Ants can solve the parallel drone scheduling traveling salesman problem

Quoc Trung Dinh*
ORLab, Faculty of Computer Science,
Phenikaa University
Ha Dong, Hanoi, Vietnam

Duc Dong Do†
Faculty of Information Technology,
VNU University of Engineering and
Technlology
Cau Giay, Hanoi, Vietnam
dongdoduc@vnu.edu.vn

Minh Hoàng Hà‡
ORLab, Faculty of Computer Science,
Phenikaa University
Ha Dong, Hanoi, Vietnam
hoang.haminh@phenikaa-uni.edu.vn

ABSTRACT

In this work, we are interested in studying the parallel drone scheduling traveling salesman problem (PDSTSP), where deliveries are split between a truck and a fleet of drones. The truck performs a common delivery tour, while the drones are forced to perform back and forth trips between customers and a depot. The objective is to minimize the completion time coming back to the depot of all the vehicles. We present a hybrid ant colony optimization (HACO) metaheuristic to solve the problem. Our algorithm is based on an idea from the literature that represents a PDSTSP solution as a permutation of all customers. And then a dynamic programming is used to decompose the customer sequence into a tour for the truck and trips for the drones. We propose a new dynamic programming combined with other problem-tailored components to efficiently solve the problem. When being tested on benchmark instances from the literature, the HACO algorithm outperforms state-of-the-art algorithms in terms of both running time and solution quality. More remarkably, we find 23 new best known solutions out of 90 instances considered.

KEYWORDS

Parallel drone scheduling, traveling salesman problem, ant colony optimization, dynamic programming.

ACM Reference Format:
Quoc Trung Dinh, Duc Dong Do, and Minh Hoàng Hà. 2021. Ants can solve the parallel drone scheduling traveling salesman problem. In *Proceedings of the Genetic and Evolutionary Computation Conference 2021 (GECCO '21).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3449639.3459342

1 INTRODUCTION

Nowadays, as technology for aerial mobility evolves, delivery means are naturally destined to make use of unmanned aerial vehicles (UAVs or drones), and multiple companies such as Amazon [1], UPS [5], DHL [2], 7-Eleven [4] and Alibaba [3] have already had plans for this transition in the near future. Using drones for delivery provides four advantages: (1) they can be operated automatically without a human driver, (2) they fly over common road networks and as a result can service truck-unreachable customers, (3) their speed is faster than trucks' speed, and (4) they are much cheaper. Yet, drones also have several disadvantages, due to their narrow operation range, small capacity, and possible need of visiting stations for charging or swapping batteries. To offset the drawbacks of both vehicles, they are coupled together to make delivery, leading to novel routing problems that attract much attention of the research community. Two leading notable problems are introduced in [20]. The first problem called the Flying Sidekick Traveling Salesman Problem (FSTSP) arises in scenarios in which the depot is relatively far from the customers and a single drone can operate in synchronization with a delivery truck. More precisely, the drone can be launched from the truck at a customer location and returns to the truck at another customer location. The second problem is motivated in the event that a large proportion of customers are located within a drone's flight range from the depot and can be serviced by a fleet of drones. A truck is also mobilized to service drone-unreachable customers which are far from the depot or require heavy parcels. In this paper, we consider the latter problem, the Parallel Drone Scheduling Traveling Salesman Problem (PDSTSP), which can be formally defined as follows.

We are given a complete undirected graph $G = (V, E)$, where the node set $V = \{0, 1, \ldots, n\}$ represents the depot (node 0) and the set of customers $\overline{V} = \{1, \ldots, n\}$. The set E represents edges linking every pair of nodes in V. A truck and a set M of m identical drones are located at the depot to deliver parcels to the customers. The truck starts from the depot, services a subset of the customers, then returns back to the depot. The remaining customers are serviced by the drones on back and forth trips from the depot to a single customer. Not every customer can be serviced by the drones due to practical constraints like the weight of the parcel or an excessive distance of the customer location from the depot. Let $V^* \subseteq \overline{V}$ denotes the set of customers that can be serviced by the drones (also called drone-eligible customers). The travel time from node i to node j of the truck is denoted as c^t_{ij}. The time required by a drone to service a customer i is denoted as c^d_i, which is calculated by doubling the time of flying from the depot to customer i. Without loss of generality, we suppose that the truck and the drones start from the depot at time 0. The objective of the PDSTSP is to minimize

*The first author
†The first corresponding author.
‡The second corresponding author.

GECCO '21, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8350-9/21/07…$15.00
https://doi.org/10.1145/3449639.3459342

the maximal time required to come back to the depot of the truck and all the drones after servicing all the customers.

The PDSTSP includes two decision layers. In the first layer, the customers are assigned to the truck and the fleet of drones. The second layer relates to the routing optimization which consist in solving two NP-hard problems: a TSP for the truck and a Parallel Machine Scheduling (PMS) problem for the drones. Figure 1 shows a feasible solution for the PDSTSP with 8 customers and two drones.

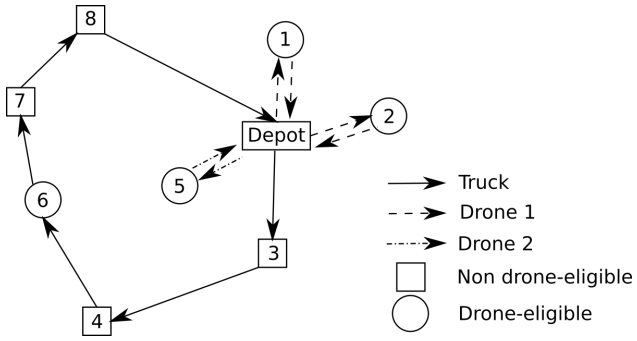

Figure 1: An example of a solution for the PDSTSP

2 RELATED WORKS

While the FSTSP and its variants are intensively studied in the literature (see, for example, [7, 9, 11, 16, 17, 21]), there are very few papers considering the PDSTSP. The problem is first introduced in [20] where the authors propose a Mixed Integer Linear Programming (MILP) formulation and a heuristic. The MILP can provide exact solutions but is time-consuming and not applicable for instances of practical size. The main principle of the heuristic is to partition the set of customers into two subsets (a vehicle set and a drone set). Then, a TSP tour is computed for the customers assigned to the truck and a parallel machine scheduling (PMS) problem is solved to assign customer requests (jobs) to drones (machines). At the end, an improvement step is performed to reassign customers either to the drone set or to the vehicle set in order to better balance vehicle and drones activities. Their method can be classified into low-level heuristics which include a solution construction followed by local search. Due to the simplicity, this deterministic algorithm provides only one solution during the search and is not expected to provide high-quality solutions as metaheuristics.

