Sohrabi, Somayeh; Ziarati, Koorush; Keshtkaran, Morteza

Article

ACS-OPHS: Ant Colony System for the Orienteering Problem with hotel selection

EURO Journal on Transportation and Logistics (EJTL)

Provided in Cooperation with:
Association of European Operational Research Societies (EURO), Fribourg

Suggested Citation: Sohrabi, Somayeh; Ziarati, Koorush; Keshtkaran, Morteza (2021) : ACS-OPHS: Ant Colony System for the Orienteering Problem with hotel selection, EURO Journal on Transportation and Logistics (EJTL), ISSN 2192-4384, Elsevier, Amsterdam, Vol. 10, Iss. 1, pp. 1-10, https://doi.org/10.1016/j.ejtl.2021.100036

EURO Journal on Transportation and Logistics 10 (2021) 100036

Contents lists available at ScienceDirect

EURO Journal on Transportation and Logistics

journal homepage: www.journals.elsevier.com/euro-journal-on-transportation-and-logistics

ACS-OPHS: Ant Colony System for the Orienteering Problem with hotel selection

Somayeh Sohrabi, Koorush Ziarati *, Morteza Keshtkaran

School of Electrical and Computer Engineering, Shiraz University, Shiraz, Iran

ARTICLE INFO

Keywords:
Routing problems
Orienteering problem
Orienteering problem with hotel selection
Intermediate facilities
Ant colony system

ABSTRACT

In this paper, an algorithm, called ACS-OPHS, is proposed to tackle the Orienteering Problem with Hotel Selection (OPHS). This algorithm is strongly based on the Ant Colony System (ACS); however, it differs from the ACS in the way the paths are constructed, in tuning a parameter of the transition rule and in the pheromone trails updating rules. The ACS-OPHS uses a bi-directional search strategy and employs a novel and fast approach to identify all feasible intermediate hotels in an offline manner. Moreover, in the ACS-OPHS, the relative importance of exploitation versus exploration is determined according to the progress of the algorithm in approaching to the global optima. The ACS-OPHS is a simple and well-performing approach to solve the OPHS. Concerning the standard benchmark instances, it outperforms the state-of-the-art algorithms in several instances and produces competitive solutions in reasonable time. This algorithm also improves the best known results of four instances with unknown optimal solutions.

1. Introduction

The Orienteering Problem (OP), which has attracted the attention of many researchers, is a combinatorial optimization problem (Tsiligirides, 1984). The Orienteering Problem with Hotel Selection (OPHS) is a variant of the OP (Divsalar et al., 2013). The OPHS is defined on a complete graph $G = (V, E)$ where V is the set of vertices and E denotes the set of edges. V has two subsets including $HOTELS = \{H_0, H_1, \ldots, H_{h-1}\}$ and $NODES = \{1, \ldots, n\}$. The score of each vertex in *HOTELS* is zero, and there is no restriction on the number of times that each of these vertices can be visited. On the other hand, each vertex in *NODES* has a specific score, s_i ($i \in NODES$), and must be visited at most once.

The goal of the OPHS is to find a tour in graph G with a maximum score. A tour is a continuous path that must be started from H_0 and finished in H_1. It must also contain a particular number of trips, D. Each trip has its own time restriction, T_j ($j \in \{1, \ldots, D\}$), that limits the subset of *NODES* that can be visited along the trip. The time limit of each trip must be satisfied regarding the Euclidean distance between the vertices. The origin and destination of each trip must be chosen from *HOTELS* and the destination of trip $i (1 \leq i < D - 1)$ is the origin of trip $i + 1$. Moreover, according to the definition of a feasible tour, the start-hotel of the first trip and the end-hotel of the last trip must be H_0 and H_1, respectively.

Henceforth the "*intermediate hotels*" term refers to the end-point of trip 1, the start-point of trip D, and the origins and destinations of the other trips. The word "*node*" refers only to a member of the *NODES* set while "*vertex*" is used to refer to a hotel or a "*node*".

There are several practical applications for the OPHS (Divsalar et al., 2013) (Divsalar et al., 2014). Planning a multi-day tour for a tourist who wants to visit a region during a specific number of days is one of them. Assume a traveling salesman who has a maximum number of working hours per day and wants to serve some requests sent from a specific area. Since he spends a limited number of days in that area, requests must be selected according to their associated profits. Planning a tour for this salesman is another application of the OPHS.

In this paper, a new algorithm, called ACS-OPHS, is proposed for solving the OPHS. This algorithm is based on the standard Ant Colony System (ACS) (Dorigo and Gambardella, 1997); however, it differs from the ACS in some aspects including:

- The path construction procedure: similar to the standard ACS proposed for the Traveling Salesman Problem (TSP) (Dorigo and Gambardella, 1997), feasible tours are built sequentially in the construction phase of the ACS-OPHS. However, in contrast to the standard ACS, two ant colonies, which are assumed to be located at the origin of the first trip and the destination of the last trip, are used. In other words, the ACS-OPHS uses a bi-directional search strategy.
- The method used to tune parameter q_0 of the transition rule: in contrast to the standard ACS, the value of q_0 is modified over the

* Corresponding author.
E-mail addresses: s.sohrabi@shirazu.ac.ir (S. Sohrabi), ziarati@shirazu.ac.ir (K. Ziarati), mkeshtkaran@cse.shirazu.ac.ir (M. Keshtkaran).

https://doi.org/10.1016/j.ejtl.2021.100036
Received 5 September 2020; Received in revised form 2 March 2021; Accepted 8 April 2021

execution time of the ACS-OPHS to have a good compromise between exploration and exploitation.
- Pheromone trail updating rules: since tours are constructed in two different directions, a bi-directional strategy is employed to update pheromone trails.

Moreover, a simple and fast offline approach is introduced for identifying the feasible intermediate hotels. Using this approach, a new point of view on the OPHS tour construction is provided. The previous methods suggested for the OPHS, construct a feasible tour as follows. Initially, a feasible sequence of hotels is determined and then, nodes are inserted in this sequence with respect to the time limitations of the trips. However, in the ACS-OPHS, a feasible tour is easily constructed using a sequential method. It means that trips can be constructed one after the other and the hotel selection occurs when the time limit of a trip prevents the other nodes from being visited.

The remainder of this paper is organized as follows: the related works are reviewed in Section 2, and the ACS-OPHS is introduced in Section 3. Section 4 is devoted to the experimental results. The conclusion and future work are presented in Section 5.

2. Related works

So far, four algorithms have been suggested for the OPHS. Divsalar et al. (2013) proposed a method based on the skewed variable neighborhood search, called SVNS. The SVNS includes three phases: initialization, improvement, and re-centering. In the initialization phase, three steps are taken to construct an initial solution. First, considering each feasible pair of hotels as the start- and end-point of each trip, an independent OP instance is solved with respect to the time limit of the trip. In this way, the potential score that can be obtained within each pair of hotels is determined. Afterwards, all feasible sequences of hotels are produced and a number of them are selected to construct the initial solution and also to use them in the improvement phase. For this purpose, the sequences with the highest potential scores are preferred. In the improvement phase, the vertices and hotels are shaken and the solution is improved using a local search procedure. The focus of the local search procedure, that contains nine operators, is to improve the order of the visited nodes along a tour. Finally, in the re-centering phase, the solution that is used to start the next iteration of the algorithm is selected.

In addition to the proposed SVNS, Divsalar et al. (2013) have introduced 229 problem instances for the OPHS. These instances have been designed using the standard OP instances with varying sizes. Considering these instances, the SVNS produces high quality solutions and have an acceptable computation time.

In 2014, Divsalar et al. (2014) proposed a memetic algorithm for the OPHS, called MA. Since the genetic algorithm operators are only used to improve the sequence of hotels, each chromosome presents a sequence of hotels in a tour. The fitness of the chromosome is equal to the score obtained along the tour. The MA has two parts: initialization and main loop. The same as the SVNS, in the initialization part, the potential score between each pair of hotels is determined and then the initial population is created. Two crossover and one mutation operators are applied in the main loop to create new solutions from the current population. Each of these new solutions, which is called offspring, is improved using a local search procedure. Then, concerning the current population and the offspring, the next population is selected. It should be mentioned that, in the MA, the sequence of hotels is improved using the crossover and mutation operators while the local search procedure improves the order of the nodes placed among the hotels.

