252.71**

Table 14: Set3: 12 extra hotels - 5 trips

Set4: 3-2/3	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	0.856
100-20-3	357	376	-	19	368	-3.08	2.13	-	2.449
100-25-2	-	-	385	5	385	-	-	0.00	2.954
100-25-3	495	568	-	25	522	-5.45	8.10	-	5.998
102-35-2	-	-	157	2	151	-	-	3.82	0.341
102-35-3	230	380	-	7	324	-40.87	14.74	-	0.641
102-40-2	-	-	210	2	210	-	-	0.00	0.81
102-40-3	299	493	-	7	389	-30.10	21.10	-	2.32
102-45-2	-	-	266	2	254	-	-	4.51	1.324
102-45-3	356	579	-	7	425	-19.38	26.60	-	3.325
Average						**-19.78**	**14.53**	**1.67**	**2.10**

Table 15: SET4: 3 extra hotels - 2 trips/3 trips

B OPHCS SVNS results

Instance	TNFC	SVNS	Gap (%)	CPU (s)
100-35	1	173	28.22	95.07
100-40	1	299	**0.00**	92.33
100-45	1	367	0.00	134.21
64-45	3	510	37.50	76.04
64-50	3	606	32.67	164.70
64-55	3	750	23.78	187.13
64-60	3	798	24.86	303.29
64-65	3	1086	2.69	346.90
64-70	3	1134	4.55	492.79
64-75	3	1176	4.85	464.50
64-80	3	1230	4.21	506.53
66-125	3	1655	0.90	852.75
T1-65	3	145	39.58	20.73
T1-70	3	205	21.15	71.46
T1-73	3	185	30.19	78.72
T1-75	3	225	16.67	46.04
T1-80	3	240	14.29	57.14
T1-85	3	225	21.05	46.81
T3-100	3	700	12.50	57.95
T3-65	3	590	3.28	86.11
T3-75	3	470	29.85	36.63
T3-80	3	630	11.27	77.03
T3-85	3	660	10.81	166.10
T3-90	3	670	12.99	114.32
T3-95	3	700	11.39	157.03
Average			**15.97**	**189.29**

Table 16: 20% range, no extra charging stations

Instance	TNFC	SVNS	Gap (%)	CPU (s)
100-35	1	173	28.22	118.11
100-40	1	299	**0.00**	110.60
100-45	1	367	**0.00**	95.11
64-45	3	708	13.24	100.82
64-50	3	750	16.67	195.59
64-55	3	948	3.66	347.59
64-60	3	948	10.73	438.11
64-65	3	1032	7.53	376.48
64-70	3	1152	3.03	500.72
64-75	3	1224	0.97	403.35
64-80	3	1254	2.34	584.25
66-125	3	1645	1.50	763.81
T1-65	3	180	25.00	41.61
T1-70	3	205	21.15	85.97
T1-73	3	215	18.87	54.98
T1-75	3	220	18.52	55.63
T1-80	3	240	14.29	119.69
T1-85	3	240	15.79	69.10
T3-100	3	710	11.25	201.66
T3-65	3	590	3.28	119.08
T3-75	3	580	13.43	160.73
T3-80	3	600	15.49	81.38
T3-85	3	660	10.81	132.12
T3-90	3	670	12.99	119.22
T3-95	3	750	5.06	202.57
Average			**10.95**	**219.13**

Table 17: 20% range. 2 extra charging stations

Instance	TNFC	SVNS	Gap (%)	CPU (s)
100-35	1	173	28.22	47.18
100-40	1	299	**0.00**	282.12
100-45	1	286	22.07	42.71
64-45	3	798	2.21	236.99
64-50	3	726	19.33	346.62
64-55	3	972	1.22	395.98
64-60	3	1014	4.52	590.21
64-65	3	1116	**0.00**	788.53
64-70	3	1140	4.04	777.26
64-75	3	1224	0.97	900.15
64-80	3	1260	1.87	732.03
66-125	3	1655	0.90	913.32
T1-65	3	230	4.17	64.88
T1-70	3	220	15.38	146.18
T1-73	3	220	16.98	223.11
T1-75	3	245	9.26	143.37
T1-80	3	250	10.71	198.60
T1-85	3	265	7.02	147.45
T3-100	3	750	6.25	311.00
T3-65	3	590	3.28	152.72
T3-75	3	600	10.45	201.00
T3-80	3	670	5.63	226.22
T3-85	3	740	**0.00**	270.82
T3-90	3	740	3.90	251.08
T3-95	3	740	6.33	485.93
Average			**7.39**	**355.02**

