Journal of Computer Science and Cybernetics, V.37, N.4 (2021), 465–493
DOI 10.15625/1813-9663/37/4/16180

THE TRAVELING SALESMAN PROBLEM WITH MULTI-VISIT DRONE

QUANG MINH HA[1], DUY MANH VU[1], XUAN THANH LE[2], MINH HOANG HA[1,*]

[1]*ORLab, Faculty of Computer Science, Phenikaa University, Nguyen Trac Street, Yen Nghia Ward, Ha Dong District, Ha Noi, Viet Nam*
[2]*Institute of Mathematics, Vietnam Academy of Science and Technology, 18 Hoang Quoc Viet Street, Cau Giay District, Ha Noi, Viet Nam*

Abstract. This paper deals with the Traveling Salesman Problem with Multi-Visit Drone (TSP-MVD) in which a truck works in collaboration with a drone that can serve up to $q \geq 1$ customers consecutively during each sortie. We propose a Mixed Integer Linear Programming (MILP) formulation and a metaheuristic based on Iterated Local Search (ILS) to solve the problem. Benchmark instances collected from the literature of the special case with $q = 1$ are used to test the performance of our algorithms. The obtained results show that our MILP model can solve a number of instances to optimality. This is the first time optimal solutions for these instances are reported. Our ILS performs better than other algorithms in terms of both solution quality and running time on several instance classes. The numerical results obtained by testing the methods on new randomly generated instances show again the effectiveness of the methods as well as the positive impact of using the multi-visit drone.

Keywords. Traveling salesman problem, multi-visit drone, mixed Integer linear programming, iterated local search.

1. INTRODUCTION

Recently, routing problems using the combination of trucks and drones to deliver services have received much attention. In the notable work of [1], the authors introduced a new transportation system in which a truck and a drone (an unmanned aerial vehicle – UAV – used for commercial purposes) are coupled to deliver parcels to customers. Being able to be launched from the truck, the drone then flies to a single customer location for the delivery before returning to the rendezvous point where it will be retrieved by the truck and being prepared for the next launches. The objective function considered in the problem is to minimize the completion time of both vehicles, i.e. the latest time both vehicles return to a depot. The problem was named as "Flying Sidekick Traveling Salesman Problem with

Dedicated to Professor Phan Dinh Dieu on the occasion of his 85th birth anniversary.

*Corresponding author.
E-mail addresses: manh.vuduy@phenikaa-uni.edu.vn(D.M. Vu); lxthanh@math.ac.vn(X.T. Le); hoang.haminh@phenikaa-uni.edu.vn(M.H. Ha) .

Drone" (FSTSP) and has opened a new research direction on routing problems with trucks and drones.

A different variant of the problem was introduced in [2]. Although the objective function of both variants is to minimize the completion time of two vehicles, the problem in the latter work assumed a different hypothesis where the launch and rendezvous points could be at the same location (meaning that the truck could launch the drone and then stay to wait for its return). In [3], the authors presented a similar problem but considered a different objective function which minimizes the total transportation cost. They called the problem as min-cost Traveling Salesman Problem with Drone (TSP-D) and the original variant FSTSP as min-time TSP-D. This paper also proposed a metaheuristic based on the Greedy Randomized Adaptive Search Procedure (GRASP) to efficiently solve both variants.

Shortly after, Carlsson and Song [4] delivered a research in which the authors also evaluated the efficiency of the TSP-D by providing numerous theoretical analysis as well as the computational results. The authors also named the problem the "horsefly routing problem" with the objective to minimize the completion time of the vehicles, which is similar to the variant proposed by [1] except that the number of drones is not strictly limited to one, but is generalized to become a parameter of the problem. Additionally, the study of the upper and lower bounds for the completion time of the TSP-D is also conducted. For the solution approach, a metaheuristic is proposed in which TSP-D solutions are iteratively built starting from an optimal TSP solution. From this work, the authors concluded that the improvement in completion time due to the use of drone is proportional to the square root of the ratio of the speeds of the truck and the drone.

In [5], the authors proposed a hybrid heuristic named HGVNS to solve two TSP-D variants proposed in [2, 1] with the min-time objective. In detail, HGVNS first creates an initial solution which is the optimal TSP tour by using a MILP solver and then applies a heuristic in which some customers on the truck tours are removed and reinserted as drone customers. Next, the initial solution is used as the input for a general variable neighbourhood search in which eight neighbourhoods are shuffled and chosen randomly. The authors conducted the experiments on three instance sets from [2, 6] and TSPLIB. The computational results show that the proposed approach can decrease delivery time by up to 67.79%.

In [7], a hybrid genetic algorithm (HGA) was proposed to solve the TSP-D problems with both min-time and min-cost objectives. The authors proposed a hybrid genetic search with dynamic population management and adaptive diversity control based on a split algorithm, problem-tailored crossover and local search operators, a new restore method to advance the convergence and an adaptive penalization mechanism to dynamically balance the search between feasible/infeasible solutions. The computational results showed that the new algorithm outperforms two existing heuristic methods of [3, 1] in terms of solution quality and improves many best known solutions found in the literature. Recent exact methods to solve the FSTSP can be found in [8, 9, 10].

Apart from the works that investigate the original TSP-D variant where one truck and one drone are used for delivery, there exists many other variants of the TSP-D as well as its generalization – the Vehicle Routing Problem with Drone (VRP-D) – that have been studied in many other researches. In [11], the authors extended the original work [1] – the FSTSP – to introduce a new problem called "The multiple Flying Sidekick Traveling Salesman Problem" (mFSTSP) where multiple drones are used with a truck to make the

deliveries. Again, the authors formulated the problem with a mixed integer programming formulation and proposed a three-phase heuristic to solve the problem with various size up to 100 customers. The computational results showed that this variant can improve the truck-only delivery problem (the TSP) by up to 23%.

The extension of TSP-D to the mTSP-D was introduced in the work of [12] where a fleet of trucks and drones are used for delivery. However, instead of fixing the pair of truck and drone (i.e. a truck only carries and works with one identical drone, not others), the problem allows the drones to return to any available trucks. The authors presented a mixed integer programming model for the mTSP-D and developed a heuristic called "Adaptive Insertion Heuristic" (ADI) to solve the problem. The computational results showed that there is a potential gain for the mTSP-D comparing to the original TSP-D.

In [13], the authors introduced a more general problem that copes with multiple trucks and drones under the objective function of minimizing the completion time. The authors named the problem "The vehicle routing problem with drone" (VRP-D) and conducted the analysis on several worst-case scenarios, from which they proposed bounds on the best possible savings in time when using drones and trucks instead of trucks alone. A further development of this research was studied in [14] where the authors extended the worst-case bounds to more generic distance/cost metrics as well as explicitly considered the limitation of battery life and cost objectives.

More recently, the authors of [15] introduced a further work of the VRP-D by proposing a mixed integer programming model for the problem (an arc-based model) to develop a branch-and-price algorithm. The experiment was run and conducted on 10 and 15-customer instances. The results showed that by using MILP solver with the proposed branch-and-price algorithm, the authors can deliver the optimal solutions of these instances in the average of 7 minutes and 4 hours for two types of instances above. Furthermore, it was reported that a VRP-D solution could save up to 20% of the cost comparing to the traditional VRP.

In all the studies mentioned above, a drone is supposed to visit a single customer during a flight. This limitation would be broken in very near future as shown in [16] where the authors studied a new variant of the TSP-D called the "Multi-visit drone routing problem". In this work, instead of being limited to carry one parcel at a time, the drone is able to carry and visit multiple customer locations before returning to the truck for recharging/taking new parcels. Additionally, the authors also considered that the drone has a fixed energy capacity instead of a fixed flight time to cope with the variety of parcels' weight. A descriptive model and various theoretical analysis were proposed. For solving the problem more effectively, a local search method was introduced using a Swap-based operator. Extensive computational results have been conducted to analyse the performance and sensitivity of the factors (i.e. drone speed and battery). Finally, the authors also extended this model to cope with more than one drone, which leads to a more general problem.

A recent work [17] proposed a similar variant of the min-cost TSP-D where a drone is able to service multiple customers before returning to the truck. It was assumed that the truck must reach the rendezvous point before the drone for security reasons. In addition, as the problem considered in [16], the authors considered the effect of varying payload on energy consumption and required that the truck cannot visit multiple customers while the drone performs its flight. They proposed a two-stage, route-based modeling approach to optimize both truck's main route and the drone's flying schedule. To construct the initial

solution, a hybrid heuristic combining a nearest-neighbor and cost-saving strategies was used. To improve this initial solution, the author applied the Simulated Annealing (SA) metaheuristic. Moreover, an experiment was conducted using randomly generated customer locations as well as a practical road network of in Changsha, China. The computational results analysed the overall performance of the SA as well as the sensitivity of different factors such as the ratio of light parcels, the drone's capacity, and drone endurance.

