Erasmus University Rotterdam
Erasmus School of Economics
Bachelor Thesis - BSc^2 in Econometrics and Economics
Major Analytics and Operations Research in Logistics

The Effectiveness of an Artificial Bee Colony for solving the Team Orienteering Problem with Hotel Selection

E.J. van der Bij (527535)

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

Abstract

In this paper, we study a team orienteering problem with hotel selection (TOPHS), which is a multi-tour version of the orienteering problem with hotel selection (OPHS) by Divsalar, Vansteenwegen and Cattrysse (2013). We introduce a MIP formulation, and describe a procedure to generate instances with known objective values from instances of the team orienteering problem (TOP).

As this problem is NP-hard, we also propose three heuristics. The first heuristic, called TSVNS, is a skewed variable neighborhood search, which has shown to be successful for the OPHS. The second heuristic, TABC, is an adaption of an artificial bee colony (ABC) optimization procedure originally designed for the TOP. The third heuristic, TAV, is also based on ABC, but uses the neighborhoods from the TSVNS heuristic.

We find that the TSVNS is the most promising algorithm, providing the lowest average gaps and a reasonable computation time. For larger instances, however, its execution time increases substantially. The TAV demonstrates potential advantages for such larger instances, as it has lower execution times than TSVNS but maintains an acceptable gap. The TABC does not perform as well, both in terms of average gap and running time.

1 Introduction

In this research, we study a multi-tour version of the Orienteering Problem with Hotel Selection (OPHS) by Divsalar et al. (2013). In the OPHS, the goal is to determine a tour, defined as a set of D connected trips d, which maximizes the sum of scores S_i of nodes i visited during those trips. One trip starts in a hotel, then visits a number of nodes (Points of Interest, or POIs), and then goes to another or the same hotel. The next trip should start from the hotel where the previous trip ended at. Each trip's length is constrained by a time limit T_d. For a more detailed explanation, we refer to Divsalar et al. (2013).

Throughout this thesis, we use the term *trip* to describe an itinerary that starts in one hotel and ends in another, without visiting intermediate hotels. We use *tour* to describe an ordered set of trips, that together form a solution to the problem. We use *node* and *POI* interchangeably, and *vertex* refers to an element from the set of nodes and hotels.

To understand the relevance of the OPHS and its multi-tour variant, let's consider an example. Imagine a company which employs an agent who is tasked to recruit students from various universities in a region. The agent has one week of time to visit different universities and conduct recruitment activities. The goal of the company is to maximize the total recruitment benefit, which is achieved by visiting universities with high recruitment potential. Thus, the agent's itinerary for this week should be optimized by choosing an optimal selection of universities and hotels visited. This problem is an example of the OPHS by Divsalar et al. (2013), with $D = 7$ trips and scores S_i determined by recruitment potential. The trip's time limit is constrained by the working hours of the agent, for example $T_d = 8$ hours, $\forall d \in \{1, ..., 7\}$.

Now imagine a situation where the company employs not one, but $m \in \mathbb{N}$ agents to recruit at universities. As visiting a certain university twice does not yield additional benefit to the company compared to visiting the university only once, we add the constraint that the tours of the agents should be disjoint in the POIs[1]. Then, the OPHS falls short, as it only considers the optimization of one tour at a time and does not consider the interdependence between two tours. In this example, we would benefit from optimizing multiple tours simultaneously, because the simultaneous optimization takes the interdependence between the possible routes into account. Notably, this cannot be achieved by simply increasing the number of days in a tour in an OPHS by $(m-1) \times D$, because the start and end hotel of each tour should remain fixed, whereas in an OPHS the intermediate hotels can be chosen freely.

This motivates the development of a multi-tour version of the OPHS, which to the best of our knowledge has not been researched before. As this extension to the OPHS shows similarities to how the Team Orienteering Problem (TOP) developed by Chao, Golden and Wasil (1996b) is an extension to the Orienteering Problem (OP), we refer to it as the Team Orienteering Problem with Hotel Selection (TOPHS). We formally introduce the TOPHS in Section 3.1 using a MIP formulation.

Note that the OPHS is a special case of the TOPHS with $m = 1$, implying that the TOPHS is a generalization. As a result of the TOPHS being more general, it is also more complex to solve. Given that the OPHS is already NP-hard (Divsalar et al., 2013), the TOPHS is NP-hard as well. As for larger instances exact computation times become obstructive, we also propose

[1]Note that they should be disjoint in the POIs, but the hotels visited may overlap.

three heuristics for the TOPHS. This is necessary, because in some applications, good solutions need to be available in real-time.

The first heuristic is a multi-tour adaption of the Skewed Variable Neighborhood Search (SVNS) for the OPHS by Divsalar et al. (2013), which we refer to as the TSVNS. As this heuristic has proven to be effective for the OPHS, it is worth exploring for the TOPHS as well. By slightly modifying neighborhoods used in this algorithm, it can be applied to the TOPHS.

The second heuristic and third heuristic are based on the Artificial Bee Colony (ABC) optimization procedure by Karaboga (2005). The reason for using an ABC-based approach is three-fold. Firstly, the skewed variable neighborhood search (SVNS) method used in (Divsalar et al., 2013) turned out to be strong, but its computation time increases substantially for larger instances (Divsalar, Vansteenwegen, Sörensen & Cattrysse, 2014). Secondly, ABC has been successfully employed in a similar problem, namely the Team Orienteering Problem with Time Windows (TOP-TW) by Cura (2014). Thirdly, it is interesting to see if it is applicable to this extension to the OPHS, as it has not been used for any OPHS problem before. If it works here, it might work for other versions of the OPHS problem as well.

As mentioned before, we implement two ABC-based heuristics. The first is an adaption of the ABC-heuristic for the TOP-TW proposed by Cura (2014). It randomly generates a solution and tries to improve the objective value by iteratively moving or exchanging random vertices. We adapt it slightly to make it applicable to the TOPHS, and refer to it as TABC.

The second ABC-based heuristic is a mixture of the ABC-heuristic of Cura (2014) and the SVNS approach by Divsalar et al. (2013). We still use the idea of an ABC-based heuristic, but also apply the neighborhoods as used in the SVNS-approach. We call this method the TAV heuristic.

An overview of the proposed heuristics is provided in Table 1. The heuristics are explained in more detail in Section 3.

Table 1: Overview of the proposed heuristics and the literature on which it is based.

Heuristic	1	2	3
Name:	TSVNS	TABC	TAV
Mainly based on:	Divsalar et al. (2013)	Cura (2014)	Karaboga (2005)

As the TOPHS is a new problem, there exist no instances with known optimal values yet. Part of this research is therefore also dedicated to constructing such instances from existing instances of the TOP (Chao et al., 1996b). The procedure to do so is discussed in greater detail in Section 3.5.

The contribution of this research can thus be summarized as follows:

- The TOPHS is motivated and modelled as multi-tour extension to the OPHS.
- A set of benchmark instances of the TOPHS is generated from existing instances of the TOP.
- The SVNS heuristic is further developed to be used for the TOPHS.
- Two ABC-based heuristics are introduced and employed for instances of the OPHS and TOPHS.
- The performance of the heuristic algorithms is assessed, for both the OPHS as well as the TOPHS.

This boils down to the following research question: *What is the effectiveness of an Artificial Bee Colony (ABC) optimization procedure compared to the Skewed Variable Neighborhood Search (SVNS) method for solving the newly developed multi-tour extension to the Orienteering Problem with Hotel Selection (OPHS)?*

The answer to this question is highly relevant as orienteering problems depend on strong and efficient heuristic methods, as high-quality solutions are sometimes needed on a real-time basis. Using exact methods is generally infeasible for larger instances given that the problems are NP-hard. Especially since the TOPHS is a new problem, the presence of a strong and efficient heuristic is crucial not only scientifically, but also to make processes in society more accurate and efficient.

The scientific relevance is that the currently existing research on the OPHS is extended in line with another commonly used model, the team orienteering problem (TOP). As also benchmark instances are developed, this invites future research on new heuristics for these extended models. Next to that, three heuristics are developed and assessed on their effectiveness. Variations on these heuristics may be applicable to and promising for similar problems.

We find that the TSVNS is the most promising algorithm, providing the lowest average gaps and a reasonable computation time. Especially for smaller instances it outperforms the other methods. For larger instances, however, its execution time increases substantially. The TAV demonstrates potential advantages for such larger instances, because it has lower execution times than TSVNS but maintains an acceptable gap. The TABC does not perform as well, both in terms of average gap and running time.

The remainder of the thesis is structured as follows. Section 2 discusses relevant literature related to orienteering problems and heuristics. The MIP formulation of the TOPHS is introduced in Section 3.1, whereas the adapted SVNS method and the ABC-based heuristics are outlined in Section 3.2 to 3.4. Moreover, Section 3.5 presents how instances with known optimal values can be generated and Section 4 concerns the analysis and the performance evaluation of the heuristic. Lastly, Section 5 summarizes the findings and Section 6 provides insights into future research directions.

2 Literature Review

2.1 Orienteering Problem

The Orienteering Problem (OP) was first developed by Tsiligirides (1984), but Golden, Levy and Vohra (1987) were the first to call it such. It can be regarded as a Travelling Salesman Problem (TSP) where not all nodes need to be visited by the salesman, but there is a time/cost limit. Also, the objective is not minimizing costs, but maximizing score over all nodes visited. This problem description matches that of a Traveling Salesman Problem with Profits (TSPP) (Feillet, Dejax & Gendreau, 2005).

As the problem is relatively old, numerous extensions have been developed, such as the Team Orienteering Problems (TOP), the Orienteering Problem with Time Windows (OPTW), Time-Dependent Orienteering Problem (TDOP), Orienteering Problem with Stochastic Profits (OPSP), Multi-Modal Orienteering Problem (MMOP), Multi-Objective Orienteering Problem

(MOOP), Multi-Period Orienteering Problem (MPOP), and a variety of combinations of those (Vansteenwegen, Souffriau & Van Oudheusden, 2011; Gunawan, Lau & Vansteenwegen, 2016; Vansteenwegen & Gunawan, 2019).

The OP is proven to be NP-hard (Golden et al., 1987). As a result, so are most of its extensions as they only add complexity or more generality to the problem. It is therefore not surprising that many authors who investigate a certain extension of the OP, also find themselves developing a heuristic.

In the remainder, we discuss extensions of the OP related to the problem that is developed in this research. For any other OP problems, we refer to the survey by Vansteenwegen and Gunawan (2019), as it is a rather complete overview of existing OP extensions.

2.2 Orienteering Problem with Hotel Selection

The OPHS is introduced by Divsalar et al. (2013). In this problem, a solution tour consists of D trips, each of which has a time limit. Note that D is not a decision variable. Furthermore, we have a set of $H + 1$ hotels and N POIs. Each POI has a score. A trip should start and end in a hotel, and the starting hotel of a certain day should be the same as the ending hotel of the day before. Furthermore, the start and end hotel of the tour are given and fixed. The goal is to construct a feasible combination of POIs and hotels that maximizes the sum of scores visited over the total tour.

Divsalar et al. (2013) created a skewed variable neighborhood search (SVNS). First, an OP instance is heuristically solved for all feasible pairs of hotels, called the sub-OP. Then, for all feasible combinations of pairs of hotels, a score is calculated by adding the OP-outcomes of all pairs in the combination. The best k hotel combinations according to this score are then used for the remainder of the algorithm, where k is fixed. The optimal POIs of the sub-OPs are added in between the hotels. As some POIs may overlap, repetitions are removed to make it feasible. Then, in the local search procedure, the hotels and POIs are shuffled, replaced, added, and removed for several iterations in an attempt to improve the solution. A solution is better if it increases the total score of the route, or decreases its length while keeping its score constant. Sometimes, also changes that do not increase the total score are accepted, to avoid being trapped in local minima.

Not only did Divsalar et al. (2013) make this heuristic, the authors also designed benchmark instances of the OPHS to test the performance with. The heuristic was capable of solving 102 of the 224 instances to optimality, with an average gap of 1.44% and an average computation time of less than two seconds. A disadvantage of this heuristic is that its performance decreases dramatically when the number of feasible hotel sequences increases (Sohrabi, Ziarati & Keshtkaran, 2020; Divsalar, Vansteenwegen, Sörensen & Cattrysse, 2014). Another disadvantage is that only a subset of all feasible hotel combinations is used during the improvement step, causing that the optimal combination is not considered anymore.

Divsalar, Vansteenwegen, Sörensen and Cattrysse (2014) developed a memetic algorithm (MA), which is able to handle larger instances of the OPHS than the SVNS, because not all feasible combinations of hotels need to be known. The MA delivers a smaller average gap and finds the optimal solution in more cases than SVNS.

Another heuristic for the OPHS is the Greedy Randomized Adaptive Search Procedure (GRASP) that uses dynamic programming to improve the order of hotels on the go, instead of considering all possible combinations beforehand (Sohrabi et al., 2020). Toledo, Riff and Neveu (2019) consider a hyperheuristic, that selects one of eleven local search methods based on the state of the problem and how successful a certain method is expected to be. Looking at the number of instances solved to optimality, this heuristic in particular was relatively successful.

A last heuristic is an Ant Colony System (ACS) heuristic by Sohrabi, Ziarati and Keshtkaran (2021). It adopts the ACS as developed by Dorigo and Gambardella (1997) to perform a bi-directional search pattern, where one ant colony searches from the starting hotel of the first trip, and the other ant colony from the arriving hotel of the last trip. As such, the route is iteratively grown from both sides until a complete solution is found. This is the first heuristic where tours are created sequentially. It provides better solutions than the other heuristics for some of the OPHS instances.

The OPHS-TW is an extension to the OPHS, and was introduced by Divsalar, Vansteenwegen, Chitsaz, Sörensen and Cattrysse (2014). It is similar to the OPHS except that now opening hours for the nodes are considered. In other words, a score is only awarded when a POI is visited within that time window. The authors also develop a heuristic, namely MA similar to Divsalar, Vansteenwegen, Sörensen and Cattrysse (2014), now called a Genetic Algorithm with embedded Variable Neighborhood Descent (GA-VND). The Genetic Algorithm focuses on the hotel selection, whereas the VND takes care of the POI placement, while taking the time windows into account. Divsalar, Emami and Vansteenwegen (2017) instead uses a Lagrangian relaxation method in combination with a heuristic for ensuring feasibility, resulting in strong results as well.

Another recent extension to the OPHS is the Bi-Objective OPHS (BO-OPHS) by Ataei, Divsalar and Saberi (2022). Apart from POIs, the authors also attach a score to hotels based on several criteria. They use a real-world case in eastern Australia to show the applicability of the problem, using real-world data such as customer reviews by using natural language processing. As the problem remains relatively small, it is solved exactly using a commercial solver.

2.3 Artificial Bee Colony Heuristic

The Artificial Bee Colony (ABC) algorithm was introduced by Karaboga (2005). It is an optimization algorithm based on the behaviour of honey bees in nature. Karaboga (2005) identifies three types of bees: scouts, employed bees and onlookers. Generally speaking, scouts do the exploration, onlookers do the exploitation, and employed bees also do exploitation, but also make sure that there is a balance between exploitation and exploration of solutions.

Initially, the algorithm starts with k random solutions to the problem, generated by k scouts. Each of the k employed bees goes to one of them, and calculates its objective value (and already does some quick local search). The employed bees then return to the hive to do the waggle dance, informing the $n \geq k$ onlooker bees about their findings. The onlooker bees are then distributed over the solutions with a probability proportional to the solution's objective value. They then perform local search to try and improve the solution. Solutions which are not improved over a preset number of iterations, are not further explored and a new random solution is generated by the scout bees, such that there are always k solutions being considered (Karaboga, 2005).

ABC is not widely applied to orienteering problems. However, Cura (2014) uses the approach for the Team Orienteering Problem with Time Windows (TOP-TW). The research shows that the ABC heuristic is capable of producing high quality solutions for the TOP-TW, comparable to other commonly-used heuristics for this problem.

2.4 Applications

The OP and OPHS specifically have numerous interesting applications. As the name of the problem is derived from the practice of orienteering, this is the first application to be discussed. In the sport of score orienteering, competitors travel by foot or other means of transport in an unknown area to find an optimal route from start to finish, while visiting as many as possible control points within a given time limit (Tsiligirides, 1984). They use a map, so at the start of the competition, they know the travel distances, bot net yet the optimal route.

Another application of the OP is that of the travelling salesman that does not have to visit each and every sales location, but instead tries to visit only the locations with the highest profits within a given time limit. Compared to the travelling salesman problem (TSP), the objective also changes from minimizing costs to maximizing sales, or total score. This problem is also called the Travelling Salesman Problem with Profits (TSPP) (Feillet et al., 2005). When multiple periods are considered, we have an instance of the OPHS.

Designing a tourist trip matches the description of OP as well, as the tourist wants to visit as many touristic highlights as possible during their holiday. As time is limited, they cannot see every inch of a city, so they need to be selective on such highlights and find an efficient route between them, which is an OP. This problem is known in literature as the Tourist Trip Design Problem (TTDP) (Gavalas, Konstantopoulos, Mastakas & Pantziou, 2014). Again, if multiple periods are considered and the tourist can travel to different hotels, this would be an OPHS.

An example of this is the tourist hotel selection problem by Ataei et al. (2022). This application used the OPHS with hotel scores together with Natural Language Processing (NLP) to get a tour where the scores are related to true customer reviews from sources such as Trustpilot. This makes the model highly adaptive.

Álvarez-Miranda, Luipersbeck and Sinnl (2018) solve a generalized clustered OP to efficiently visit different categories of Pokémon in the popular game of Pokémon Go. This is a clustered problem, which means that each POI is distributed over different locations, and visiting one of its locations is sufficient to get the score of the POI, such that neither of its locations need to be visited in later stages of the tour. In the Pokémon analogy, this means that the same Pokémon species may be found at different locations in a city, but the player only needs one of each species, so visiting one of these locations is enough.

The problems are also relevant for military missions. (Divsalar et al., 2013) mention a situation where a submarine does as many as possible missions (POIs), while resulting to save zones (hotels) for provisioning. As provisioning needs to occur regularly, there is a time limit on each trip. The number of trips and the start and ending location are fixed, meaning that this is an instance of the OPHS.

Another real-life instance of an OPHS is truck planning with mandatory intermediate stops. For example, imagine a truck driver that wants to replenish as many as possible locations where

the score received depends on demand. Assume the truck has a sufficiently high capacity. As the trip length is restricted by law, the truck driver needs to rest at rest stations next to the highway, which can be considered as the hotels. If the start and end locations are known, this is also an OPHS (Divsalar, Vansteenwegen, Sörensen & Cattrysse, 2014).

3 Methodology

3.1 MIP Formulation

In this section, we introduce the modifications necessary to change the OPHS into a TOPHS. First, we introduce notation for the model in Table 3.1. It is based on the MIP from Divsalar et al. (2013).