Since the instances introduced in (Divsalar et al., 2013) were small in terms of the Total Number of Feasible Sequences of hotels (TNFS), 176 new and larger instances have been designed by Divsalar et al. (2014). The experiments indicate that the TNFS is an important property of the OPHS instances. The difficulty of an instance has a direct relationship with its TNFS value. Using the new instances, the main drawback of the SVNS has been disclosed. Although this algorithm provides even better solutions than the MA for small instances, considering the new large instances, the computation time of the SVNS is not acceptable at all. The bottleneck of the SVNS is at the initialization phase when it finds all feasible sequences of hotels. In contrast to the SVNS, the MA has a reasonable computation time even for instances with large TNFS values.

In 2019, Toledo et al. (2020) proposed a hyper-heuristic to tackle the OPHS. Solving the OPs, the same as the SVNS and MA, the potential scores between each pair of hotels are determined. Then, the hyper-heuristic creates the initial population and selects an individual from the population using tournament selection. The selected solution is shaken employing four feasible heuristics. At the next step, a local search procedure containing eleven low-level heuristics is used to improve the tour. Finally, a member of the population is replaced by the newly generated solution. The algorithm is terminated when a stopping criteria has been met. In fact, the proposed hyper-heuristic is based on the large neighborhood search. This algorithm produces competitive solutions for the standard OPHS instances in reasonable time. It should be mentioned that ten instances have been ignored by Toledo et al. (2020) during their experiments.

Sohrabi et al. (2019) proposed an algorithm based on the greedy randomized adaptive search procedure, called GRASP. As opposed to the previous algorithms, GRASP does not determine the potential score between each pair of hotels. Instead, a new approach based on dynamic programming have been introduced for the hotel selection. GRASP has two phases: construction and local search. In the construction phase, a feasible solution is created. For this purpose, using the dynamic programming idea, a feasible sequence of hotels is created. Then, two procedures are used to insert nodes among the hotels. When there is no node left that can be added, the local search phase is started to improve the solution. GRASP also uses the proximate optimality principle (Fleurent and Glover, 1999). Thus, each time a node is added to the solution, a procedure including four operators is used to improve the tour: two operators to improve the order of the nodes, one operator based on the dynamic programming idea to make better the sequence of hotels, and another operator to improve the sequence of hotels as well as the selection of the nodes.

Using the standard instances for the Traveling Salesman Problem with Hotel Selection (TSPHS), Sohrabi et al. (2019) have also created new 76 benchmark instances. These instances differ from the previous ones in the way they have been created and in terms of the number of nodes. The results of the experiments indicate that GRASP outperforms the MA especially when the number of trips is less than six.

The Ant Colony Optimization (ACO), which takes inspiration from the foraging behavior of ants, is a swarm intelligence algorithm. So far, several ACO methods have been applied successfully to the OP (Liang et al., 2002) (Ke and Feng, 2008) and its variants, including the Team Orienteering Problem (TOP) (Ke et al., 2008), multi-objective version of the OP (Schilde et al., 2009), Team Orienteering Problem with Time Windows (TOPTW) (Gambardella et al., 2012), Time-Dependent Orienteering Problem (TD-OP) (Verbeeck et al., 2014), Time-Dependent Orienteering Problem with Time windows (TD-OPTW) (Verbeeck et al., 2017), and Multi-Objective Time-Dependent Orienteering Problem (MOTDOP) (Mei et al., 2016). In most of these algorithms, solutions are created sequentially, and each ant successively chooses nodes to be added to the path following a unidirectional search strategy. For more information about the variants of the OP and the other types of solution methods applied for solving them, we refer the interested reader to (Gunawan et al., 2016) and (Vansteenwegen and Gunawan, 2019).

From another point of view, the OPHS is a variant of routing problems with intermediate facilities. We refer the interested reader to (Schiffer et al., 2019) in which this type of routing problems has been described and categorized.

Since, according to the literature, the ACO has a good background in solving the OPs with respect to their graph based nature, in this paper, the ACO has been successfully applied to tackle the OPHS as an example

of the routing problem with intermediate facilities.

3. Proposed algorithm

The iterative procedure of the ACS-OPHS is started by initializing the parameters of the algorithm. In order to construct feasible tours, the ACS-OPHS uses two colonies of ants, whose nests are located in H_0 and H_1 (Section 3.1). Each ant changes pheromone levels of edges within its tour locally (Section 3.3). To improve the solution created by each ant, a local search procedure is employed (Section 3.4). At the end of each iteration, the global best solution is updated if the best constructed solution is better than that. Afterwards, pheromone trails are updated globally to amplify exploitation (Section 3.3). Then, the next iteration of the algorithm is started. The ACS-OPHS is iterated until a specific stopping criteria has been met. The pseudocode of the ACS-OPHS is shown in Fig. 1. In the following sub-sections, each step of the ACS-OPHS is described in more detail. We have used $T_{x,y}$ to show the Euclidean distance between vertex x and vertex y. T_d and s_j are the time limit of trip d and the score of node j, respectively. Moreover, as mentioned before, *HOTELS* is the set of all available hotels.

3.1. Construction phase

In the ACS-OPHS, two ant colonies, called C_0 and C_1, with a similar individuals number, m, are used. Nests of C_0 and C_1 are located in H_0 and H_1, respectively. Each ant of colony C_0 starts building a feasible tour from the first trip while the tour construction procedure is begun from the last trip by the ants of colony C_1.

Assume that an ant from colony C_0, after walking along a partial tour, is now in trip d and the last visited hotel and vertex are H_s and i, respectively. The set $FV(i)$ of feasible vertices that can be visited after i is defined as follows:

$$FV(i) = \{\, j \in UNVISITED \mid \exists H_k \in FEH[d][s] \;:\; L + T_{i,j} + T_{j,H_k} \le T_d \,\} \tag{1}$$

In this definition, *UNVISITED* is the set of not yet visited nodes within the current partial tour and $FEH[d][s]$ is the set of feasible end-hotels for trip d with H_s as the origin of the trip. Additionally, L denotes the current length of trip d. If there is no feasible node that can be visited after vertex i, $FV(i)$ is defined with respect to the feasible end-hotels that can be visited after i:

$$FV(i) = \{H_k \in FEH[d][s] \mid L + T_{i,H_k} \le T_d\} \tag{2}$$

Our proposed method to construct FEH is discussed in Section 3.2. This method follows an offline fashion, which means that $FEH[d][s]$ is determined without concerning vertex i and the other vertices visited before i.

The same definitions are valid for an ant from colony C_1 that after traversing a reversed partial tour from H_1 is now in trip d and the last visited hotel and vertex are H_e and i, respectively.

Initializing the parameters of the ACS-OPHS: α, β, q_0, ρ, τ_0, m, and the stopping criteria
while the stopping criteria has not been met
    Consider two ant colonies whose nests are in H_0 and H_1: C_0 and C_1
    for each ant of the colonies
        Move the ant from its nest and construct a feasible tour
        Update pheromone trails locally
        Apply the local search procedure to improve the constructed solution
    Change the global best solution if a better solution has been found
    Update pheromone trails globally

Fig. 1. The general structure of the ACS-OPHS.

$$FV(i) = \{\, j \in UNVISITED \mid \exists H_k \in FSH[d][e] \;:\; L + T_{j,i} + T_{H_k,j} \le T_d \,\} \tag{3}$$

or

$$FV(i) = \{H_k \in FSH[d][e] \mid L + T_{H_k,i} \le T_d\} \tag{4}$$

$FSH[d][e]$ represents the set of feasible start-hotels of trip d when the destination of this trip is H_e. Our proposed method to construct the *FSH* is also described in Section 3.2.