In this research, we study a more general variant of the FSTSP in [1] in which the drone can deliver several parcels in a flight. We call the problem as the Traveling Salesman Problem with Multi-Visit Drone (TSP-MVD). Compared with the problems considered in [17, 16], our problem are more general when we allow the truck can visit multiple customers when the drone flies in the air. We also consider that the launching and retrieving times of the drone are significant, leading to more complex synchronization operations. However, to simplify the model, we do not take into account the effect of parcels' load on energy consumption of the drone. Our contributions in this paper are as follows:

- We propose a new MILP model for a TSP-D variant in which the drone can deliver several parcels during a flight, from which small-size instances can be solved to optimality.
- We propose an Iterated Local Search (ILS) that could effectively solve the TSP-MVD. Our algorithm include several problem-tailored components such as: (i) a solution representation based on a new split method to transform a TSP tour to a TSP-MVD solution optimally (given the relative orders of the nodes are fixed), (ii) four new local search operators to specifically handle the "multi-visit" characteristic of the drone, and (iii) adapted perturbation operators.
- We conduct several experiments to test the performance of our proposed methods. Our MILP model can find optimal solutions for a number of open benchmark instances in the literature while our ILS can perform better when compared with existing algorithms on several class of instances in terms of solution quality and running time. The impact of the limit of drone-visits on the TSP-MVD solutions is also conducted to investigate the benefit of using the multi-visit drone.

This paper is organized as follows. After this introduction, Section 2 gives the precise description of the TSP-MVD. In Section 3, we construct a mixed integer linear programming (MILP) formulation for the TSP-MVD. Section 4 discusses our Iterated Local Search (ILS) algorithm to solve this problem. Numerical experiments are presented in Section 5. Finally, Section 6 closes this paper with some conclusions.

2. PROBLEM DESCRIPTION

In this section we give a precise description of the TSP-MVD. We are given n different customer locations, each one must be served exactly once by either a truck (driver-operated vehicle) or a drone (unmanned aerial vehicle) operating in coordination with the truck. The truck and the drone must depart from a single depot (distribution center), and return to that depot only when all the customers are served. Although the physical location of the depot exists uniquely, for the sake of distinguishing the departure and arrival events of the vehicles at the depot, we assign to it two node numbers 0 and $n+1$. The two vehicles depart from the depot at node 0 and return to the depot at node $n+1$. Let $V = \{0, 1, \ldots, n+1\}$ be

the set of nodes representing the customer locations (numbered from 1 to n) and the depot together with its duplication (numbered by 0 and $n+1$).

At the depot, the two vehicles can depart or return either independently or in tandem. The drone may make multiple sorties during the delivery service. A drone sortie starts by launching the drone either at the depot or from a customer location. During a drone sortie, both the truck and the drone can visit multiple nodes to deliver parcels to corresponding customers. However, due to some technical characteristics, the drone payload and flight endurance are restricted, hence the number of drone visits in each sortie is limited. We assume that the maximum number of customer locations that can be served by the drone in each sortie is q ($q \geq 1$). We assume furthermore that the drone cannot reconnect with the truck at some intermediate location, hence a drone sortie must end at either the depot or a customer location where the drone is retrieved by the truck. We can represent a drone sortie as a 3-tuple $\langle i, J, k\rangle$ in which i is the launch node where the truck releases the drone for delivering, J is a sequence of at most q nodes corresponding to the customers served by the drone in the sortie, and k is the rendezvous node where the drone returns to the truck. Since the truck also visits the launch node and the rendezvous node in each sortie of the drone, the customers at these nodes are better served by the truck driver. Therefore, for a drone sortie $\langle i, J, k\rangle$ we have $i \notin J, k \notin J$, $i \neq k$, $|J| \leq q$, the nodes in J are served by the drone, while the nodes i and k are served by truck if they are not depot nodes.

Before each sortie, the drone is loaded with parcels corresponding to some customers, and its battery is changed if needed. If the drone is transported by the truck between two consecutive sorties, these actions can be done on the truck to save time. In that case, the drone can be launched right after its travel to the new launch node. However, if at the rendezvous node of a sortie the drone is launched again for a new sortie, then it requires a service time s_L before launching to load parcels and to change battery. A service time s_R is required for the truck driver to retrieve the drone in the end of each drone sortie. At the rendezvous nodes of the drone sorties, two vehicles are required to wait for each other. While waiting for the truck before being retrieved, the drone is assumed to be in a constant flight. Furthermore, due to the limited battery capacity, the drone has an endurance e which is its maximum operating time without charging. The drone endurance must be enough for each drone sortie $\langle i, J, k\rangle$, i.e., the total travel time of the drone from the launch node i through the customer nodes in J to the rendezvous node k and the time of retrieving the drone at node k cannot exceed e. The drone must be retrieved before it runs out of battery, therefore the total travel time of the truck during each drone sortie $\langle i, J, k\rangle$ and its time to retrieve the drone at node k cannot exceed the drone endurance e.

Due to some technical reasons, not all customer requests can be fulfilled by the drone. For examples, some parcels exceed the drone payload capacity, some large-size parcels cannot be handled by the drone, some parcels require customer signature, or some customer locations are not safe for landing the drone. Therefore, not all customer nodes can be visited by the drone. We denote $V_D \subseteq \{1, \ldots, n\}$ the set of customer nodes that can be served by the drone.

While not in a sortie, the drone is carried and recharged by the truck. If the drone is retrieved by the truck at some customer node k, it may be re-launched from that node. However, once the drone is launched from a customer node i, it must not return to that node to be retrieved by the truck. Moreover, the truck may not revisit any customer nodes to

retrieve the drone. In this context, neither the truck nor the drone may revisit any customers.

The objective of the TSP-MVD is to minimize the completion time of both vehicles (i.e., the time required to serve all customers and return both vehicles to the depot at node $n + 1$). It is worth noting that the TSP-MVD can be considered as an extension of the TSP-D studied in [1]. In the TSP-D, the drone is restricted to serve exactly one customer in each of its sorties, which corresponds to the case $q = 1$ in our setting. In our TSP-MVD, we allow the drone to deliver parcels to multiple customers before returning to the truck.

In the next section we propose a mixed integer programming formulation for the TSP-MVD. For the mathematical formulation of the problem, we introduce some more notations and concepts. Let V_0 be the set of nodes from which the vehicles may depart, i.e., $V_0 = \{0, 1, \ldots, n\}$. Let V_+ be the set of nodes to which the vehicles may arrive, i.e., $V_+ = \{1, \ldots, n + 1\}$. Let τ_{ij} (resp., τ'_{ij}) be the time required for the truck (resp., the drone) to travel from node $i \in V_0$ to node $j \in V_+$ if the vehicle can visit these nodes. The drone is *active* at a node if the node belongs to some sortie of the drone, otherwise it is called *inactive* at that node. A node is called an *operation node* if it is a launch node or an rendezvous node of the drone. Under this setting, for a drone sortie $\langle i, J, k\rangle$, the nodes i and k are operation nodes, and the drone is active at all nodes in $J \cup \{i, k\}$.

3. MIXED INTEGER LINEAR PROGRAMMING FORMULATION

We represent a feasible TSP-MVD solution as the union of its *components*. Each component consists of either the path of the truck together with the path of the drone in a drone sortie, or the path of the truck carrying the drone on when the drone is not in a sortie. For simplicity, we label each component by the name of its starting node. In Example 1 we give a detail description of a solution to a TSP-MVD instance and its components.

Example 1.

Figure 1 illustrates a solution to a TSP-MVD with 10 customer nodes (labeled from 1 to 10) and a depot (represented by node 0 and node 11). In this solution, the truck follows the paths represented by continuous arrows, while the drone follows the paths represented by dashed arrows. The solution is composed of five components labeled by their starting nodes (0, 2, 6, 7, 9). In component 0, the truck starts from the depot node 0 and comes to serve customers at nodes 1 and 2 while carrying the drone. In component 2, both vehicles start at node 2 which is the starting node of this component. The drone is launched from node 2, then comes to serve nodes 4 and 5 before flying to node 6 to be retrieved there. The truck starts at node 2, then come to serve nodes 3 and 6, and retrieves the drone at node 6. Node 6 in turn is the starting node of component 6, which consists of only one continuous arrow from node 6 to node 7. That means in component 6 the truck carries the drone and comes directly from node 6 to node 7 to serve the customer there. From node 7, which is the starting node of component 7, the truck goes directly to serve customer 9, while the drone is launched to serve node 8 before coming to be retrieved at node 9. At this node, the drone is launched again to join component 9. It flies to serve the customer at node 10 before returning to the depot at node 11. In this component, the truck goes directly from node 9 to the depot. In total, the drone makes three sorties during the delivery service. Its first sortie belongs to component 2 and consists of the drone path 2–4–5–6. The second sortie belongs to component 7 and consists of the drone path 7–8–9. The last drone sortie is in component

9, in which the drone starts from node 9, then coming to serve customer 10 before returning to the depot. In the solution, the operation nodes are 2, 6, 7, 9, 11, while the drone is active at nodes 2, 4, 5, 6, 7, 8, 9, 10, 11.