Table 2: Definition of variables for the TOPHS

Notation	Description	Set
D	Number of trips in the route	$\mathcal{D} = \{1, ..., D\}$
$H + 1$	Number of hotels (hotel 0 is the starting hotel in trip 1, hotel H is the ending hotel in trip D)	$\mathcal{H} = \{0, ..., H\}$
N	Number of POIs	$\mathcal{N} = \{H + 1, ..., H + N\}$
Q	Number of team members	$\mathcal{Q} = \{1, ..., Q\}$
S_i	Score of POI i	
$t_{i,j}$	Symmetric travel time between vertex i and j	
T_d	Travel time limit for trip d	
T	Travel time limit for any whole tour	
$x_{i,j,q,d}$	Binary variable indicating if j is visited right after i by team member q in trip d	
u_i	Order in which the POIs are visited. If i is visited before j in the same trip, then $u_i < u_j$	

Apart from introducing multiple tours instead of one, another addition is the variable T, which denotes the maximum total travel time per tour. This variable may not be necessary in all cases (for example when $\sum_{d=1}^{D} T_d \leq T$), but it is added to increase the flexibility and application possibilities of the problem. If such a time limit is not present in the case under study, T can of course always be set to infinity, eliminating the corresponding constraint. For even more flexibility, an index q could be added (T_q), to give each team member a time limit for their tour. However, we assume all team members to be symmetrical, implying $T_q = T$.

Then, the model is represented by equations (1) - (13).

$$\max \quad \sum_{d=1}^{D}\sum_{q=1}^{Q}\sum_{i=0}^{H+N}\sum_{j=0}^{H+N} S_i x_{i,j,q,d} \tag{1}$$

$$\text{subject to} \quad \sum_{q=1}^{Q}\sum_{i=1}^{H+N} x_{0,i,q,1} = Q \tag{2}$$

$$\sum_{q=1}^{Q}\sum_{i=0}^{H+N} x_{i,H,q,D} = Q \tag{3}$$

$$\sum_{h=0}^{H}\sum_{i=0}^{H+N} x_{h,i,q,d} = 1 \qquad d \in \mathcal{D}, q \in \mathcal{Q} \tag{4}$$

$$\sum_{h=0}^{H}\sum_{i=0}^{H+N} x_{i,h,q,d} = 1 \qquad d \in \mathcal{D}, q \in \mathcal{Q} \tag{5}$$

$$\sum_{i=0}^{H+N} x_{i,h,q,d} = \sum_{i=0}^{H+N} x_{h,i,q,d+1} \qquad d \in \{1,...,D-1\}, h \in \mathcal{H}, q \in \mathcal{Q} \tag{6}$$

$$\sum_{i=0}^{H+N} x_{i,k,q,d} = \sum_{i=0}^{H+N} x_{k,i,q,d} \qquad k \in \mathcal{N}, d \in \mathcal{D}, q \in \mathcal{Q} \tag{7}$$

$$\sum_{d=1}^{D}\sum_{q=1}^{Q}\sum_{i=0}^{H+N} x_{k,i,q,d} \leq 1 \qquad k \in \mathcal{N} \tag{8}$$

$$\sum_{i=0}^{H+N}\sum_{j=0}^{H+N} t_{i,j} x_{i,j,q,d} \leq T_d \qquad d \in \mathcal{D}, q \in \mathcal{Q} \tag{9}$$

$$\sum_{d=1}^{D}\sum_{i=0}^{H+N}\sum_{j=0}^{H+N} t_{i,j} x_{i,j,q,d} \leq T \qquad q \in \mathcal{Q} \tag{10}$$

$$u_i + 1 - u_j \leq (N-1)\left(1 - \sum_{q=1}^{Q}\sum_{d=1}^{D} x_{i,j,q,d}\right) \qquad i,j \in \mathcal{N} \tag{11}$$

$$u_i \in \mathbb{N} \qquad i \in \mathcal{N}, d \in \mathcal{D}, q \in \mathcal{Q} \tag{12}$$

$$x_{i,j,q,d} \in \mathbb{B} \qquad i,j \in \mathcal{H} \cup \mathcal{N}, d \in \mathcal{D}, q \in \mathcal{Q} \tag{13}$$

The objective function (1) maximizes the total score obtained by visiting POIs over all trips and all team members. Constraint (2) ensures that each tour starts in hotel 0, and constraint (3) ensures that each tour ends in hotel H. Constraints (4) and (5) ensure that each trip respectively starts at a hotel and ends at a hotel. Constraints (6) guarantee that each trip is started from the hotel where the previous trip ended. Constraints (7) ensure that if a POI is visited, it is also left in the same trip. Constraints (8) limit the number of visits to any POI by one, over all trips and all team members, whereas the trip and tour length are regulated by constraints (9) and (10), respectively. The subtour elimination constraints are constraints (11), which is similar to the one in Divsalar et al. (2013). They give the order of the POIs in each of the trips using an increasing value of the variable u_i. Constraints (12) and (13) limit the range of the decision

variables and ensure integrality of the problem.

Note that the variables $t_{i,j}, T_d$ and T can be interpreted as travel distance, travel time, or any other quantity that makes the transition between two vertices expensive.

Note that if $T \geq \sum_{d=1}^{D} T_d$ and the start and end hotel are the same, an upper bound on the TOPHS objective value can be derived from an OPHS instance with the same hotels and POIs, $Q \times D$ trips and trip time limits $T'_f = T_{f-nD}, \forall n \in \mathbb{N}, f \leq (n+1)D \leq Q \times D$. For a proof, we refer to Appendix A.1. If the start and end hotel are not the same, we can also construct an OPHS as an upper bound for the optimum of the TOPHS. The proof of that case is presented in Appendix A.2.

3.2 Skewed Variable Neighborhood Search for the TOPHS (TSVNS)

The first heuristic used to solve the TOPHS is based on the SVNS heuristic by Divsalar et al. (2013). We call our generalization the team-SVNS, or TSVNS.

The general structure is presented in Algorithm 1. Note that we changed the while loop of Divsalar et al. (2013) slightly to prevent an infinite loop resulting from $NoImprovement$ becoming too high while K is not reaching $Kmax$. Another change is that local search is now performed within the vertices shake function, and that we check if the solution after vertices shake is an improvement. This is in contrast to Divsalar et al. (2013), which performs local search after the vertices shake and only checks for improvement after also the hotels shake is performed.

The initialization phase is discussed in Section 3.2.1. In the improvement phase, the current solution is explored and exploited. For the improvement phase, the parameter $NoImprovementMax$ limits the number of subsequent iterations where no improvement was found, and $Kmax$ limits the number of subsequent iterations where X is not updated. These conditions are the conditions for the loop. In the loop, the vertices of the current solution are changed to diversify the search (see Section 3.2.2). If the best solution is then not improved, the hotels are exchanged for another set of hotels (see Section 3.2.3), after which again local search is applied. If the solution is improved after any change, the current best solution is updated.

Notably, under some condition, a suboptimal solution may also be used to recenter the solution currently under investigation. In this paper, we use the condition that the current solution has an objective value not further than $MaxPercentageWorse$ percent away from the best found objective value. This part of the algorithm ensures that we are not stuck in a local optimum and promotes exploring apparently suboptimal branches.

Essentially, this implementation of the TSVNS heuristic is very similar to that of the SVNS heuristic of Divsalar et al. (2013). In fact, when setting $Q = 1$ in the TSVNS heuristic, it reduces to our the SVNS heuristic. Therefore, solving OPHS instances with the TSVNS heuristic would replicate the results of Divsalar et al. (2013).

For the rest of the discussion of the algorithm, it is worth mentioning that we consider a solution X to the TOPHS as a better solution than Y (denoted $X > Y$), if either X has a higher objective value than Y, or if the objectives are the same but X has a shorter total distance/time travelled than Y.

Algorithm 1 Structure of the TSVNS heuristic

Initialization

$M \leftarrow$ matrix for all pairs of hotels with potential scores using a sub-OP heuristic

$L' \leftarrow$ list of best feasible combinations of hotels

$L \leftarrow$ list of the NUFC best feasible combinations of hotels, based on M and L'

$X \leftarrow$ initial solution based on L

Improvement

$bestX \leftarrow X$, $K \leftarrow 1$, $NoImprovement \leftarrow 0$

while $NoImprovement < NoImprovementMax$ **and** $K < Kmax$ **do**

  $X'' \leftarrow$Vertices-Shake(X)

  if $X'' > bestX$ **then** $X \leftarrow X''$, $bestX \leftarrow X$, $K \leftarrow 1$, $NoImprovement \leftarrow 0$, **continue**

  $X' \leftarrow$Hotels-Shake(X, L)

  $X'' \leftarrow$Local-Search(X')

  if $X'' > bestX$ **then** $X \leftarrow X''$, $bestX \leftarrow X$, $K \leftarrow 1$, $NoImprovement \leftarrow 0$

  else

    $NoImprovement$++

    if X'' is slightly worse than X **then** $X \leftarrow X''$, $K \leftarrow 1$

    else K++

return $bestX$

3.2.1 Initialization Phase

As follows from Algorithm 1, the first step is to find the potential score between all pairs of hotels, for each trip $d \in \mathcal{D}$. This is done using a Sub-OP procedure, similar to Divsalar et al. (2013). A Sub-OP is an orienteering problem between two hotels with the time limit set equal to T_d. It is solved using the greedy Algorithm 2, which uses local search methods from Section 3.2.4. As this algorithm is relatively straight-forward, it takes little computation time. The four methods used in this algorithm are discussed in Section 3.2.4.

Algorithm 2 Sub-OP

X $\leftarrow$ initial solution with only the start hotel and the end hotel

Level $\leftarrow 0$

Methods $\leftarrow$ [Insert, Replacement, Two-Opt, Move-Best]

while Level < 4 **do**

  $X' \leftarrow$ Methods[Level](X)

  if $X' > X$: $X \leftarrow X'$, Level$\leftarrow 0$, **continue**

  Level++

return X

Next, the list of feasible combinations of hotels is obtained. This is done by obtaining all permutations of size $D-1$ from the set of hotels, and then appending the start and end hotel to the front and back respectively. This creates a tour of D trips, and its feasibility is easily checked using the trip time limits T_d and T. By taking all Q-permutations of these tours, a

feasible combination of hotels is obtained for the TOPHS.

All these combinations of hotels are subsequently given a score, which is the sum of sub-OP scores of each pair of hotels in each tour of the combination. This score is called the Heuristic Estimated Score (HES), because some nodes may appear in multiple solutions to the sub-OPs and better sub-OP solutions may exist. Based on this HES, the list of combinations is sorted and the best NUFC (Number of Used Feasible Combinations of hotels, a parameter of the algorithm) are selected.

Note that these NUFC combinations do not contain any POIs yet. To construct an initial solution, we apply three methods similar to (Divsalar et al., 2013), each of which creates a feasible solution. The feasible solution with the highest outcome is then used as the initial solution.

The first two methods iteratively solve the Sub-OP of Algorithm 2 for each pair of hotels in the solution, taking into account the nodes that have already been added to previous trips. The first method does this starting from the first trip of the first route, whereas the second method starts at the last trip of the last route. After that, local search of Section 3.2.4 is applied to improve the solution to the first and second method. The third method just applies local search as described in Section 3.2.4 to the initial empty combination of hotels.

3.2.2 Vertices Shake

In the Vertices Shake method, some of the vertices in the current solution are replaced by new ones in order to explore the solution space for a given set of hotels. Two methods are used to remove vertices. In the first method, the first half of the vertices in each trip is deleted, after which local search (see Section 3.2.4) is applied. In the second method, the last half of the vertices in each trip is deleted, also followed by local search. The best result is then returned.

3.2.3 Hotels Shake

In the Hotels Shake method, the hotels in the tour are replaced by another one of the NUFC feasible combinations of hotels from the list L in Algorithm 1. It is determined randomly with equal probability which hotel combination is selected.

After replacing the hotels, the set of tours may become infeasible. If this is the case, nodes are removed from the infeasible trips according to an iterative procedure, where the node with the highest ratio of time saved by a removal over score is removed. This is repeated until the set of tours is again feasible.

3.2.4 Local Search

In Local Search, the current solution is exploited by adding, moving or replacing POIs in the set of tours. The implementation of this method is presented in Algorithm 3. As the Local Search procedure is the core of the TSNVS algorithm, the neighborhoods need to be carefully chosen. As the TOPHS is an extension of the OPHS and can be seen as Q simultaneously optimized OPHS instances, we expect the methods used for OPHS by (Divsalar et al., 2013) to work well. Therefore, we use the nine neighborhoods from that paper.

As the TOPHS has an extra dimension compared to the TOPHS, namely the fact that we now have Q tours instead of one, we should slightly change some of the neighborhoods, to allow

for example swapping nodes from one tour to another. In the remainder of this section follows a short description of each neighborhood, based on Divsalar et al. (2013).

Insert: for all non-included POIs, identify the cheapest location to insert it in the solution, if possible. Insert the POI that results in the largest increase in total score, relative to additional travel time.

Move-Best: for all POIs in the solution, identify the cheapest location to move it to. After the move is done, consider the next POI.

Two-Opt: within each trip in each tour, identify the pair of POIs that can be swapped resulting in the highest decrease in travel time. After the swap is done, consider the next trip.

Swap-Trips: for each pair of trips, identify the pair of POIs (one coming from each trip) for that can be swapped resulting in the highest decrease in travel time, while keeping both trips feasible. After the swap is done, consider the next pair of trips.

Extract-Insert: for each POI, identify if it is possible to increase the score by removing it and inserting as many other vertices as possible (using *Insert*). If this is possible, keep the change and consider the next POI.

Extract2-Insert: same as Extract2-Insert, except that a sequence of two POIs in the same trip is now considered for exclusion.

Extract5-Insert: same as Extract-Insert, except that a sequence of five POIs in the same trip is now considered for exclusion. If one possibility is found, the move is performed and the neighborhood search is terminated.

Extract-Move-Insert: for each POI, identify if it is possible to increase total score by (1) removing the node, (2) moving another node in the tour and (3) inserting a new node. If this is the case, perform the move and consider the next POI.

Replacement: for each non-included POI, identify the cheapest position to insert it in each trip. If this move is feasible, insert this POI. If not, all other POIs in the trip with a lower score are considered for removal. If removing one of these POIs makes insertion of the initial POI feasible, perform the replacement and consider the next trip.

Note that these neighborhoods only change the choice and order of included POIs, not the hotels. Therefore, in the local search algorithm, the hotels are considered fixed.

Algorithm 3 Local Search

```
X ← initial solution
Level ← 0
Methods ← [Insert, Move-Best, Two-Opt, Swap-Trips, Extract-Insert, Extract2-Insert,
Extract5-Insert, Extract-Move-Insert, Replacement]
while Level < 9 do
    X' ← Methods[Level](X)
    if X' > X: X ← X', Level← 0, continue
    Level++
return X
```

3.2.5 Parameters Overview

This TSVNS has several parameters that require tuning. Divsalar et al. (2013) already found appropriate parameters for their OPHS instance set. As the TOPHS is based on the OPHS and because we did not greatly change the methods, we stick with the values they used. Also, the authors found that small changes in the parameter values did not significantly change the performance of the algorithm.

The values they found are the following: $NoImprovementMax = 50$, $NUFC = 250$, $MaxPercentageWorse = 0.3$, $Kmax = \max\{0.25 \times \min\{NUFC, TNFC\}, 1\}$.

Here, $TNFC$ denotes the total number of feasible combination of hotels in the instance of the problem. For example, if there are only three distinct hotels and two trips per route, then there are at most three feasible combinations of hotels. This means that only when $TNFC$ is larger than $NUFC$, we do not consider all possible combinations in the algorithm.

3.2.6 Theoretical Problems with TSVNS

One difficulty with the SVNS of Divsalar et al. (2013) is that its computation time increases substantially when the number of hotels in an OPHS increases. This is a result of the algorithm considering a subOP for each pair of hotels, and the fact that it heuristically solves all feasible combinations of hotels before starting the improvement phase. Because of that, the initialization phase can take obstructively long. We have a similar problem for the TSVNS as hotels are selected in a similar fashion.

Another possible disadvantage is that the TSVNS only selects the $NUFC$ combinations of hotels which have the highest HES. This implies that some feasible combinations of hotels are not considered in the improvement step, even though they may give a higher feasible solution than the set of the best $NUFC$ hotels.

Both difficulties advocate the use of a more randomized algorithm. The ABC-based algorithms of Section 3.3 and 3.4 may therefore be a successful alternative.

3.3 Artificial Bee Colony Heuristic (TABC)

The second heuristic used in this research is a modified version of the ABC-based heuristic by Cura (2014). They applied the heuristic to a similar problem, namely the team orienteering problem with time windows (TOP-TW). By changing its algorithm slightly, a heuristic for the TOPHS can be developed.

An important factor in using this heuristic is the solution representation. Instead of defining a set of feasible tours as a solution and performing neighborhood searches by inserting nodes from a pool of nodes, all nodes are already present in the solution. One can construct a solution as one sequence, with N POIs, $Q \times (D-1)$ hotels and $Q-1$ tour separators (TSs), which indicate where the previous tour ends and the next one starts. Note that as the tours are separated by a TS, there is no need to explicitly insert the start and end hotels of each tour. An example of a solution representation is given in Table 3.

To convert this solution to a feasible set of tours, the tours are separated around the TS operator and the start and end hotel are inserted. Note that some time limits may be violated as we inserted all nodes into the solution. To solve that, all nodes are visited until one of the

time restrictions is violated. In the example in Table 3 and given appropriate time limits, node 5 can be visited in the first trip of tour 1, but node 4 cannot be reached within the trip distance limit and is skipped because of that. In this fashion, we can construct the two tours as used in the TSVNS heuristic. Note that the two solutions in Table 3 are then equivalent.

The reason for using this representation is that it makes randomly changing the order of vertices a lot easier. Note that all solutions automatically have the same length. Namely, they always consist of N POIs, $(D-1) \times Q$ hotels and $(Q-1)$ TSs.

Table 3: Solution representation of the same feasible solution for each of the heuristics. TS denotes an operator that separates the tours. Parameters: $N = 8, Q = 2, H = 2$ (so vertices 0, 1 and 2 are the hotels).

Heuristic	Solution representation						
TSVNS, TAV	1:	[**0**	5	**1**	3	**2**]
	2:	[**0**	8	**0**	6	**2**]
TABC	[5	4	**1**	3	7	TS
		8	**0**	6	9	10]

Algorithm 4 shows how this heuristic is implemented. It is based on how a bee hive explores its surroundings for the highest amount of nectar (the highest objective value). For a given number of iterations, the nEB employed bees and the nOB onlooker bees explore the neighborhood of the L current solutions by swapping and moving vertices in a solution, and reversing the order between two vertices. The employed bees choose each of the L solutions with equal probability, whereas the onlooker bees take the amount of nectar in each solution into account. Then, with a probability of μ, new random solutions are generated by the nSB scout bees. After each change in one of the solutions, the current best solution is updated, such that the variable *best* in Algorithm 4 always reflects the best found solution.

Algorithm 4 Implementation TABC heuristic

```
sols ← L randomly generated feasible solutions
best ← Best solution from sols
for i in 1:maxIter do
    for b in 1:nEB, do employedBee(sols)
    P← waggleDance()
    for b in 1:nOB, do onlookerBee(P, sols)
    if Random draw from UNIF(0,1) < μ then
        for b in 1:nSB, do scoutBee(sols)
```

3.3.1 Generating random solutions

In the first step of Algorithm 4, L random solutions are generated. This is done by creating a list of all N POIs, a random subset of $(D-1) \times Q$ hotels and $Q-1$ TSs. Then, this list is shuffled to create a solution in line with Table 3.

Note that this solution is not necessarily feasible, even after deleting all POIs, as some combinations of hotels may be infeasible due to travel time restrictions. Therefore, generating a

random solution is tried until either a feasible solution is found, or time runs out of the set time limit, implying that there is likely no feasible solution.