Table 18: 20% range, 6 extra charging stations

Instance	TNFC	SVNS	Gap (%)	CPU (s)
100-30	1	173	**0.00**	12.93
100-35	1	206	14.52	28.65
100-40	1	299	**0.00**	22.96
100-45	2	367	**0.00**	26.82
64-45	3	666	18.38	43.59
64-50	3	858	4.67	38.45
64-55	3	954	3.05	93.74
64-60	3	1050	1.13	114.49
64-65	3	1116	**0.00**	207.96
64-70	3	1170	1.52	186.71
64-75	3	1230	0.49	274.44
64-80	3	1284	**0.00**	318.28
66-125	3	1655	0.90	381.50
T1-65	3	240	**0.00**	12.62
T1-70	3	260	**0.00**	31.21
T1-73	3	250	5.66	18.99
T1-75	3	270	**0.00**	28.88
T1-80	3	275	1.79	40.76
T1-85	3	280	1.75	34.70
T3-100	3	750	6.25	39.21
T3-65	3	560	8.20	16.78
T3-75	3	640	4.48	50.04
T3-80	3	650	8.45	35.76
T3-85	3	740	**0.00**	56.12
T3-90	3	740	3.90	67.57
T3-95	3	780	1.27	37.49
Average			**3.32**	**85.41**

Table 19: 40% range, no extra charging stations

Instance	TNFC	SVNS	Gap (%)	CPU (s)
100-30	1	173	**0.00**	7.85
100-35	1	206	14.52	13.69
100-40	1	299	**0.00**	12.49
100-45	2	367	**0.00**	13.77
64-45	3	816	**0.00**	88.16
64-50	3	882	2.00	120.99
64-55	3	972	1.22	179.53
64-60	3	1062	**0.00**	187.54
64-65	3	1116	**0.00**	382.22
64-70	3	1170	1.52	295.83
64-75	3	1230	0.49	307.42
64-80	3	1284	**0.00**	411.29
66-125	3	1665	0.30	563.11
T1-65	3	235	2.08	19.04
T1-70	3	260	0.00	33.61
T1-73	3	245	7.55	26.77
T1-75	3	270	**0.00**	35.63
T1-80	3	270	3.57	49.39
T1-85	3	285	**0.00**	32.29
T3-100	3	760	5.00	48.25
T3-65	3	570	6.56	31.04
T3-75	3	650	2.99	51.33
T3-80	3	680	4.23	45.27
T3-85	3	740	**0.00**	51.75
T3-90	3	770	**0.00**	100.95
T3-95	3	790	**0.00**	**60.93**
Average			**2.00**	**121.93**

Table 20: 40% range, 2 extra charging stations

Instance	TNFC	SVNS	Gap (%)	CPU (s)
100-30	1	173	**0.00**	25.87
100-35	1	206	14.52	20.41
100-40	1	299	**0.00**	45.24
100-45	2	343	6.54	42.80
64-45	3	816	**0.00**	190.09
64-50	3	882	2.00	299.71
64-55	3	972	1.22	413.77
64-60	3	1050	1.13	500.74
64-65	3	1116	**0.00**	592.53
64-70	3	1170	1.52	1315.84
64-75	3	1230	0.49	715.69
64-80	3	1284	**0.00**	763.39
66-125	3	1660	0.60	817.37
T1-65	3	240	**0.00**	64.96
T1-70	3	260	**0.00**	70.09
T1-73	3	245	7.55	106.88
T1-75	3	270	**0.00**	87.67
T1-80	3	270	3.57	102.08
T1-85	3	285	**0.00**	79.11
T3-100	3	770	3.75	130.56
T3-65	3	610	**0.00**	70.42
T3-75	3	650	2.99	128.92
T3-80	3	700	1.41	131.32
T3-85	3	740	**0.00**	155.20
T3-90	3	740	3.90	149.62
T3-95	3	780	1.27	159.10
Average			**2.02**	**276.13**