Recall that i is the last visited vertex. Each ant selects the next vertex of its partial tour by applying the following transition rule:

$$j = \begin{cases} \underset{k \in FV(i)}{\operatorname{argmax}} \{\tau_{ik}^{\alpha} * \eta_{ik}^{\beta}\} & q \le q_0 \quad (exploitation) \\ J \quad otherwise \quad (exploration) \end{cases} \tag{5}$$

Here, $q_0 \in [0,1]$ is a parameter that determines the importance of exploitation versus exploration, q is a uniform random number over $[0,1]$, and J is a vertex that is chosen randomly using the following probabilistic distribution:

$$p_{ik} = \begin{cases} \dfrac{\tau_{ik}^{\alpha} * \eta_{ik}^{\beta}}{\sum_{k' \in FV(i)} \tau_{ik'}^{\alpha} * \eta_{ik'}^{\beta}} & k \in FV(i) \\ 0 \quad otherwise \end{cases} \tag{6}$$

In other words, with probability q_0, the next vertex is the member of $FV(i)$ with the highest value of $\tau_{ik}^{\alpha} * \eta_{ik}^{\beta}$; otherwise, a roulette wheel procedure is used to select a vertex from $FV(i)$. In this case, the probability of choosing each vertex $k \in FV(i)$ is determined using equation (6).

As it can be seen in equations (5) and (6), each ant uses two kinds of information in order to make a decision: the pheromone information (τ_{ik}), and the heuristic information (η_{ik}). In the ACS-OPHS, η_{ik} is defined as follows:

$$\eta_{ik} = \begin{cases} s_j / T_{i,k} & vertex\ j\ is\ a\ node \\ 1 / T_{i,k} & vertex\ j\ is\ a\ hotel, \end{cases} \tag{7}$$

α and β, in equations (5) and (6), are the other parameters of the ACS. They determine the relative importance of the pheromone trails and the heuristic information.

3.2. Identifying feasible intermediate hotels

As described in the previous section, two matrices, called FSH (Feasible Start-Hotels) and FEH (Feasible End-Hotels), are used to construct feasible tours. The FSH and FEH are two-dimensional matrices with h columns and D rows. $FSH[i][e]$ is the set of feasible start-hotels for trip i when the destination of this trip is H_e and $FEH[i][s]$ includes feasible end-hotels of trip i when this trip is started from H_s.

These matrices are constructed before the start of the main procedure of the ACS-OPHS. In other words, initially, concerning the time limits, the feasible start- and end-hotels of each trip are determined. Then, regarding this information, feasible tours are constructed in the construction phase at each iteration of the algorithm.

The FSH and FEH are defined using an auxiliary graph, $G^a = (V', E')$. V' is the set of vertices and contains $h \times (D-1) + 2$ nodes that are located in $D+1$ levels. Vertex $v_j^i \in V'$ is in level i and represents H_j as the start hotel of trip i. Since the origin of trip 1 must be H_0, the first level of G^a has only one node, called v_0^1. Similarly, since H_1 is the only feasible destination of trip D, only one node, called v_1^{D+1}, is in the last level of G^a. The number of nodes in each of the other $D-1$ levels is equal to h. Fig. 2 demonstrates the graphical representation of the vertices in G^a.

Concerning the time limits of the trips, the set of edges, E', is constructed as shown in Fig. 3.

Drawing an edge between v_j^i and v_k^{i+1} indicates that there might be a

feasible sequence of hotels in which trip i is started from H_j and ends in H_k. There is a doubt in the existence of this sequence since concerning the time limits of trips $i+1$ to D, it is possible that there is not any feasible sub-sequence of hotels starting from H_k. However, there is at least one feasible sub-sequence of hotels for trips 1 to i in which trip i ends in H_k. An example with 8 hotels and 4 trips is shown in Fig. 4. As it can be seen, concerning only the first two trips, there are three feasible sub-sequences of hotels in which trip 2 ends in H_2 (dotted lines). However, there is not any feasible sequence of hotels in which H_2 is the end-hotel of trip 2.

Now the FSH and FEH can be constructed. Initially, for each i and j, $FSH[i][j]$ and $FEH[i][j]$ are empty. Starting from v_1^{D+1}, the graph is explored, and if edge (v_j^i, v_k^{i+1}) is visited then:

- hotel H_k is a feasible end-hotel for trip i when the origin of this trip is H_j. Thus, $FEH[i][j] = FEH[i][j] \cup \{H_k\}$.
- hotel H_j is a feasible start-hotel for trip i when this trip must be ended in H_k. Thus, $FSH[i][k] = FSH[i][k] \cup \{H_j\}$.

In other words, this edge indicates that there is at least one feasible sequence of hotels in which trip i starts from H_j and ends in H_k. The construction procedure of the FSH and FEH is as shown in Fig. 5.

The time complexity of the procedure utilized for constructing the FSH and FEH is $O(DH^2)$. Recall that D and H are the number of trips and hotels, respectively. Since these two parameters usually do not have a very large value in the applications of the OPHS, the procedure introduced in this section is not time consuming.

3.3. Pheromone trails updating rules

After creating a feasible tour, the pheromone levels of the edges in the tour are updated locally using the following equation:

$$\tau_{ij}^{new} = (1-\rho)\tau_{ij}^{old} + \tau_0 \tag{8}$$

$\rho \in [0,1]$ is the evaporation rate. The initial value of the pheromone level (τ_0) is set to $1/\sum_{i \in NODES} s_i$. The local update occurs in the direction that the tour is constructed (i.e., from H_0 to H_1 or vice versa). In other words, if arc (i, j) is traversed by an ant of colony C_0, τ_{ij} must be updated; however, if an ant of colony C_1 goes through this edge, τ_{ji} must be changed.

As discussed in Section 3.1, each ant of both colonies construct a feasible tour. After that, the global best solution is updated if a better solution is found. Finally, according to the following equation, the pheromone levels are globally updated:

$$\tau_{ij}^{new} = (1-\rho)\tau_{ij}^{old} + \rho\Delta\tau_{ij}$$

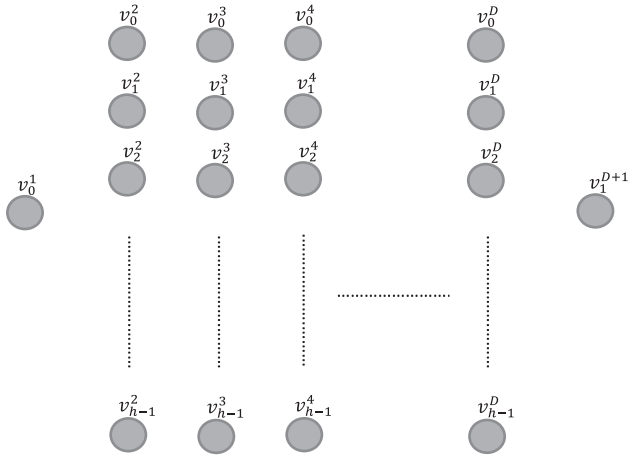


Fig. 2. The graphical representation of the vertices inG^a.

$$\Delta\tau_{ij} = S_{ib}^0 \big| S_{ib}^1 \big| S_{gb} \tag{9}$$

S_{ib}^0 and S_{ib}^1 are the scores of the best solutions constructed by the ants of C_0 and C_1, respectively. Moreover, S_{gb} is the score of the global best solution. With respect to the global best solution, the global update is performed in both directions. It means that if arc (i, j) is traversed, both τ_{ij} and τ_{ji} are updated. However, similar to the local updating, an unidirectional scheme is employed to change the pheromone values regarding S_{ib}^0 and S_{ib}^1 (i.e., updating only occurs in the direction that the solution is created).