3.3.2 Employed Bees

The function employedBee($sols$) in Algorithm 4 randomly selects two of the solutions in $sols$, called k and l. Then, a random vertex i in l is chosen, which is either a Hotel, a POI or a TS. Next, the location of i in k and l is recorded as r_k and r_l respectively.

The employed bees explore three neighborhoods of l, each of which is chosen with equal probability. The three neighborhoods are SWAP(i), INSERT(i) and INVERT(i), and how they are implemented depends on the type of i, as not all moves from Cura (2014) are allowed for all types of vertices. The neighborhoods are explained in Section 3.3.6.

After the neighborhood is explored and neighbour l_{new} is discovered, the objective value of l_{new} is calculated. The new solution is accepted with a probability of one if the new objective value is better, and it may also be accepted if it is not according to a certain probability. This probability is based on the probability calculation in a Simulated Annealing algorithm (Cura, 2014), namely $\exp\{(f(l_{new}) - f(l))/\rho\}$, where $f(x)$ denotes the objective value of solution x and ρ is a parameter. If accepted, l is replaced by l_{new} in $sols$.

3.3.3 Waggle Dance

In a waggle dance, employed bees inform the other bees about the amount of nectar available in the location they explored. The onlookers can then distribute themselves over these locations in a proportional way.

Our algorithm simulates this by assigning a probability to each location in $sols$. This is done in proportion to their objective values. Thus, P in Algorithm 4 is a vector of L probabilities.

3.3.4 Onlooker Bees

Onlooker bees explore the neighborhood of the food sources even further, after having been informed by the employed bees using the waggle dance. Specifically, they select food sources k and l with a probability $P[k]$ and $P[L]$. After that, they do explore the neighborhoods of l in exactly the same manner as the employed bees.

3.3.5 Scout Bees

When a food source is exhausted, scout bees go to a random new location. They inform the other bees of this new location, so that they can explore it further.

In our implementation, it is hard to determine when a solution is exhausted, or when no single (sequence of) move(s) can improve the objective value. Therefore, in each iteration, the nSB scout bees are employed with a small probability μ.

The scout bee function does the following. First, a random food source l is selected from the L current food sources. Then, all vertices are swapped with each other and all vertices are moved to all possible positions, to find the feasible solution with the highest objective value. This solution is then saved. This practice is explained in more detail in Algorithm 5.

Algorithm 5 Scout Bee

$l \leftarrow L$ random solution from $sols$
$numVertices \leftarrow N + (D-1) \times Q + (Q-1)$ ▷ number of POIs + hotels + TS in each solution
for i, j in $1 : numVertices$ **do**
    $l_{new} \leftarrow$ swap ith and jth vertex in l
    if l_{new} is feasible & $l_{new} > l$ **then** $l \leftarrow l_{new}$
for i, j in $1 : numVertices$ **do**
    $l_{new} \leftarrow$ insert ith vertex at jth position in l
    if l_{new} is feasible & $l_{new} > l$ **then** $l \leftarrow l_{new}$

3.3.6 Neighborhoods

Both the employed bees and the onlooker bees explore three types of neighborhoods, each of which is chosen with equal probabilities. The three neighborhoods are SWAP(i), INSERT(i) and INVERT(i).

SWAP(i) swaps to vertices with each other, by swapping the elements at index r_k and r_l in l. Note that this is not always possible, as swapping hotels or TSs may imply that the number of hotels in each tour is not satisfied anymore. As a result, the range of values that r_k may take is restricted depending on the type of i. Similar complications also occur for the other neighborhoods. In the remainder of this section, we therefore describe how each heuristic is implemented for each type of i to account for such complications.

Firstly, if i is a POI, it can only swap with other POIs, with hotels in the same tour or with TSs between the same hotels to prevent a change in the number of hotels per tour. If r_k does not satisfy this requirement, no move is applied.

Secondly, if i is a hotel, it can only swap with other hotels or with POIs in the same tour. If r_k does not satisfy this requirement, the hotel at r_l in l is replaced with a random other hotel from $\mathcal{H}$.

Lastly, when i is a TS, it can swap with other TSs, which has no effect as a TS has no specific meaning in the final solution. It can also swap with any POI between the same pair of hotels as i. Any other move again causes an imbalance in the number of hotels in each tour. If r_k does not satisfy this requirement, no move is applied.

INSERT(i) removes the vertex at index r_l from l and inserts it at index r_k in l. If i is a POI, this is always possible. If i is a hotel, it is possible only if r_k is in the same tour as i. Otherwise, the hotel at r_l in l is replaced with a random other hotel from $\mathcal{H}$. If i is a TS, the insert move can be applied only if r_k is between the same hotels as i. Otherwise, no move is applied.

The last neighborhood is INVERT(i). It reverses the order of the vertices in l between and including indices r_l and r_k.

If i is a POI, this is possible if r_l and r_k belong to the same tour in l. It is also possible if the number of to be reversed hotels in the route to which r_l belongs, is the same as the number of to be reversed hotels in the route to which r_k belongs. If r_k does not satisfy any of these requirements, then either SWAP(i) or INSERT(i) is applied, with equal probability.

If i is a hotel, the same condition applies. If this condition is not satisfied, we replace the hotel at r_l in l with a random other hotel from $\mathcal{H}$. Lastly, if i is a TS, it can invert if index r_k in

l corresponds to a TS as well, or if r_k is between the same pair of hotels as r_l. Again, no move is applied otherwise.

Lastly, for all neighborhoods, note that it may happen that hotel i is not present in k. In that case, r_k is undefined. Then, for all heuristics, we replace the hotel at r_l in l with a random other hotel from $\mathcal{H}$. This is preferred over doing no move at all, as it promotes exploring solutions with a different order or subset of the hotels.

The solution after the move is not necessarily feasible, as the order and choice of hotels may have changed. In that case, the original solution is still used, without having applied any move.

Contrary to the TOP implementation by Cura (2014), the moves allowed for each of the neighborhoods is highly restricted. This may negatively influence the performance of the heuristic in terms of exploitation of a given solution. This motivates the use of the TAV heuristic.

3.3.7 Parameters

The TABC algorithm as above has several parameters that need to be set. Firstly, we have the number of current solutions considered (L). Then, we specify the size and distribution of the bee hive by the number of employed bees (nEB), onlooker bees (nOB) and scout bees (nSB). Next to that, ρ is a value that determines the likelihood of accepting a solution that is worse than the current solution. A lower value promotes exploration over exploitation, by increasing this likelihood. Additionally, we have μ which specifies for each iteration the probability that the scout bees are employed. Lastly, the maximum number of iterations ($maxIter$) needs to be set. As larger problems likely need more iterations, it makes sense to relate this parameter to the parameters of the model, as in Cura (2014).

Another value that can be considered as a parameter is the time limit. If a feasible random solution cannot be found within this time limit, the heuristic algorithm quits.

Cura (2014) obtained the following parameters in their parameter tuning: $L = 10, nEB = 10, nOB = 5, nSB = 1, \rho = 0.3, \mu = 0.00015$. Also, $maxIter = (N + Q - 1) \times 8000$, where $N + Q - 1$ is the number of elements in a solution to the TOP. In our case, we need to consider the factor $N + (D - 1) \times Q + (Q - 1)$, which is the number of elements in a solution to the TOPHS.

Lastly, one could consider the time limit a parameter of the heuristic.

3.4 Artificial Bee Colony Heuristic with Neighborhood Search (TAV)

The third heuristic that is implemented in this research is also based on the Artificial Bee Colony optimization procedure. However, it is not inherited from any other ABC-based heuristic for an OP specifically. Instead, the approach is based on the initial idea of ABC by Karaboga (2005). It is extended with the neighborhood search and vertex shake methods from the TSVNS heuristic, contrary to the TABC heuristic which mostly uses random moves. The motivation for this is that the random moves of the TABC heuristic are highly restricted, limiting the possibilities for exploitation, as discussed in Section 3.3.6.

As this heuristic is a combination of an ABC-based optimization procedure and the TSVNS local search neighborhoods, we call this heuristic the Team-ABC-VNS or TAV heuristic for short.

Algorithm 6 summarizes the implementation of the TAV. Initially k random feasible solutions are generated by the scout bees. These k solutions are then iteratively improved by the employed bees and the onlooker bees. If a solution strain is not successful, i.e., no improvement has been achieved over a given number of iterations, the employed bees ask the scout bee to replace it with a new random one. This process is repeated until a stopping criterion is met, such as a fixed time limit or number of iterations.

The behaviour of scout bees is further explained in Section 3.4.1, that of the employed bees in Section 3.4.2, and that of the onlooker bees in Section 3.4.4. Section 3.4.3 explains the waggle dance.

Again, we consider a solution X to the TOPHS as a better solution than Y (denoted $X > Y$), if either X has a higher objective value than Y, or if the objectives are the same but X has a shorter total distance/time travelled than Y.

Algorithm 6 Artificial Bee Colony Heuristic for the TOPHS

$X \leftarrow$ empty result matrix, to be filled with k potential solutions
$C \leftarrow$ vector of length k keeping track of the number of iterations without improvement
$bestX \leftarrow scoutBee()$
for i in 1:k **do** $X[i] \leftarrow scoutBee()$, $C[i] \leftarrow 0$
while stopping criterion not satisfied **do**
  for i in 1:k **do**
    $X[i] \leftarrow employedBee(X[i])$
    if $X[i] > bestX$ **then** $bestX = X[i]$, $C[i] = 0$
    else $C[i]++$
    if $C[i] > l$ **then** ▷ l is the number of allowed iterations without improvement
      $X[i] \leftarrow scoutBee()$, $C[i] \leftarrow 0$
      $X[i] \leftarrow employedBee(X[i])$
  $P \leftarrow waggleDance(X)$
  for j in 1:n **do**
    $X \leftarrow onlookerBee(X, P)$
    if for any i, $X[i] > bestX$ **then** $bestX = X[i]$, $C[i] = 0$
return $bestX$

3.4.1 Scout Bee

In Algorithm 7, scout bees are employed. They generate a random feasible solution from all nodes and hotels, given the restrictions set by T_d and T. This boils down to creating Q tours that are mutually exclusive in the POIs. Therefore, let us first describe how the tours are generated randomly.

First, $D - 1$ hotels are randomly selected from $\mathcal{H}$. We call this set with $D - 1$ hotels $\mathcal{H}'$. Then the elements from the set $\mathcal{H}' \cup \mathcal{N}$ are put in a random order. Next, the start and end hotel are appended at the front and back respectively.

This random tour is likely infeasible due to exceeded travel time limits. From all infeasible trips, the first POI is removed, until either the trip is feasible or no nodes are left. We chose

to remove the first node of the infeasible trips instead of the node with the smallest ratio of score over decrease in trip length by removing it, as to minimize computational efforts. Also, this promotes diversification of the initial solutions and therefore prevents being stuck in local optima.

If no nodes are left in the trip and the trip is infeasible, then this combination of hotels is infeasible and one needs to restart at selecting $D-1$ hotels randomly. If there exists a feasible combination of hotels, then at one point, all Q tours are feasible. In our implementation, we set a time limit for trying to generate this set of Q tours. If this time limit is exceeded, then the problem is assumed to be infeasible and the TAV heuristic gives no output.

Now that there are Q tours, it does not mean that they are disjoint in the POIs. Therefore, from all duplicate nodes, the one with the highest ratio of decrease in length after removal over score is removed, until there are no duplicates left. As a result, the solution is feasible and the scout bee has completed its task. Making the solution feasible is also summarized in Algorithm 8.

Algorithm 7 scoutBee()

V = null
while V is null **do**
  $\mathcal{H}' \leftarrow D-1$ random hotels from $\mathcal{H} = \{0, ..., H\}$
  $V \leftarrow \mathcal{H}' \cup \mathcal{N}$ where $\mathcal{N} = \{H+1, ..., H+N\}$
  $V \leftarrow shuffle(V)$ ▷ $shuffle()$ shuffles the set
  $V \leftarrow \{0\} \cup V \cup \{H\}$
  $V \leftarrow makeFeasible(V)$ ▷ See algorithm 8
return V

Algorithm 8 makeFeasible(V)

for Infeasible trip d in V **do** ▷ Remove violations of T_d
  while d infeasible **do**
    if no POIs in d **then**
      return null ▷ Infeasible combination of hotels
    Remove POI with greatest ratio of marginal distance / score
while V not feasible & duplicate POIs in V **do** ▷ Remove duplicates
  Remove duplicate POI with with greatest ratio of marginal distance / score
while V not feasible **do** ▷ Remove violations of T
  if no POIs in V **then**
    return null ▷ Infeasible combination of hotels
  Remove POI with greatest ratio of marginal distance / score
return V

3.4.2 Employed Bee

During the exploration and exploitation phase in an ABC optimization procedure, employed bees are used (Karaboga, 2005). They are in charge of improving the solution using local search

methods and informing the bee hive about the quality of the result. The first task is discussed in this section, whereas the second task is discussed in Section 3.4.3. Next to that, if a certain solution strain has not resulted in any improvement of the existing result for a sufficient number of iterations, an employed bee also instructs the scout bee to find a new random solution. This is handled by the general TAV implementation in Algorithm 6.

To improve a given solution, we first perform a vertices shake similar to the TSVNS. Contrary to the method discussed for the TSVNS (Section 3.2.2), this is only done for a randomly chosen tour instead of all Q tours. The reason for using a vertices shake is that in local search, the algorithm tries to find an optimal neighbour of a solution until a local optimum is reached. Because of that, performing local search on a solution to local search has no effect. By slightly modifying the solution each time local search is performed, we therefore prevent getting stuck in local optima. However, doing a vertices shake in all Q tours would diversify the new solution too much from the original solution, meaning that exploitation of the original solution would be reduced. Changing only one tour seems reasonable to preserve both exploitation and exploration of our solution, especially given that the scout bees already do the randomization / exploration.

After the vertex shake, the local optimum is found using local search. The neighborhoods used are the same as for TSVNS, namely *Insertion, Move-Best, Two-Opt, Swap-Trips, Extract-Insert, Extract2-Insert, Extract5-Insert, Extract-Move-Insert* and *Replacement*, as discussed in Section 3.2.4. Therefore, we use the local search method from Algorithm 3.

The complete algorithm for the employed bee is outlined in Algorithm 9. Note that in this implementation local search is applied as part of the vertices shake procedure. Therefore, there is no need to apply it again to the solution obtained from the vertices shake procedure.

Algorithm 9 employedBee(X[i])

$q \leftarrow$ random integer between 1 and Q
$L \leftarrow$ X[i] where the first half of the nodes in all trips of tour q is removed
$R \leftarrow$ X[i] where the last half of the nodes in all trips of tour q is removed
$L \leftarrow$ localSearch(L), $R \leftarrow$ localSearch(R) ▷ See Algorithm 3
if $L > R$ **then return** L, **else return** R

3.4.3 Waggle Dance

When the onlooker bees return, they do a so-called waggle dance to inform the hive which of the k results are promising and should be explored further. The n onlooker bees distribute themselves over these solution in a fraction that is proportional to the potential of the solution. Our analogy to this is that the potential can be calculated as the fraction of the objective value over the sum of all objective values. This is implemented in Algorithm 10.

Algorithm 10 waggleDance()

$S \leftarrow 0$, $P \leftarrow$ vector of length k which contains the solution's probability scores
for i in 1:k **do** $S \leftarrow S + obj(X[i])$ ▷ obj(x) calculates objective value of x
for i in 1:k **do** $P[i] = obj(X[i])/S$
return P

3.4.4 Onlooker Bee

The last type of bee is the onlooker bee. We have n of them and they are distributed over the k solutions to explore their neighborhood. A promising solution (meaning it has a higher objective value) gets more onlooker bees, so its neighborhood is explored more intensively. The exploration of the onlooker bees occurs in the same way as the employed bees, just their distribution among the solutions differs. The implementation is depicted in Algorithm 11.

Algorithm 11 onlookerBee(X, P)

$i \leftarrow$ index chosen in accordance to the probability mass function defined by P
$X[i] \leftarrow$ employedBee($X[i]$) ▷ See Algorithm 9
return X

3.4.5 Parameter Overview

We have several parameters, namely the number of solutions considered at any point in time (k), the number of onlooker bees (n), the maximum number of iterations without an improvement of solution k (l) and the stopping criterion of algorithm. For the stopping criterion, we use a maximum number of iterations $maxIter$. This leaves us with four parameters that require tuning. As this is a new heuristic, we have no optimal parameters from the literature. Again, the time limit can also be regarded as a parameter.

Note that contrary to the TABC heuristic, the number of scout bees is not specified in the TAV heuristic. The reason is that it is assumed that an employed bee temporarily takes on the task of a scout bee if needed, as in Karaboga (2005). Also, the number of employed bees is assumed to be equal to the number of solutions k, and exactly one employed bee is allocated to each solution.

3.5 Generation of instances with known optimal values

This research deals with assessing the performance of the heuristics. For that, it is necessary to have benchmarks of the TOPHS, meaning that we need an instance and a solution. Luckily, these can be generated straightforwardly, using existing instances of the Team Orienteering Problem (TOP) from Tsiligirides (1984); Chao (1993); Chao, Golden and Wasil (1996a); Chao et al. (1996b). They provide us with seven sets of in total 387 instances of different sizes.

To transform them into TOPHS instances, we follow a similar procedure as Divsalar et al. (2013) used when they transformed instances from an OP to an OPHS. The procedure starts from the optimal solution of a TOP instance, which has Q tours, one for each team member. The time limit for each tour is T. For the TOPHS, these tours are split into D trips each. This is done by first setting $T_d = T/D, \forall d \in \{1, ..., D-1\}$.

The next step is to set the $D-1$ intermediate hotels exactly at the nodes that can be reached on the existing optimal routes within each of the cumulative T_ds, using a similar procedure as in Divsalar et al. (2013). This gives rise to $D-1$ trips. Finding the last parameter T_D is a little more tedious: it cannot just be set to $T_D = T/D$, because generally not all points on the optimal routes are visited at multiples of time T/D. Therefore, we need to set $T_D > T/D$. The size required is calculated by checking for all routes the time required to travel from the $D-1$th

intermediate hotel to the endpoint, via the optimal route. T_D is then the maximum of these values over all routes.

Now, the problems at hand are complete TOPHS benchmarks. By construction, the optimal value does not change: because the hotels are on the optimal route, there is no added distance by adding the hotels, meaning we basically still have the TOP. The time limits T_d implicitly decide the hotels visited and T makes sure that the total route time is limited. Thus, these instances are ready to be used for testing the heuristics.

The only difficulty lies in finding the optimal solutions to the TOP. These are calculated using a commercial solver (CPLEX). A disadvantage is that the TOP is NP-hard as well, making larger instances too complex to solve to optimality within reasonable computation time. For this part of the research, we therefore focus on smaller instances of the TOP. To be precise, for each of the seven sets of TOP instances from the literature and for each value of $Q \in \{2, 3, 4\}$, we use the instance with the highest objective value which is solved to optimality within one hour of computation time. In other words, for a given set of instances and value of Q, we try to solve all TOP instances using CPLEX, and we use the instance with the highest objective value, given that it is solved within one hour. As each set contains three values of Q, we have $3 \times 7 = 21$ instances with a known optimal value.

In the next step, for each of the 21 TOP instances, we create a set of TOPHS instances using the procedure described above for D trips per tour. In this research, $D \in \{1, 2, 3\}$. The set of all such instances is called SET 1, which therefore consists of $21 \times 3 = 63$ instances with a known optimal value.