Mbiadou Saleu et al. [22] propose the first metaheuristic composed of two steps. The first step (or coding step) transforms a feasible PDSTSP solution into a customer sequence T, also called giant tour or TSP tour. The second step (or decoding step) decomposes the customer sequence into a tour T_t for the truck and a set of trips T_d for the drones. This decomposition must satisfy the requirements that the customers in the truck tour follows the same order as in sequence T and the customers that are not on the tour T_t will be serviced by the drones. The decoding procedure is expressed as a bicriteria shortest path problem on a acyclic graph $G^T = (V^T, A^T)$ constructed as follows. $V^T = \overline{V} \cup \{0, n+1\}$ represents the set of customers added by two copies of the depot: the original depot 0 and its copy $n+1$. Given two nodes i and j in V^T such that i is before j in the sequence T, arc i, j exists in A^T if and only if all the customers between i and j can be serviced by the drones. Each arc $(i, j) \in A^T$ is associated with a cost vector (c^1_{ij}, c^2_{ij}), where c^1_{ij} and c^2_{ij} represent the costs incurred for the truck and the drones, respectively, if the truck travels directly from i to j and all the customers between i and j are serviced by the drones: $c^1_{ij} = c^t_{ij}$ and $c^2_{ij} = 1/m \sum_{k \in V^T : i <_T k <_T j} c^d_k$.

After the acyclic graph is created, the shortest path from the original depot 0 to its copy $n+1$ is found by dynamic programming. The principle is to progressively associate a list of labels $\mathcal{L}(i)$ with each node $i \in V^T$. A procedure goes through the graph from the origin depot node 0 to the destination depot node $n+1$ by following the order defined in sequence T. For a given node i, list $\mathcal{L}(i)$ is constructed by adding a new label for every label in $\mathcal{L}(j)$, where j is a predecessor node of i and $\mathcal{L}(j)$ is the list of labels at node j. The new label is simply defined by adding the arc cost. To limit the number of possibly generated labels, several bounding mechanisms and a dominance rule are used. However, the procedure still can be very slow and memory-consuming because these techniques cannot help completely avoid creating uncontrollable quantities of labels on large-scale instances.

More recently, Dell'Amico et al. [10] propose a matheuristic in which the authors use the same idea of coding and decoding steps as in [22]. However, instead of dynamic programming, a MILP approach is used to handle the decoding phase. For a given TSP tour T, the authors add constraints that enforce binary variables $x_{i'j'}$ equal to zero to a pure MILP formulation of the problem. Here, i' and j' are two customers such that i' lies after j' in the TSP tour T and variables $x_{i'j'}$ represent the appearance of arc (i', j') in the solution. Solving this modified formulation provides optimal solutions (w.r.t the order of the customers in T) for the splitting approach and allows the procedure to exploit a wider search space than the dynamic programming-based approach. However, the computational time required to solve MILP models might be very expensive. For this reason, when attacking large instances, the authors add further constraints to the MILP model. In this way the time required to solve the MILP is reduced, making it possible to carry out more iterations of the heuristic in a given time limit. The drawback is that the constraints reduce the search space region explored by the MILP, potentially increasing the number of iterations required to converge.

In this research, we aim at developing a new metaheuristic for the problem. Our algorithm is mainly based on Ant Colony Optimization (ACO) algorithm, a metaheuristic technique that uses artificial ants to solve combinatorial optimization problems which can be reduced to finding good paths through graphs. ACO is a class of optimization algorithms based on the actions of an ant colony. The interested readers are referred to [12] for the detailed descriptions of ant behavior relating to ACO.

The use of ant colonies is first applied to the Traveling Salesman Problem (TSP) in [13] and several variants of the Vehicle Routing Problem (VRP) in [6, 8]. To the best of our knowledge, the ACO algorithm has never been applied to solve the PDSTSP.

The main contribution of our research is to design a hybrid ant colony optimization (HACO) algorithm integrated with several components adapted from other metaheuristics. We follow the idea

of representing a PDSTSP solution as a giant tour of the previous researches. We propose an easy-to-implement and more efficient split procedure from which we can better control the trade-off between quality and speed, a deconstruction phase borrowed from the idea of the large neighborhood search, and fast local searches. The experiments on PDSTSP benchmark instances show a dominance of our algorithm over the two algorithms in terms of solution quality and speed. The rest of the paper is organized as follows. Section 3 describes our metaheuristic. Section 4 analyzes the performance of the HACO on the PDSTSP benchmark data sets. Finally, our conclusions are given in the Section 5.

3 HYBRID ANT COLONY OPTIMIZATION ALGORITHM FOR THE PDSTSP

One of the most important factors to consider when designing ACO algorithms is to define pheromone trails. As mentioned earlier, the PDSTSP involves two main decision layers: the assignment of customers to the truck or the drone and the sequence of customers in the truck tour. Therefore, we manage two corresponding pheromone trails to handle these decisions: μ_i^a to decide if customer i is assigned to truck or drone and μ_{ij}^s to find the next customer j to service from the customer i in the truck tour. Our algorithm includes three main components: solution construction, local search, and pheromone trail update. The overview of our method is presented in Algorithm 1. We now discuss these procedures in detail.