3.4. Local search operators

Fig. 6 shows the pseudocode of the procedure that is employed to improve a constructed tour. This procedure consists of four operators. In the following, these operators are described in order of usage:

a) **ImproveHotels-DP:** following a dynamic programming strategy, this procedure improves the sequence of hotels with respect to the nodes visited along the tour. This local search operator has been introduced by Sohrabi et al. (2019) for the OPHS.
b) **Exchange:** following the best improvement strategy, it swaps two nodes from two different trips, whose exchange makes the most decrease in the traveled time. This operator is iterated until no more improvements are possible (Sohrabi et al., 2019).
c) **2-opt:** this operator reduces the length of each trip as far as possible following the best improvement strategy. The length of the trip is reduced iteratively. At each iteration, the pair of nodes, for which reversing the order of visited nodes among them makes the most reduction in length, is determined. Then, accordingly, the tour is modified (Sohrabi et al., 2019).
d) **Insert:** this operator determines the best insertion place of each unvisited node concerning time restrictions of the trips. Then, following the best improvement strategy, the node that makes the most increase in the score and then leads to the least increase in the length, is included in the tour. This operator is iterated until no more insertions are possible.

4. Experimental results

According to (Dorigo and Gambardella, 1997), α, β, ρ, and m are set to 1, 5, 0.9, and 10, respectively. If a better solution is not found over 1000 consecutive iterations, the algorithm is stopped. The initial value of q_0 is set to 0.9. During the execution of the ACS-OPHS, the value of q_0 is changed to establish an equilibrium between exploitation and exploration. If the global best solution is not improved over 400 consecutive iterations, the value of q_0 is altered to 0.2 to increase exploration. This parameter is set to 0.9 when a new global best tour is found. This modification encourages exploitation (i.e., searching the space around the new global best solution).

The ACS-OPHS has been applied to 405 benchmark instances created by Divsalar et al. (2013) (Divsalar et al., 2014) and 76 instances designed by Sohrabi et al. (2019). The instances created by Divsalar et al. (2013) (Divsalar et al., 2014) are divided into 17 sets regarding the number of hotels and the number of trips. 16 problem sets are referred to as SET X–Y, where X is the number of hotels (excluding H_0 and H_1) and Y is the number of trips. 395 of the OPHS instances are included in these sets. One other set, which is SET 4, contains ten instances with three hotels (excluding H_0 and H_1) and two or three trips. The optimal solutions are unknown for five out of ten instances in SET 4. All the instances can be accessed from http://www.mech.kuleuven.be/en/cib/op. Using the TSPHS instances with at most 500 customers, Sohrabi et al. (2019) have created 76 instances. These instances are divided into five sets. The sets are referred to as "SET X-Y-Z" where X is the number of hotels excluding H_0 and H_1, Y is the number of trips, and Z is the name of the problem (VRP or TSP) whose instances have been used by Vansteenwegen et al.

$E' = \{(v_0^1, v_k^2) | v_k^2 \in Level\ 2, T_{H_0,H_k} \leq T_1\ and\ T_{H_k,H_1} \leq \sum_{l=2}^{D} T_l\}$

for $d = 2$ to D

  for each $v_j^d \in Level\ d$

    if $\exists v_i^{d-1} \in Level\ d-1 : (v_i^{d-1}, v_j^d) \in E'$

      $E' = E' \cup \{(v_j^d, v_k^{d+1}) | v_k^{d+1} \in Level\ d+1, T_{H_j,H_k} \leq T_d\ and\ T_{H_k,H_1} \leq \sum_{l=d+1}^{D} T_l\}$

Fig. 3. The construction procedure of the edges inG^a.

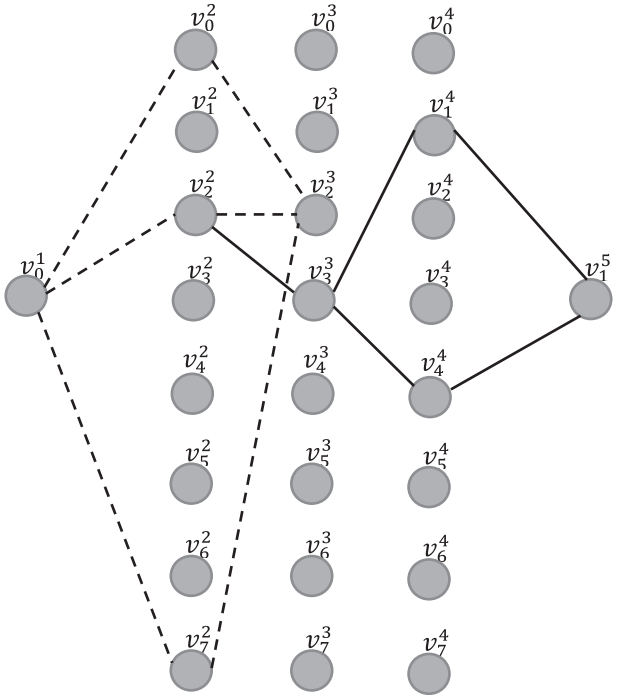


Fig. 4. Feasible sequences of hotels andG^a.

(2012) to construct the TSPHS instances.

As described in Section 2, four algorithms have been suggested for the OPHS until now: SVNS, MA, a hyper-heuristic that is called HH in the following subsections, and GRASP. The ACS-OPHS is compared with all of these algorithms except the SVNS since the SVNS cannot construct

X: The incumbent solution of the local search
N: Sequence of four neighborhood structures including N_1: ImproveHotels-DP, N_2: Exchange, N_3: 2-opt, and N_4: Insert
$i \leftarrow 1$
while $i < 5$
  $X' \leftarrow$ Apply neighborhood structure N_i on X
  if X' is better than X
    $i \leftarrow 1$
    $X \leftarrow X'$
  else
    $i = i + 1$

Fig. 6. The local search procedure.

Table 1
The configurations of the PCs on which each algorithm has been run.

Algorithm	CPU	RAM
GRASP,MA	Intel Core i7 with 3.5 GHz	4 GB
ACS	Intel Core i7 with 2.5 GHz	8 GB
HH	Intel Core i7 with 3.40 GHz	16 GB

feasible solutions for all OPHS instances in a reasonable time (Divsalar et al., 2014). In the following subsections, the results associated with the MA and GRASP are the values reported in (Sohrabi et al., 2019). The results reported in (Toledo et al., 2020) have also been used to compare the ACS-OPHS with HH. Table 1 presents the configurations of the PCs on which the code of each algorithm has been run. It should be mentioned that all the algorithms have been implemented in C++ and have been run three times over each instance. Moreover, to have some additional comparisons, the ACS-OPHS is also run for 15 and 30 times on each

for each $v_k^D \in Level\ D$

  if $(v_k^D, v_1^{D+1}) \in E'$

    $FEH[D][k] = \{H_1\}$

    $FSH[D][1] = FSH[D][1] \cup \{H_k\}$

for $d = D$ down to 2

  for each $v_k^d \in Level\ d$

    if $FEH[d][k] \neq \phi$ then

      for each $v_j^{d-1} \in Level\ d-1$

        if $(v_j^{d-1}, v_k^d) \in E'$

          $FEH[d-1][j] = FEH[d-1][j] \cup \{H_k\}$

          $FSH[d-1][k] = FSH[d-1][k] \cup \{H_j\}$

Fig. 5. Construction procedure of the FSH and FEH.

Table 2
The comparison of the ACS-OPHS and the other three algorithms.