Then, we construct a second set, called SET 2. This set consists of all instances of SET 1 with $D = 2$ trips per tour, but where we add m extra possible hotels per instance. The extra hotels are placed at the same location as vertices at an index corresponding to a multiple of $\lfloor N/\left(2 + (D-1)Q + m\right) \rfloor$, similar to how Divsalar et al. (2013) constructed their SET2. Note that the presence of these extra hotels by construction never changes the optimal solution. In this research, $m \in \{1, 2, 3\}$, so SET 2 also consists of $21 \times 3 = 63$ instances.

Furthermore, we create a set called SET 3. We follow the same procedure as for SET 1, but instead of adding hotels at the location of the POIs at the end of each trip, we now choose random POI locations for the hotels. Instead of keeping the POI to which we allocate a hotel, we now remove the POI, so it is essentially replaced by a hotel. We do this for $D = 2$ trips per tour. Additionally, we also replace $m \in 1, 2, 3$ other random POIs with hotels. As such, the optimal value is unknown to us, and we have no optimal value to benchmark our results against. For the trip distances, we set $T_d = T/2$ for both trips. SET 3 then contains $21 \times 3 = 63$ instances.

As the previously described procedure depends on being able to exactly solve a TOP instance within one hour of computation time, the instances are not particularly large. Therefore, we also create an additional SET 4 with larger instances. We use a similar procedure as for SET 3, but instead of using the solved instances of the TOP, we now use the largest instance for each value of Q in each set of TOP instances. Again, we use $m \in 1, 2, 3$. As some instances might then overlap with SET 3, we only include new instances in SET 4. SET 4 then consists of at most $21 \times 3 = 63$ instances, with unknown optimal value.

Note that as SET 3 and SET 4 have new hotels which are not necessarily on the optimal TOP route, the instances may not be feasible.

All four sets of instances are used to study the effectiveness each of the three heuristics relative to CPLEX. We impose a time limit of five minutes for each method.

4 Results

In this section, the performance of each heuristics with respect to the other heuristics and CPLEX is studied. For CPLEX, the MIP of Section 3.1 is used. A guide to the code used can be found in Appendix B.

The algorithms in Section 3 are implemented in Java, and are run on a Windows 10 computer with a hexa-core Intel Core i5-10500 CPU at 3.10 GHz and 16.0 GB of RAM. The heuristics all run on a single core, whereas CPLEX self-optimizes the number of cores used.

In Section 4.1, we briefly discuss the parameters used. In Section 4.2, CPLEX and the three heuristics are employed on the OPHS instances from Divsalar et al. (2013), which can be interpreted as TOPHS instances with $Q = 1$ and $T = \infty$. In Section 4.3, the methods are employed on the TOPHS instances as described in Section 3.5.

Note that in this section, only aggregated results are discussed. The outcomes of each solving method and for each instance can be found in Appendix C for the OPHS instances of Section 4.2, and in Appendix D for the TOPHS instances of Section 4.3.

4.1 Parameter Tuning

To determine appropriate values of the parameters, we take a random sample of 20 instances of the OPHS and TOPHS. Based on the average gap and average computation time, we determine appropriate settings of the parameters, for each heuristic. The results of this parameter tuning procedure are presented in Table 4.

For the TSVNS heuristic, we consider the same values as Divsalar et al. (2013) for their SVNS heuristic. Similar to what the authors find, the results are not much affected by small changes in the parameters, both in terms of computation time and average gap. Therefore, we determine that it is appropriate to use the same values as they do.

For the TABC heuristic, we deviate from the values that Cura (2014) use for their ABC-based heuristic for the TOP. The reason for that is the following. Recall that the neighborhoods of the TABC heuristic are constrained in the moves that they are allowed to make, because the number of hotels in each tour must stay the same after each move. Therefore, some solutions cannot straightforwardly be reached from a given solution. As a result, the neighborhood search of the TABC is less powerful in exploration than the original ABC-based heuristic for the TOP. To compensate for that, we need to increase the degree of randomization elsewhere in the algorithm. This is done using more scout bees (a higher nSB) and a higher probability of employing the scout bees (a higher μ). Because the algorithm for the scout bees is computationally rather intensive, and because now more scout bees are employed, we also reduce the maximum number of iterations compared to Cura (2014). To promote exploration even further, we also increase L. The resulting parameter settings, as shown in Table 4, turn out to be the most competitive for the sample of instances.

Lastly, for the TAV, no parameter settings have been inherited from existing literature as

the heuristic is newly introduced. The outcomes in Table 4 seem most appropriate given average gap and calculation time of the sample of instances.

The value of the time limit parameter is not reported in Table 4, as we set it to five minutes for all methods. The reason for choosing five minutes, is that it should be possible to find a reasonable solution within five minutes in real world applications.

Table 4: Parameter tuning for each of the heuristics, based on 20 OPHS and TOPHS instances.

TSVNS			TABC			TAV		
Param.	Considered	Chosen	Param.	Considered	Chosen	Param.	Considered	Chosen
NIM[1]	25, 50, 100	50	L	5, 10, 20	20	k	5, 10, 20	5
$NUFC$	150, 250, 350	250	nEB	5, 10, 20	10	n	5, 10, 20	5
$Kmax$	$NUFC/2$,	$NUFC/4$	nOB	2, 5, 10	5	l	5, 10, 20	5
	$NUFC/4$,		nSB	1, 2, 5	5	$maxIter$	10, 15, 20	15
	$NUFC/8$		ρ	0.1, 0.3, 0.5	0.3			
MPW[2]	0.2, 0.3, 0.4	0.3	μ	0.015,	0.015			
				0.0015,				
				0.00015				
			$maxIter$[3]	$80 \times n$,	$800 \times n$			
				$800 \times n$,				
				$8000 \times n$				

[1] $NoImprovementMax$

[2] $MaxPercentageWorse$

[3] n is the number of vertices in a solution, so $N + (D-1) \times Q + (Q-1)$

4.2 Performance Analysis for the OPHS

Firstly, we study the case where $Q = 1, T = \infty$, as it allows us to use the OPHS instances from Divsalar et al. (2013). Namely, when $Q = 1$ and $T = \infty$, the TOPHS reduces to an OPHS.

We deployed CPLEX and all three heuristics on all instances discussed by Divsalar et al. (2013). The results per instance are presented in Tables 10 to 18 in Appendix C. The aggregated results presented in Table 5 and Table 6 are discussed here.

Table 5 shows the average gap and average computation time of each method, separated by the set (as defined in Divsalar et al. (2013)), number of trips in the tour (D) and the number of 'extra' hotels (EH), which is the total number of hotels excluding the start and end hotels. The gap is defined as the relative difference between the known optimum and the heuristic value with respect to the optimum. Note that for some instances the objective is not known. For these instances, the gap is not reported with respect to the objective, but with respect to the best known feasible solution found with CPLEX within five minutes. The weighted average over all instances with known optimum is also reported, to compare the performance of all methods.

Table 5: Average gap and average computation time for the OPHS problem (TOPHS with $Q = 1$ and $T = \infty$), for each of the solving methods. D: number of trips, EH: number of hotels, except start and end hotel, #: number of instances in this set

Instance				CPLEX		TSVNS		TABC		TAV	
SET	D	EH	#	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)
1	2	1	35	5.10	151.14	1.64	5.14	4.76	21.25	1.18	10.33
1	3	2	35	8.1	234.31	1.38	5.09	8.26	18.40	4.99	8.72
1	4	3	35	13.56	235.42	1.67	9.95	12.60	19.30	10.76	7.69
2	3	5	35	9.80	240.20	1.34	7.17	7.09	22.14	4.87	9.10
2	4	6	35	9.26	224.84	1.74	18.24	13.01	20.99	12.26	7.85
3	4	10	22	36.70	300.00	4.02	144.11	15.49	59.96	9.60	66.36
3	5	12	22	43.15	300.00	4.77	128.32	19.34	48.90	13.79	59.87
4	2	3	5	9.76	273.72	0.92	1.54	3.58	80.86	0.97	2.13
4[1]	3	3	5	0.00	300.00	-90.3	4.0	-90.29	79.16	-92.39	4.52
Weighted Average[2]:				15.22	234.71	2.10	33.91	10.65	28.45	7.56	19.24

[1] Instances have no known objective value, so the gap presented is the gap with respect to the best feasible solution found using CPLEX.

[2] Only instances with known optimal value are included.

From this table, we observe that CPLEX has a relatively high gap, which is to be expected as it is an exact algorithm. Therefore, if it does not terminate, there may be a substantial gap, which is indeed what we see here.

Regarding the three heuristics, we see that the TSVNS algorithm results in the lowest average gap over all instances. However, the average computation time is the highest over all heuristics. This seems to be mainly caused by the instances where many additional hotels have been added (SET3). The cause of this is that the TSVNS approach lists all feasible combinations of hotels and solves a subOP for each pair of hotels before the improvement phase. This is computationally intensive, especially when the number of hotels is largee. As we do not have this problem for the TABC and TAV heuristics, we expect their calculation times to be substantially lower, which is indeed what we see.

Overall, the TSVNS still outperforms the other heuristics, as the average computation times seem comparable but the gap is substantially lower that of the other heuristic. Nevertheless, the TAV seems to be the best alternative when the instances are larger, providing slightly larger gaps but substantially lower computation times.

Another conclusion that can be drawn from these results is that the TABC heuristic does not perform too well, compared to both the TSVNS and the TAV heuristic, both in terms of average computation time and average gap. This may be due to the nature of the hotels and the problem of allocating them such that the route between them achieves a maximum score, which may not be fully captured by the algorithm as it mainly performs random neighborhood searches.

Table 6: Relative comparison table for each solving method, and for each set of instances of the OPHS. D: number of trips, EH: number of hotels, except start and end hotel, #: number of instances in this set.

Instance Set				CPLEX			TSVNS			TABC			TAV		
Set	D	EH	#	Opt[1]	≥[2]	>[3]	Opt	≥	>	Opt	≥	>	Opt	≥	>
1	2	1	35	19	21	5	14	24	5	3	3	0	17	24	1
1	3	2	35	10	13	1	18	31	14	6	6	0	9	14	2
1	4	3	35	13	14	3	15	31	19	4	3	0	4	5	1
2	3	5	35	13	14	2	18	30	15	6	7	0	8	12	3
2	4	6	35	13	14	7	14	26	17	4	2	0	2	4	2
3	4	10	22	0	0	0	2	21	19	0	0	0	0	3	1
3	5	12	22	0	0	0	3	21	21	0	0	0	0	1	1
4	2	3	5	2	2	1	3	4	0	1	2	0	3	4	0
4	3	3	5	-	0	0	-	3	2	0	2	1	0	2	0
Total:			229	70	78	19	87	191	112	24	25	1	43	69	11

[1] The column 'Opt' denotes the number of instances for which this method found the known optimum.

[2] The column '≥' denotes the number of instances for which this method achieved the best objective value of all methods.

[3] The column '>' denotes the number of instances for which this method achieved a higher objective than all other methods.

From Table 6, it also follows that the TSVNS is the best performing method in general, as it outperforms the other methods in terms of objective value for 112 instances, which is almost half of all instances. For 191 instances, it did not perform worse than any other heuristic, which is more than any other method. Nevertheless, there exist instances for which the other methods outperform the objective value that TSVNS finds, but this is comparatively a small number.

Again, we also see that the TAV is more successful than the TABC. This provides evidence that the random neighborhoods of the TABC are not as appropriate as the more informed searches of the TAV and TSVNS.

4.3 Performance Analysis for the TOPHS

Now, we study cases where $Q > 1$. We generate the instances as explained in Section 3.5. SET 1, SET 2 and SET 3 contain 63 instances each, whereas SET 4 contains 39 instances.

Again, CPLEX and all heuristics are deployed on these instances. The results for all separate instances can be found in Tables 19 to 30 in Appendix D. The aggregated results are presented in Table 7 and Table 8 and are discussed here.

Table 7: Average gap and average computation time for the TOPHS instances, for each of the solving methods. D: number of trips per tour, AH: number of hotels additional to those needed to satisfy the number of trips (start hotel, end hotel and $(D-1) \times Q$ intermediate hotels). #: number of instances in this set.

Instance				CPLEX		TSVNS		TABC		TAV	
SET	D	AH	#	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)
1	1	0	21	0.27	164.12	1.30	1.66	1.35	20.26	0.22	3.67
1	2	0	21	2.96	186.50	0.20	2.44	2.73	24.69	1.33	2.10
1	3	0	21	5.99	216.57	0.48	99.55	6.04	22.17	4.96	1.79
2	2	1	21	0.88	184.78	0.30	2.31	3.23	21.16	2.27	2.24
2	2	2	21	1.50	191.89	0.55	2.51	2.89	25.50	2.84	2.12
2	2	3	21	1.64	194.56	0.42	3.31	3.32	29.04	3.66	2.22
3[1]	2	1	21	0.00	100.12	-3.20	1.34	-1.38	19.93	-1.95	1.47
3[1]	2	2	21	0.00	108.31	-7.27	1.31	-5.52	22.99	-5.31	1.42
3[1]	2	3	21	0.00	98.91	-5.35	1.61	-3.36	21.39	-4.66	1.38
4[1]	2	1	13	0.00	300.00	-142.88	52.66	-132.17	37.93	-136.07	27.38
4[1]	2	2	13	0.00	300.00	-156.87	62.60	-138.68	34.47	-148.13	27.35
4[1]	2	3	13	0.00	300.00	-146.85	74.52	-131.72	25.46	-132.89	24.71
Weighted Average[2]:				2.20	189.74	0.54	18.63	3.26	23.80	2.55	2.36

[1] Instances have no known objective value, so the gap presented is the gap with respect to the best feasible solution found using CPLEX.

[2] Only instances with known optimal value are included.

In Table 7, we find again that the TSVNS algorithm achieves the lowest gap to the known objective values on average. Also for the instances with unknown objective value, it achieves the highest objective value on average. Notably, the average gap is smaller than for CPLEX, whereas the computation time is substantially lower.

For the case where we have three trips per tour, the high average computation time is caused by some instances running out of the 300 second time limit. As these are larger instances, this is in line with our expectations that the TSVNS is slower for larger instances because it has to iterate over all possible pairs of hotels and construct all feasible combinations of hotels. We also see this behaviour for SET 4, which consists of the largest instances available. In these cases TABC and TAV are quicker, and they also achieve an acceptable gap to the best known values.

In general, TAV slightly outperforms TABC in terms of average gap, and it is a lot quicker. TAV thus seems to be a better method than TABC, which may again be caused by the limited number of ways in which the TABC neighborhoods can change a solution. This is even the case for the TOP instances (SET 1 with $D = 1$), even though the TABC is derived from an ABC heuristic for the TOP (see Section 3.3).

TAV achieves a lower gap than TSVNS, but the gaps still seem reasonable to work with. The benefit that TAV is a lot quicker than TSVNS may outweigh the disadvantage of the slightly higher gap in real world applications. Especially for larger instances, the TSVNS running times may become obstructive, making the TAV algorithm more promising.

Table 8: Relative comparison table for each solving method, and for each set of instances of the OPHS. D: number of trips per tour, AH: number of hotels additional to those needed to satisfy the number of trips (start hotel, end hotel and $(D-1) \times Q$ intermediate hotels). #: number of instances in this set.

Instance Set				CPLEX			TSVNS			TABC			TAV		
Set	D	AH	#	Opt[1]	≥[2]	>[3]	Opt	≥	>	Opt	≥	>	Opt	≥	>
1	1	0	21	18	19	2	16	16	0	16	16	0	17	18	2
1	2	0	21	15	15	0	18	21	4	18	21	4	15	16	0
1	3	0	21	13	13	1	17	19	5	13	13	0	12	13	0
2	2	1	21	15	17	2	17	18	1	12	12	0	15	15	1
2	2	2	21	15	17	3	17	18	3	13	13	0	12	12	0
2	2	3	21	13	14	1	17	20	6	13	13	0	11	11	0
3	2	1	21	-	12	0	-	19	3	-	15	0	-	17	2
3	2	2	21	-	12	0	-	20	5	-	15	0	-	15	1
3	2	3	21	-	13	0	-	21	4	-	15	0	-	17	0
4	2	1	13	-	0	0	-	9	9	-	1	1	-	3	3
4	2	2	13	-	0	0	-	10	7	-	3	0	-	5	3
4	2	3	13	-	2	0	-	11	9	-	3	0	-	2	1
Total:			228	89	134	9	102	202	56	85	140	5	82	144	13

[1] The column 'Opt' denotes the number of instances for which this method found the known optimum.

[2] The column '≥' denotes the number of instances for which this method achieved the best objective value of all methods.

[3] The column '>' denotes the number of instances for which this method achieved a higher objective than all other methods.

From Table 8, we see that the TSVNS heuristic outperforms all other methods in 56 cases, and is not outperformed in 202 cases, which is almost 90%. The second best heuristic (by some margin) in this perspective is the TAV, which outperforms the other methods in 13 cases. Notably, 7 of those cases are in SET 4 which contains larger instances. TAV might thus have an edge in larger instances, when TSVNS is more likely to hit the running time limit.

It can be concluded that the TSVNS method is still the best performing method in terms of objective value, but if computation time is relevant, the TAV may be a good alternative with slightly lower objective values but a lot quicker computation.

TABC again does not seem to be better than the other methods, even though it outperforms the other methods in five cases.

5 Conclusion

In this research, the Team Orienteering Problem with Hotel Selection (TOPHS) is introduced as a multi-tour extension of the Orienteering Problem with Hotel Selection (OPHS) proposed by Divsalar et al. (2013). Given the NP-hard nature of the problem, the study aims to assess the effectiveness of three heuristics specifically tailored for TOPHS.

The first heuristic, termed the TSVNS algorithm, is an extension of the heuristic proposed by Divsalar et al. (2013) for OPHS. The other two heuristics are based on an artificial bee colony (ABC) search approach, with one, referred to as TABC, implementing relatively random moves, while the other, known as TAV, utilizes neighborhoods related to those of TSVNS. The primary

objective of this research is to compare the performance of these ABC-based heuristics against the SVNS-based heuristic for solving the TOPHS.

To evaluate the effectiveness of the heuristics, we introduce a procedure to generate TOPHS instances from the team orienteering problem (TOP). We have generated 126 instances with known optimal values and 102 larger instances with an unknown optimal value.

After tuning the parameters of each heuristic, an exact solver and the three heuristics are employed on instances of the OPHS. The results reveal that, on average, TSVNS yields the best performance. However, for larger instances, the computation times significantly increase due to the initialization phase of the algorithm, which involves heuristically solving an orienteering problem for all pairs of hotels and exploring all feasible hotel combinations before starting the neighborhood search algorithm.

Although TAV exhibits a larger average gap compared to TSVNS, it achieves lower average running times. Therefore, the TAV may have an edge for larger instances, where the longer execution time of the TSVNS may become obstructive. Nevertheless, TSVNS remains superior for smaller instances. Notably, TAV outperforms TABC in terms of both running time and average gap.

Regarding TOPHS instances, TSVNS once again outperforms the other algorithms, as it achieves the best solution over all methods in nearly 90% of cases within reasonable average computation time. However, the TAV offers a substantially lower average computation time with a higher but still acceptable average gap.

The TABC heuristic continues to underperform, primarily due to its limited freedom to move to a neighbour. This restriction arises from the fact that the number of hotels per tour may not change when transitioning to a neighbour.