Algorithm 1 An overview of the HACO algorithm

```
1: f(S_best) ← +∞
2: Initialize pheromone values
3: while the stopping condition is not met do
4:     f(S_local) ← +∞
5:     for k = 1 to nb_ant do
6:         S ← SolutionConstruction()
7:         S ← LocalSearch(S)
8:         if f(S) < f(S_local) then
9:             S_local ← S
10:        end if
11:    end for
12:    Update pheromone values using S_local
13:    if f(S_local) < f(S_best) then
14:        S_best ← S_local
15:    end if
16: end while
```

3.1 Ants construct PDSTSP solutions

We now explain `SolutionConstruction()`, the first procedure called in Algorithm 1, which constructs initial solutions. The pseudo-code of the procedure is given in Algorithm 2. Its main components are described as followed:

`SelectDroneNode`: the goal is to partially select a subset of customers serviced by the drones. We iterate over all the drone-eligible customers; each is selected with the probability of r_d. Suppose after this step we have a set of n_p selected customers. Among those we then select $\min(n_p, \lfloor r'_d \times n_d \rfloor)$ customers i with highest pheromone values μ_i^a. Here, n_d is the number of customers serviced by the drones in the current best found solution. It is set to null in the first iteration where a best solution has not existed. The parameter r'_d is the ratio of customers to be retained to be serviced by the drones in the current iteration. Preliminary tests show that the impact of r'_d on the performance of the algorithm is not really clear. Therefore, to keep the parameter tuning simple, we decide to set r'_d to 0.5. These customers are recorded in set N_d. They are combined with the customers visited by the drones found in the split procedure below to create the final set of customers serviced by the drones in the solution of the current iteration. These customers are then assigned to the drones by a greedy heuristic presented in the split procedure. This drone node selection brings us two advantages. First, it can be considered as a removal operator in the classical Large Neighborhood Search (LNS) metaheuristic [23] and allows the algorithm to flexibly control the balance between the intensification and diversification. Second, it helps to reduce the number of customers considered in the next procedures which are more expensive in terms of computation time, thus speeding up our algorithm in overall.

Algorithm 2 Solution construction procedure

```
1: procedure SolutionConstruction
2:     N_d ← SelectDroneNode(n_d) ▷ Select a set of customers to be serviced by the drones
3:     T ← AntTSPTour(N_t)                ▷ N_t = V \ N_d
4:     T ← ImproveTSP(T)
5:     S ← Split(T)
6:     return S
7: end procedure
```

`AntTSPTour`: this procedure is to build a giant TSP tour T visiting the depot and all the remaining customers other than those in N_d. For that matter, we apply a randomized nearest-neighbor construction procedure using the roulette wheel selection with the probability of selecting the next customer j to visit from the customer i proportional with $\frac{\mu_{ij}^s}{c_{ij}^t+1}$. Note that, μ_{ij}^s and $\frac{1}{c_{ij}^t+1}$ represent the pheromone trail and heuristic information, respectively. This means that if the pheromone trail on edge (i, j) is more concentrated and j is closer to i, customer j has more chance to be the next location visited after customer i.

`ImproveTSP`: in this procedure, the truck route T obtained from the `AntTSPTour` procedure is then improved using the well-known 3-opt local search move [18]. We note that this local search is widely used in ACO algorithms to successfully solve the TSP [19, 24].

`Split`: this procedure is the backbone of our algorithm. It computes the best partition of the customer set between the truck and the unique drone with the constraint that the customers in the vehicle tour follows the same order as in TSP tour T. Similarly to [22], this problem is modeled as a shortest path problem in an acyclic directed graph G^T and is solved by dynamic programming.

Our split procedure is described in Algorithm 3. Node $i \in H$ is associated with nb_l+1 labels; each has the form $\mathcal{L}(i, p)$ representing the cost incurred by the truck given that the cost incurred by the drones is approximately less than or equal to $\frac{p}{nb_l}$ of the objective value of the current best solution, where p is an integer ranging from

0 to nb_l. At the initialization, $\mathcal{L}(i,p)$ is set to $+\infty$ for every node i and every value of p (Line 4). The procedure starts by assigning label $\mathcal{L}(0,0)$ to 0 (Line 7). It then goes through the graph from the origin depot node 0 to the destination depot node $n+1$ and all the values of p from 0 to nb_l. From a given node i and for a given value p, the labels at j are updated by formulas in lines 12-14. Here, c^1_{ij} and c^2_{ij} have the same meanings as in the split algorithm of [22] described above, and $f(S_{best})$ is the objective value of the current best solution.

In Line 20, we have a set of $nb_l + 1$ labels; from each of them we derive a truck path τ_t and a set of customers serviced by the drones $\overline{N_d}$. The well-known longest processing time heuristic [22] is applied on the $\overline{N_d}$ to provide a feasible PMS solution (procedure `GreedyHeuristic` in Line 26). As such, the customers are sorted in the decreasing order of their processing time. Then, they are successively assigned to the drone with the least flying time.

It is worth mentioning that when nb_l is set to infinity, our split is similar to the procedure proposed in [22]. The value of nb_l has an important impact on the performance of our algorithm. If nb_l is too large, the dynamic programming could explore more labels, leading to high running time but we would have higher chance to find better solutions. On the other hand, when nb_l is small, the performance of the split could decrease, but its computation time would be faster and we could perform more iterations of the algorithm. Therefore, to have the best performance of the overall algorithm, the value of nb_l should be selected such that it gives a good trade-off between the speed and the quality of the split procedure. A preliminary experiment shows that nb_l equal to 100 satisfies our objective. This value is selected in our work.