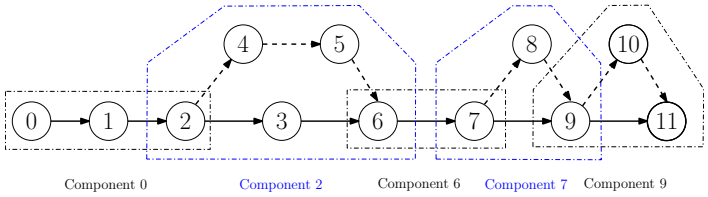

Figure 1: A solution to a TSP-MVD with its components

3.1. Determining components' nodes

To determine the components of a solution to TSP-MVD, we introduce binary variables $z_k (k \in V_0)$ in which

$$z_k = \begin{cases} 1 & \text{if node } k \in V_0 \text{ is the starting node of a component,} \\ 0 & \text{otherwise.} \end{cases}$$

Since the truck must start from the depot for its first departure, node 0 must be the starting node of a component, hence by the variable definition we have

$$z_0 = 1. \tag{1}$$

Apart from the depot node 0, if a node starts a component, then it is also the last node of the previous component. To unify this structure on the whole solution, we impose

$$z_{n+1} = 1 \tag{2}$$

to indicate that the depot node $n+1$ is the last node of the last component.

To determine which node belongs to which component and is served by which vehicle, we introduce two sets of binary variables x_{ki} and x'_{ki} $(k \in V_0, i \in V)$ in which

$$x_{ki} = \begin{cases} 1 & \text{if node } i \in V \text{ belongs to component } k \text{ and is served by the truck,} \\ 0 & \text{otherwise,} \end{cases}$$

$$x'_{ki} = \begin{cases} 1 & \text{if node } i \in V \text{ belongs to component } k \text{ and the drone is active at } i, \\ 0 & \text{otherwise.} \end{cases}$$

Note that, by definition, the drone may be active at the starting node and the last node of a component but inactive at the other nodes of the component. This corresponds to the situation in which this component is between two consecutive drone sorties. In spirit of the component concept, each node must belong to at least one component to guarantee that it is served. A node can belong up to two components when it is a customer node at which the drone is launched or retrieved. Since the depot node 0 cannot belong to any component other than component 0, we have

$$x_{k0} = 0 \qquad \forall k \in V_0 \backslash \{0\} \tag{3}$$

$$x'_{k0} = 0 \qquad \forall k \in V_0 \backslash \{0\} \tag{4}$$

Since the delivery service stops at the depot node $n+1$, this last node must belong to exactly one component. This fact can be represented by

$$\sum_{k \in V_0} x_{k,n+1} = 1. \tag{5}$$

By definition of solution components, a component starts at the node of the same label. Furthermore, if the node is not the depot, then it must be an operation node. Therefore we have

$$x_{kk} = z_k \quad \forall k \in V_0, \tag{6}$$

$$x'_{kk} = z_k \quad \forall k \in V_0 \backslash \{0\}. \tag{7}$$

For the customer nodes we impose

$$z_i + x_{ki} - x'_{ki} \leq 1 \quad \forall i \in V_0 \backslash \{0\}, k \in V_0 \backslash \{i\}, \tag{8}$$

$$z_i + x'_{ki} - x_{ki} \leq 1 \quad \forall i \in V_0 \backslash \{0\}, k \in V_0 \backslash \{i\}, \tag{9}$$

$$\sum_{k \in V_0 \backslash \{i\}} x_{ki} \leq 1 \quad \forall i \in V_0 \backslash \{0\}, \tag{10}$$

$$\sum_{k \in V_0 \backslash \{i\}} x'_{ki} \leq 1 \quad \forall i \in V_0 \backslash \{0\}, \tag{11}$$

$$\sum_{k \in V_0 \backslash \{i\}} x_{ki} + \sum_{k \in V_0 \backslash \{i\}} x'_{ki} - z_i = 1 \quad \forall i \in V_0 \backslash \{0\}. \tag{12}$$

These constraints cover the following requirements on the customer nodes.

- If the customer node $i \in V_0 \backslash \{0\}$ is not an operation node (i.e., $z_i = 0$), then it must belong to exactly one component. This is guaranteed by (10)–(12), since they imply that either $\sum_{k \in V_0 \backslash \{i\}} x_{ki} = 1$ or $\sum_{k \in V_0 \backslash \{i\}} x'_{ki} = 1$, that respectively means that the node is served by either the truck or the drone in some component $k \neq i$.
- If the customer node $i \in V_0 \backslash \{0\}$ is an operation node, then it must be the starting node of component label i (i.e., $z_i = 1$). This node must also be the last node of some component $k^* \neq i$. The synchronization of the two vehicles at this node means that $x_{k^*i} = x'_{k^*i} = 1$, while $x_{ki} = x'_{ki} = 0$ for other components $k \notin \{i, k^*\}$. This synchronization is ensured by constraints (8), (9), together with (12).

Since the drone can only serve the customer nodes within the set V_D, the other customer nodes must be served by the truck. Keeping in mind that each node belongs to at least one component, this is ensured by imposing

$$\sum_{k \in V_0} x_{ki} \geq 1 \quad \forall i \in V \backslash V_D. \tag{13}$$

To completely determine the node members of components, we have the following additional constraints

$$\sum_{i \in V} x'_{ki} \leq (q+2) z_k \quad \forall k \in V_0, \tag{14}$$

$$\sum_{i \in V} x_{ki} \leq (n+2) z_k \quad \forall k \in V_0. \tag{15}$$

These constraints ensure that if k is not the starting node of a component, then no node belongs to a component starting from node k. Note that the constraints (14) guarantee furthermore that the drone can serve at most q customer locations during a sortie.

3.2. Determining vehicles' paths in components

After determining the node members of components, our next step is to determine the order of each node in the vehicles' paths constituting the components. We say that an arc (i, j) is a *truck arc* if the truck travels from node i to node j. Similarly, we say that (i, j) is a *drone arc* if the drone flies from node i to node j. We introduce two more sets of binary variables y_{kij} and y'_{kij} $(k \in V_0, i \in V_0, j \in V_+, i \neq j)$ in which

$$y_{kij} = \begin{cases} 1 & \text{if } (i,j) \text{ is a truck arc in component } k, \\ 0 & \text{otherwise,} \end{cases}$$

$$y'_{kij} = \begin{cases} 1 & \text{if } (i,j) \text{ is a drone arc in component } k, \\ 0 & \text{otherwise.} \end{cases}$$

To eliminate sub-tours in the paths of the two vehicles, we introduce non-negative continuous variables $u_i (i \in V)$. Using these variables, we can determine the vehicles' paths in the components as follows. Each component contains exactly one truck path. The structure of the truck path in each component is satisfied by imposing the following constraints.

$$\sum_{i \in V_0 \setminus \{j\}} y_{kij} = x_{kj} \quad \forall k \in V_0, j \in V_+ \setminus \{k\}, \tag{16}$$

$$\sum_{j \in V_+ \setminus \{i,k\}} y_{kij} \geq x_{ki} - z_i \quad \forall k \in V_0, i \in V_0 \setminus \{0,k\}, \tag{17}$$

$$2 \sum_{j \in V_+ \setminus \{i,k\}} y_{kij} \leq x_{ki} - z_i + 1 \quad \forall k \in V_0, i \in V_0 \setminus \{0,k\}, \tag{18}$$

$$\sum_{i \in V_0 \setminus \{k\}} y_{kik} = 0 \quad \forall k \in V_0, \tag{19}$$

$$\sum_{i \in V_+ \setminus \{k\}} y_{kki} = z_k \quad \forall k \in V_0, \tag{20}$$

$$u_i - u_j + (n+1) y_{kij} \leq n \quad \forall k \in V_0, i \in V_0, j \in V_+, i \neq j, \tag{21}$$

$$u_i - u_j + (n+1) y'_{kij} \leq n \quad \forall k \in V_0, i \in V_0, j \in V_+, i \neq j, \tag{22}$$

$$u_i \geq 0 \quad \forall i \in V, \tag{23}$$

$$u_i \leq n+1 \quad \forall i \in V. \tag{24}$$

More precisely, by constraints (16), if node j is served by the truck in component k but is not the starting node of the component, then there must be exactly one truck arc from another node in the component going to this node. Otherwise, if node j is not in component k, then there is no arc in this component going to j. By constraints (17) and (18), if node i is served by the truck in component k but is not the first or the last node (i.e., $z_i = 0$ and $x_{ki} = 1$), then we have $\sum_{j \in V_+ \setminus \{i,k\}} y_{kij} = 1$, or equivalently, there must be exactly one truck arc going out node i. If i is the last node of component $k \neq i$ (i.e., $x_{ki} = 1$ and $z_i = 1$), then it follows from (18) that there is no truck arc in the component going out this node. Constraints (19) imply that within each component there is no truck arc going to the starting node. By constraints (20), if k is the starting node of a component (i.e., $z_k = 1$), then there must be exactly one truck arc in this component going out this node. Otherwise, if there is no component with label k (i.e., $z_k = 0$), then there is no truck arc in such empty

component. Constraints (21)-(24) eliminate sub-tours of the truck and the drone since they imply that no node is revisited by the vehicles.