Table 21: 40% range, 6 extra charging stations

Instance	TNFC	SVNS	Gap (%)	CPU (s)
100-30	1	173	**0.00**	3.99
100-35	1	241	**0.00**	8.97
100-40	2	299	**0.00**	6.40
100-45	3	367	**0.00**	9.46
64-45	3	816	**0.00**	26.87
64-50	3	876	2.67	31.70
64-55	3	978	0.61	95.00
64-60	3	1050	1.13	101.31
64-65	3	1116	**0.00**	152.23
64-70	3	1170	1.52	238.91
64-75	3	1218	1.46	231.62
64-80	3	1284	**0.00**	242.69
66-125	3	1655	0.90	567.69
T1-65	3	240	**0.00**	8.39
T1-70	3	260	**0.00**	24.61
T1-73	3	245	7.55	31.62
T1-75	3	270	**0.00**	30.89
T1-80	3	270	3.57	16.61
T1-85	3	285	**0.00**	16.92
T3-100	3	800	**0.00**	25.85
T3-65	3	610	**0.00**	31.13
T3-75	3	630	5.97	24.99
T3-80	3	710	**0.00**	40.70
T3-85	3	740	**0.00**	55.36
T3-90	3	760	1.30	29.79
T3-95	3	780	1.27	32.46
Average			**1.07**	**80.24**

Table 22: 60% range, no extra charging stations

Instance	TNFC	SVNS	Gap (%)	CPU (s)
100-30	1	173	**0.00**	2.47
100-35	1	241	**0.00**	13.70
100-40	2	299	**0.00**	31.66
100-45	3	367	**0.00**	49.07
64-45	3	798	2.21	75.75
64-50	3	882	2.00	132.62
64-55	3	966	1.83	181.49
64-60	3	1062	**0.00**	232.19
64-65	3	1116	**0.00**	209.17
64-70	3	1170	1.52	278.14
64-75	3	1230	0.49	308.47
64-80	3	1284	**0.00**	382.01
66-125	3	1670	**0.00**	693.29
T1-65	3	240	**0.00**	26.12
T1-70	3	260	**0.00**	24.28
T1-73	3	250	5.66	30.18
T1-75	3	270	**0.00**	26.88
T1-80	3	280	**0.00**	30.33
T1-85	3	285	**0.00**	23.04
T3-100	3	790	1.25	66.67
T3-65	3	610	**0.00**	60.90
T3-75	3	650	2.99	81.32
T3-80	3	690	2.82	96.21
T3-85	3	740	**0.00**	70.26
T3-90	3	760	1.30	77.50
T3-95	3	790	**0.00**	61.44
Average			**0.85**	**125.58**

Table 23: 60% range, 2 extra charging stations

Instance	TNFC	SVNS	Gap (%)	CPU (s)
100-30	1	173	**0.00**	1.56
100-35	1	241	**0.00**	13.59
100-40	2	299	**0.00**	33.30
100-45	3	367	**0.00**	55.32
64-45	3	804	1.47	184.11
64-50	3	882	2.00	333.12
64-55	3	972	1.22	487.23
64-60	3	1062	**0.00**	541.72
64-65	3	1116	**0.00**	474.20
64-70	3	1170	1.52	450.16
64-75	3	1230	0.49	578.00
64-80	3	1284	**0.00**	541.48
66-125	3	1665	0.30	580.06
T1-65	3	240	**0.00**	62.47
T1-70	3	255	1.92	56.83
T1-73	3	245	7.55	64.86
T1-75	3	270	**0.00**	51.98
T1-80	3	265	5.36	69.15
T1-85	3	285	**0.00**	104.43
T3-100	3	800	**0.00**	115.22
T3-65	3	610	**0.00**	50.22
T3-75	3	650	2.99	120.42
T3-80	3	710	**0.00**	117.37
T3-85	3	710	4.05	99.50
T3-90	3	740	3.90	112.03
T3-95	3	780	1.27	99.45
Average			**1.31**	**207.61**