Algorithm 3 Split procedure

```
1: procedure Split
2:     for i = 0 to n do
3:         for p = 0 to nb_l do
4:             L(i,p) ← +∞
5:         end for
6:     end for
7:     L(0,0) ← 0
8:     for i = 0 to |T| − 1 do
9:         for p = 0 to nb_l do
10:            for j = i + 1 to |T| do
11:                if arc (i,j) ∈ G^T then
12:                    p' = p + ⌊0.5 + nb_l × c^2_ij / f(S_best)⌋
13:                    if L(j,p') > L(i,p) + c^1_ij then
14:                        L(j,p') = L(i,p) + c^1_ij
15:                    end if
16:                end if
17:            end for
18:        end for
19:    end for
20:    L* ← labels L(|T|,p) ∀p = 0,..,nb_l
21:    f(S) ← +∞
22:    for each label l in L* do
23:        τ_t ← truck path that led to label l
24:        time_t ← length of τ_t
25:        N_d ← all nodes that are not in the path
26:        τ_d ← GreedyHeuristic(N_d)
27:        time_d ← time coming back to the depot of the latest
           drone in τ_d
28:        if f(S) > max(time_t, time_d) then
29:            S ← solution corresponds with τ_t and τ_d
30:            f(S) ← max(time_t, time_d)
31:        end if
32:    end for
33:    return S and f(S)
34: end procedure
```

3.2 Local search operators

Previous researches show that the integration of local search in ACO algorithms improves their performance in overall when applied to solve vehicle routing problems (see, for example, [6, 8]). Moreover, in the solution construction above, a customer is assigned to each drone in the greedy fashion. So far, our algorithm has not had any mechanism to repair bad drone-customer assignments created by the simple greedy heuristic. Therefore, we decide to use two local search operators to improve the quality of the algorithm, as well as to handle the decision related to the assignment of the customers to each drone as follows:

- `SwapTruck-Drone`: This operator tries to swap a customer serviced by the truck and a customer serviced by the drone to search for a better solution.
- `SwapDrone-Drone`: This operator swaps two customers serviced by two different drones to find an improved solution.

Our local search operators are performed in the best improvement fashion. Verifying if a move creates a saving in the objective function can be done in $O(m)$, where m is the number of drones. This procedure is fast and its running time can be negligible if the amount of drones is small. However, when the number of drones is large enough, computing the change in the objective function value in $O(1)$ is necessary to reduce the running time of the overall algorithm. For this purpose, it is sufficient to use an additional data structure that records three drones with the longest, second longest, and third longest flying time. Note that we must record three drones, instead of one, to handle the cases where there are more than two drones with the equal longest flying times.

3.3 Pheromone trail update

Pheromone trail update (line 12 of Algorithm 1) is a crucial step in ant colony optimization as it guides ants to find good solutions. We use the Smooth Max-Min Ant System method proposed in [14, 15], a refined version of the well-known pheromone boundary rule [25], to update the pheromone trails as follows:

$$\mu^a_i = \begin{cases} \rho^a \mu^a_i + (1-\rho^a)\mu_{min} \text{ if customer } i \text{ is serviced by the truck} \\ \rho^a \mu^a_i + (1-\rho^a)\mu_{max} \text{ if customer } i \text{ is serviced by the drones} \end{cases}$$

$$\mu^s_{ij} = \begin{cases} \rho^s \mu^s_{ij} + (1-\rho^s)\mu_{max} \text{ if truck travels from } i \text{ to } j \\ \rho^s \mu^s_{ij} + (1-\rho^s)\mu_{min} \text{ otherwise} \end{cases}$$

where $\rho^l \in (0, 1]$ ($l \in \{a, s\}$) is the evaporation parameter; μ_{min} and μ_{max} are the lower and upper bounds of the pheromone trails, respectively. The value of these parameters is set as in Table 1. As can be derived from these formula, the greater the trail μ_i^a of a customer i is, the higher probability it is serviced by the drones with. And if the trail μ_{ij}^s of an arc from customer i to customer j is larger, this arc has more chance to appear in the next solution.

4 EXPERIMENTS

Our algorithm is implemented in C++. All experiments are carried out on a 2.10GHz Intel E5-2683 v4. In this section, we mainly investigate the performance of our HACO by comparing with reference algorithms proposed in [10, 22], which are run on an Intel core(TM) i5-6200U CPU at 2.30Ghz and an Intel(R) Xeon(R) E5-2620 v4 at 2.10GHz processors. These machines have single-thread rating performance approximately similar to our processor according to Passmark Software (https://www.cpubenchmark.net/). Since different CPU speed conversion techniques can provide very different results, we decided to present the raw running times, letting the readers choose their preferred approach.

The PDSTSP instances considered in our experiment have been generated from classic TSPLIB instances by [22]. The number in each instance name corresponds to the number of customers, varying from 48 to 229. Euclidean distances are used for the drones while Manhattan distances are used for the truck. For each original TSP instance, several PDSTSP were generated by modifying the following settings: (i) the depot location, which is either in the center of the customers or in one corner of the customers' region; (ii) the percentage of drone-eligible customers, ranging from 0% to 100%; (iii) the speed of the drones, which is expressed as a factor of the truck speed, with values ranging from 1 to 5; (iv) the number of drones, that is from 1 to 5.

The six parameters required to be tuned in our HACO algorithm are: the lower bound μ_{min} and the upper bound μ_{max} of the pheromone trails, the evaporation parameters ρ^a and ρ^s, the number of ants nb_{ant}, and the ratio of customers being kept to be serviced by the drones at the beginning of the solution construction phase r_d. The upper bound μ_{max} is always set to 1.0 and the lower bound μ_{min} is set to $\frac{\mu_{max}}{r_p \times n}$, where r_p is a given ratio taking the values in $\{0.1, 0.5, 1.0, 2.0, 3.0, 5.0\}$. The parameters ρ^a and ρ^s are set to one of five values $\{0.70, 0.80, 0.85, 0.90, 0.95\}$. The number of ants nb_{ant} depends on the number of nodes n in the graph and is computed as $nb_{ant} = r_a \times n$. There are three values for the ratio r_a: 0.5, 1.0, and 2.0. And finally, the ratio of selecting the customers to be serviced by the drones r_d is set to one of three values $\{0.3, 0.5, 0.7\}$. Table 1 summarizes the possible values for our parameters:

This results 900 combinations of parameters $\{r_p, \rho^a, \rho^s, r_a, r_d\}$ for the test. The training set is picked from 15 largest instances of the benchmark with 229 nodes. For each combination of the settings, the algorithm is run 10 times per instance. As shown in the following experiments, our algorithm in general provides good results in the running time of less than 200 seconds on each instance. However, because each PDSTSP instance is solved in a time limit of 300 seconds (by the algorithm of [22]) and 1,200 seconds (by the best algorithm version of [10] in term of solution quality), we decide to limit our algorithm in 300 seconds for each execution for a fair comparison. Thus, we have totally solved $900 \times 15 \times 10 = 135,000$ problems in the experiment. Two following quantitative measures are used to compare the combinations: the average relative gap to the best solutions found and the number of times that a setting generates the best solution. The obtained results show that the average gaps among settings are not significantly different. Therefore, we select the best setting based on the number of times it provides the best solutions. Using this technique, we adopt the following parameter settings: $\{r_p = 2.0, \rho^a = 0.95, \rho^s = 0.80, r_a = 1.0, r_d = 0.5\}$.