Unlike the truck, in each component there are two different situations for the drone: either there exists exactly one drone path, or there is no drone path at all. In the former situation, the drone flies in one path from the starting node to the last node of the component. In the latter situation, the drone is active only at the operation nodes of the component: it is retrieved at the starting node, then inactive when carried by the truck, and launched again at the last node of the component. To cover these cases and the structure of the drone path (if exists) in each component, the following conditions must be satisfied.

(D1) If j is not a node in component k, then no drone arc in component k comes to j. This also ensures that for each node there is at most one drone arc going to.

(D2) If the drone is not active at node j, then there is no drone arc going to j.

(D3) If the drone serves node j, then there must be exactly one drone arc going to j and one drone arc going out of j. Furthermore, these two drone arcs must be in the same component.

(D4) If the drone is launched at node k, then there must be one drone arc in component k going out of k, and no drone arc in this component going to k.

(D5) If the drone is retrieved at node j of component k, then this must be the last node of the component (i.e., no drone arc in this component going out of j).

(D6) In each sortie, the drone must serve at least one customer.

(D7) The drone must be launched right after a component without drone sortie.

To ensure the above conditions, we impose the following constraints.

$$\sum_{i\in V_0\setminus\{j\}} y'_{kij} \le x'_{kj} \quad \forall k \in V_0, j \in V_+\setminus\{k\}, \tag{25}$$

$$\sum_{i\in V_0\setminus\{j\}} y'_{kij} \ge x'_{kj} - z_j \quad \forall k \in V_0, j \in V_+\setminus\{k\}, \tag{26}$$

$$\sum_{i\in V_+\setminus\{j,k\}} y'_{kji} \ge x'_{kj} - z_j \quad \forall k \in V_0, j \in V_0\setminus\{k\}, \tag{27}$$

$$2\sum_{i\in V_+\setminus\{j,k\}} y'_{kji} \le x'_{kj} - z_j + 1 \quad \forall k \in V_0, j \in V_0\setminus\{k\}, \tag{28}$$

$$\sum_{j\in V_0\setminus\{k\}} y'_{kjk} = 0 \quad \forall k \in V_0, \tag{29}$$

$$\sum_{i\in V} x'_{ki} \ge 3\sum_{j\in V_+\setminus\{k\}} y'_{kkj} \quad \forall k \in V_0, \tag{30}$$

$$x'_{ik} - \sum_{j\in V_0\setminus\{k\}} y'_{ijk} \le \sum_{j\in V_+\setminus\{k\}} y'_{kkj} \quad \forall k \in V_0\setminus\{0\}, i \in V_0\setminus\{k\}. \tag{31}$$

Conditions (D1) and (D2) are guaranteed by constraints (25). Constraints (25) and (26) together imply that, if the drone serves node j (i.e., $x'_{kj} = 1$ for some $k \in V_0$ and $z_j = 0$), there must be one drone arc in component k going to j. In this setting, constraints (27) and (28) imply furthermore that there must be one drone arc in the same component k going out of j. Therefore condition (D3) is satisfied. The latter requirement of condition

(D4) is ensured by constraints (29). Note that, by definition of the variables, the drone is launched at node k if and only if $\sum_{j\in V_+\setminus\{k\}} y'_{kkj} = 1$, and this is already satisfied by (25) and (26). Constraints (28) imply furthermore that when the drone is retrieved at node j of component k (i.e., when $x_{kj} = 1$ and $z_j = 1$), then there is no drone arc in component k going out of j. Hence (28) ensure condition (D5). Constraints (30) guarantee that if the drone is launched at node k, then it is active at the first and the last nodes of the component k and at least one customer node in this component, hence condition (D6) is ensured. By constraints (31), if k is the last node of a component i with no drone arc going to (i.e., $x'_{ik} = 1$ and $\sum_{j\in V_0\setminus\{k\}} y'_{ijk} = 0$), then it implies that $\sum_{j\in V_+\setminus\{k\}} y'_{kkj} = 1$, or equivalently, the drone must be launched at node k. So the condition (D7) is also satisfied.

3.3. Drone endurance and completion time

To model the objective of the TSP-MVD and to satisfy the constraints related to the drone endurance, we introduce non-negative continuous variables t_k ($k \in V_0$) in which t_k is the total time amount to complete service for component k. Since we aim to minimize the completion time of both vehicles, or equivalently, the total time amount to complete services for all components, the objective can be represented as follows.

$$\min \sum_{k\in V_0} t_k. \tag{A}$$

As the drone can be launched from the depot at any time, the truck drive does not need to spend the service time s_L to launch the drone at node 0. Therefore, the endurance constraint for the drone in the first component is ensured by

$$\sum_{\substack{i\in V_0\\ j\in V_+\setminus\{0,i\}}} \tau'_{ij} y'_{0ij} + s_R \sum_{j\in V_+} y'_{00j} \le e. \tag{32}$$

Indeed, the first sum in the left hand side of (32) is the total service time of the drone in component 0 (starting from the depot node 0). If the drone is launched at the depot node 0 (which means $\sum_{j\in V_+} y'_{00j} = 1$), it must be used in component 0, hence the second sum in the left hand side of (32) is the service time to retrieve the drone at the last node of component 0. So the left hand side of (32) represents the total active time of the drone in component 0, which must not exceed the drone endurance e.

The endurance constraints for the drone in the other components are satisfied by

$$t_k \le e + M(1 - \sum_{j\in V_+\setminus\{k\}} y'_{kkj}) \quad \forall k \in V_0\setminus\{0\}, \tag{33}$$

in which M is a large real number ($M = (n+1)\max\{\tau_{ij}, \tau'_{ij} \mid i \in V_0, j \in V_+\}$ is sufficient). If the drone is launched at the first node of component k (i.e., $\sum_{j\in V_+\setminus\{k\}} y'_{kkj} = 1$), then by (33) we have $t_k \le e$, which means that the total service time for component k is within the drone endurance.

The total time amount to complete service for component k is computed by

$$t_k = \max\{t^t_k, t^d_k\}, \tag{B}$$

in which t^t_k (resp., t^d_k) is the total time amount for the truck (resp., the drone) to complete its service for component k. By similar arguments for the left hand side of (32), the completion time for the drone in component k is computed by

$$t_k^d = \sum_{\substack{i \in V_0 \\ j \in V_+ \setminus \{k,i\}}} \tau'_{ij} y'_{kij} + s_R \sum_{j \in V_+} y'_{kkj}. \tag{C}$$

For the truck, we need to take into account the service time for the driver to launch the drone if it is launched at the rendezvous node of some component to start a new sortie. To recognize this situation, we use the binary variables $l_k (k \in V_0)$ in which

$$l_k = \begin{cases} 1 & \text{if the drone is launch again at the last node of component } k, \\ 0 & \text{otherwise.} \end{cases}$$

The concordance of variables l_k with their meaning is ensured by imposing

$$\sum_{j \in V_+} y'_{iij} + x_{ki} - 1 \leq l_k \qquad \forall k \in V_0, i \in V_+ \setminus \{k\}. \tag{34}$$

These constraints guarantee that l_k is set to 1 if and only if there is a launching action at the last node i of component k (i.e., $\sum_{j \in V_+} y'_{iij} = x_{ki} = 1$). Then, the completion time for the truck in component k is given by

$$t_k^t = \sum_{\substack{i \in V_0 \\ j \in V_+ \setminus \{k,i\}}} \tau_{ij} y_{kij} + s_R \sum_{j \in V_+} y'_{kkj} + s_L l_k. \tag{D}$$

In cooperation of (A) with (B), (C), (D), we come up with the following constraints to compute the completion time in components.

$$\sum_{\substack{i \in V_0 \\ j \in V_+ \setminus \{k,i\}}} \tau'_{ij} y'_{kij} + s_R \sum_{j \in V_+} y'_{kkj} \leq t_k \quad \forall k \in V_0, \tag{35}$$

$$\sum_{\substack{i \in V_0 \\ j \in V_+ \setminus \{k,i\}}} \tau_{ij} y_{kij} + s_R \sum_{j \in V_+} y'_{kkj} + s_L l_k \leq t_k \quad \forall k \in V_0. \tag{36}$$

To the end, we come up with the following MILP for the TSP-MVD

$$\begin{aligned}
\min \quad & \sum_{k \in V_0} t_k, \\
\text{s.t.} \quad & (1) - (36), \\
z_k \in \{0,1\} \quad & \forall k \in V, \\
x_{ki} \in \{0,1\} \quad & \forall k \in V_0, i \in V, \\
x'_{ki} \in \{0,1\} \quad & \forall k \in V_0, i \in V, \\
y_{kij} \in \{0,1\} \quad & \forall k \in V_0, i \in V_0, j \in V_+ \setminus \{i\}, \\
y'_{kij} \in \{0,1\} \quad & \forall k \in V_0, i \in V_0, j \in V_+ \setminus \{i\}, \\
l_k \in \{0,1\} \quad & \forall k \in V_0, \\
t_k \geq 0 \quad & \forall k \in V_0.
\end{aligned}$$

4. AN ITERATED LOCAL SEARCH FOR TSP-MVD

In this section, following the paradigm of Iterated Local Search (ILS, see e.g., [18]), we propose a metaheuristic for solving the TSP-MVD problem. The key idea is to focus our search on a smaller set of the solution space instead of the whole solution space by using an implicit solution representation called giant tours. We define a giant tour as any TSP

tour that starts from the depot at node 0, goes through all customer nodes and returns to the depot at node $n+1$. After applying the split procedure described in Section 4.1. on a giant tour, we obtain a feasible TSP-MVD solution. Such TSP-MVD solution associated with giant tour $\mathcal{T}$ is denoted by $s_{\mathcal{T}}$. The value of $\mathcal{T}$, also called the completion time of $s_{\mathcal{T}}$ is denoted as $f(\mathcal{T})$. The overall framework of our ILS algorithm for the TSP-MVD is described in Algorithm 1.