Table 24: 60% range, 6 extra charging stations

C OPHCS benchmark algorithm results

Instances	TNFC	SVNS	Gap (%)	CPU (s)
64-50	3	786	12.67	12.246
64-55	3	936	4.88	13.678
64-60	3	1008	5.08	21.546
64-70	3	1104	7.07	28.919
64-75	3	1182	4.37	29.36
66-125	3	1625	2.69	65.065
T1-73	3	240	9.43	2.143
T3-75	3	570	14.93	6.562
T3-85	3	690	6.76	4.622
T3-95	3	760	3.80	2.834
Average			**7.17**	**18.70**

Table 25: Benchmark: 20% range, no extra charging stations

Instances	TNFC	SVNS	Gap (%)	CPU (s)
64-50	3	822	8.67	10.043
64-55	3	960	2.44	15.731
64-60	3	1008	5.08	17.398
64-70	3	1110	6.57	31.298
64-75	3	1128	8.74	30.027
66-125	3	1625	2.69	64.006
T1-73	3	225	15.09	2.724
T3-75	3	580	13.43	5.954
T3-85	3	690	6.76	4.642
T3-95	3	760	3.80	2.456
Average			**7.33**	**18.43**

Table 26: Benchmark: 20% range, 2 extra charging stations

Instances	TNFC	SVNS	Gap (%)	CPU (s)
64-50	3	810	10.00	11.55
64-55	3	966	1.83	17.117
64-60	3	1014	4.52	20.426
64-70	3	1146	3.54	27.108
64-75	3	1194	3.40	28.172
66-125	3	1655	0.90	61.733
T1-73	3	225	15.09	3.189
T3-75	3	600	10.45	6.239
T3-85	3	700	5.41	8.89
T3-95	3	760	3.80	3.038
Average			**5.89**	**18.75**

Table 27: Benchmark: 20% range, 6 extra charging stations

Instances	TNFC	SVNS	Gap (%)	CPU (s)
64-50	3	876	2.67	11.342
64-55	3	966	1.83	18.914
64-60	3	1014	4.52	17.781
64-70	3	1152	3.03	31.103
64-75	3	1194	3.40	29.687
66-125	3	1640	1.80	56.919
T1-73	3	250	5.66	2.243
T3-75	3	640	4.48	3.403
T3-85	3	700	5.41	4.183
T3-95	3	750	5.06	3.792
Average			**3.78**	**17.94**

Table 28: Benchmark: 40% range, no extra charging stations

Instances	TNFC	SVNS	Gap (%)	CPU (s)
64-50	3	876	2.67	13.345
64-55	3	966	1.83	17.429
64-60	3	1050	1.13	20.24
64-70	3	1170	1.52	32.882
64-75	3	1188	3.88	40.631
66-125	3	1640	1.80	56.826
T1-73	3	250	5.66	3.548
T3-75	3	650	2.99	3.495
T3-85	3	710	4.05	6.787
T3-95	3	760	3.80	3.343
Average			**2.93**	**19.85**

Table 29: Benchmark: 40% range, 2 extra charging stations

Instances	TNFC	SVNS	Gap (%)	CPU (s)
64-50	3	876	2.67	12.669
64-55	3	966	1.83	16.723
64-60	3	1050	1.13	24.491
64-70	3	1170	1.52	35.609
64-75	3	1236	**0.00**	31.777
66-125	3	1655	0.90	65.387
T1-73	3	250	5.66	2.022
T3-75	3	650	2.99	4.254
T3-85	3	720	2.70	4.909
T3-95	3	780	1.27	3.94
Average			**2.07**	**20.18**

Table 30: Benchmark: 40% range, 6 extra charging stations

Instances	TNFC	SVNS	Gap (%)	CPU (s)
64-50	3	876	2.67	11.593
64-55	3	930	5.49	16.657
64-60	3	1014	4.52	19.028
64-70	3	1104	7.07	33.89
64-75	3	1206	2.43	41.148
66-125	3	1605	3.89	65.866
T1-73	3	250	5.66	3.024
T3-75	3	600	10.45	4.574
T3-85	3	720	2.70	4.978
T3-95	3	780	1.27	3.006
Average			**4.61**	**20.38**