In conclusion, the TSVNS heuristic outperforms the ABC-based heuristics for solving the TOPHS. However, the TAV heuristic demonstrates potential advantages for larger instances, as it offers better running times while still maintaining an acceptable gap.

6 Discussion

In this section, we critically evaluate the obtained conclusions. We address some points in the analysis of the heuristics that may have lead to biases or some implementation choices that may affect the final outcomes. Ultimately, we also give suggestions for future research.

Firstly, even though we have performed parameter tuning in this research, only a small number of potential parameter values have been analyzed. A more profound analysis might arrive at more competitive heuristics, especially for the TABC and the TAV, which were more sensitive to changes in the parameters than TSVNS.

Secondly, the neighborhoods of the TSVNS are implemented using an idea similar to the SVNS for the OPHS by Divsalar et al. (2013). These neighborhoods are not specifically tailored to the TOPHS, which could reduce effectiveness slightly. A similar problem occurred for the TABC, where the range of reachable neighbours was restricted due to the condition that the number of hotels needs to be the same in every tour. Designing neighborhoods more specifically tailored to the OPHS would be beneficial for this.

Another consideration for the TABC heuristic is that we made a solution representation

feasible by removing vertices from the end of each trip (see Section 3.3). We chose this approach to minimize computation time and promote exploration, but it is less likely to create an optimal tour. This decision may have had influence on the effectiveness of the TABC heuristic.

Additionally, all the heuristics depend on a random number generator. Therefore, these outcomes are unique and another run could give different values and conclusions. A perhaps more insightful way to study the heuristics would be to run each instance multiple times and take averages.

Furthermore, we mainly use relatively small instances in our research. The reason is that we need an optimal solution to the TOP to generate instances of the TOPHS with a known optimal value. As a result, SET 1, 2 and 3 all have relatively little POIs in their optimal tours. This may bias our results towards these smaller instances. It would be interesting to study larger instances as well, as the TSVNS might not be as competitive anymore due to its computationally advanced initialization algorithm. However, this requires solving large TOP instances.

Notably, the results for the TSVNS heuristic for the OPHS have substantially higher execution times than the SVNS heuristic in Divsalar et al. (2013), even though the same instances were used and for one tour, the TSVNS and SVNS are the same. The difference in computation times is therefore surprising. This may be caused by the coding language used: we used Java and not C++, which is known to be more efficient. Another reason may be differences in implementation or the usage of a different draw from the random number generator. Lastly, the used number of cores may influence the results: we used only one, whereas the number of cores used by Divsalar et al. (2013) is unknown.

As the TOPHS problem is new in the literature, not many solving methods exist. Future research could therefore focus on more competitive heuristics than the ones discussed in this research, such as for example simulated annealing or ant colony search.

It is also interesting to study variants of the TOPHS problem. For example, one could add time windows or opening hours to the POIs and hotels. Another potential improvement is to allow the hotels to have a score as well. As such, a hotel further away may be preferred over the closest one. This is a more realistic scenario.

One particular problem improvement is optimizing the total length of the set of tours, besides optimizing the sum of scores. The reason is that during the research, we found that some problem instances have optimal solutions that share the same objective value, whereas their lengths differed substantially. To prevent this, one could think of a bi-objective TOPHS, which first maximizes the sum of scores and then, in case of equality, minimizes the total length. Especially in a case where travelled distance is actually costly, minimizing this is socially relevant.

References

Álvarez-Miranda, E., Luipersbeck, M. & Sinnl, M. (2018). Gotta (efficiently) catch them all: Pokémon go meets orienteering problems. *European Journal of Operational Research*, *265*(2), 779–794.

Ataei, M., Divsalar, A. & Saberi, M. (2022). The bi-objective orienteering problem with hotel selection: An integrated text mining optimisation approach. *Information Technology and Management*, 1–29.

Chao, I.-M. (1993). *Algorithms and solutions to multi-level vehicle routing problems.* University of Maryland, College Park.

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

Chao, I.-M., Golden, B. L. & Wasil, E. A. (1996b). The team orienteering problem. *European Journal of Operational Research*, *88*(3), 464-474.

Cura, T. (2014). An artificial bee colony algorithm approach for the team orienteering problem with time windows. *Computers & Industrial Engineering*, *74*, 270–290.

Divsalar, A., Emami, S. & Vansteenwegen, P. (2017). A lagrange relaxation for the orienteering problem with hotel selection and time windows. In *Verolog, date: 2017/07/10-2017/07/12, location: Amsterdam.*

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., Chitsaz, M., Sörensen, K. & Cattrysse, D. (2014). Personalized multi-day trips to touristic regions: a hybrid ga-vnd approach. In *Evolutionary computation in combinatorial optimisation* (pp. 194–205).

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.

Dorigo, M. & Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, *1*(1), 53–66.

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

Gavalas, D., Konstantopoulos, C., Mastakas, K. & Pantziou, G. (2014). A survey on algorithmic approaches for solving tourist trip design problems. *Journal of Heuristics*, *20*, 291–328.

Golden, B. L., Levy, L. & Vohra, R. (1987). The orienteering problem. *Naval Research Logistics (NRL)*, *34*(3), 307–318.

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.

Karaboga, D. (2005, 01). An idea based on honey bee swarm for numerical optimization, technical report - tr06. *Technical Report, Erciyes University*.

Sohrabi, S., Ziarati, K. & Keshtkaran, M. (2020). A greedy randomized adaptive search pro-

cedure for the orienteering problem with hotel selection. *European Journal of Operational Research*, *283*(2), 426–440.

Sohrabi, S., Ziarati, K. & Keshtkaran, M. (2021). ACS-OPHS: Ant colony system for the orienteering problem with hotel selection. *EURO Journal on Transportation and Logistics*, *10*, 100036.

Toledo, A., Riff, M.-C. & Neveu, B. (2019). A hyper-heuristic for the orienteering problem with hotel selection. *IEEE Access*, *8*, 1303–1313.

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

Vansteenwegen, P. & Gunawan, A. (2019). Orienteering problems. *EURO Advanced Tutorials on Operational Research*.

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

A Proof upperbound TOPHS

A.1 Start and end hotel are the same

Given that we have a TOPHS instance R where (1) the start and end hotel are the same and (2) where

$$T \geq \sum_{d=1}^{D} T_d, \tag{14}$$

we can construct an OPHS instance O with an objective value at least as large as the TOPHS instance. In other words, if $f(x)$ denotes the objective of problem x, then

$$f(R) \leq f(O). \tag{15}$$

Proof: Firstly, note that an optimal solution to the TOPHS instance R with Q tours and D trips per tour has a total of $Q \times D$ trips. If we append these trips behind each other, we have one tour of $Q \times D$ trips.

To make the trip distances correspond to the trip distance limits of the TOPHS, we define the trip limits as follows. Define T'_f as the trip distance limit in the new tour for all $f \in \mathbb{N}, f \leq Q \times D$. Then,

$$T'_f = T_{f-nD}, \forall n \in \{0\} \cup \mathbb{N} \text{ s.t. } f - nD \leq D \tag{16}$$

or, equivalently

$$T'_f = T_{f-nD}. \forall n \in \{0\} \cup \mathbb{N} \text{ s.t. } f \leq (n+1)D. \tag{17}$$

As also $n < Q$ or $n + 1 \leq Q$, we have

$$T'_f = T_{f-nD}, \forall n \in \{0\} \cup \mathbb{N} \text{ s.t. } f \leq (n+1)D \leq Q \times D. \tag{18}$$

.

So now we have a tour where each trip limit corresponds to the same trip in the original TOPHS R. Note that this is just a different way to represent the same solution to TOPHS R.

We now construct the OPHS O. Let O have the same hotels and POIs as R. Let O have $Q \times D$ trips. Let the time limits of the trips of O be given by $T'_f, f \leq Q \times D$. Let the start hotel and end hotel of O be the same as those of R.

Now, we have an OPHS. As the optimal solution to the TOPHS R is still feasible in this form of the OPHS O by condition (1), we have that

$$f(R) \leq f(O). \tag{19}$$

This proves the statement. □

Note that we used condition (1) to assure that the TOPHS solution is feasible in the OPHS, because if condition (1) would not hold, not all trips in the OPHS would start in the same hotel as where the previous trip ended. Also, we use condition (2) as the OPHS formulation does not

allow us to restrict the time limit on a subset of sequential trips.

A.2 Start and end hotel differ

Given that we have a TOPHS instance R where

$$T \geq \sum_{d=1}^{D} T_d, \tag{20}$$

we can construct an OPHS instance O with an objective value at least as large as the TOPHS instance. In other words, if $f(x)$ denotes the objective of problem x, then

$$f(R) \leq f(O). \tag{21}$$

Proof: Firstly, note that the optimal solution to the TOPHS instance R with Q tours and D trips per tour has a total of $Q \times D$ trips. If we append these trips behind each other, we have one tour of $Q \times D$ trips. However, these trips cannot directly follow after each other, because the end hotel of the ith trip with $i = m \times D, m \in \mathbb{N}, m < Q$ differs from the start hotel of the following trip.

Therefore, we need to define a new trip between the ith and $i+1$th trip, $i = m \times D, m \in \mathbb{N}, m < Q$. This gives us $Q-1$ additional trips, bringing the total to $Q \times D + Q - 1 = Q \times (D+1) - 1$.

For this new tour, we define the trip limits as follows. Define T'_f as the trip distance limit in the new tour for all $f \in \mathbb{N}, f \leq Q \times (D+1) - 1$. Also, define T_0 =the travel time from the end hotel to the start hotel. Then,

$$T'_f = T_{f-n(D+1)}, \forall n \in \{0\} \cup \mathbb{N} \text{ s.t. } f - n(D+1) \leq D. \tag{22}$$

So now we have a tour where each trip limit corresponds to the same trip in the original TOPHS R. Note that this is just a different way to represent the same solution to TOPHS R.

We now construct the OPHS O. Let O have the same hotels and POIs as R. Let O have $Q \times (D+1) - 1$ trips. Let the time limits of the trips of O be defined by $T'_f, f \in \mathbb{N}, f \leq Q \times (D+1) - 1$. Let the start hotel and end hotel of O be the same as those of R.

Now, we have an OPHS. As the optimal solution to the TOPHS R is still feasible in this form of the OPHS O, we have that

$$f(R) \leq f(O). \tag{23}$$

This proves the statement. □

Note that we use condition (21) as the OPHS formulation does not allow us to restrict the time limit on a subset of sequential trips.

B Code

This Appendix gives an introduction to the code used. We used Java for all code files.

Each class belongs to one of three categories: problem, solution and solver. The problem category defines the problem, its vertices and its parameters. The solution category defines the parts that belong to a data structure representing a solution to the TOPHS problem. The solver category represents the heuristics and the exact solving methods. An overview is given in Table 9.

The code can be used by opening the project in a Java IDE, and running the method *main* in the Main class. This produces the results similar to those in the paper.

For parameter tuning, the main methods in the TSVNS, TABC and TAV classes can be run.

Now follows a brief description of all classes and their most important variables and methods.

Table 9: Code classes used in this research, in alphabetical order per category

Problem	Solution	Solver
OPHS	Hotel	ExactOPHS
TOP	Node	ExactTOP
TOPHS	POI	ExactTOPHS
	Tour	Main
	TourSeparator	TABC
	TTour	TAV
	TTourRep	TSVNS

B.1 Solution

Firstly, we discuss the data structure for the solution. We start with the Node data structure, as it is a superclass of the Hotel, POI and TourSeparator class.

B.1.1 Node

A Node contains a set of (x, y)-coordinates and a name. Apart from getter methods, it contains a method *this.distanceTo(other)*, which returns the Cartesian distance from *this* to *other*. Note that the definition of a Node differs compared to the main text of the paper: Node here means the same as *vertex* in the main text.

B.1.2 Hotel

Subclass of Node with no additional methods.

B.1.3 POI

Subclass of Node, which apart from the (x, y)-coordinates and name also records a score.

B.1.4 TourSeparator

Subclass of Node, which is used to indicate the start of a new tour for the TOPHS instances in the TABC heuristic. This class does not contain any additional methods or instance variables compared to a Node.

B.1.5 Tour

Contains a list of Nodes called *nodes*, a list of doubles called *tds* representing the time limits per trip and a double called *tmax* representing the time limit for a whole tour. Important methods defined within the class are:

- *getVal()*, which returns the sum of the scores of all POIs in *nodes*.
- *getLength()*, which returns the sum of Cartesian distances between all subsequent pairs of Nodes in *nodes*.
- *isBetterThan(other)*, which returns TRUE if this tour either has a higher value or if the value is the same but it has a lower length.
- *makeTrips()*, which returns a list of lists of Nodes representing the separate trips in the tour
- *isFeasible()*, which determines if the time requirements are satisfied for the tour and no duplicates are present

B.1.6 TTour

Contains a list of Tour objects representing a solution to the TOPHS problem. Its most important methods are *getVal(), getLength(), isBetterThan(other)* and *isFeasible()*, which have a similar meaning as in the Tour class.

B.1.7 TTourRep

Contains a list of Nodes that acts as the representation of a solution for the TABC heuristic. Its most important method is *makeTTour()*, which converts the TTourRep into a feasible TTour, if possible (and null otherwise). Apart from that, it also contains the methods *getVal(), getLength(), isBetterThan(other)* and *isFeasible()*, which have a similar meaning as in the Tour class.

B.2 Problem

The classes in the problem category collect all parameters of the instances that we discuss in our research, as well as store the list of nodes in the problem.

B.2.1 OPHS

The OPHS class contains several instance variables relating to the various parameters found in the OPHS. These include:

- *name*: name of the instance
- *optimalValue*: optimal value of the instance, if known
- *pois*: list of all POIs in the problem
- *hotels*: list of all Hotels in the problem
- *startHotel* and *endHotel*: the first respectively the last hotel in a tour which is a solution to the OPHS
- *numTrips*: number of trips
- *Tmax*: maximal travel time for the full tour
- *Td*: list of travel time limits for each trip separately

The constructor takes a file name from the repository by Divsalar et al. (2013) as input (which we saved in the folder "Input/OPHS/"). It then automatically creates an instance of the OPHS.

Note that in the folder "Input/OPHS/" we also included a file named "instances.txt", which maps each instance to its optimal value according to Divsalar et al. (2013).

B.2.2 TOP

Similar to the OPHS class, but then defines the parameters of the TOP. This includes among others also the number of tours to be created, which is stored as an integer *teamSize*.

Again, the constructor takes a file name from the repository by Tsiligirides (1984); Chao (1993); Chao et al. (1996a, 1996b) as input, which we saved in the folder "Input/TOP/".

B.2.3 TOPHS

Similar to the OPHS class, with an additional parameter for the number of tours to be created, referred to as *teamSize*.

The constructor takes again a file name, which corresponds to the file names given when creating the TOPHS instances from the TOP instances. They are stored in the folder "Input/TOPHS/".

This class also has the methods *SET1*, *SET2* and *SET3*, which create the instances of the TOPHS as described in Section 3.5. Note that the instances of SET 4 can be generated using the method *SET3* as well.

Another useful method is *getTOPHS(ophs)* which takes an OPHS instance as argument and converts it to a TOPHS problem (by copying the parameters and setting *teamSize=1*).

B.3 Solution

This category of classes involve the implementation of CPLEX and heuristic solving methods.

B.3.1 ExactOPHS

Implementation of the OPHS MIP from Divsalar et al. (2013) in CPLEX. After creating an instance of this class, by passing an OPHS instance to the constructor. The method *solve(tlim)* attempts to solve the instance given *tlim* as a time limit.

If feasible (then *this.getStatus()* returns "Optimal" or "Feasible"), the outcomes can be retrieved using the methods *getOptVal()* and *getX()*.

B.3.2 ExactTOP

Implementation of the TOP MIP from Vansteenwegen et al. (2011) in CPLEX. After creating an instance of this class, by passing a TOP instance to the constructor. The method *solve(tlim)* attempts to solve the instance given *tlim* as a time limit.

The method *makeInstances()* runs the exact TOP algorithm for all TOP instances. Next, it exports the optimal solution (or best known feasible solution if the time limit is reached) to a file in the folder "Output/TOP/" which can be used for generating the TOPHS instances.

B.3.3 ExactTOPHS

Implementation of the TOP MIP from Section 3.1 in CPLEX. After creating an instance of this class, by passing a TOPHS instance to the constructor. The method *solve(tlim)* attempts to solve the instance given *tlim* as a time limit.

If feasible (then *this.getStatus()* returns "Optimal" or "Feasible"), the outcomes can be retrieved using the methods *getOptVal()* and *getX()*.

B.3.4 Main

The main class contains two methods, *mainOPHS* and *mainTOPHS*. These methods solve (a given subset of) all instances of respectively the OPHS and TOPHS using the exact method and each of the heuristics, given a set time limit. They result in a text file of the form "Output/allOutputOPHS[]" and "Output/allOutputTOPHS[]" respectively, where [] denotes a range of values ran. These files form the basis for our analysis in this paper.

B.3.5 TABC

This is the implementation of the TABC heuristic, as described in Section 3.3. After creating an instance for a given TOPHS and a set of parameters, the method *solve(tlim)* can be called with a given time limit *tlim*. It returns the TTour that the algorithm found using its search procedure.

The class also contains a method *paramTuning* that runs a sample of a given number of instances for a set of different parameter values, to assess what parameter settings are most appropriate.

B.3.6 TAV

This is the implementation of the TAV heuristic, as described in Section 3.4. After creating an instance for a given TOPHS and a set of parameters, the method *solve(tlim)* can be called with a given time limit *tlim*. It returns the TTour that the algorithm found using its search procedure. Similar to the TABC, it also includes a *paramTuning* method.

B.3.7 TSVNS

This is the implementation of the TSVNS heuristic, as described in Section 3.2. After creating an instance for a given TOPHS and a set of parameters, the method *solve(tlim)* can be called with a given time limit *tlim*. It returns the TTour that the algorithm found using its search procedure. Similar to the TABC, it also includes a *paramTuning* method.