Algorithm 1: Iterated local search for TSP-MVD.

Data: A TSP-MVD instance.
Result: Best found giant tour $\mathcal{T}^*$ and its corresponding TSP-MVD solution $s_{\mathcal{T}^*}$.
Initialization: Set values for parameters n_{NI}, $n_{\mathcal{H}}$, r, y, *acceptProb*, ;
$nIter = 0$;
$\mathcal{T}_0 = InitialTour()$;
$s_{\mathcal{T}_0} = Split(\mathcal{T}_0)$;
$s_{\mathcal{T}_0} = LocalSearch(s_{\mathcal{T}_0})$;
$\mathcal{T}_0 = Restore(s_{\mathcal{T}_0})$;
$\mathcal{T}' = \mathcal{T}_0$, $\mathcal{T}^* = \mathcal{T}_0$, $\mathcal{H} = \{\mathcal{T}_0\}$;
while $nIter < n_{NI}$ **do**
    $nIter = nIter + 1$;
    $\mathcal{T} = Perturb(\mathcal{T}')$;
    $s_{\mathcal{T}} = Split(\mathcal{T})$;
    $s_{\mathcal{T}} = LocalSearch(s_{\mathcal{T}})$;
    $\mathcal{T} = Restore(s_{\mathcal{T}})$;
    if $f(\mathcal{T}) < f(\mathcal{T}^*)$ **then**
        $\mathcal{T}^* = \mathcal{T}$;
    if $f(\mathcal{T}) < f(\mathcal{T}')$ **then**
        $\mathcal{H} = \mathcal{H} \cup \{\mathcal{T}\}$;
        if $|\mathcal{H}| > n^{\mathcal{H}}_{max}$ **then**
            Sort elements of $\mathcal{H}$ in the increasing order of $f(\mathcal{T}), \mathcal{T} \in \mathcal{H}$;
            Remove the worst elements in $\mathcal{H}$ until $|\mathcal{H}| = n^{\mathcal{H}}_{min}$;
    if $AcceptanceCriterion(\mathcal{T}, \mathcal{T}', acceptProb)$ **then**
        $\mathcal{T}' = \mathcal{T}$;
    else
        $\mathcal{T}'$ = the giant tour of index $\lfloor r^y |\mathcal{H}| \rfloor$ in $\mathcal{H}$;
$s_{\mathcal{T}^*} = Split(\mathcal{T}^*)$;

Our ILS algorithm starts with a giant tour $\mathcal{T}_0$ obtained by *InitialTour* procedure. The giant tour $\mathcal{T}_0$ is used as the input for the splitting algorithm to get a TSP-MVD solution $s_{\mathcal{T}_0}$. This solution is then educated by using *LocalSearch* procedure and then restored by using *Restore* procedure to have an updated giant tour. The solution corresponding to this giant tour becomes a local optimum to be used as the input for the first iteration. We initialize the history list $\mathcal{H}$ by pushing this updated giant tour into the list. Then, in each iteration, we first save a copy $\mathcal{T}'$ of the current giant tour $\mathcal{T}$, and then "shake" $\mathcal{T}'$ by using *Perturb* procedure. The perturbed giant tour $\mathcal{T}$ is then split by using the splitting algorithm described in Section 4.1. to obtain its corresponding TSP-MVD solution. This solution is then passed to *LocalSearch* procedure to improve its quality (i.e., to reduce its value). After being educated, the giant tour $\mathcal{T}$ is updated by using *Restore* procedure. We then update the best giant tour and its value. In the next step, if the updated giant tour $\mathcal{T}$ is better than

the previous one $\mathcal{T}'$ (in sense of its value), then we add the giant tour $\mathcal{T}$ to a history list $\mathcal{H}$. This list of the past giant tours is then sorted in the ascending order of the tours' values, so that we can remove the worst tours in $\mathcal{H}$ to keep the number of giant tours in the list varying from $n^{\mathcal{H}}_{min}$ to $n^{\mathcal{H}}_{max}$. Then, by using *AcceptanceCriterion* procedure, we decide which giant tour in the list $\mathcal{H}$ will be chosen for the next iteration. If giant tour $\mathcal{T}$ we are considering is accepted, we use it as the current giant tour for the next iteration. Otherwise, we choose a random giant tour in $\mathcal{H}$. We terminate the algorithm when the number of iterations reaches a fixed limit. For the algorithm execution, we use the following parameters:

- n_{NI}: maximum number of iterations,
- $n^{\mathcal{H}}_{min}$, $n^{\mathcal{H}}_{max}$: minimum and maximum numbers of the best tours kept in the history list $\mathcal{H}$,
- *acceptProb*: probability of accepting bad moves in *AcceptanceCriterion*,
- r: a random number in $(0, 1]$,
- y: degree of randomness.

In the following subsections, we discuss in detail the procedures used in our ILS algorithm, that are *Split*, *InitialTour*, *Perturb*, *LocalSearch*, *Restore*, and *AcceptanceCriterion*.

4.1. Splitting algorithm

In this section, we introduce the splitting algorithm for decoding a giant tour into a feasible TSP-MVS solution. This splitting algorithm is an adaption of the one introduced in [3], which was used for the TSP-D, to the context of our TSP-MVD.

Roughly speaking, the algorithm starts with a TSP tour from the depot through all customer nodes, then optimally selects the nodes that could be served by the drone while respecting the relative order of these nodes in the tour. More precisely, we are given a TSP tour, denoted $\mathcal{T}$, from the depot node 0 through all customer nodes and then return to the depot at node $n + 1$. If we denote by $\mathcal{T}[i]$ the i^{th} node on the tour $\mathcal{T}$, then $\mathcal{T}[0] = 0$ and $\mathcal{T}[n + 1] = n + 1$ are the depot and its duplication. We construct an auxiliary directed graph $H_{\mathcal{T}} = (V_{\mathcal{T}}, A_{\mathcal{T}})$ associated with the tour $\mathcal{T}$ as follows. The node set $V_{\mathcal{T}}$ consists of the indices of the nodes on tour $\mathcal{T}$, i.e., $V_{\mathcal{T}} = \{0, 1, \ldots, n + 1\}$. Each arc in $A_{\mathcal{T}}$ has form (i, j) with $i < j$, that corresponds to the situation in which both vehicles start from $\mathcal{T}[i]$ and consecutively travel to serve customers at the subsequent nodes $\mathcal{T}[i + 1], \ldots, \mathcal{T}[j]$. If in such a situation we allow the drone to make at most one sortie in coordination with the truck, then the smallest completion time for the path from $\mathcal{T}[i]$ to $\mathcal{T}[j]$ is considered as a cost c_{ij} assigned to arc (i, j). Additionally, to speed up the splitting algorithm, we only reduce the number of arcs in $A_{\mathcal{T}}$ by using a threshold λ, which is the maximum number of nodes between nodes i and j in tour $\mathcal{T}$, i.e., we only allow arc (i, j) satisfying condition $|\{\mathcal{T}[i], \ldots \mathcal{T}[j]\}| \leq \lambda$. This threshold helps the algorithm to avoid computing the arcs that are too long and might not be used in good TSP-MVD solutions.