Parameter	Value	Note
μ_{max}	1.0	the upper bound of the pheromone trails
r_p	0.1, 0.5, 1.0, 2.0, 3.0, 5.0	the ratio to compute the lower bound of the pheromone trails μ_{min}
ρ^a	0.70, 0.80, 0.85, 0.90, 0.95	the evaporation parameter to handle the decision related to the drone-truck assignment
ρ^s	0.70, 0.80, 0.85, 0.90, 0.95	the evaporation parameter to handle the sequence decision in the truck tour
r_a	0.5, 1.0, 2.0	the ratio to define the number of ants nb_{ant}
r_d	0.3, 0.5, 0.7	the probability to select a part of customers serviced by the drones at the beginning of the construction solution phase

Table 1: Values used for the parameter tuning

Using the parameter setting above, we compare our algorithm with two state-of-the-art metaheuristics recently proposed in [10, 22]. In the following, we call these two algorithms as SDFGQ18 and DMN20 using the initial letters of authors' last names, respectively. We note that these algorithms were run once and only one solution was reported for each instance. To better observe the behavior of the HACO, we run our algorithm 10 times; each in 300 seconds for a fair comparison with the two reference algorithms. For each instance, four solutions of HACO are recorded for the comparison: the solution of the first run (corresponding to the single run of the algorithm), the best solution, the worst solution, and the average solution over 10 runs. The results are summarized in Table 2 and the detailed results can be found in the Appendix.

Data	SDFGQ18	DMN20	HACO					
			z_1	z_{best}	z_{worst}	$\bar{z}$	$\bar{t}$	$\#_{BKS}$
att48	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0
berlin52	0.17	0.00	0.00	0.00	0.00	0.00	2.35	0
eil101	1.44	0.05	0.01	0.00	0.28	0.19	27.51	1
gr120	1.67	3.85	1.58	0.00	2.45	0.88	61.75	7
pr152	2.41	0.87	0.40	0.00	2.19	0.87	40.48	4
gr229	8.89	3.56	0.99	0.05	1.65	0.97	168.63	11

Table 2: Comparison of HACO with existing algorithms on the benchmark instances

In the table above, the first column shows the name of the instance class. The next two columns represent the total gaps of the solutions provided by SDFGQ18 and DMN20 to the best found solutions (including ones found in this paper) on each class of instances. For the HACO algorithm, we report the gaps for the first, best, worst, and average solutions over 10 executions in Columns z_1, z_{best}, z_{worst}, and $\bar{z}$ respectively. The column $\bar{t}$ presents the average running time to reach the best solutions of the algorithm. And

finally, the column $\#_{BKS}$ reports the number of new best known solutions found by the HACO.

As can be seen in the table, the results clearly show a competitive performance of our method. The total gaps provided by our first and best solutions are at least as good as those of the other methods on all instance classes. Remarkably, we improve 23 best known solutions in total. The values in the column $\overline{z}$ are quite small (never exceeding 1%), leading to small fluctuations of the solutions over 10 executions from the best solutions. This is an indicator showing the stability of our algorithm. The worst solutions even outperform the best solutions of the other methods in many cases. In particular, they provide better total gaps than both methods SDFGQ18 and DMN20 on the largest instances with 229 customers. The running time of the algorithm is fixed to 300 seconds but it spends only 168.63 seconds on average to reach the best solutions on the largest instances. An interesting observation is that the running time to reach the best solutions on the instances pr152 is shorter than that on the instances gr120, which have smaller size. And in addition, we can improve more best known solutions on gr120 instances than on pr152 instances (7 vs 4). A further analysis is performed and it turns out that the customers in the pr152 instances distribute in clusters, possibly making the instances easier to solve.

Since the comparisons mostly conducted on average values of aggregated gaps of solutions may not be quite comprehensive, it may not clear which algorithm has the best performance. We now carry out the Kruskal–Wallis one-way analysis of variance by ranks as a non-parametric method over the obtained outcomes in terms of gaps to the best known solutions of the algorithms. The objective is to check whether the differences observed between our employed algorithm and those in the literature are statistically significant. The null hypothesis of the Kruskal–Wallis test is that the mean ranks of the groups are the same or the population medians are equal, while the alternative hypothesis is that the population medians are not all equal. In such ranking system, the smallest value is assigned to the first rank, the next smallest possesses the second rank, etc.

Data	SDFGQ18	DMN20	z_1-HACO	Chi-Square	df	Asymp. Sig.
att48	23	23	23	0	2	1.000
berlin52	24	22.5	22.5	0.130	2	0.937
eil101	27.97	21.23	**19.80**	3.307	2	0.191
gr120	22.13	27.40	**19.47**	2.834	2	0.242
pr152	28.53	21.73	**18.73**	4.385	2	0.112
gr229	32.93	22.23	**13.83**	15.938	2	0.000

Table 3: The mean ranks of groups and test statistics in Kruskal–Wallis test

For the fair comparison, the solutions obtained from the first run of HACO are selected in this experiment. In Table 3, columns SDFGQ18, DMN20, and z_1-HACO represent the mean ranks of the three methods. Three last columns report the test statistics: Chi-Square, degree of freedom (df), and p-value (labeled Asymp. Sig.). As illustrated in Table 3, our proposed HACO outperforms the others. It possesses the first rank and shows a better performance on four instance classes: eil101, gr120, pr152, and gr229. In this calculation, DMN20 has the second best outcome except on instance class gr120 where SDFGQ18 performs better. However, only on the instance class gr229 with the largest size, the p-value is less than the significance level (0.000 < 0.05) leading to the rejection of the null hypothesis. This consequence implies that there is statistically significant difference between performances of considered algorithms in this test. This can be explained by the fact that on the smaller-size instances, which are in general easier, the algorithms provide good solutions of almost the same quality. And as a result, the performance difference of our algorithm is observed more clearly on larger instances.