Instances		Optimal Solutions				MA vs. ACS		GRASP vs. ACS		GRASP vs. ACS15		gap					Avg. CPU Time (s)			
SET	#Ins.	ACS	ACS15	MA	GRASP	MA	ACS	GRASP	ACS	GRASP	ACS	ACS	ACS15	MA	GRASP	HH	ACS	ACS15	MA	GRASP	HH
1–2	35	27	28	14	28	0	21	1	0	0	0	0.25	0.24	2.82	0.24	1.81	1.70	1.86	0.82	2.41	1.9
2–3	35	24	26	23	26	1	6	2	0	0	0	0.34	0.29	0.43	0.29	0.34	1.77	1.84	0.67	3.20	1.19
4 (OPT)	5	4	4	4	4	0	0	0	0	0	0	0.76	0.76	0.76	0.76	–	1.70	1.70	0.56	2.46	–
5–3	35	26	26	24	26	0	4	0	0	0	0	0.29	0.29	0.41	0.29	0.34	1.91	1.94	0.68	4.17	1.19
3–4	35	25	27	23	26	0	5	3	0	0	1	0.35	0.30	0.46	0.31	0.31	1.75	1.81	0.52	3.94	0.87
6–4	35	26	26	23	26	0	4	0	0	0	0	0.33	0.33	0.39	0.33	0.31	1.78	1.88	0.56	5.06	0.9
10–4	22	10	12	7	11	0	9	3	0	2	1	0.30	0.25	0.96	0.25	0.39	7.92	8.54	3.08	31.60	7.11
12–4	22	11	12	6	10	1	11	4	2	1	3	0.37	0.24	1.24	0.31	0.72	7.85	8.99	2.97	34.95	7.1
15–4	22	7	11	5	12	3	12	7	0	3	0	0.65	0.29	1.32	0.23	0.61	8.46	8.91	3.05	39.60	7.43
10–5	22	9	11	6	10	3	7	5	0	1	2	0.91	0.51	1.30	0.54	0.67	7.33	8.23	2.61	33.12	5.18
12–5	22	8	9	6	9	1	8	5	0	3	2	0.94	0.88	1.43	0.60	0.65	8.28	8.56	2.44	35.85	5.37
15–5	22	11	11	7	10	1	10	3	3	1	3	0.68	0.51	1.40	0.67	0.88	7.63	8.43	2.55	40.56	5.87
10–6	22	10	10	9	10	2	8	5	2	2	2	0.51	0.39	1.24	0.38	0.56	9.21	9.04	2.17	34.28	4.22
12–6	22	10	11	6	11	1	10	5	1	2	2	0.47	0.35	1.63	0.47	0.71	7.80	8.31	2.15	37.51	4.5
15–6	22	10	10	8	10	0	10	4	6	2	6	0.68	0.60	1.39	0.81	0.92	8.45	8.69	2.35	43.62	5.08
15–8	13	4	4	1	2	0	8	0	7	0	7	0.87	0.75	2.95	2.72	1.87	10.13	11.09	2.52	69.63	5.43
15–10	9	2	3	2	0	1	5	0	9	0	9	2.10	0.91	3.78	6.15	1.82	12.97	13.92	2.29	93.73	5.28
All Ins.	400	224	241	174	231	14	138	47	30	17	38	0.52	0.40	1.24	0.60	–	5.42	5.78	1.71	24.24	–

Column "Instances" presents the name and the number of instances for each set. For this comparison, only five out of ten instances with known optimal values in SET 4 have been considered. Column "Optimal Solutions" shows the number of instances for which each algorithm produces the optimal solutions. In the next columns, with respect to the quality of the best solutions, each pair of algorithms are compared. For example, in column "MA vs. ACS", it is presented that for how many of the instances in each set the ACS-OPHS provides better solutions than the MA and vice versa. Column "gap" shows the average gap of each algorithm in each set. The last column also contains the average CPU times in each set. To compute the average CPU time of each algorithm in each set, we perform as follows. For each instance, the average time taken for each run of the algorithm is computed. Then, the mean of these values in each set is reported as the average CPU time in that set. A similar approach is used to obtain the average gap in each set. Besides reporting these measures for each set, in the last row of columns "gap" and "Avg. CPU Time (s)", the average gaps and average CPU times over all 400 instances are shown. Please note that the average *total* execution time of the algorithms can be obtained by multiplying these values by three except for the ACS15 for which the corresponding value must be multiplied by 15.

instance. In the following sections, these variants of our proposed method are called ACS15 and ACS30, respectively.

The best solution of an algorithm for an instance is the one that is found during all specified number of runs of the algorithm on the instance. In all the experiments, the best solutions provided by the algorithms are compared. To compare the algorithms in terms of the execution time, average CPU times of the runs are considered in the following sections. To obtain the total execution time of a specific algorithm, the average CPU time is multiplied by the number of times the algorithm has been run. In the appendix, Tables A1 to A.22 show the best, average, and worst solutions that the ACS-OPHS produces over three runs on each instance. The average execution time for each instance is also available in these tables.

4.1. The solutions of the 405 benchmark instances introduced by (Divsalar et al. (2013) (Divsalar et al., 2014)

In Table 2, regarding the execution time and quality of solutions, the ACS-OPHS, MA, GRASP and HH have been compared. A measure called *gap* is also used that is determined for each instance as follows:

$$gap = \left(\frac{optimal\ value - best_{Alg}}{optimal\ value} \right) \times 100 \tag{10}$$

$best_{Alg}$ is the value of the best solution constructed by algorithm *Alg*.

The ACS-OPHS outperforms the MA in terms of:

a. The number of instances for which the optimal solutions are found,
b. The number of instances for which the solutions provided by the ACS-OPHS is better than that of the MA,
c. The average gap in each set.

The slight differences between the computation times of these two algorithms are negligible.

According to the results, the solutions provided by GRASP have higher quality than those of the ACS-OPHS when the number of trips is less than six. However, the ACS-OPHS outperforms this algorithm in SET 15-6, 15-8 and 15-10. Regarding several comparisons, Sohrabi et al. (2019) claimed that GRASP is a better approach to solve the OPHS than the MA if the number of trips is less than six. In most of the sets including instances with less than six trips, there is a little difference between the average gap of the ACS-OPHS and GRASP. Moreover, these two algorithms provide the optimal solutions for almost the same number of instances. Thus, we elicit that the ACS-OPHS is at least as good as GRASP.

In contrast to GRASP, ACS-OPHS can provide high quality solutions for instances with even more than six trips. Moreover, the execution time of ACS-OPHS is significantly better than that of GRASP.

Concerning the execution time of the ACS-OPHS and GRASP, to have a fair comparison, the ACS-OPHS has also been run 15 times on each instance. This version of the ACS-OPHS is called ACS15. As shown in Table 2, the ACS15 outperforms GRASP in 38 instances while there are 17 instances for which GRASP finds better solutions than the ACS15. Moreover, the ACS15 can find the optimal tours for more instances than GRASP. The superiority of the ACS15 with respect to the average gaps especially in the sets with more than six trips is also remarkable. According to these results, we can conclude that the ACS15 is as powerful as GRASP and, in some cases, it performs even significantly better than GRASP.

Unfortunately, the detailed results of the HH are not available. This issue makes it impossible to have some comprehensive comparisons with this algorithm. The ACS-OPHS is compared with the HH only in terms of the average gap and the average CPU time in each set. The average gap of the HH is better than that of the ACS-OPHS only in six sets. It should be noted that the results of the HH for ten instances in SET 4 are unavailable. According to (Toledo et al., 2020), the HH

Table 3
The five instances without known optimal solutions in SET 4.

Instances		Results				Avg. Execution Time of Each Run (s)			
Name	BKS	ACS	ACS15	MA	GRASP	ACS	ACS15	MA	GRASP
100-20-3-3.ophs	368	**368**	**368**	**368**	**368**	3.31	2.54	0.66	3.02
100-25-3-3.ophs	528	526	**528**	524	**528**	3.35	3.31	1.26	6.45
102-35-3-3.ophs	324	**324**	**324**	**324**	**324**	0.95	1.05	0.52	1.90
102-40-3-3.ophs	389	**389**	**389**	386	**389**	1.78	1.98	0.58	3.01
102-45-3-3.ophs	447	**447**	**447**	444	**447**	2.75	2.66	0.58	3.93

Table 4
Statistical tests.

Z-Test TestID	α		Z	P-Value	Lower bound for difference
1	0.05	$P_{ACS} - P_{MA} = 0.125$	3.56	0.000	0.067
Normality Test					
TestID				Anderson-Darling	P-Value
1	ACS; MA			12.354; 9.084	<0.005; <0.005
2	ACS15; GRASP			0.476; 1.439	0.218; <0.005
Mann-Whitney Test					
TestID	α			Wilcoxon	P-Value
1	0.05			24148	0.0000
2	0.05			488	0.0019

produces the optimal solutions for 217 instances while our proposed method finds the optimal tours for 224 instances.