The computation of the arc costs is precisely described in Algorithm 2. For the algorithm initialization, we assume that only truck is available to serve the customers, hence the cost of each arc $(i, j) \in A_{\mathcal{T}}$ is set by the truck travel time along the path from $\mathcal{T}[i]$ to $\mathcal{T}[j]$ of the tour $\mathcal{T}$. If the coordination of the truck with a drone sortie can reduce the completion time for such a path, then we update the arc cost by the better completion time. Since the drone cannot make a sortie between two adjacent nodes on the tour, we only need to update the

cost c_{ij} when $\mathcal{T}[i]$ and $\mathcal{T}[j]$ are non-adjacent nodes on tour $\mathcal{T}$. As such, we use variable *cost* to save the best computed value for c_{ij} after checking all feasible options for drone sortie. Such a sortie has the form $\langle \mathcal{T}[i], \{\mathcal{T}[k], \ldots, \mathcal{T}[k+m-1]\}, \mathcal{T}[j] \rangle$ in which the drone serves $m \leq q$ nodes $\mathcal{T}[k], \ldots, \mathcal{T}[k+m-1]$ in this order on the path from $\mathcal{T}[i]$ to $\mathcal{T}[j]$. In the context of such sortie, the nodes $\mathcal{T}[k], \ldots, \mathcal{T}[k+m-1]$ served by the drone are excluded from the truck path. For each of such drone sorties, the total time amount we really use on the path from $\mathcal{T}[i]$ to $\mathcal{T}[j]$ is the maximum completion time between the drone and the truck for this path. The completion time must be checked to ensure that it is within the drone endurance. To incorporate the fact that the truck needs to prepare for the drone before each launch, we also add the drone launch time s_L to the completion time if the launch node $\mathcal{T}[i]$ is not the depot.

Algorithm 2: Cost calculation in TSP-MVD splitting algorithm.

Data: A TSP tour $\mathcal{T}$ from node 0 through all customer nodes.
Result: Cost c_{ij} associated to each arc (i, j) in $H_{\mathcal{T}}$.

Initialization: $\forall (i,j) \in A_{\mathcal{T}}$: c_{ij} = truck travel time along the path from $\mathcal{T}[i]$ to $\mathcal{T}[j]$.
for $i = 0 \to n-1$ **do**
  for $j = i+2 \to \min(i+\lambda, n+1)$ **do**
    $cost = c_{ij}$;
    for $m = 1 \to \min\{q, j-i-1\}$ **do**
      for $k = i+1 \to j-m$ **do**
        if $\mathcal{T}[h] \in V_D$ for all $h = k, \ldots, k+m-1$ **then**
          $droneTime$ = s_R plus drone travel time during its sortie $\langle \mathcal{T}[i], \{\mathcal{T}[k], \ldots, \mathcal{T}[k+m-1]\}, \mathcal{T}[j] \rangle$;
          $truckTime$ = s_R plus truck travel time along the path $(\mathcal{T}[i], \mathcal{T}[i+1], \ldots, \mathcal{T}[k-1], \mathcal{T}[k+m], \mathcal{T}[k+m+1], \ldots, \mathcal{T}[j])$;
          if $droneTime \leq e$ and $truckTime \leq e$ **then**
            $usedTime = \max\{droneTime, truckTime\}$;
            if $i \neq 0$ **then**
              $usedTime = usedTime + s_L$;
            if $usedTime < cost$ **then**
              $cost = usedTime$;
    $c_{ij} = cost$;

Having assigned the corresponding cost to each arc in $H_{\mathcal{T}}$, we can compute a cheapest path from node 0 to node $n+1$ in this auxiliary graph. Such cheapest path can be found by applying the well-known Dijkstra algorithm [**?**]. If there is any arc of form $(i, i+1)$ in the cheapest path, then we convert the arc to a truck path from node $\mathcal{T}[i]$ directly to node $\mathcal{T}[i+1]$. For any arc (i, j) in the cheapest path with $j \geq i+2$, we convert the arc to the truck path together with the drone sortie (if exists) constituting the cost c_{ij} computed in Algorithm 2. After converting all arcs of the cheapest path, we have spit the given TSP tour $\mathcal{T}$ into a feasible solution to the TSP-MVD.

4.2. Getting an initial giant tour

To generate an initial solution for the ILS algorithm, we first construct a TSP tour based on a constructive heuristic so-called k-cheapest insertion, with $k \in \mathbb{Z}^+$. This construction is

the same as the one in [3] used for the TSP-D. For the sake of completeness, we present here the construction details.

We start with a subtour consisting of only the depot and a customer node. Then, step by step, we extend the subtour by inserting an unvisited node. We define an insertion by an ordered set (i, k, j) in which k is an unvisited node while i and j are two consecutive nodes on the current subtour. The cost of such insertion is defined by $\tau_{ik} + \tau_{kj} - \tau_{ij}$. This cost is the increment of time amount when the truck travels along the path $i - k - j$ instead of going directly from i to j. In each step, we randomly choose an insertion (i, k, j) among the n_{insert} insertions of the least costs, and insert node k into the location between i and j, then mark k as a visited node. Here, n_{insert} is a pre-defined parameter and is set to 3 in our experiments. When all customer nodes are visited and added, the tour construction stops and we obtain a giant tour $\mathcal{T}_0$. We return it as the output of *InitialTour* procedure.

4.3. Local search operators

The *LocalSearch* procedure is to educate a TSP-MVD solution obtained by splitting a giant tour. For this procedure, we use the following move operators.

- Truck Swap 1-1: exchanging two random positions on the truck tour.
- Truck Swap 2-1: swapping two consecutive nodes $i, i+1$ with another random node j on the truck tour.
- Truck Swap 2-2: swapping two pairs of consecutive nodes $\{i, i+1\}$ and $\{j, j+1\}$ on the truck tour.
- Truck-only Relocation 1-1: relocating a random truck-only node i after a random node j. For a truck-only node we mean a node that does not belong to any drone sortie.
- Truck-only Relocation 2-1: relocating a random pair of truck-only nodes $\{i, j\}$ after a random node k. We also consider the moves where $\{i, j\}$ are reversed to $\{j, i\}$ when being relocated to k.
- 2-opt: swapping two pairs of consecutive nodes $\{i, j\}$ and $\{u, v\}$ in the truck tour to have $\{i, u\}$ and $\{j, v\}$.
- Drone delivery creation: changing a path $i - j - k$ on the truck tour, in which j is a truck-only node, into a drone sortie in which i is the launch node, j becomes a drone node, and k is the rendezvous node.
- Drone delivery relocation: relocating the sequence of drone visited nodes J in a drone sortie $\langle i, J, k\rangle$ to have another drone sortie $\langle i', J, k'\rangle$ with new launch and rendezvous nodes i' and k' in the truck tour.
- Drone delivery removal: removing a drone sortie $\langle i, J, k\rangle$ by reinserting the sequence of nodes J into the position right after a random node j in the truck tour.
- Inter drone-truck swap 1-1: swapping the launch or rendezvous node of a drone sortie with a drone node of another drone sortie.
- Inter truck-rdv swap 1-1: swapping the launch and rendezvous nodes of a drone sortie.
- Drone sequence insertion: inserting a truck-only node i to a random position in a sequence of drone visited nodes J of a drone sortie.
- Drone sequence removal: removing a drone node $j \in J$ in a drone sortie $\langle i, J, k\rangle$ by reinserting it to the position right after a node j' in the truck tour.

- Drone sequence swap: swapping two drone nodes belonging to two different drone sorties.
- Drone sequence permuting: replacing the sequence of nodes J of a drone sortie $\langle i, J, k\rangle$ by a random permutation of its nodes.

The last four operators are the new ones that we propose to solve the TSP-MVD, while the other operators are adapted from the ones proposed in [3, 7], that were used for the TSP-D. In each call of *LocalSearch* procedure in Algorithm 1, we use all but in a random order of the above move operators. Furthermore, to speed up the performance of *LocalSearch* procedure, we apply the idea of a concept so-called *granular threshold* (see [19]). That is, in each call of the operators, we avoid scanning all the neighborhoods of the considering nodes, and only scan the neighborhoods consisting of 10% of nodes that are closest to each of the considering nodes.

4.4. Perturbation technique

The aim of *Perturb* procedure in Algorithm 1 is to have a new starting point for the ILS by shaking a giant tour. It is observed from [1] that the quality of the giant tour has a clear impact on the value of its associated TSP-D solution. Based on this observation, our perturbation technique is designed for the TSP-MVD in such a way that it keeps the positions of the nodes in the truck tour, and only "shake" a subset of the drone nodes to avoid too much different (and possibly bad) new giant tours. Algorithm 3 describes more precisely the *Perturb* procedure used in our ILS. The procedure takes the current giant tour $\mathcal{T}$ as its input and returns a perturbed giant tour $\mathcal{T}'$. We initialize $\mathcal{T}'$ by a tour of the same structure as $\mathcal{T}$ but all nodes have label -1. At first, we compute the TSP-MVD solution $s_{\mathcal{T}}$ associated with $\mathcal{T}$ by applying the splitting algorithm. This TSP-MVD solution $s_{\mathcal{T}}$ is extracted into two parts: the truck tour $\mathcal{T}^*$ and the list of drone sorties *droneSorties*. We copy all nodes on the truck tour $\mathcal{T}^*$ to $\mathcal{T}'$, keeping their position indices as in the giant tour $\mathcal{T}$. Then, we go through each node in the drone sorties and decide with probability of 0.5 whether to copy this node to the $\mathcal{T}'$ or not, keeping its position index as in $\mathcal{T}$. Finally, we randomly locate each of the uncopied nodes in the drone sorties to the unassigned positions in $\mathcal{T}'$, the ones with label -1.