5 CONCLUSION

In this paper, we propose a hybrid ant colony optimization algorithm to solve the Parallel Drone Scheduling Traveling Salesman Problem, which recently receives the attention of the research community. Our algorithm combines the ideas of ant colony optimization, large neighborhood search, and local search-based metaheuristics. A new dynamic programming is proposed to split the TSP tour into a feasible PDSTSP solution in a fast and efficient way. The experiments on the benchmark instances show that our algorithm outperforms existing approaches in terms of both computation time and solution quality to become the state-of-the-art method proposed for the problem. In particular, 23 improved best known solutions have been found by our algorithm.

Our future work consists in embedding our new split procedure in the framework of other nature-inspired metaheuristics (e.g., genetic algorithm) to develop more complex algorithms and compare them with the proposed HACO. Further research effort will be devoted to adapt our HACO algorithm to solve other routing variants with drones such as the FSTSP.

APPENDIX

In the following tables, we report the detailed results for the benchmark data from [22]. In each table, the first block shows the instance settings with the information: the percentage of drone-eligible customers over the total (el), the drone speed (sp), the number of drones (#), and the depot location (dp). The second block reports the best solutions found by the two existing metaheuristics. The last block presents the objective values of solutions provided by the HACO: the solutions in the single run mode (column z_1), the best solutions (column z_{best}), the worst solutions (column z_{worst}), the average solutions (column $\overline{z}$), the average time ($\overline{t}$) and the average iterations ($\overline{iter}$) over 10 runs when the best solutions are found. The last row computes, for each instance class, the total gaps in percentage of the solutions generated by different methods to the best found solutions. The numbers in bold represent our improved best known solutions.

Instance settings				SDFGQ18	DMN20	HACO					
el	sp	#	dp			z_1	z_{best}	z_{worst}	$\bar{z}$	$\bar{t}$	$\overline{iter}$
80	2	1	1	29954	29954	29954	29954	29954	29954	0.07	2.5
80	2	1	2	33798	33798	33798	33798	33798	33798	0.14	4.5
0	2	1	1	42136	42136	42136	42136	42136	42136	0.02	1
20	2	1	1	38662	38662	38662	38662	38662	38662	0.04	2.4
40	2	1	1	31592	31592	31592	31592	31592	31592	0.03	1.8
60	2	1	1	30788.8	30788.8	30788.8	30788.8	30788.8	30788.8	0.05	2.2
100	2	1	1	27784	27784	27784	27784	27784	27784	0.18	3.8
80	1	1	1	33234	33234	33234	33234	33234	33234	0.07	2.6
80	3	1	1	29142	29142	29142	29142	29142	29142	0.07	2.7
80	4	1	1	28686	28686	28686	28686	28686	28686	0.08	3
80	5	1	1	28610	28610	28610	28610	28610	28610	0.04	1.8
80	2	2	1	28686	28686	28686	28686	28686	28686	0.07	2.5
80	2	3	1	28610	28610	28610	28610	28610	28610	0.05	1.8
80	2	4	1	28610	28610	28610	28610	28610	28610	0.02	1
80	2	5	1	28610	28610	28610	28610	28610	28610	0.02	1
Gap				0.0	0.0	0.0	0.0	0.0	0.0		

Table 4: Results of the algorithms on the instance *att48* from [22]

Instance settings				SDFGQ18	DMN20	HACO					
el	sp	#	dp			z_1	z_{best}	z_{worst}	$\bar{z}$	$\bar{t}$	$\overline{iter}$
80	2	1	1	6386.48	6386.48	6386.48	6386.48	6386.48	6386.48	10.71	260.5
80	2	1	2	7830	7830	7830	7830	7830	7830	0.09	2.4
0	2	1	1	9675	9675	9675	9675	9675	9675	0.02	1.1
20	2	1	1	9350	9350	9350	9350	9350	9350	0.02	1
40	2	1	1	8300	8300	8300	8300	8300	8300	0.02	1
60	2	1	1	7410	7410	7410	7410	7410	7410	0.05	2.3
100	2	1	1	6192	6192	6192	6192	6192	6192	23.23	327.6
80	1	1	1	7450	7450	7450	7450	7450	7450	0.09	2.6
80	3	1	1	5656.56	5656.56	5656.56	5656.56	5656.56	5656.56	0.59	16
80	4	1	1	5290.65	5290.65	5290.65	5290.65	5290.65	5290.65	0.1	3.6
80	5	1	1	5190	5190	5190	5190	5190	5190	0.04	2
80	2	2	1	5299.81	5290.65	5290.68	5290.68	5290.68	5290.68	0.15	4.5
80	2	3	1	5190	5190	5190	5190	5190	5190	0.02	1
80	2	4	1	5190	5190	5190	5190	5190	5190	0.02	1
80	2	5	1	5190	5190	5190	5190	5190	5190	0.03	1
Gap				0.17	0.0	0.0	0.0	0.0	0.0		

Table 5: Results of the algorithms on the instance *berlin52* from [22]