C Results per instance of the OPHS

Table 10: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of OPHS SET1 with 2 trip and 1 extra hotel.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
T1-65	240	240	0.00	6.56	240	0.00	1.04	235	2.08	2.02	240	0.00	2.26
T1-70	260	260	0.00	5.64	260	0.00	0.91	225	13.46	2.11	260	0.00	2.19
T1-73	265	265	0.00	7.47	245	7.55	1.10	250	5.66	1.97	265	0.00	2.45
T1-75	270	270	0.00	3.97	270	0.00	1.03	265	1.85	1.95	270	0.00	2.58
T1-80	280	280	0.00	7.60	265	5.36	1.17	270	3.57	3.03	270	3.57	2.81
T1-85	285	285	0.00	11.44	280	1.75	1.22	265	7.02	2.30	280	1.75	3.20
T3-65	610	610	0.00	33.36	610	0.00	1.06	600	1.64	2.95	610	0.00	2.12
T3-75	670	670	0.00	10.14	650	2.99	1.14	630	5.97	2.47	670	0.00	2.55
T3-80	710	710	0.00	35.52	690	2.82	1.40	700	1.41	1.73	700	1.41	2.96
T3-85	740	740	0.00	37.78	740	0.00	1.31	690	6.76	2.66	740	0.00	3.06
T3-90	770	770	0.00	12.54	770	0.00	1.35	730	5.19	1.81	770	0.00	3.12
T3-95	790	790	0.00	7.79	780	1.27	1.45	770	2.53	2.35	790	0.00	3.26
T3-100	800	800	0.00	38.68	800	0.00	2.00	790	1.25	2.89	800	0.00	3.37
T3-105	800	790	1.25	300.00	800	0.00	1.53	770	3.75	3.06	800	0.00	3.67
64-45	816	816	0.00	12.90	798	2.21	3.51	804	1.47	25.40	816	0.00	9.16
64-50	900	876	2.67	300.00	876	2.67	5.02	870	3.33	20.99	876	2.67	11.61
64-55	984	948	3.66	300.00	972	1.22	6.11	942	4.27	19.58	972	1.22	14.61
64-60	1062	996	6.21	300.00	1056	0.56	13.07	984	7.34	18.06	1050	1.13	20.44
64-65	1116	1116	0.00	115.38	1116	0.00	9.75	1062	4.84	17.70	1116	0.00	23.50
64-70	1188	1146	3.54	300.00	1170	1.52	13.94	1110	6.57	22.01	1170	1.52	26.96
64-75	1236	1230	0.49	300.00	1212	1.94	12.28	1176	4.85	19.01	1218	1.46	32.59
64-80	1284	1212	5.61	300.00	1278	0.47	17.53	1230	4.21	12.19	1266	1.40	35.50
66-40	575	495	13.91	300.00	570	0.87	1.30	540	6.09	25.59	570	0.87	3.08
66-45	650	615	5.38	300.00	615	5.38	1.72	615	5.38	19.51	645	0.77	5.24
66-50	730	480	34.25	300.00	715	2.05	4.19	705	3.42	24.14	690	5.48	5.42
66-55	825	590	28.48	300.00	805	2.42	3.44	680	17.58	17.92	785	4.85	7.67
66-60	915	620	32.24	300.00	890	2.73	4.77	820	10.38	17.87	890	2.73	9.08
66-125	1670	1485	11.08	300.00	1665	0.30	24.86	1595	4.49	14.79	1665	0.30	51.32
66-130	1680	1580	5.95	300.00	1680	0.00	34.54	1675	0.30	14.76	1680	0.00	53.10
100-30	173	173	0.00	28.38	173	0.00	0.42	173	0.00	58.39	173	0.00	0.96
100-35	241	241	0.00	10.05	227	5.81	0.89	241	0.00	56.12	227	5.81	2.42
100-40	299	299	0.00	63.45	283	5.35	1.59	298	0.33	89.51	298	0.33	3.19
100-45	367	367	0.00	41.10	367	0.00	2.28	310	15.53	65.37	367	0.00	3.29
102-50	181	153	15.47	300.00	181	0.00	0.33	181	0.00	82.39	181	0.00	0.82
102-60	243	223	8.23	300.00	243	0.00	0.81	233	4.12	69.26	233	4.12	1.95
Average:			5.10	151.14		1.64	5.14		4.76	21.25		1.18	10.33

Table 11: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of OPHS SET1 with 3 trips and 2 extra hotels.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
T1-65	240	215	10.42	300.00	240	0.00	0.97	235	2.08	1.71	210	12.50	1.54
T1-70	260	260	0.00	78.72	260	0.00	1.04	205	21.15	1.22	250	3.85	1.79
T1-73	265	265	0.00	64.41	265	0.00	1.04	265	0.00	2.20	240	9.43	1.86
T1-75	270	270	0.00	115.35	270	0.00	1.07	230	14.81	2.38	260	3.70	1.87
T1-80	280	280	0.00	57.75	280	0.00	1.27	270	3.57	2.23	270	3.57	2.40
T1-85	285	275	3.51	300.00	285	0.00	1.82	260	8.77	2.38	265	7.02	2.47
T3-65	610	520	14.75	300.00	610	0.00	0.98	530	13.11	2.37	610	0.00	1.58
T3-75	670	630	5.97	300.00	670	0.00	1.26	580	13.43	2.03	650	2.99	2.21
T3-80	710	610	14.08	300.00	680	4.23	1.25	650	8.45	2.39	640	9.86	2.42
T3-85	740	710	4.05	300.00	740	0.00	1.45	640	13.51	2.56	670	9.46	2.59
T3-90	770	770	0.00	170.32	740	3.90	1.50	670	12.99	2.39	770	0.00	2.91
T3-95	790	790	0.00	85.00	790	0.00	1.57	740	6.33	2.76	720	8.86	3.23
T3-100	800	750	6.25	300.00	800	0.00	1.70	760	5.00	1.87	800	0.00	3.30
T3-105	800	760	5.00	300.00	800	0.00	1.96	750	6.25	1.88	750	6.25	3.37
64-45	816	582	28.68	300.00	762	6.62	3.13	726	11.03	12.72	720	11.76	6.14
64-50	900	648	28.00	300.00	870	3.33	4.70	792	12.00	14.16	876	2.67	8.37
64-55	984	972	1.22	300.00	948	3.66	6.07	918	6.71	12.74	966	1.83	12.03
64-60	1062	930	12.43	300.00	1050	1.13	15.54	984	7.34	12.98	996	6.21	14.11
64-65	1116	996	10.75	300.00	1098	1.61	9.42	1068	4.30	14.26	1038	6.99	18.72
64-70	1188	1128	5.05	300.00	1152	3.03	19.03	1134	4.55	17.03	1146	3.54	20.59
64-75	1236	1122	9.22	300.00	1206	2.43	18.29	1176	4.85	18.25	1200	2.91	28.54
64-80	1284	1236	3.74	300.00	1254	2.34	16.21	1200	6.54	14.23	1260	1.87	32.01
66-40	575	570	0.87	300.00	570	0.87	1.24	445	22.61	18.52	425	26.09	2.09
66-45	650	615	5.38	300.00	645	0.77	1.71	530	18.46	8.17	645	0.77	2.75
66-50	730	675	7.53	300.00	680	6.85	2.20	575	21.23	15.50	570	21.92	4.22
66-55	825	575	30.30	300.00	805	2.42	4.80	735	10.91	14.09	785	4.85	5.29
66-60	915	860	6.01	300.00	890	2.73	3.85	820	10.38	12.43	890	2.73	7.07
66-125	1670	1225	26.65	300.00	1640	1.80	23.62	1545	7.49	11.17	1640	1.80	49.20
66-130	1680	1280	23.81	300.00	1670	0.60	24.74	1615	3.87	21.51	1660	1.19	49.44
100-30	173	173	0.00	9.35	173	0.00	0.37	173	0.00	69.79	173	0.00	0.88
100-35	241	241	0.00	15.60	241	0.00	0.81	223	7.47	41.78	241	0.00	2.26
100-40	299	299	0.00	56.48	299	0.00	0.90	299	0.00	55.10	299	0.00	2.41
100-45	367	367	0.00	47.84	367	0.00	1.46	367	0.00	58.74	367	0.00	4.19
102-50	181	181	0.00	300.00	181	0.00	0.35	181	0.00	74.94	181	0.00	0.44
102-60	243	196	19.34	300.00	243	0.00	0.90	243	0.00	95.58	243	0.00	0.76
Average:			8.09	234.31		1.38	5.09		8.26	18.40		4.99	8.72

Table 12: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of OPHS SET1 with 4 trips and 3 extra hotels.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
T1-65	240	240	0.00	135.91	240	0.00	1.65	200	16.67	1.81	215	10.42	1.18
T1-70	260	260	0.00	72.17	260	0.00	2.07	210	19.23	2.01	230	11.54	1.49
T1-73	265	265	0.00	40.99	265	0.00	2.16	215	18.87	1.85	240	9.43	1.63
T1-75	270	270	0.00	45.22	270	0.00	2.15	225	16.67	3.09	235	12.96	1.73
T1-80	280	280	0.00	225.01	270	3.57	2.90	240	14.29	2.75	270	3.57	2.08
T1-85	285	285	0.00	65.75	285	0.00	2.87	270	5.26	2.57	265	7.02	1.81
T3-65	610	610	0.00	228.06	610	0.00	2.05	570	6.56	2.93	520	14.75	1.51
T3-75	670	670	0.00	237.70	650	2.99	3.10	540	19.40	1.41	590	11.94	2.06
T3-80	710	620	12.68	300.00	710	0.00	2.74	560	21.13	1.79	580	18.31	1.85
T3-85	740	620	16.22	300.00	700	5.41	3.05	610	17.57	2.21	620	16.22	2.25
T3-90	770	670	12.99	300.00	720	6.49	3.09	630	18.18	2.38	700	9.09	2.57
T3-95	790	790	0.00	263.57	760	3.80	3.46	630	20.25	3.37	710	10.13	2.72
T3-100	800	690	13.75	300.00	760	5.00	4.13	700	12.50	3.80	710	11.25	2.81
T3-105	800	720	10.00	300.00	790	1.25	4.94	730	8.75	2.24	710	11.25	3.39
64-45	816	564	30.88	300.00	816	0.00	4.13	756	7.35	14.04	672	17.65	4.85
64-50	900	654	27.33	300.00	870	3.33	7.42	840	6.67	23.51	756	16.00	7.71
64-55	984	882	10.37	300.00	972	1.22	11.40	966	1.83	16.62	864	12.20	8.51
64-60	1062	684	35.59	300.00	1002	5.65	14.19	948	10.73	23.27	936	11.86	10.92
64-65	1116	1056	5.38	300.00	1116	0.00	22.68	972	12.90	11.54	1008	9.68	15.11
64-70	1188	1008	15.15	300.00	1134	4.55	26.87	1110	6.57	10.12	1104	7.07	18.25
64-75	1236	1098	11.17	300.00	1206	2.43	31.33	1170	5.34	16.89	1122	9.22	22.76
64-80	1284	1026	20.09	300.00	1260	1.87	46.49	1182	7.94	16.90	1200	6.54	25.65
66-40	575	570	0.87	300.00	570	0.87	2.25	440	23.48	13.89	435	24.35	1.56
66-45	650	215	66.92	300.00	645	0.77	3.15	550	15.38	19.36	525	19.23	2.21
66-50	730	590	19.18	300.00	715	2.05	4.61	685	6.16	16.94	595	18.49	3.04
66-55	825	620	24.85	300.00	805	2.42	5.57	730	11.52	25.39	655	20.61	3.73
66-60	915	430	53.01	300.00	910	0.55	6.97	775	15.30	16.78	765	16.39	5.41
66-125	1670	1215	27.25	300.00	1635	2.10	55.77	1515	9.28	18.73	1600	4.19	44.55
66-130	1680	1060	36.90	300.00	1645	2.08	59.89	1570	6.55	23.28	1625	3.27	49.85
100-30	173	173	0.00	21.28	173	0.00	0.43	173	0.00	88.74	NaN[1]	NaN	NaN
100-35	241	241	0.00	34.46	241	0.00	0.79	241	0.00	50.11	241	0.00	1.72
100-40	299	299	0.00	141.77	299	0.00	0.90	299	0.00	56.15	299	0.00	1.97
100-45	367	367	0.00	127.97	367	0.00	1.41	367	0.00	71.32	367	0.00	3.28
102-50	181	181	0.00	300.00	181	0.00	0.50	123	32.04	45.78	161	11.05	0.39
102-60	243	185	23.87	300.00	243	0.00	0.98	130	46.50	61.90	243	0.00	0.80
Average:			13.56	235.42		1.67	9.95		12.60	19.30		10.76	7.69

[1] The heuristic was unable to find a feasible solution within the time limit

Table 13: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of OPHS SET2 with 3 trips and 5 extra hotels.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
T1-65	240	240	0.00	175.10	240	0.00	1.29	210	12.50	2.52	240	0.00	1.60
T1-70	260	260	0.00	141.11	260	0.00	1.51	255	1.92	1.88	230	11.54	1.75
T1-73	265	265	0.00	107.13	265	0.00	1.61	220	16.98	1.73	255	3.77	1.96
T1-75	270	270	0.00	131.74	270	0.00	1.76	240	11.11	2.03	250	7.41	2.21
T1-80	280	280	0.00	151.60	280	0.00	2.01	280	0.00	1.56	265	5.36	2.31
T1-85	285	250	12.28	300.00	280	1.75	2.00	260	8.77	2.06	270	5.26	2.31
T3-65	610	500	18.03	300.00	610	0.00	1.46	530	13.11	1.30	590	3.28	1.83
T3-75	670	570	14.93	300.00	670	0.00	1.89	620	7.46	2.39	580	13.43	2.09
T3-80	710	630	11.27	300.00	690	2.82	2.27	630	11.27	2.74	670	5.63	2.56
T3-85	740	690	6.76	300.00	740	0.00	2.27	710	4.05	2.04	700	5.41	2.62
T3-90	770	770	0.00	280.14	750	2.60	2.39	700	9.09	3.47	720	6.49	3.02
T3-95	790	790	0.00	227.30	790	0.00	2.60	710	10.13	2.41	790	0.00	3.05
T3-100	800	710	11.25	300.00	800	0.00	2.69	780	2.50	2.60	760	5.00	3.32
T3-105	800	770	3.75	300.00	800	0.00	2.91	770	3.75	2.62	780	2.50	3.42
64-45	816	816	0.00	187.07	816	0.00	5.48	732	10.29	17.01	780	4.41	7.08
64-50	900	810	10.00	300.00	870	3.33	7.45	870	3.33	16.61	858	4.67	10.11
64-55	984	792	19.51	300.00	948	3.66	9.52	948	3.66	12.89	972	1.22	13.09
64-60	1062	1062	0.00	172.75	1020	3.95	12.04	978	7.91	27.56	1008	5.08	13.84
64-65	1116	1056	5.38	300.00	1098	1.61	14.86	1056	5.38	16.96	1080	3.23	19.76
64-70	1188	1116	6.06	300.00	1146	3.54	27.28	1104	7.07	10.22	1152	3.03	24.67
64-75	1236	1188	3.88	300.00	1212	1.94	22.61	1158	6.31	11.60	1206	2.43	27.75
64-80	1284	1188	7.48	300.00	1278	0.47	24.58	1212	5.61	22.22	1266	1.40	33.02
66-40	575	495	13.91	300.00	570	0.87	1.85	525	8.70	15.47	460	20.00	2.51
66-45	650	600	7.69	300.00	645	0.77	2.81	555	14.62	19.42	560	13.85	3.11
66-50	730	510	30.14	300.00	680	6.85	3.56	635	13.01	6.83	645	11.64	3.92
66-55	825	405	50.91	300.00	785	4.85	4.51	715	13.33	25.85	665	19.39	5.90
66-60	915	860	6.01	300.00	860	6.01	5.99	820	10.38	18.10	890	2.73	7.23
66-125	1670	1475	11.68	300.00	1645	1.50	35.50	1605	3.89	18.62	1635	2.10	47.84
66-130	1680	1605	4.46	300.00	1675	0.30	39.31	1645	2.08	20.08	1675	0.30	54.04
100-30	173	173	0.00	8.31	173	0.00	0.38	173	0.00	118.76	173	0.00	0.79
100-35	241	241	0.00	33.43	241	0.00	0.83	241	0.00	56.69	241	0.00	2.23
100-40	299	299	0.00	106.57	299	0.00	0.92	299	0.00	74.02	299	0.00	2.32
100-45	367	367	0.00	84.81	367	0.00	1.49	367	0.00	77.47	367	0.00	3.93
102-50	181	181	0.00	300.00	181	0.00	0.31	145	19.89	76.65	181	0.00	0.40
102-60	243	30	87.65	300.00	243	0.00	0.91	243	0.00	80.58	243	0.00	0.72
Average:			9.80	240.20		1.34	7.17		7.09	22.14		4.87	9.10

Table 14: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of OPHS SET2 with 4 trips and 6 extra hotels.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
T1-65	240	210	12.50	300.00	240	0.00	3.80	200	16.67	2.54	195	18.75	1.33
T1-70	260	260	0.00	17.69	260	0.00	4.28	215	17.31	1.46	215	17.31	1.62
T1-73	265	265	0.00	34.35	265	0.00	4.53	220	16.98	2.47	220	16.98	1.62
T1-75	270	270	0.00	13.63	270	0.00	4.92	215	20.37	1.30	225	16.67	1.85
T1-80	280	280	0.00	188.75	275	1.79	5.65	240	14.29	3.16	255	8.93	2.05
T1-85	285	280	1.75	300.00	285	0.00	5.77	230	19.30	1.62	255	10.53	2.08
T3-65	610	550	9.84	300.00	610	0.00	4.41	560	8.20	2.67	510	16.39	1.48
T3-75	670	570	14.93	300.00	650	2.99	5.44	560	16.42	1.37	550	17.91	2.19
T3-80	710	710	0.00	34.30	710	0.00	5.93	630	11.27	2.10	570	19.72	1.87
T3-85	740	740	0.00	214.16	700	5.41	6.55	620	16.22	3.04	630	14.86	2.22
T3-90	770	770	0.00	124.95	720	6.49	6.93	720	6.49	3.06	650	15.58	2.56
T3-95	790	790	0.00	150.99	750	5.06	7.62	670	15.19	2.87	710	10.13	2.75
T3-100	800	800	0.00	195.80	760	5.00	8.69	720	10.00	1.38	720	10.00	3.04
T3-105	800	780	2.50	300.00	760	5.00	8.77	730	8.75	2.71	750	6.25	3.32
64-45	816	630	22.79	300.00	798	2.21	14.79	630	22.79	18.67	660	19.12	5.72
64-50	900	660	26.67	300.00	864	4.00	18.60	768	14.67	18.76	756	16.00	7.31
64-55	984	894	9.15	300.00	972	1.22	25.97	870	11.59	15.18	852	13.41	8.80
64-60	1062	936	11.86	300.00	1002	5.65	32.08	1002	5.65	12.38	888	16.38	11.51
64-65	1116	1026	8.06	300.00	1116	0.00	42.97	1050	5.91	21.44	1056	5.38	14.71
64-70	1188	996	16.16	300.00	1152	3.03	48.64	1104	7.07	17.78	1098	7.58	19.37
64-75	1236	1194	3.40	300.00	1206	2.43	56.65	1140	7.77	15.14	1188	3.88	22.95
64-80	1284	972	24.30	300.00	1260	1.87	68.32	1188	7.48	16.35	1206	6.07	25.97
66-40	575	420	26.96	300.00	570	0.87	4.35	405	29.57	15.11	440	23.48	1.70
66-45	650	500	23.08	300.00	645	0.77	6.18	495	23.85	24.43	525	19.23	2.57
66-50	730	480	34.25	300.00	715	2.05	8.86	580	20.55	27.39	610	16.44	3.41
66-55	825	695	15.76	300.00	805	2.42	10.85	690	16.36	11.93	670	18.79	3.92
66-60	915	860	6.01	300.00	910	0.55	13.71	725	20.77	17.61	815	10.93	5.51
66-125	1670	1310	21.56	300.00	1655	0.90	97.59	1555	6.89	16.61	1630	2.40	43.99
66-130	1680	1135	32.44	300.00	1660	1.19	99.38	1585	5.65	15.27	1660	1.19	46.51
100-30	173	173	0.00	13.27	173	0.00	0.47	173	0.00	72.17	173	0.00	0.96
100-35	241	241	0.00	35.61	241	0.00	0.79	241	0.00	82.84	NaN	NaN	NaN
100-40	299	299	0.00	74.00	299	0.00	0.91	299	0.00	58.97	NaN	NaN	NaN
100-45	367	367	0.00	171.98	367	0.00	1.42	367	0.00	57.35	367	0.00	3.17
102-50	181	181	0.00	300.00	181	0.00	0.60	129	28.73	114.67	161	11.05	0.43
102-60	243	243	0.00	300.00	243	0.00	1.29	188	22.63	52.91	211	13.17	0.69
Average:			9.26	224.84		1.74	18.22		13.01	20.99		12.26	7.85