In addition to the specifically designed perturbation for TSP-MVD described above, two more standard perturbation methods are also used to add more diversification to this process, as follows:

- Random reinsertion: we randomly select p nodes and remove them from the giant tour $\mathcal{T}$ of the current TSP-MVD solution. These nodes are then shuffled and reinserted to the giant tour following the k-cheapest-insertion strategy that is described in [3]. In this paper, the number of nodes p is randomly selected in the set $\{0.1|V|, 0.2|V|, \ldots 0.6|V|\}$, k is randomly chosen between $\{3, 4, 5\}$ for each node waiting to be re-inserted.
- PMX-style operator: This perturbation follows the PMX operator [20] with the two chromosomes are the giant tour $\mathcal{T}$ of the current TSP-MVD solution and a random giant-tour selected in the history $\mathcal{H}$ of ILS.

In each iteration of the ILS, a perturbation strategy is chosen randomly among these three alternative operators.

Algorithm 3: *Perturb* procedure for TSP-MVD.

Data: A giant tour $\mathcal{T}$ and its associated solution $s_{\mathcal{T}}$
Result: A giant tour $\mathcal{T}'$ perturbed from $\mathcal{T}$.
Initialization: $\mathcal{T}' = \mathcal{T}$ and then replace node labels in $\mathcal{T}'$ by -1;
$s_{\mathcal{T}} = Split(\mathcal{T})$;
$\mathcal{T}^*$ = the truck tour in $s_{\mathcal{T}}$;
$droneSorties$ = the list of nodes served by drone in $s_{\mathcal{T}}$;
foreach node i in $\mathcal{T}^*$ **do**
    ind_i = index of i in $\mathcal{T}$;
    $\mathcal{T}'[ind_i] = i$;
foreach node j in $droneSorties$ **do**
    r = random number between (0, 1];
    if $r \leq 0.5$ **then**
        ind_j = index of j in $\mathcal{T}$;
        $\mathcal{T}'[ind_j] = j$;
$I = \{i \in \{0, \ldots, n+1\} \mid \mathcal{T}'[i] = -1\}$ = array of unassigned positions in $\mathcal{T}'$;
$U = \{\mathcal{T}[i] \mid i \in I\}$ = array of uncopied nodes in $s_{\mathcal{T}}$;
Randomly reshuffle U;
for $k = 1$ to $|I|$ **do**
    $\mathcal{T}'[I[k]] = U[k]$;
return $\mathcal{T}'$;

4.5. Restore method

The splitting method described in Section 4.1. converts some nodes of a giant tour to drone nodes in order to obtain a feasible TSP-MVD solution. Conversely, *Restore* method reinserts the drone nodes in a given feasible TSP-MVD solution to the truck tour in order to obtain a new giant tour. The purpose of *Restore* method is to speed up the convergence speed of the search since the giant tour is updated with the "educated" sequences. In Algorithm 4 we describe *Restore* procedure for the TSP-MVD more precisely. Given a TSP-MVD solution composed of a truck tour and set of drone sorties, a giant tour $\mathcal{T}$ is initialized with nodes from the truck tour. Then, for each drone sortie $\langle i, J, k \rangle$, we insert the whole set J of drone nodes to a random position between launch node i and rendezvous node k in $\mathcal{T}$.

Algorithm 4: *Restore* procedure for TSP-MVD.

Data: A TSP-MVD solution s.
Result: A giant tour $\mathcal{T}$.
Initialization: $truckTour$ = truck tour in s, $droneSorties$ = set of drone sorties in s, $\mathcal{T} = truckTour$;
foreach drone sortie $\langle i, J, k \rangle$ **do**
    if $k - i = 1$ **then**
        insert J right before k in $\mathcal{T}$, respecting order of nodes in J;
    else
        $randNode$ = random node between i and k (including k) in $\mathcal{T}$;
        Insert J right before $randNode$ in $\mathcal{T}$, respecting order of nodes in J;

4.6. Acceptance criterion

The procedure *AcceptanceCriterion* used in Algorithm 1 decides whether we should accept a new generated giant tour $\mathcal{T}$ based on the previously perturbed one $\mathcal{T}'$. If the new giant tour $\mathcal{T}$ is better than the previous one $\mathcal{T}'$ in sense of its value, we immediately accept it. Otherwise, we only accept this new giant tour with a probability of $acceptProb$ which decreases gradually. That is, we set $acceptProb = acceptProb/r_{ac}$ after each time a worse giant tour is allowed. Here, $r_{ac} > 1$ is a given value set to 1.0002 defined by experiments. By doing this, we allow the search to explore wider regions at the early iterations to find diversified solutions for the history list $\mathcal{H}$. Moreover, when approaching the final iterations of the procedure, the probability of accepting bad solutions would be very small, making Algorithm 1 to focus on finding better solutions.

Algorithm 5: *AcceptanceCriterion*

Data: New generated giant tour $\mathcal{T}$, previously perturbed giant tour $\mathcal{T}'$, value of $acceptProb$.
Result: $true$ if $\mathcal{T}$ is accepted, $false$ if $\mathcal{T}$ is not accepted.

```
if f(T) < f(T') then
    return true;
else
    randProb = random value between (0, 1] ;
    if randProb ≤ acceptProb then
        acceptProb = acceptProb/r_ac ;
        return true;
    else
        return false;
```

5. NUMERICAL EXPERIMENTS

In this section, we present the computational results of the MILP formulation in Section 3. and ILS algorithm in Section 4.. Our main aim is to validate the MILP model and show the efficiency of the ILS algorithm in comparison with the existing methods for solving TSP-MVD. We also analyze the impact of drone capacity on the performance of the ILS algorithm as well as the quality of obtained solutions.

5.1. Experimental setup

We used CPLEX 12.9 as MILP solver and run the MILP formulation on a computer with an Intel Xeon E5620 2.4 GHz processor and 16 GB of RAM. The ILS algorithm was implemented in C++ and all experiments of the metaheuristic were conducted on a computer with Intel i7-6700 3.4 GHz and 16 GB of RAM.

To generate TSP-MVD instances, we use the same set of TSP-D instances proposed in [3] but allow the drone can serve up to $q \geq 1$ customers in each of its sortie. For the sake of completeness, we present here the construction of the tested instances. Each TSP-MVD instance is characterized by the locations of the depot as well as the customers, the subset of customers that cannot be served by the drone, traveling time of each vehicle between

Table 1: Information of groups of instances for the TSP-MVD

Instance label	Number of customers	Area (km^2)	Average distance
A1 to A5	10	100	7.43
B1 to B10	50	100	7.13
C1 to C10	50	500	15.45
D1 to D10	50	1000	22.19
E1 to E10	100	100	7.14
F1 to F10	100	500	15.21
G1 to G10	100	1000	21.59

the nodes, drone launch time, drone retrieve time, and drone endurance. For each tested instance, we randomly generated locations for a number of customers within a square in a plane. We generated 65 such instances and classified them into 7 groups according to the number of customers and the area of the square, as shown in Table 1.

The fourth column in Table 1 indicates the average Euclidean distance between customer locations. Similarly to the settings in [1], for all generated instances, we set the speed of the truck and the drone to 40 km per hour, the default drone endurance e was set to 20 minutes, both the launch time s_L and and retrieve time s_R were set to 1 minute, and 20% of customers chosen randomly could not be served by the drone. Furthermore, the depot location was set to the bottom left vertex of the square in each tested instance. The maximum number of drone visits per sortie was set to $q = 2$ by default. In addition, we also use the TSP-D instances proposed in [1] to test the performance of our approaches.

We selected values for parameters used in Algorithm 1 by experiments. The maximum number of iterations n_{NI} was set to 5000. To manage the history list $\mathcal{H}$, we set the maximum number $n^{\mathcal{H}}_{max}$ of the best tours keeping in the history list to 25, the minimum number $n^{\mathcal{H}}_{min}$ to 10. The degree of randomness y was set to 2. The value for $acceptProb$ was fixed to 0.5. For the splitting algorithm, the threshold λ was set to $0.2|V|$.

5.2. Results on the small instances of [1]

The aim of this experiment is to validate the MILP model and assess the performance of our ILS algorithm on the set of 72 TSP-D instances of 10-customer size from [1]). For each instance, we consider two settings for the drone endurance: $e = 20$ (minutes) and $e = 40$ (minutes). On each tested instance, our MILP model is run in a time limit of 16 hours while our ILS algorithm is run 10 times and the best objective value as well as the average objective value are reported. We compare the quality of obtained solutions to the ones found by existing methods for the TSP-D: Hybrid Genetic Search Algorithm (HGA, see [7]), the MILP model of [1], the best solutions obtained from a set of methods in [1] including the MILP model. HGA was run 10 times and the best result was reported. The detailed results are reported in Table 9 in Appendix. In that table, column "Instance" gives the name of each tested TSP-D instance, column e gives the corresponding values of drone endurance (in minutes). Columns $MILP_o$, $BEST$, and HGA respectively report the objective values of: solutions obtained by the MILP model of [1], the best solutions found by the set of methods in [1], and the best solutions provided by HGA. The objective values, gap values, and running time in seconds of our MILP model are presented in Columns $MILP_n$, Gap_M

and $time_M$. And finally, Columns ILS, $\overline{ILS}$ report the best and average objective values obtained from 30 runs of our ILS algorithm for each instance. There, all reported objective values are in minutes.