Table 31: Benchmark: 60% range, no extra charging stations

Instances	TNFC	SVNS	Gap (%)	CPU (s)
64-50	3	876	2.67	11.881
64-55	3	966	1.83	14.061
64-60	3	1050	1.13	16.756
64-70	3	1170	1.52	29.216
64-75	3	1236	**0.00**	33.638
66-125	3	1670	**0.00**	64.082
T1-73	3	250	5.66	3.038
T3-75	3	650	2.99	3.134
T3-85	3	720	2.70	5.383
T3-95	3	780	1.27	2.881
Average			**1.98**	**18.41**

Table 32: Benchmark: 60% range, 2 extra charging stations

Instances	TNFC	SVNS	Gap (%)	CPU (s)
64-50	3	876	2.67	13.254
64-55	3	966	1.83	14.405
64-60	3	1014	4.52	18.731
64-70	3	1170	1.52	27.472
64-75	3	1236	**0.00**	28.202
66-125	3	1665	0.30	53.381
T1-73	3	250	5.66	2.99
T3-75	3	650	2.99	3.435
T3-85	3	720	2.70	6.325
T3-95	3	780	1.27	3.082
Average			**2.34**	**17.13**

Table 33: Benchmark: 60% range, 6 extra charging stations

D Structure of the code

In this section of the appendix, we give a short overview of the structure of our code. All code is written in Java and is included in the provided zip file. In the zip-file, we also include a manual that explains how the presented results can be obtained.

D.1 Code for the OPHS

We solve the OPHS with a skewed variable neighborhood search. The variable neighborhood search is programmed in the $MainVNS$ class. To perform the described shake moves, this class uses the $ShakeMachine$ class. We construct a new $ShakeMachine$ object, which performs both the vertices shakes and the hotels-shake. After all shake moves, the solution is improved by means of local search. The methods for the local search are introduced in the $LocalSearch$ class. The $Vertex$ class allows us to create objects that represent hotels and vertices with coordinates and a score associated to them. We use objects of the $Tour$ class, to represent the OPHS tour. The $BigReader$ class allows us to run the SVNS algorithm for a whole set of instances.

D.2 Code for generating the OPHCS instances

Since the OPHCS is a new problem, no benchmark instances are available for it. We create our own benchmark instances based on the available OPHS instances of Divsalar et al. (2013). For this, we make use of the $InstanceCreator$ class. In the main method of this class, the user specifies for which instances new OPHCS instances should be created. With the $MakeNewInstance$ method, the user creates new OPHCS instances for capacities equal to $20\%, 40\%$ and 60% of the known length of the optimal OPHS tour. These new instances do not contain extra charging stations. To include extra charging stations, the user runs the $MakeNewInstanceExtraCS$ method with as its input the number of extra charging stations to include.

D.3 Code for the OPHCS

We solve the OPHCS with a skewed variable neighborhood search. The variable neighborhood search is programmed in the $MainVNSChargingStations$ class. To perform the described shake moves, this class uses the $ShakeMachineChargingStations$ class. We construct a new $ShakeMachineChargingStations$ object, which performs both the vertices shakes and the hotels-shake. After all shake moves, the solution is improved by means of local search. The methods for the local search are introduced in the $LocalSearchChargingStations$ class. The $Vertex$ class allows us to create objects that represent hotels and vertices with coordinates and a score associated to them. We use objects of the $TourChargingStation$ class, to represent the OPHCS tour.The $BigReaderOPHCS$ class allows us to run the SVNS algorithm for a whole set of instances.

D.4 Code for the benchmark algorithm

Our benchmark algorithm starts with solving the OPHS. The variable neighborhood search and the repair is programmed in the $MainVNSRepair$ class. The classes used by the $MainVNSRepair$ class are identical to those that the SVNS class for the OPHS uses. The $BigReaderBenchmark$ class allows us to run the SVNS algorithm for a whole set of instances.