Table 15: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of OPHS SET3 with 4 trips and 10 extra hotels.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
64-75	1236	858	30.58	300.00	1212	1.94	63.20	1110	10.19	13.95	1164	5.83	23.00
64-80	1284	1170	8.88	300.00	1284	0.00	68.39	1188	7.48	17.91	1278	0.47	27.25
66-125	1670	1405	15.87	300.00	1640	1.80	94.26	1575	5.69	15.41	1640	1.80	46.87
66-130	1680	950	43.45	300.00	1675	0.30	102.35	1620	3.57	15.60	1650	1.79	45.84
100-50	412	330	19.90	300.00	408	0.97	3.42	371	9.95	93.73	408	0.97	4.61
100-60	504	373	25.99	300.00	504	0.00	9.21	412	18.25	79.68	464	7.94	6.32
100-70	590	513	13.05	300.00	578	2.03	24.28	492	16.61	77.37	514	12.88	9.34
100-80	652	489	25.00	300.00	626	3.99	39.20	478	26.69	54.21	594	8.90	13.50
100-90	725	501	30.90	300.00	706	2.62	54.21	550	24.14	69.84	569	21.52	15.74
100-100	782	454	41.94	300.00	760	2.81	71.18	612	21.74	58.72	615	21.36	22.23
100-110	835	470	43.71	300.00	796	4.67	83.82	664	20.48	95.46	723	13.41	29.10
100-120	894	603	32.55	300.00	825	7.72	126.95	717	19.80	37.28	755	15.55	39.41
100-130	956	611	36.09	300.00	918	3.97	146.44	775	18.93	32.64	780	18.41	44.84
100-140	1013	527	47.98	300.00	919	9.28	153.19	777	23.30	50.35	893	11.85	61.59
100-150	1057	632	40.21	300.00	1024	3.12	187.18	914	13.53	55.09	934	11.64	80.33
100-160	1114	502	54.94	300.00	1010	9.34	210.47	870	21.90	88.19	998	10.41	89.65
100-170	1164	484	58.42	300.00	1056	9.28	245.80	956	17.87	71.60	1039	10.74	112.34
100-180	1201	712	40.72	300.00	1096	8.74	286.91	1029	14.32	76.11	1100	8.41	128.89
100-190	1234	683	44.65	300.00	1147	7.05	300.00	1018	17.50	71.42	1115	9.64	134.04
100-200	1261	785	37.75	300.00	1204	4.52	300.00	1122	11.02	91.12	1165	7.61	146.16
100-210	1284	540	57.94	300.00	1237	3.66	300.00	1134	11.68	61.32	1186	7.63	164.56
100-240	1306	564	56.81	300.00	1297	0.69	300.00	1225	6.20	92.05	1275	2.37	214.37
Average:			36.70	300.00		4.02	144.11		15.49	59.96		9.60	66.36

Table 16: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of OPHS SET3 with 5 trips and 12 extra hotels.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
64-75	1236	966	21.84	300.00	1188	3.88	54.12	1056	14.56	8.75	1140	7.77	20.55
64-80	1284	1098	14.49	300.00	1236	3.74	58.13	1170	8.88	25.87	1194	7.01	26.25
66-125	1670	765	54.19	300.00	1620	2.99	88.54	1460	12.57	19.84	1610	3.59	37.80
66-130	1680	1025	38.99	300.00	1660	1.19	94.05	1425	15.18	5.20	1640	2.38	44.99
100-50	412	393	4.61	300.00	393	4.61	2.18	393	4.61	79.87	NaN	NaN	NaN
100-60	504	393	22.02	300.00	504	0.00	14.48	348	30.95	38.27	415	17.66	4.66
100-70	590	437	25.93	300.00	590	0.00	24.54	466	21.02	83.84	402	31.86	6.63
100-80	652	505	22.55	300.00	652	0.00	34.08	517	20.71	94.50	466	28.53	9.75
100-90	725	406	44.00	300.00	709	2.21	49.93	537	25.93	77.51	571	21.24	15.54
100-100	782	584	25.32	300.00	752	3.84	62.45	571	26.98	63.98	644	17.65	20.43
100-110	835	560	32.93	300.00	798	4.43	75.57	632	24.31	42.61	698	16.41	26.30
100-120	894	575	35.68	300.00	827	7.49	89.50	704	21.25	52.89	739	17.34	29.09
100-130	956	349	63.49	300.00	882	7.74	113.23	722	24.48	60.24	803	16.00	40.41
100-140	1013	524	48.27	300.00	939	7.31	136.45	733	27.64	31.46	837	17.37	49.29
100-150	1057	423	59.98	300.00	965	8.70	140.06	861	18.54	37.09	904	14.47	59.38
100-160	1114	331	70.29	300.00	1014	8.98	175.12	851	23.61	47.26	946	15.08	77.86
100-170	1164	540	53.61	300.00	1048	9.97	200.06	876	24.74	37.44	1021	12.29	92.56
100-180	1201	532	55.70	300.00	1111	7.49	239.29	1025	14.65	52.71	1031	14.15	98.37
100-190	1234	771	37.52	300.00	1165	5.59	271.23	988	19.94	38.62	1088	11.83	116.49
100-200	1261	449	64.39	300.00	1183	6.19	300.00	1092	13.40	69.26	1173	6.98	133.86
100-210	1284	277	78.43	300.00	1210	5.76	300.00	1034	19.47	44.02	1205	6.15	154.37
100-240	1306	326	75.04	300.00	1270	2.76	300.00	1148	12.10	64.62	1256	3.83	192.71
Average:			43.15	300.00		4.77	128.32		19.34	48.90		13.79	59.87

Table 17: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of OPHS SET4 with 3 trips and 2 extra hotels.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
100-20	247	247	0.00	195.99	247	0.00	1.13	244	1.21	96.36	247	0.00	1.79
100-25	385	385	0.00	272.58	382	0.78	4.74	337	12.47	53.43	381	1.04	4.74
102-35	157	139	11.46	300.00	151	3.82	0.38	151	3.82	97.09	151	3.82	0.81
102-40	210	145	30.95	300.00	210	0.00	0.69	210	0.00	83.40	210	0.00	1.34
102-45	266	249	6.39	300.00	266	0.00	0.75	265	0.38	74.03	266	0.00	1.96
Average:			9.76	273.72		0.92	1.54		3.58	80.86		0.97	2.13

Table 18: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of OPHS SET4 with 3 trips and 3 extra hotels. Note that the gap is calculated relative to the best feasible solution of the CPLEX method, as the optimal value is unknown.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap*	CPU	Obj.	Gap*	CPU	Obj.	Gap*	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
100-20	N/A	314	0.00	300.00	367	-16.88	2.88	349	-11.15	78.39	350	-11.46	3.62
100-25	N/A	491	0.00	300.00	526	-7.13	9.62	480	2.24	60.50	501	-2.04	9.38
102-35	N/A	114	0.00	300.00	300	-163.16	1.27	324	-184.21	79.56	324	-184.21	2.05
102-40	N/A	169	0.00	300.00	383	-126.63	2.33	385	-127.81	108.88	383	-126.63	2.99
102-45	N/A	186	0.00	300.00	442	-137.63	4.03	429	-130.65	68.48	442	-137.63	4.55
Average:			0.00	300.00		-90.29	4.03		-90.31	79.16		-92.39	4.52

D Results per instance of the TOPHS

Table 19: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of the TOPHS SET 1 with 1 trip per tour (TOP).

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
p1.2.r	280	270	3.57	300.00	275	1.79	0.64	270	3.57	1.54	280	0.00	1.35
p1.3.f	40	40	0.00	197.95	40	0.00	0.05	40	0.00	2.25	40	0.00	0.10
p1.4.h	45	45	0.00	300.00	45	0.00	0.08	45	0.00	2.07	45	0.00	0.13
p2.2.k	275	275	0.00	161.16	275	0.00	0.14	270	1.82	0.74	275	0.00	0.32
p2.3.k	200	200	0.00	20.39	200	0.00	0.15	200	0.00	0.88	200	0.00	0.19
p2.4.k	180	180	0.00	300.00	180	0.00	0.11	180	0.00	0.76	180	0.00	0.15
p3.2.t	800	800	0.00	87.71	800	0.00	0.76	800	0.00	1.80	800	0.00	1.63
p3.3.e	200	200	0.00	300.00	200	0.00	0.15	200	0.00	2.05	200	0.00	0.26
p3.4.g	220	220	0.00	300.00	220	0.00	0.29	220	0.00	2.41	220	0.00	0.32
p4.2.d	531	528	0.56	300.00	483	9.04	7.70	468	11.86	21.95	522	1.69	19.02
p4.3.d	335	330	1.49	300.00	298	11.04	1.82	314	6.27	59.58	332	0.90	4.53
p4.4.d	38	38	0.00	5.59	38	0.00	0.10	38	0.00	63.31	38	0.00	0.18
p5.2.c	50	50	0.00	92.44	50	0.00	0.12	50	0.00	21.33	50	0.00	0.23
p5.3.c	20	20	0.00	4.21	20	0.00	0.08	20	0.00	17.37	20	0.00	0.15
p5.4.d	20	20	0.00	6.57	20	0.00	0.09	20	0.00	23.29	20	0.00	0.17
p6.2.n	1260	1260	0.00	35.32	1230	2.38	10.16	1200	4.76	14.69	1248	0.95	22.26
p6.3.n	1170	1170	0.00	300.00	1134	3.08	6.78	1170	0.00	22.36	1158	1.03	14.97
p6.4.n	1068	1068	0.00	40.76	1068	0.00	5.41	1068	0.00	20.68	1068	0.00	10.70
p7.2.a	30	30	0.00	62.81	30	0.00	0.05	30	0.00	63.13	30	0.00	0.09
p7.3.b	46	46	0.00	300.00	46	0.00	0.09	46	0.00	26.79	46	0.00	0.16
p7.4.b	14	14	0.00	31.62	14	0.00	0.06	14	0.00	56.47	14	0.00	0.11
Average:			0.27	164.12		1.30	1.66		1.35	20.26		0.22	3.67

Table 20: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of the TOPHS SET 1 with 2 trips per tour.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
p1.2.r	280	265	5.36	300.00	280	0.00	0.70	245	12.50	1.93	265	5.36	1.08
p1.3.f	40	40	0.00	8.55	40	0.00	0.12	40	0.00	3.18	40	0.00	0.12
p1.4.h	45	45	0.00	300.00	45	0.00	0.35	45	0.00	2.29	45	0.00	0.18
p2.2.k	275	275	0.00	268.86	275	0.00	0.16	275	0.00	0.72	275	0.00	0.22
p2.3.k	200	200	0.00	6.97	200	0.00	0.26	200	0.00	0.80	200	0.00	0.20
p2.4.k	180	180	0.00	300.00	180	0.00	0.66	180	0.00	1.18	180	0.00	0.18
p3.2.t	800	800	0.00	243.79	800	0.00	1.16	760	5.00	2.24	750	6.25	1.59
p3.3.e	200	200	0.00	300.00	200	0.00	0.44	200	0.00	2.78	200	0.00	0.35
p3.4.g	220	220	0.00	300.00	220	0.00	1.54	220	0.00	4.39	220	0.00	0.38
p4.2.d	531	486	8.47	300.00	523	1.51	6.11	431	18.83	57.42	496	6.59	8.35
p4.3.d	335	330	1.49	300.00	331	1.19	1.94	311	7.16	58.85	331	1.19	2.54
p4.4.d	38	38	0.00	7.17	38	0.00	0.26	38	0.00	64.59	38	0.00	0.35
p5.2.c	50	50	0.00	268.55	50	0.00	0.17	50	0.00	22.93	50	0.00	0.28
p5.3.c	20	20	0.00	1.38	20	0.00	0.13	20	0.00	18.37	20	0.00	0.23
p5.4.d	20	20	0.00	1.42	20	0.00	0.22	20	0.00	29.46	20	0.00	0.32
p6.2.n	1260	1086	13.81	300.00	1242	1.43	15.36	1218	3.33	21.55	1158	8.10	12.96
p6.3.n	1170	1140	2.56	300.00	1170	0.00	13.32	1080	7.69	11.02	1164	0.51	8.52
p6.4.n	1068	1068	0.00	64.95	1068	0.00	7.97	1038	2.81	21.12	1068	0.00	5.63
p7.2.a	30	30	0.00	36.79	30	0.00	0.08	30	0.00	98.05	30	0.00	0.14
p7.3.b	46	32	30.43	300.00	46	0.00	0.15	46	0.00	45.37	46	0.00	0.27
p7.4.b	14	14	0.00	8.15	14	0.00	0.15	14	0.00	50.34	14	0.00	0.22
Average:			2.96	186.50		0.20	2.44		2.73	24.69		1.33	2.10

Table 21: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of the TOPHS SET 1 with 3 trips per tour.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
p1.2.r	280	270	3.57	300.00	280	0.00	5.78	235	16.07	2.84	240	14.29	0.81
p1.3.f	40	40	0.00	108.55	40	0.00	49.93	40	0.00	2.68	40	0.00	0.19
p1.4.h	45	45	0.00	300.00	45	0.00	300.00	45	0.00	3.27	45	0.00	0.29
p2.2.k	275	275	0.00	142.16	265	3.64	0.91	200	27.27	0.79	240	12.73	0.27
p2.3.k	200	200	0.00	300.00	200	0.00	194.21	200	0.00	1.06	200	0.00	0.26
p2.4.k	180	180	0.00	300.00	180	0.00	300.00	160	11.11	1.26	140	22.22	0.20
p3.2.t	800	760	5.00	300.00	800	0.00	14.61	760	5.00	2.78	710	11.25	1.30
p3.3.e	200	200	0.00	300.00	200	0.00	71.13	200	0.00	3.85	200	0.00	0.42
p3.4.g	220	210	4.55	300.00	220	0.00	300.00	220	0.00	3.40	220	0.00	0.56
p4.2.d	531	496	6.59	300.00	519	2.26	4.52	439	17.33	44.34	519	2.26	6.77
p4.3.d	335	252	24.78	300.00	335	0.00	14.04	263	21.49	38.84	289	13.73	1.68
p4.4.d	38	38	0.00	8.58	38	0.00	5.92	38	0.00	65.30	38	0.00	0.51
p5.2.c	50	50	0.00	300.00	50	0.00	1.57	50	0.00	21.72	50	0.00	0.29
p5.3.c	20	20	0.00	7.96	20	0.00	23.54	20	0.00	13.46	20	0.00	0.31
p5.4.d	20	20	0.00	19.18	20	0.00	289.44	20	0.00	12.51	20	0.00	0.46
p6.2.n	1260	1014	19.52	300.00	1242	1.43	60.69	1062	15.71	18.85	1140	9.52	11.26
p6.3.n	1170	804	31.28	300.00	1170	0.00	155.04	1020	12.82	9.57	1044	10.77	6.96
p6.4.n	1068	1068	0.00	300.00	1038	2.81	30.55	1068	0.00	33.88	990	7.30	4.18
p7.2.a	30	30	0.00	41.76	30	0.00	2.17	30	0.00	88.55	30	0.00	0.21
p7.3.b	46	32	30.43	300.00	46	0.00	240.90	46	0.00	29.13	46	0.00	0.38
p7.4.b	14	14	0.00	19.82	14	0.00	25.57	14	0.00	67.52	14	0.00	0.36
Average:			5.99	216.57		0.48	99.55		6.04	22.17		4.96	1.79

Table 22: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of the TOPHS SET 2 with 2 trips per tour and 1 additional hotel.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
p1.2.r	280	265	5.36	300.00	280	0.00	0.82	250	10.71	2.26	280	0.00	1.11
p1.3.f	40	40	0.00	6.27	40	0.00	0.15	40	0.00	1.73	40	0.00	0.11
p1.4.h	45	45	0.00	107.58	45	0.00	0.40	40	11.11	2.74	40	11.11	0.17
p2.2.k	275	275	0.00	300.00	275	0.00	0.20	275	0.00	0.86	240	12.73	0.26
p2.3.k	200	200	0.00	4.42	200	0.00	0.28	200	0.00	1.13	200	0.00	0.18
p2.4.k	180	180	0.00	300.00	180	0.00	0.62	160	11.11	1.24	160	11.11	0.18
p3.2.t	800	760	5.00	300.00	800	0.00	2.06	710	11.25	2.30	800	0.00	1.53
p3.3.e	200	200	0.00	300.00	200	0.00	0.44	200	0.00	2.45	200	0.00	0.32
p3.4.g	220	220	0.00	300.00	220	0.00	1.51	220	0.00	3.52	220	0.00	0.40
p4.2.d	531	529	0.38	300.00	523	1.51	6.17	499	6.03	62.19	500	5.84	8.77
p4.3.d	335	327	2.39	300.00	331	1.19	2.00	328	2.09	55.48	335	0.00	2.44
p4.4.d	38	38	0.00	8.68	38	0.00	0.27	38	0.00	32.28	38	0.00	0.33
p5.2.c	50	50	0.00	300.00	50	0.00	0.17	50	0.00	16.02	50	0.00	0.24
p5.3.c	20	20	0.00	1.81	20	0.00	0.14	20	0.00	19.67	20	0.00	0.19
p5.4.d	20	20	0.00	1.71	20	0.00	0.21	20	0.00	17.68	20	0.00	0.34
p6.2.n	1260	1206	4.29	300.00	1254	0.48	11.67	1176	6.67	16.64	1224	2.86	15.23
p6.3.n	1170	1158	1.03	300.00	1134	3.08	10.18	1074	8.21	19.04	1122	4.10	9.09
p6.4.n	1068	1068	0.00	106.10	1068	0.00	10.82	1062	0.56	21.53	1068	0.00	5.66
p7.2.a	30	30	0.00	33.07	30	0.00	0.08	30	0.00	58.75	30	0.00	0.14
p7.3.b	46	46	0.00	300.00	46	0.00	0.15	46	0.00	55.28	46	0.00	0.24
p7.4.b	14	14	0.00	10.83	14	0.00	0.15	14	0.00	51.54	14	0.00	0.22
Average:			0.88	184.78		0.30	2.31		3.23	21.16		2.27	2.24