The detailed results of the ACS-OPHS for the five instances with unknown optimal solutions in SET 4 are shown in Table 3. The solutions produced by the ACS-OPHS for these instances are the same as those of GRASP. Therefore, regarding these instances, no improvement has been achieved using the ACS-OPHS.

To validate the above comparisons, some statistical tests, which are explained as follows, have been performed using the *gap* measure. The results of these tests are shown in Table 4.

Test 1- ACS-OPHS vs. MA:

The description of the test is the same as the one considered by Sohrabi et al. in (Sohrabi et al., 2019). This test indicates that the performance of the ACS-OPHS is significantly better than that of the MA. Initially, the following hypotheses have been considered and then the Z-test has been used:

H0. : The proportion of the instances for which the ACS-OPHS produces the optimal solution is less than or equal to that of the MA

H1. : The proportion of the instances for which the ACS-OPHS produces the optimal solution is larger than that of the MA

The test results show that H_0 is rejected since P-Value is less than 0.05. The proportion of the instances for which the ACS-OPHS can find the optimal result is at least 6.7% more than that of the MA.

Considering 172 instances for which both algorithms cannot find the optimal tour, the following hypotheses have also been considered:

H1. The quality of the solutions provided by the ACS-OPHS is higher than that of the MA

In this test, H_0 is also rejected using the Mann–Whitney as a non-parametric test. A nonparametric test has been used since, according to Anderson-Darling test, the distributions of the gaps are not normal. Using the ACS-OPHS instead of the MA, the average gap of these 172 instances is reduced from 2.04% to 1.17%.

Test 2- ACS15 vs. GRASP:

Regarding the solutions produced by ACS15 and GRASP for the instances of SET 15-6, SET 15-8, and SET 15-10, the following hypotheses have been defined:

H0. When both algorithms cannot find the optimal solution, the quality of the solution provided by ACS15 is the same as that of the GRASP

H1. When both algorithms cannot find the optimal solution, the quality of the solution provided by the ACS15 is higher than that of the GRASP.

This test has been performed concerning 25 instances in the three sets for which both algorithms cannot find the optimal result. Since the Mann–Whitney test is significant at 0.0019, it can be concluded that, considering the largest instances in terms of the TNFS value, ACS-OPHS outperforms GRASP. However, in the other sets, the performance of these two methods are very similar.

4.2. *The solutions of the 76 benchmark instances introduced by (Sohrabi et al. (2019)*

The results of the comparison of the MA, GRASP, and ACS-OPHS are shown in Table 5. Regarding the execution times of the algorithms, the results of GRASP with 20 iterations (Sohrabi et al., 2019) have been used. The properties of the instances in each set are in the first three columns. Similar to Table 2, in the next columns, each pair of the algorithms are compared with respect to the quality of the solutions. The last column shows the average CPU time of each algorithm in seconds.

Similar to the MA and GRASP, the ACS-OPHS finds the optimal solutions of 12 instances in SET 4-4-VRP. The number of instances for which the ACS-OPHS provides a better solution than the MA is more than the number of instances for which the MA finds a better solution than the ACS-OPHS. The performance of the ACS-OPHS is better than GRASP with

Table 5
The solutions of the 76 benchmark instances.

SET	#Hotels	#Trips	MA vs. ACS		GRASP-20 iter vs. ACS		Avg CPU Time (s)		
			MA	ACS	GRASP	ACS	ACS	MA	GRASP-20 iter
4-4-VRP	6	4	**2**	**2**	**4**	0	52.51	37.76	58.42
2-3-TSP	4	3	2	**12**	**9**	3	48.95	62.50	45.75
4-4-TSP	6	4	3	**9**	3	**8**	60.79	50.25	36.98
9-7-TSP	10,11	7	4	**8**	1	**14**	37.18	30.12	30.24
8-4-TSP	10	4	**6**	**6**	**6**	5	59.46	54.21	55.71
Total Ins.	–	–	17	**37**	23	**30**	51.79	46.85	45.59

Table 6
The ACS-OPHS improves the best-known solutions of four instances.

Instances	BKS		ACS-OPHS results	
	Value	Time (s)	Value	Time(s)
tsp225-H4-T3-S3	10590	1037.30	**10620**	81.59
rd100-H6-T4-S3	4490	37.46	**4500**	8.91
eil76-H11-T7-S3	3800	28.38	**3810**	5.66
eil76-H10-T4-S4	4290	32.62	**4300**	8.48

20 iterations especially when the number of trips is more than six (i.e. SET 9-7-TSP). Before, the same conclusion has also been drawn regarding the instances introduced by Divsalar et al. (2013) (Divsalar et al., 2014).

There are 20 instances for which the ACS-OPHS produces better solutions than both the MA and GRASP with 20 iterations. As shown in Table 6, the ACS-OPHS also improves the best-known results of four instances with unknown optimal solutions. Previously, the best known results of these instances have been produced by GRASP with 500 iterations. Although the execution time of the ACS-OPHS is remarkably less than this version of GRASP, the ACS-OPHS can find better results for these instances. The solutions found by the ACS-OPHS for these instances are available in the appendix. This is another evidence that indicates the power of the ACS-OPHS in comparison with the state-of-the-art algorithms.

4.3. Effects of the two main characteristics of the ACS-OPHS

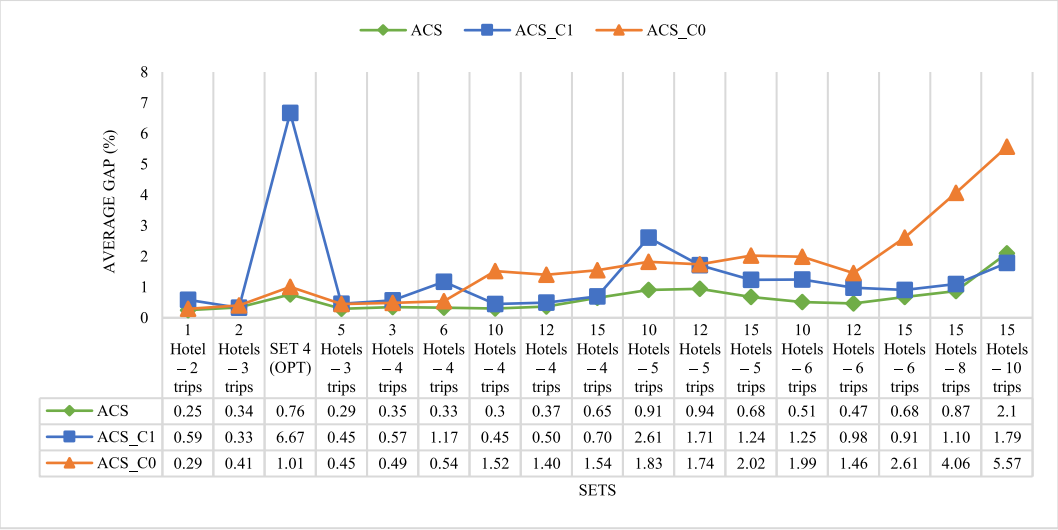
Our proposed ACS method has two main characteristics: it uses two colonies and modifies the value of q_0 during the execution time. To evaluate the influences of these properties on the quality of the solutions, two experiments are performed using 400 instances with known optimal solutions introduced by Divsalar et al. (2013) (Divsalar et al., 2014):

a) The effect of using two colonies:

Three versions of the ACS-OPHS are compared. In the first one, only colony C_0 is used (i.e., at each iteration of the algorithm, 20 ants start their tours from H_0). This version is called ACS_C0. ACS_C1 is the second version that only uses colony C_1. The third version is the original algorithm. With respect to *gap*, these algorithms have been compared. The results of this test are shown in Fig. 7. While the average gap over 400 instances with known optimal solutions is 0.52% for the ACS-OPHS, this measure equals to 1.35% and 1% for ACS_C0 and ACS_C1, respectively.