Instance settings				SDFGQ18	DMN20	HACO					
el	sp	#	dp			z_1	z_{best}	z_{worst}	$\bar{z}$	$\bar{t}$	$\overline{iter}$
80	2	1	1	564	564	564	564	564	564	30.38	140
80	2	1	2	650	648.98	648.98	648.98	648.98	648.98	42.08	182.4
0	2	1	1	819	819	819	819	819	819	1.89	12.9
20	2	1	1	738	736	736	736	736	736	14.77	114.6
40	2	1	1	646	646	646	646	646	646	1.75	12.5
60	2	1	1	578	578	578	578	578	578	5.28	30.1
100	2	1	1	561.41	560	560	560	560	560	71.6	150.8
80	1	1	1	650	650	650	650	650	650	6.18	26.2
80	3	1	1	504	504	504	**503.944**	504.088	504.003	78.66	381.5
80	4	1	1	456	456	456	456	456	456	4.95	25.9
80	5	1	1	420.83	421	420.833	420.833	420.833	420.833	86	468.8
80	2	2	1	456	456	456	456	456	456	46.06	217.4
80	2	3	1	395	395	395	395	396	395.7	13.73	62
80	2	4	1	346.68	346	346	346	346	346	5.38	23.7
80	2	5	1	319.74	318	318	318	318	318	3.95	17.2
Gap				1.44	0.05	0.01	0.0	0.28	0.19		

Table 6: Results of the algorithms on the instance *eil101* from [22]

Instance settings				SDFGQ18	DMN20	HACO					
el	sp	#	dp			z_1	z_{best}	z_{worst}	$\bar{z}$	$\bar{t}$	$\overline{iter}$
80	2	1	1	1414	1420.76	1414	1414	1414	1414	11.06	34.4
80	2	1	2	1730	1726	1736	1726	1740	1729.6	111.66	318.1
0	2	1	1	2006	2006	2006	2006	2006	2006	5.45	24.5
20	2	1	1	1736	1736	1736	1736	1736	1736	19.38	88
40	2	1	1	1624	1624	1624	1624	1624	1624	22.22	90.7
60	2	1	1	1494	1494	1494	1494	1494	1494	11.04	41.8
100	2	1	1	1414.8	1416	1414	**1414**	1414	1414	25.33	32.7
80	1	1	1	1592	1592	1592	1592	1597	1593.4	79.94	233.2
80	3	1	1	1289.27	1291	1286.16	**1284.74**	1288	1287.49	44.6	146.3
80	4	1	1	1189.71	1192	1186	**1186**	1186	1186	36.22	126
80	5	1	1	1112	1114	1113.23	1112	1113.23	1112.25	120.01	431.6
80	2	2	1	1188.51	1197	1186	**1186**	1186.13	1186.01	71.25	228.5
80	2	3	1	1044.65	1050	1044	**1044**	1045.7	1044.17	76.37	231.2
80	2	4	1	946.04	946.04	946	**943**	946	944.177	153.55	434.2
80	2	5	1	880	881	883	**878.91**	883	880.749	138.12	330.3
Gap				1.67	3.85	1.58	0.0	2.45	0.88		

Table 7: Results of the algorithms on the instance *gr120* from [22]

Instance settings				SDFGQ18	DMN20	HACO					
el	sp	#	dp			z_1	z_{best}	z_{worst}	$\bar{z}$	$\bar{t}$	$\overline{iter}$
80	2	1	1	76008	76008	76008	76008	76008	76008	53.01	88.7
80	2	1	2	76556	76556	76556	76556	76556	76556	2.1	3.7
0	2	1	1	86596	86596	86596	86596	86596	86596	1.69	4.2
20	2	1	1	82504	82504	82504	82504	82504	82504	1.83	4.6
40	2	1	1	77372	77316	77236	**77236**	77316	77252	73.83	169.1
60	2	1	1	76786	76786	76786	76786	76786	76786	12.56	25.1
100	2	1	1	74468	74302	74560	74302	74560	74385.3	81.88	49
80	1	1	1	80164	79952	79992	79952	79992	79980	47.67	77.7
80	3	1	1	72936	72936	72936	72936	72936	72936	68.1	118.2
80	4	1	1	70412	70328	70148	**70148**	70148	70148	20.59	36.8
80	5	1	1	67798	67798	67798	67798	67858	67846	18.55	34.2
80	2	2	1	70244	70405.45	70148	**70148**	70160	70149.2	54.53	93
80	2	3	1	65062.1	64720.3	64626	**64626**	65648	65034.8	79.16	131.8
80	2	4	1	60027.4	59772	59772	59772	59772	59772	56.8	90.8
80	2	5	1	56336.1	56262	56262	56262	56262	56262	34.9	48.2
Gap				2.41	0.87	0.40	0.0	2.19	0.87		

Table 8: Results of the algorithms on the instance *pr152* from [22]

Instance settings				SDFGQ18	DMN20	HACO					
el	sp	#	dp			z_1	z_{best}	z_{worst}	$\bar{z}$	$\bar{t}$	$\overline{iter}$
80	2	1	1	1794.84	1785.86	1785.24	**1781.16**	1785.94	1784.36	172.87	99.8
80	2	1	2	1913.74	1911.58	1911.58	1911.58	1911.58	1911.58	144.34	78.6
0	2	1	1	2020.16	2017.24	2017.24	2017.24	2017.24	2017.24	38.37	29
20	2	1	1	1862.76	1860.14	1860.14	1860.14	1860.14	1860.14	73.67	55.9
40	2	1	1	1828.02	1827.02	1824.8	**1824.8**	1825.92	1825.37	172.57	128.5
60	2	1	1	1807.5	1797.37	1798.12	**1796.9**	1798.12	1797.5	141.84	92.3
100	2	1	1	1498.05	1496.29	1496.97	1496.97	1497.45	1497.03	225.9	35.7
80	1	1	1	1865	1863.12	1861.98	**1861.98**	1863.16	1862.87	184.17	102
80	3	1	1	1735.16	1725.45	1720.16	**1717.92**	1721.88	1720.23	194.74	115.3
80	4	1	1	1679.33	1675.82	1669.5	**1668.08**	1669.5	1668.66	214.61	132.7
80	5	1	1	1642.04	1629.38	1623.76	**1623.06**	1623.84	1623.59	196.6	125.4
80	2	2	1	1686.75	1673.72	1669.12	**1668.34**	1670.3	1669.28	209.51	124.2
80	2	3	1	1603.9	1592.52	1584.84	**1582.92**	1585.64	1584.22	198.95	116.7
80	2	4	1	1518.62	1526.92	1517.84	**1516.72**	1519.92	1518.72	191.87	102.9
80	2	5	1	1483.68	1467.76	1467.12	**1464.98**	1468.62	1467.22	169.46	74.9
Gap				8.89	3.56	0.99	0.05	1.65	0.97		

Table 9: Results of the algorithms on the instance *gr229* from [22]

REFERENCES

[1] 2013. Amazon.com, Inc. Amazon prime air. http://www.amazon.com/primeair.