Table 23: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of the TOPHS SET 2 with 2 trips per tour and 2 additional hotels.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
p1.2.r	280	260	7.14	300.00	280	0.00	0.97	255	8.93	2.82	245	12.50	1.15
p1.3.f	40	40	0.00	3.53	40	0.00	0.12	40	0.00	3.78	40	0.00	0.11
p1.4.h	45	40	11.11	300.00	45	0.00	0.36	45	0.00	2.35	45	0.00	0.18
p2.2.k	275	275	0.00	281.25	275	0.00	0.22	260	5.45	0.96	240	12.73	0.24
p2.3.k	200	200	0.00	7.01	200	0.00	0.51	200	0.00	1.21	200	0.00	0.18
p2.4.k	180	180	0.00	300.00	180	0.00	0.81	180	0.00	0.74	180	0.00	0.18
p3.2.t	800	750	6.25	300.00	800	0.00	1.32	760	5.00	2.37	760	5.00	1.62
p3.3.e	200	200	0.00	300.00	200	0.00	0.44	200	0.00	3.23	200	0.00	0.37
p3.4.g	220	220	0.00	300.00	220	0.00	2.33	220	0.00	2.09	220	0.00	0.39
p4.2.d	531	529	0.38	300.00	523	1.51	6.23	484	8.85	68.54	521	1.88	8.51
p4.3.d	335	335	0.00	300.00	331	1.19	2.03	323	3.58	60.75	331	1.19	2.36
p4.4.d	38	38	0.00	6.17	38	0.00	0.28	38	0.00	65.18	38	0.00	0.32
p5.2.c	50	50	0.00	300.00	50	0.00	0.17	50	0.00	16.91	45	10.00	0.25
p5.3.c	20	20	0.00	6.57	20	0.00	0.14	20	0.00	16.88	20	0.00	0.24
p5.4.d	20	20	0.00	5.88	20	0.00	0.22	20	0.00	14.50	20	0.00	0.32
p6.2.n	1260	1242	1.43	300.00	1206	4.29	11.55	1188	5.71	17.21	1182	6.19	12.38
p6.3.n	1170	1110	5.13	300.00	1116	4.62	13.02	978	16.41	14.47	1098	6.15	8.84
p6.4.n	1068	1068	0.00	65.40	1068	0.00	11.64	996	6.74	16.34	1026	3.93	6.33
p7.2.a	30	30	0.00	29.93	30	0.00	0.09	30	0.00	99.61	30	0.00	0.14
p7.3.b	46	46	0.00	300.00	46	0.00	0.17	46	0.00	91.21	46	0.00	0.25
p7.4.b	14	14	0.00	23.92	14	0.00	0.14	14	0.00	34.37	14	0.00	0.24
Average:			1.50	191.89		0.55	2.51		2.89	25.50		2.84	2.12

Table 24: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of the TOPHS SET 2 with 2 trips per tour and 3 additional hotels.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
p1.2.r	280	265	5.36	300.00	280	0.00	1.06	235	16.07	2.30	270	3.57	1.20
p1.3.f	40	40	0.00	10.04	40	0.00	0.12	40	0.00	3.44	40	0.00	0.12
p1.4.h	45	45	0.00	300.00	45	0.00	0.35	45	0.00	1.53	45	0.00	0.17
p2.2.k	275	270	1.82	300.00	275	0.00	0.27	260	5.45	0.83	260	5.45	0.23
p2.3.k	200	200	0.00	6.18	200	0.00	0.71	200	0.00	0.73	200	0.00	0.21
p2.4.k	180	180	0.00	300.00	180	0.00	1.24	180	0.00	1.14	160	11.11	0.18
p3.2.t	800	730	8.75	300.00	800	0.00	1.92	760	5.00	2.74	750	6.25	1.80
p3.3.e	200	200	0.00	300.00	200	0.00	0.60	200	0.00	3.27	200	0.00	0.31
p3.4.g	220	210	4.55	300.00	220	0.00	2.32	220	0.00	2.31	210	4.55	0.38
p4.2.d	531	520	2.07	300.00	524	1.32	8.19	437	17.70	97.14	457	13.94	7.87
p4.3.d	335	326	2.69	300.00	331	1.19	2.02	328	2.09	75.12	295	11.94	2.45
p4.4.d	38	38	0.00	9.86	38	0.00	0.29	38	0.00	81.40	38	0.00	0.34
p5.2.c	50	50	0.00	200.22	50	0.00	0.17	50	0.00	21.54	45	10.00	0.21
p5.3.c	20	20	0.00	1.39	20	0.00	0.14	20	0.00	15.68	20	0.00	0.20
p5.4.d	20	20	0.00	1.45	20	0.00	0.22	20	0.00	26.78	20	0.00	0.31
p6.2.n	1260	1248	0.95	300.00	1212	3.81	14.17	1200	4.76	20.73	1218	3.33	13.79
p6.3.n	1170	1074	8.21	300.00	1140	2.56	18.05	1104	5.64	20.78	1092	6.67	9.67
p6.4.n	1068	1068	0.00	211.62	1068	0.00	17.29	930	12.92	19.10	1068	0.00	6.62
p7.2.a	30	30	0.00	36.48	30	0.00	0.09	30	0.00	31.50	30	0.00	0.15
p7.3.b	46	46	0.00	300.00	46	0.00	0.15	46	0.00	96.81	46	0.00	0.26
p7.4.b	14	14	0.00	8.44	14	0.00	0.15	14	0.00	84.95	14	0.00	0.22
Average:			1.64	194.56		0.42	3.31		3.32	29.04		3.66	2.22

Table 25: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of the TOPHS SET 3 with 2 trips per tour and 1 additional hotel. Note that gaps are calculated as the relative difference to the best known feasible solution found by the CPLEX method.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
p1.2.r	N/A	200	0.00	300.00	210	-5.00	1.06	200	0.00	1.95	215	-7.50	0.77
p1.3.f	N/A	15	0.00	0.30	15	0.00	0.04	15	0.00	2.09	15	0.00	0.07
p1.4.h	N/A	15	0.00	0.36	15	0.00	0.06	15	0.00	1.93	15	0.00	0.07
p2.2.k	N/A	165	0.00	73.67	175	-6.06	0.23	165	0.00	0.64	165	0.00	0.14
p2.3.k	N/A	100	0.00	3.33	100	0.00	0.12	100	0.00	0.88	100	0.00	0.11
p2.4.k	N/A	75	0.00	2.17	85	-13.33	0.10	75	0.00	0.88	75	0.00	0.09
p3.2.t	N/A	600	0.00	300.00	650	-8.33	1.06	650	-8.33	2.22	610	-1.67	1.22
p3.3.e	N/A	110	0.00	13.69	110	0.00	0.10	110	0.00	1.45	110	0.00	0.12
p3.4.g	N/A	90	0.00	3.36	90	0.00	0.11	90	0.00	2.87	90	0.00	0.16
p4.2.d	N/A	382	0.00	300.00	435	-13.87	3.98	416	-8.90	95.11	435	-13.87	5.47
p4.3.d	N/A	infeasible[1]											
p4.4.d	N/A	infeasible[1]											
p5.2.c	N/A	20	0.00	0.51	20	0.00	0.09	20	0.00	8.35	20	0.00	0.19
p5.3.c	N/A	15	0.00	0.74	15	0.00	0.10	15	0.00	15.39	15	0.00	0.17
p5.4.d	N/A	20	0.00	0.96	20	0.00	0.15	20	0.00	14.19	20	0.00	0.25
p6.2.n	N/A	1056	0.00	300.00	1122	-6.25	8.20	1110	-5.11	14.53	1134	-7.39	11.35
p6.3.n	N/A	912	0.00	300.00	984	-7.89	9.48	948	-3.95	17.44	972	-6.58	7.04
p6.4.n	N/A	174	0.00	300.00	174	0.00	0.35	174	0.00	11.25	174	0.00	0.58
p7.2.a	N/A	0	0.00	0.82	0	0.00	0.04	0	0.00	59.81	0	0.00	0.04
p7.3.b	N/A	0	0.00	1.13	0	0.00	0.07	0	0.00	55.20	0	0.00	0.07
p7.4.b	N/A	0	0.00	1.22	0	0.00	0.11	0	0.00	72.40	0	0.00	0.09
Average:			0.00	100.12		-3.20	1.34		-1.38	19.93		-1.95	1.47

[1] Due to removal of POIs, solutions may become infeasible. Computation times are not included in the average.

Table 26: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of the TOPHS SET 3 with 2 trips per tour and 2 additional hotels. Note that gaps are calculated as the relative difference to the best known feasible solution found by the CPLEX method.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
p1.2.r	N/A	200	0.00	300.00	215	-7.50	0.95	215	-7.50	1.75	215	-7.50	0.79
p1.3.f	N/A	15	0.00	0.30	15	0.00	0.04	15	0.00	2.27	15	0.00	0.06
p1.4.h	N/A	15	0.00	0.50	15	0.00	0.06	15	0.00	1.90	15	0.00	0.08
p2.2.k	N/A	160	0.00	52.94	170	-6.25	0.21	160	0.00	0.57	160	0.00	0.14
p2.3.k	N/A	85	0.00	1.62	85	0.00	0.17	85	0.00	0.59	85	0.00	0.08
p2.4.k	N/A	70	0.00	0.12	70	0.00	0.23	70	0.00	0.85	70	0.00	0.06
p3.2.t	N/A	560	0.00	300.00	580	-3.57	0.89	570	-1.79	1.01	570	-1.79	0.96
p3.3.e	N/A	110	0.00	1.78	110	0.00	0.23	110	0.00	1.61	100	9.09	0.12
p3.4.g	N/A	70	0.00	1.66	70	0.00	0.18	70	0.00	2.19	70	0.00	0.10
p4.2.d	N/A	277	0.00	300.00	394	-42.24	3.28	337	-21.66	69.90	350	-26.35	4.27
p4.3.d	N/A	139	0.00	300.00	150	-7.91	0.71	150	-7.91	98.28	150	-7.91	0.88
p4.4.d	N/A	infeasible[1]											
p5.2.c	N/A	20	0.00	0.71	20	0.00	0.10	20	0.00	19.59	20	0.00	0.18
p5.3.c	N/A	15	0.00	0.96	15	0.00	0.10	15	0.00	5.55	15	0.00	0.18
p5.4.d	N/A	20	0.00	1.07	20	0.00	0.14	20	0.00	9.06	20	0.00	0.24
p6.2.n	N/A	768	0.00	300.00	1122	-46.09	7.70	1098	-42.97	15.66	1086	-41.41	10.93
p6.3.n	N/A	870	0.00	300.00	978	-12.41	8.52	960	-10.34	16.75	984	-13.10	6.75
p6.4.n	N/A	558	0.00	300.00	666	-19.35	2.45	660	-18.28	21.15	654	-17.20	2.38
p7.2.a	N/A	0	0.00	1.20	0	0.00	0.04	0	0.00	82.69	0	0.00	0.04
p7.3.b	N/A	0	0.00	1.60	0	0.00	0.07	0	0.00	45.76	0	0.00	0.07
p7.4.b	N/A	0	0.00	1.83	0	0.00	0.10	0	0.00	62.58	0	0.00	0.10
Average:			0.00	108.31		-7.27	1.31		-5.52	22.99		-5.31	1.42

[1] Due to removal of POIs, solutions may become infeasible. Computation times are not included in the average.

Table 27: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of the TOPHS SET 3 with 2 trips per tour and 3 additional hotels. Note that gaps are calculated as the relative difference to the best known feasible solution found by the CPLEX method.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
p1.2.r	N/A	195	0.00	300.00	220	-12.82	0.64	200	-2.56	0.98	215	-10.26	0.66
p1.3.f	N/A	20	0.00	0.55	20	0.00	0.05	20	0.00	2.09	20	0.00	0.08
p1.4.h	N/A	15	0.00	0.77	15	0.00	0.12	15	0.00	1.18	15	0.00	0.07
p2.2.k	N/A	170	0.00	64.30	170	0.00	0.09	170	0.00	0.54	170	0.00	0.11
p2.3.k	N/A	135	0.00	0.79	135	0.00	0.17	135	0.00	0.57	135	0.00	0.08
p2.4.k	N/A	30	0.00	0.33	30	0.00	0.17	30	0.00	0.64	30	0.00	0.05
p3.2.t	N/A	540	0.00	300.00	580	-7.41	1.06	560	-3.70	1.02	580	-7.41	0.94
p3.3.e	N/A	50	0.00	4.15	50	0.00	0.10	50	0.00	2.18	50	0.00	0.08
p3.4.g	N/A	120	0.00	0.92	120	0.00	0.15	120	0.00	1.91	120	0.00	0.18
p4.2.d	N/A	289	0.00	300.00	333	-15.22	2.80	323	-11.76	87.01	317	-9.69	3.95
p4.3.d	N/A	infeasible[1]											
p4.4.d	N/A	infeasible[1]											
p5.2.c	N/A	15	0.00	0.82	15	0.00	0.08	15	0.00	17.75	15	0.00	0.13
p5.3.c	N/A	15	0.00	0.90	15	0.00	0.10	15	0.00	11.74	15	0.00	0.20
p5.4.d	N/A	10	0.00	1.07	10	0.00	0.18	10	0.00	11.90	10	0.00	0.16
p6.2.n	N/A	822	0.00	300.00	1134	-37.96	7.93	1080	-31.39	14.17	1122	-36.50	9.91
p6.3.n	N/A	840	0.00	300.00	984	-17.14	8.01	906	-7.86	10.91	954	-13.57	6.30
p6.4.n	N/A	648	0.00	300.00	720	-11.11	8.79	690	-6.48	11.70	720	-11.11	3.21
p7.2.a	N/A	0	0.00	1.25	0	0.00	0.04	0	0.00	80.93	0	0.00	0.04
p7.3.b	N/A	0	0.00	1.57	0	0.00	0.08	0	0.00	81.02	0	0.00	0.07
p7.4.b	N/A	0	0.00	1.78	0	0.00	0.10	0	0.00	68.19	0	0.00	0.09
Average:			0.00	98.91		-5.35	1.61		-3.36	21.39		-4.66	1.38

[1] Due to removal of POIs, solutions may become infeasible. Computation times are not included in the average.

Table 28: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of the TOPHS SET 4 with 2 trips per tour and 1 additional hotel. Note that gaps are calculated as the relative difference to the best known feasible solution found by the CPLEX method.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
p1.3.r	N/A	170	0.00	300.00	180	-5.88	0.79	175	-2.94	1.83	170	0.00	0.41
p1.4.r	N/A	105	0.00	300.00	125	-19.05	0.97	110	-4.76	2.30	105	0.00	0.26
p3.3.t	N/A	420	0.00	300.00	490	-16.67	1.37	480	-14.29	1.96	480	-14.29	0.62
p3.4.t	N/A	360	0.00	300.00	440	-22.22	3.43	430	-19.44	2.10	430	-19.44	0.52
p4.2.t	N/A	616	0.00	300.00	1242	-101.62	71.28	1155	-87.50	78.75	1198	-94.48	107.67
p4.3.t	N/A	337	0.00	300.00	1124	-233.53	111.94	1069	-217.21	50.84	1089	-223.15	54.50
p4.4.t	N/A	656	0.00	300.00	1054	-60.67	271.35	1008	-53.66	51.05	1060	-61.59	44.76
p5.2.z	N/A	1010	0.00	300.00	1575	-55.94	9.27	1505	-49.01	20.03	1590	-57.43	16.25
p5.3.z	N/A	390	0.00	300.00	1425	-265.38	14.46	1315	-237.18	15.69	1400	-258.97	10.10
p5.4.z	N/A	650	0.00	300.00	1220	-87.69	32.72	1125	-73.08	13.16	1130	-73.85	6.60
p7.2.t	N/A	177	0.00	300.00	1075	-507.34	38.77	1020	-476.27	82.06	1078	-509.04	65.30
p7.3.t	N/A	247	0.00	300.00	925	-274.49	44.51	901	-264.78	78.08	879	-255.87	35.42
p7.4.t	N/A	243	0.00	300.00	746	-207.00	83.76	773	-218.11	95.23	731	-200.82	13.52
Average:			0.00	300.00		-142.88	52.66		-132.17	37.93		-136.07	27.38

Table 29: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of the TOPHS SET 4 with 2 trips per tour and 2 additional hotels. Note that gaps are calculated as the relative difference to the best known feasible solution found by the CPLEX method.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
p1.3.r	N/A	125	0.00	300.00	160	-28.00	0.78	155	-24.00	2.33	145	-16.00	0.39
p1.4.r	N/A	110	0.00	300.00	120	-9.09	1.50	120	-9.09	2.33	110	0.00	0.25
p3.3.t	N/A	460	0.00	300.00	490	-6.52	2.32	490	-6.52	2.23	490	-6.52	0.65
p3.4.t	N/A	390	0.00	300.00	460	-17.95	3.63	460	-17.95	2.18	460	-17.95	0.54
p4.2.t	N/A	425	0.00	300.00	1304	-206.82	66.78	1192	-180.47	70.79	1226	-188.47	100.58
p4.3.t	N/A	369	0.00	300.00	1187	-221.68	120.65	1110	-200.81	75.54	1186	-221.41	64.59
p4.4.t	N/A	375	0.00	300.00	1008	-168.80	300.00	929	-147.73	59.52	992	-164.53	38.33
p5.2.z	N/A	975	0.00	300.00	1515	-55.38	15.20	1445	-48.21	14.41	1520	-55.90	15.05
p5.3.z	N/A	580	0.00	300.00	1420	-144.83	20.67	1330	-129.31	14.60	1385	-138.79	9.25
p5.4.z	N/A	345	0.00	300.00	1210	-250.72	47.46	1135	-228.99	12.01	1130	-227.54	5.43
p7.2.t	N/A	245	0.00	300.00	1048	-327.76	27.22	996	-306.53	78.89	1089	-344.49	62.73
p7.3.t	N/A	319	0.00	300.00	936	-193.42	62.97	877	-174.92	71.00	963	-201.88	37.88
p7.4.t	N/A	180	0.00	300.00	915	-408.33	144.64	771	-328.33	42.22	796	-342.22	19.86
Average:			0.00	300.00		-156.87	62.60		-138.68	34.47		-148.13	27.35

Table 30: Results of the solving methods CPLEX, TSVNS, TABC and TAV for each instance of the TOPHS SET 4 with 2 trips per tour and 3 additional hotel. Note that gaps are calculated as the relative difference to the best known feasible solution found by the CPLEX method.

		CPLEX			TSVNS			TABC			TAV		
		Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU	Obj.	Gap	CPU
Inst.	Opt.		(%)	(s)		(%)	(s)		(%)	(s)		(%)	(s)
p1.3.r	N/A	150	0.00	300.00	160	-6.67	0.81	160	-6.67	1.36	160	-6.67	0.33
p1.4.r	N/A	125	0.00	300.00	120	4.00	2.06	125	0.00	1.88	110	12.00	0.24
p3.3.t	N/A	530	0.00	300.00	530	0.00	2.42	530	0.00	2.46	500	5.66	0.59
p3.4.t	N/A	400	0.00	300.00	430	-7.50	6.80	410	-2.50	1.78	410	-2.50	0.45
p4.2.t	N/A	580	0.00	300.00	1254	-116.21	65.96	1212	-108.97	55.63	1261	-117.41	88.26
p4.3.t	N/A	449	0.00	300.00	1169	-160.36	168.72	1059	-135.86	39.45	1058	-135.63	58.28
p4.4.t	N/A	455	0.00	300.00	1139	-150.33	300.00	1074	-136.04	42.24	1009	-121.76	38.35
p5.2.z	N/A	780	0.00	300.00	1580	-102.56	12.27	1485	-90.38	14.38	1545	-98.08	13.67
p5.3.z	N/A	560	0.00	300.00	1410	-151.79	22.76	1365	-143.75	11.67	1350	-141.07	8.04
p5.4.z	N/A	210	0.00	300.00	1245	-492.86	67.90	1180	-461.90	20.03	1155	-450.00	5.28
p7.2.t	N/A	402	0.00	300.00	1119	-178.36	32.56	996	-147.76	34.60	1075	-167.41	55.50
p7.3.t	N/A	346	0.00	300.00	925	-167.34	84.90	910	-163.01	38.00	912	-163.58	30.48
p7.4.t	N/A	187	0.00	300.00	896	-379.14	201.61	777	-315.51	67.45	825	-341.18	21.75
Average:			0.00	300.00		-146.85	74.52		-131.72	25.46		-132.89	24.71