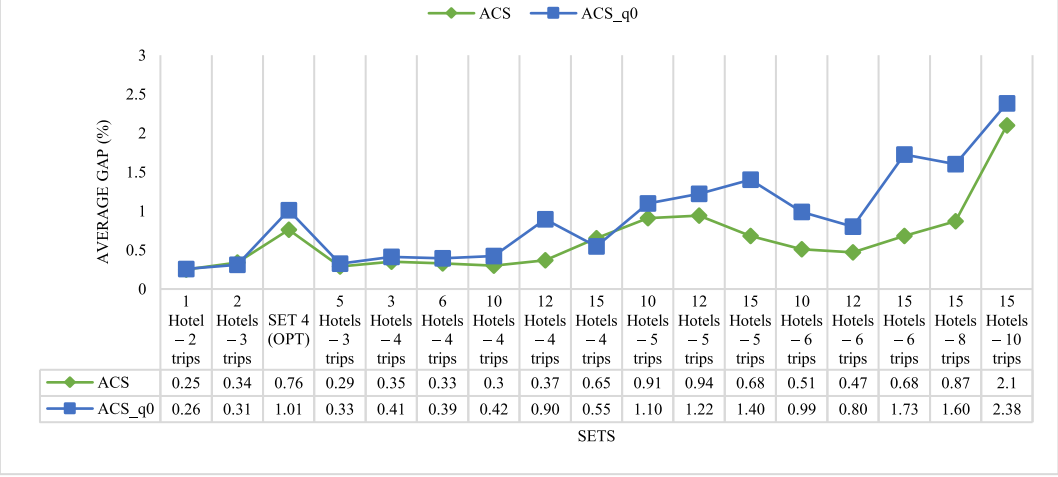
b) The effect of changing the value of q_0:

As mentioned before, the ACS-OPHS changes the value of q_0 during its execution to establish an equilibrium between exploitation and



	1 Hotel – 2 trips	2 Hotels – 3 trips	SET 4 (OPT)	5 Hotels – 3 trips	3 Hotels – 4 trips	6 Hotels – 4 trips	10 Hotels – 4 trips	12 Hotels – 4 trips	15 Hotels – 4 trips	10 Hotels – 5 trips	12 Hotels – 5 trips	15 Hotels – 5 trips	10 Hotels – 6 trips	12 Hotels – 6 trips	15 Hotels – 6 trips	15 Hotels – 8 trips	15 Hotels – 10 trips
ACS	0.25	0.34	0.76	0.29	0.35	0.33	0.3	0.37	0.65	0.91	0.94	0.68	0.51	0.47	0.68	0.87	2.1
ACS_C1	0.59	0.33	6.67	0.45	0.57	1.17	0.45	0.50	0.70	2.61	1.71	1.24	1.25	0.98	0.91	1.10	1.79
ACS_C0	0.29	0.41	1.01	0.45	0.49	0.54	1.52	1.40	1.54	1.83	1.74	2.02	1.99	1.46	2.61	4.06	5.57

Fig. 7. Evaluating the effect of using two colonies.



	1 Hotel – 2 trips	2 Hotels – 3 trips	SET 4 (OPT)	5 Hotels – 3 trips	3 Hotels – 4 trips	6 Hotels – 4 trips	10 Hotels – 4 trips	12 Hotels – 4 trips	15 Hotels – 4 trips	10 Hotels – 5 trips	12 Hotels – 5 trips	15 Hotels – 5 trips	10 Hotels – 6 trips	12 Hotels – 6 trips	15 Hotels – 6 trips	15 Hotels – 8 trips	15 Hotels – 10 trips
ACS	0.25	0.34	0.76	0.29	0.35	0.33	0.3	0.37	0.65	0.91	0.94	0.68	0.51	0.47	0.68	0.87	2.1
ACS_q0	0.26	0.31	1.01	0.33	0.41	0.39	0.42	0.90	0.55	1.10	1.22	1.40	0.99	0.80	1.73	1.60	2.38

Fig. 8. The influence of changing the value ofq_0.

Table 7
The number of runs (NR) within which the best solutions are obtained by the ACS30.

	Number of instances
$1 \leq$ NR ≤ 5	368
$5 <$ NR ≤ 10	19
$10 <$ NR ≤ 15	9
$15 <$ NR ≤ 20	2
$20 <$ NR ≤ 25	5
$25 <$ NR ≤ 30	2

exploration. Fig. 8 presents the results of comparing the original algorithm with the variant of the ACS-OPHS that does not employ this policy (called ACS_q0). As it can be seen, the modification of this parameter has a significant effect on the quality of the solutions especially when the number of trips is more than five. It should be mentioned that these instances are the hardest OPHS instances (Divsalar et al., 2014). For example, as you can see, in sets such as SET 15-6 or SET 15-8, the average gap decreases by about 50 percent using this technique.

4.4. Checking the stability of the ACS-OPHS

In this section, the stability of the ACS-OPHS is studied considering 405 standard benchmark instances created by Divsalar et al. (2013) (Divsalar et al., 2014). To this aim, the following investigations have been done on the best solutions produced after running the ACS-OPHS 3, 15, and 30 times on each instance. It should be noted that, ACS-OPHS, ACS15, and ACS30 refer to these three versions of the algorithm.

Initially, the impact of increasing the number of runs on the quality of the solutions is investigated. The ACS-OPHS, ACS15, and ACS30 find the optimal solutions for 224, 241, and 242 instances, respectively. Although by increasing the number of runs from 3 to 15, the optimal solutions are found for 17 more instances, there is only one more instance for which the optimal solution is produced after changing the number of runs from 15 to 30. While there are 51 instances for which the ACS15 produces better solutions than the ACS-OPHS, there are only 9 instances in which the ACS30 outperforms the ACS15. In addition, the solutions produced by the ACS30 are better than those of the ACS-OPHS in 55 instances. Therefore, it can be concluded that the improvements are more remarkable when the number of runs is increased from 3 to 15 versus when it is changed from 3 to 30.

As the second experiment, with respect to the results produced by the ACS30, we analyze the number of runs within which the best solutions are obtained. The results of this experiment are presented in Table 7. The best solutions are found during the first three runs for 350 out of 405 instances (i.e., 86.42% of the instances). Moreover, as shown in Table 7, the best solutions are produced during the first five runs for 90.86% and during the first ten runs for 95.56% of the instances. These results show that three or five are suitable number of times for running the proposed algorithm.

Fig. 9 shows the average standard deviation of the values of the solutions produced by the ACS15 in each set. To determine these values, initially, the standard deviation of 15 solutions found for each instance is calculated; then, the average of these values in each set is obtained. Although the average standard deviation is influenced by both the number of hotels and the number of trips, it seems that the number of trips has more impact on this measure. This is expected since when the number of trips increases, more different solutions are produced following the bi-directional search strategy used in the ACS-OPHS.

In Table 8, considering the average standard deviation in each set, the ACS15 and ACS30 are compared. These results and the results of the

Table 8
The impact of increasing the number of runs on the average standard deviation in each set.

	Average SD	
Sets	ACS15	ACS30
1 Hotel – 2 trips	0.55	0.56
2 Hotels – 3 trips	0.98	1.08
SET 4	0.89	0.85
3 Hotels – 4 trips	1.47	1.57
5 Hotels – 3 trips	1.15	1.18
6 Hotels – 4 trips	1.81	1.80
10 Hotels – 4 trips	6.17	7.08
10 Hotels – 5 trips	6.44	7.14
10 Hotels – 6 trips	10.68	10.58
12 Hotels – 4 trips	4.89	5.68
12 Hotels – 5 trips	4.01	4.65
12 Hotels – 6 trips	8.81	8.38
15 Hotels – 4 trips	7.38	7.59
15 Hotels – 5 trips	10.21	9.70
15 Hotels – 6 trips	6.23	6.33
15 Hotels – 8 trips	14.14	14.02
15 Hotels – 10 trips	20.82	19.21