[2] 2014. DHL International GmbH. DHL parcelcopter launches initial operations for research purposes. http://www.dhl.com/en/press/releases/releases_2014/group/dhl_parcelcopter_launches_initial_operations_for_research_purposes.html.

[3] 2015. Leo Kelion. Alibaba begins drone delivery trials in China. http://www.bbc.com/news/technology-31129804.

[4] 2016. Andrew Liptak. 7-Eleven just made the first commercial delivery by drone. http://www.theverge.com/2016/7/23/12262468/7-11-first-retailer-deliver-food-drone..

[5] 2016. Rodrique Ngowi. UPS testing drones for use in its package delivery system. http://phys.org/news/2016-09-ups-drones-package-delivery.html.

[6] John E. Bell and Patrick R. McMullen. 2004. Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics* 18, 1 (2004), 41–48.

[7] Maurizio Boccia, Adriano Masone, Antonio Sforza, and Claudio Sterle. 2021. A column-and-row generation approach for the flying sidekick travelling salesman problem. *Transportation Research Part C: Emerging Technologies* 124, 102913 (2021).

[8] Bernd Bullnheimer, Richard F. Hartl, and Christine Strauss. 1999. *Applying the ANT System to the Vehicle Routing Problem.* Springer US, Boston, MA, 285–296.

[9] Júlia Cária de Freitas and Puca Huachi Vaz Penna. 2020. A variable neighborhood search for flying sidekick traveling salesman problem. *International Transactions in Operational Research* 27, 1 (2020), 267–290.

[10] Mauro Dell'Amico, Roberto Montemanni, and Stefano Novellani. 2020. Matheuristic algorithms for the parallel drone scheduling traveling salesman problem. *Annals of Operations Research* 289, 2 (2020), 211–226.

[11] Mauro Dell'Amico, Roberto Montemanni, and Stefano Novellani. 2021. Modeling the flying sidekick traveling salesman problem with multiple drones. *Networks* forthcoming (2021).

[12] Marco Dorigo, Gianni Di Caro, and Luca Maria Gambardella. 1999. Ant Algorithms for Discrete Optimization. *Artificial Life* 5, 2 (1999), 137–172.

[13] Marco Dorigo and Luca Maria Gambardella. 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evolutionary Computation* 1, 1 (1997), 53–66.

[14] Dong Do Duc, Huy Q. Dinh, and Huan Hoang Xuan. 2008. On the Pheromone Update Rules of Ant Colony Optimization Approaches for the Job Shop Scheduling Problem. In *Intelligent Agents and Multi-Agent Systems, 11th Pacific Rim International Conference on Multi-Agents, PRIMA 2008, Hanoi, Vietnam, December 15-16, 2008. Proceedings (Lecture Notes in Computer Science)*, The Duy Bui, Tuong Vinh Ho, and Quang-Thuy Ha (Eds.), Vol. 5357. Springer, 153–160.

[15] Dong Do Duc, Sy Vinh Le, and Huan Hoang Xuan. 2013. ACOHAP: an efficient ant colony optimization for the haplotype inference by pure parsimony problem. *Swarm Intelligence* 7, 1 (2013), 63–77.

[16] Quang Minh Ha, Yves Deville, Pham Quang Dung, and Minh Hoàng Hà. 2020. A hybrid genetic algorithm for the traveling salesman problem with drone. *Journal of Heuristics* 26, 2 (2020), 219–247.

[17] Quang Minh Ha, Yves Deville, Quang Dung Pham, and Minh Hoàng Hà. 2018. On the min-cost traveling salesman problem with drone. *Transportation Research Part C: Emerging Technologies* 86 (2018), 597–621.

[18] S. Lin. 1965. Computer solutions of the traveling salesman problem. *The Bell System Technical Journal* 44, 10 (1965), 2245–2269.

[19] Max Manfrin, Mauro Birattari, Thomas Stützle, and Marco Dorigo. 2006. Parallel Ant Colony Optimization for the Traveling Salesman Problem. In *Ant Colony Optimization and Swarm Intelligence*, Marco Dorigo, Luca Maria Gambardella, Mauro Birattari, Alcherio Martinoli, Riccardo Poli, and Thomas Stützle (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 224–234.

[20] Chase C Murray and Amanda G Chu. 2015. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies* 54 (2015), 86–109.

[21] Chase C. Murray and Ritwik Raj. 2020. The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones. *Transportation Research Part C: Emerging Technologies* 110 (2020), 368–398.

[22] Raïssa G. Mbiadou Saleu, Laurent Deroussi, Dominique Feillet, Nathalie Grangeon, and Alain Quilliot. 2018. An iterative two-step heuristic for the parallel drone scheduling traveling salesman problem. *Networks* 72, 4 (2018), 459–474.

[23] Paul Shaw. 1998. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *Principles and Practice of Constraint Programming - CP98, 4th International Conference, Pisa, Italy, October 26-30, 1998, Proceedings (Lecture Notes in Computer Science)*, Michael J. Maher and Jean-Francois Puget (Eds.), Vol. 1520. Springer, 417–431.

[24] Thomas Stützle and Marco Dorigo. 1999. ACO algorithms for the Traveling Salesman Problem. In *Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications*, M. M. Mkel Jacques Périaux K. Miettinen, Pekka Neittaanmki (Ed.). John Wiley & Sons, 163–183.

[25] Thomas Stützle and Holger H. Hoos. 2000. MAX-MIN Ant System. *Future Generation Computer Systems* 16, 8 (2000), 889–914.