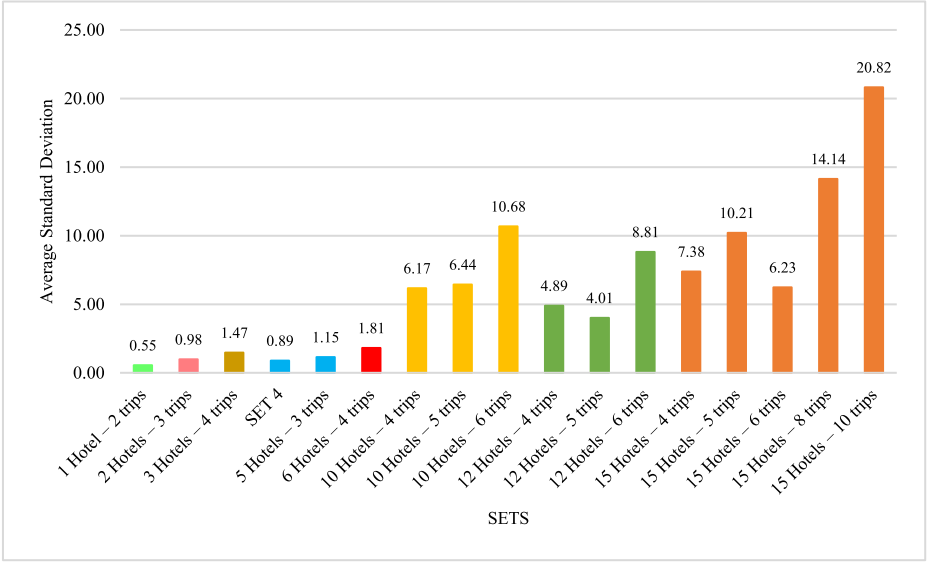


Fig. 9. The average standard deviation of the values of the solutions produced by the ACS15 in each set.

previous investigations indicate that, in our proposed method, there is a quite well equilibrium between exploration and exploitation. Considering an OPHS instance, our algorithm does not always converge to a similar solution in all runs; however, it produces solutions that are distributed in a good region of the feasible search space (i.e., a region that mostly includes the optimal or near optimal solution for that instance). When the number of trips and hotels increases and so the problem gets harder to solve, the exploration of our algorithm is also enhanced to amplify the probability of finding the optimal solution. However, the exploration is not enhanced unlimitedly. Using the pheromone trails updating rules and the local search procedure, a suitable level of exploitation is provided, as well.

5. Conclusion

In this paper, we proposed an ACO algorithm for the OPHS. This algorithm follows the standard procedure of the ACS and is denominated as ACS-OPHS. A novel and fast approach was also introduced to identify the feasible intermediate hotels. Using this approach in the ACS-OPHS, tours are constructed sequentially, which is a new point of view on the tour construction procedure for this problem. Moreover, in contrast to some proposed methods in the literature, this idea makes the ACS-OPHS independent from solving the OPs between all the pairs of the hotels.

The experimental results show that the ACS-OPHS is a well-performing approach to solve the OPHS. Regarding the quality of the solutions, it outperforms the state-of-the-art algorithms in several instances. It also improves the best known results of four instances with unknown optimal solutions.

While GRASP has a weak performance when the number of trips is more than six, the ACS-OPHS produces high quality solutions even for the instances with large number of hotels and trips. The main drawback of the GRASP proposed in the literature to solve the OPHS is that it is relatively time consuming. It is worth investigating the execution time reduction of GRASP using the sequential tour construction procedure that have been described in this article. So, to eliminate some drawbacks of GRASP, this algorithm can be combined with the ACS-OPHS in the future.

The procedure introduced for identifying feasible intermediate hotels can be adapted for solving other problems with intermediate facilities. Future research might also focus on adapting the ACS-OPHS to solve similar problems. As an example, if we can fix the number of trips in the TSPHS in such a way that all the nodes can be visited, an adapted version of the ACS-OPHS can be applied to construct a good solution.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to show their gratitude to the editor and two anonymous reviewers for their expert advices that improved this manuscript.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.ejtl.2021.100036.

References

Divsalar, A., Vansteenwegen, P., Cattrysse, D., 2013. A variable neighborhood search method for the orienteering problem with hotel selection. Int. J. Prod. Econ. 145 (1), 150–160. https://doi.org/10.1016/j.ijpe.2013.01.010.

Divsalar, A., Vansteenwegen, P., Sörensen, K., Cattrysse, D., 2014. A memetic algorithm for the orienteering problem with hotel selection. Eur. J. Oper. Res. 237 (1), 29–49. https://doi.org/10.1016/j.ejor.2014.01.001.

Dorigo, M., Gambardella, L.M., 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans. Evol. Comput. 1 (1), 53–66. https://doi.org/10.1109/4235.585892.

Fleurent, C., Glover, F., 1999. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. Inf. J. Comput. 11 (2), 198–204. https://doi.org/10.1287/ijoc.11.2.198. CEC '02. Proceedings of the 2002 Congress on, 20.

Gambardella, L.M., Montemanni, R., Weyland, D., 2012. Coupling ant colony systems with strong local searches. Eur. J. Oper. Res. 220 (3), 831–843. https://doi.org/10.1016/j.ejor.2012.02.038.

Gunawan, A., Lau, H.C., Vansteenwegen, P., 2016. Orienteering Problem: a survey of recent variants, solution approaches and applications. Eur. J. Oper. Res. 255 (2), 315–332. https://doi.org/10.1016/j.ejor.2016.04.059.

Ke, L., Feng, Z., 2008. "A new ant colony optimization approach for the orienteering problem," in Intelligent Control and Automation. In: WCICA 2008. 7th World Congress on, 2008, pp. 2027–2032. https://doi.org/10.1109/WCICA.2008.4593236.

Ke, L., Archetti, C., Feng, Z., 2008. Ants can solve the team orienteering problem. Comput. Ind. Eng. 54 (3), 648–665. https://doi.org/10.1016/j.cie.2007.10.001.

Liang, Y.C., Kulturel-Konak, S., Smith, A.E., 2002. "Meta heuristics for the orienteering problem," in Evolutionary Computation. In: 02, vol. 1, pp. 384–389. https://doi.org/10.1109/CEC.2002.1006265.

Mei, Y., Salim, F.D., Li, X., 2016. Efficient meta-heuristics for the multi-objective time-dependent orienteering problem. Eur. J. Oper. Res. 254 (2), 443–457. https://doi.org/10.1016/j.ejor.2016.03.053.

Schiffer, M., Schneider, M., Laporte, G., Walther, G., 2019. Vehicle routing and location routing with intermediate stops: a review. Transp. Sci. Publ. 53 (2), 319–343. https://pubsonline.informs.org/doi/10.1287/trsc.2018.0836.

Schilde, M., Doerner, K.F., Hartl, R.F., Kiechle, G., 2009. Metaheuristics for the bi-objective orienteering problem. Swarm Intell. 3 (3), 179–201. https://doi.org/10.1007/s11721-009-0029-5.

Sohrabi, S., Ziarati, K., Keshtkaran, M., 2019. A greedy randomized adaptive search procedure for the orienteering problem with hotel selection. Eur. J. Oper. Res. 283 (2), 426–440. https://doi.org/10.1016/j.ejor.2019.11.010.

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

Tsiligirides, T., 1984. Heuristic methods applied to orienteering. J. Oper. Res. Soc. 35 (9), 797–809. https://doi.org/10.1057/jors.1984.162.

Vansteenwegen, P., Gunawan, A., 2019. Orienteering Problems Models and Algorithms for Vehicle Routing Problems with Profits.

Vansteenwegen, P., Souffriau, W., Sörensen, K., 2012. The travelling salesperson problem with hotel selection. J. Oper. Res. Soc. 63 (2), 207–217. https://doi.org/10.1057/jors.2011.18.

Verbeeck, C., Sörensen, K., Aghezzaf, E.H., Vansteenwegen, P., 2014. A fast solution method for the time-dependent orienteering problem. Eur. J. Oper. Res. 236 (2), 419–432. https://doi.org/10.1016/j.ejor.2013.11.038.

Verbeeck, C., Vansteenwegen, P., Aghezzaf, E.H., 2017. The time-dependent orienteering problem with time windows: a fast ant colony system. Ann. Oper. Res. 254 (1–2), 481–505. https://doi.org/10.1007/s10479-017-2409-3.