Table 2: Instance that ILS improves the best known solution

Instance	e	$BEST$	HGA	ILS	$\overline{ILS}$
440v8	20	62.796	62.576	**62.222**	62.222

It can be seen from Table 9 that our MILP model can solve 58 over 72 instances to optimality. It is worth mentioning that the MILP model of [1] was run in a time limit of 1800 seconds and could not find any optimal solution. Thus, this is the first time optimal solutions for such instances are reported. Especially, we could find incorrect solutions for three instances found by heuristic methods of [1] (numbers in red in Table 9). ILS algorithm performs at least as good as the existing methods. In particular, our ILS algorithm improves one best known solution from [7] that is reported in Table 2.

5.3. The results on the related instances of [3]

We now validate the MILP formulation and access the solution quality of the ILS algorithm on TSP-MVD instances of small size with 10 customers. Table 3 (resp., Table 4) reports the numerical results of this experiment in case of $q = 1$ (resp., $q = 2$). For each instance, we run the ILS algorithm 10 times and report the best objective value obtained from these runs in Column *obj*. In addition, the number of times that ILS reaches the optimal solution is shown in Column *opt*. The running time of our metaheuristic is quite small (never bypassing 2 seconds), we do not report this information in the result tables. We solve each instance by the MILP formulation to optimality, and report the running time in seconds as well as the optimal objective value in Columns *time* and *obj*, respectively.

Table 3: Numerical results on the TSP-MVD instances of small size (case $q = 1$)

Instance	TSP	MILP		ILS	
	obj	*obj*	*time*	*obj*	*opt*
A1	1.00733	0.8582	7502	0.8582	10
A2	0.95588	0.8996	28722	0.8996	10
A3	0.98568	0.8729	14510	0.8729	10
A4	0.94464	0.8995	8166	0.8995	10
A5	0.98568	0.8664	19100	0.8664	10

To see how the coordination with the drone improves the completion time of serving all customers, we compute the optimal solution for each tested instance when using only the truck, and report the corresponding optimal objective value to the TSP columns. These TSP optimal objective values also give us a baseline to compare the quality of solutions obtained by using the MILP formulation and the ILS algorithm.

It can be seen from Tables 3 and 4 that the ILS algorithm can reach the optimal solutions in all runs on all tested instances. The results also confirm the evidence that allowing the drone to carry one more parcel can reduce the completion time of the whole system.

Table 4: Numerical results on the TSP-MVD instances of small size (case $q = 2$)

Instance	TSP	MILP		ILS	
	obj	*obj*	*time*	*obj*	*opt*
A1	1.00733	0.8465	26465	0.8465	10
A2	0.95588	0.8758	17411	0.8758	10
A3	0.98568	0.8676	23321	0.8676	10
A4	0.94464	0.8620	13318	0.8620	10
A5	0.98568	0.8664	15321	0.8664	10

Table 5: Performance of ILS and HGA on TSP-D instances of large size

Instances	obj_{HGA}	$time_{HGA}$	obj_{ILS}	$time_{ILS}$	Gap (%)
B1 to B10	116.75	0.60	116.92	0.41	0.15
C1 to C10	221.42	0.50	221.28	0.38	**-0.06**
D1 to D10	310.83	0.50	309.50	0.32	**-0.43**
E1 to E10	188.47	4.28	188.82	2.69	0.19
F1 to F10	312.53	5.26	317.40	2.92	1.56
G1 to G10	415.94	4.80	421.72	2.81	1.39
Mean		2.66		**1.59**	0.46

Table 6: Instances that ILS improves the best known solutions in instance set of [3]

	BKS	ILS	Gap (%)
C1	215.07	215.00	**-0.03**
C5	223.06	220.50	**-1.15**
C6	234.01	233.67	**-0.15**
C8	234.26	233.71	**-0.23**
C10	226.17	225.93	**-0.11**
D1	306.39	304.73	**-0.54**
D2	313.93	311.80	**-0.68**
D3	295.86	294.23	**-0.55**
D4	323.72	323.42	**-0.09**
D5	321.46	319.17	**-0.71**
D6	313.21	313.11	**-0.03**
D8	293.76	289.48	**-1.46**
D9	317.85	316.04	**-0.57**
D10	305.51	303.09	**-0.79**
E9	189.76	189.07	**-0.36**
E10	189.45	188.96	**-0.26**
G1	417.92	416.70	**-0.29**

In addition, we compare our ILS algorithm with HGA on the set of 60 TSP-D instances of larger size (B1 to G10 as described in Section 5.1.). Table 5 reports numerical results obtained from these tests. The first column of the table gives the groups of instances. In the second (resp., the fourth) column, we report the average of best objective values of

instances in each group by using HGA (resp., our ILS algorithm). The third (resp., the fifth) column reports the average running time over all instances in each group of HGA (resp., our ILS algorithm). All the mentioned values are in minutes. The last column shows the average improvement (in percent, over all instances in each group) of the best objective value obtained by our ILS algorithm in comparison to the one obtained by HGA. The negative value in this percentage indicates an improvement of our ILS over HGA in sense of objective value.

It follows from Table 5 that our ILS algorithm can perform better than HGA on the instances C1 to C10 and D1 to D10 while showing inferior results comparing to HGA in the remaining instances. In average, ILS performs 0.46% worse than HGA. However, ILS runs much faster than HGA (approximately 40%). Moreover, as can be seen from Table 6, ILS can improve 17 best known solutions (BKS) of HGA reported in [7]. This result is remarkable because ILS is much simpler with less parameters and designed to solve a more general problem. The detailed results of this experiment can be found in Tables 11 in Appendix.

5.4. Impact of the maximum number of drone visits per sortie

To analyze the impact of the maximum number q of drone visits per sortie on the TSP-MVD solutions, we run our ILS algorithm on the 50-customer and 100-customer instances mentioned in Section 5.1. For each instance, the algorithm is run 30 times and three values $q = 1, 2, 5$ are considered. For each pair of tested instance and value of q, we report obj^q_{best} as the best objective value and obj^q_{ave} as the average objective value over 30 runs of ILS. To observe the obtained savings when allowing more number of drone visits per sortie, for each $q = 2$ and $q = 5$ we compute

$$improve^q = \frac{obj^1_{best} - obj^q_{best}}{obj^1_{best}} \times 100\%,$$

which tells us how the best objective value changes in comparison with the one in which only a single drone visit per sortie is allowed. A positive value of $improve^q$ indicates an improvement in the solution objective value when the drone is allowed to serve up to q customers instead of a single customer per sortie.

Table 7 (resp., Table 8) reports the numerical results of this experiment on the set of 50-customer TSP-MVD instances (resp., 100-customer TSP-MVD instances). It can be observed from these tables that by allowing drone to visit multiple customer locations, we can significantly reduce the completion time of the system. In comparison to the situation $q = 1$, if the drone can serve one more customer per sortie (i.e., $q = 2$), the completion time reduces averagely 7.62% for 50 customers, and 10.83% for 100 customers. If the capacity of the drone can be enlarged to serve up to 5 customers per sortie (i.e., $q = 5$), we can save the completion time averagely 12.46% for 50 customers, and 20.67% for 100 customers.

It can be observed furthermore that the improvement in completion time depends under-linearly on parameter q. In fact, by increasing the number of drone visits per sortie by just one, we gain approximately 8% better in the objective value. However, increasing four more visits per drone sortie does not increase the completion time improvement by the same ratio, since our gain is only doubled approximately.

Moreover, we can also observe a significant impact on raising number of drone visits per sortie among high-density instances (the ones with labels B and E) where our gain on the

Table 7: Impact of parameter q solutions of 50-customer instances

	$q = 1$ (TSP-D)		$q = 2$			$q = 5$		
Instance	obj^1_{best}	obj^1_{ave}	obj^2_{best}	obj^2_{ave}	$improve^2$(%)	obj^3_{best}	obj^3_{ave}	$improve^3$(%)
B1	115.72	118.45	100.93	104.06	-12.78	85.53	88.44	-26.09
B2	118.39	119.96	102.90	105.02	-13.08	89.94	93.26	-24.03
B3	116.21	118.79	99.76	103.43	-14.16	87.73	91.24	-24.51
B4	118.99	120.65	102.00	104.69	-14.28	86.18	88.46	-27.57
B5	115.78	118.48	101.56	105.91	-12.28	90.14	92.75	-22.15
B6	115.26	117.97	99.06	102.79	-14.06	81.77	86.00	-29.06
B7	115.53	116.63	101.09	103.64	-12.50	88.36	90.92	-23.52
B8	117.90	118.28	102.66	104.56	-12.93	89.39	91.81	-24.18
B9	117.72	118.69	102.54	105.06	-12.90	90.04	93.30	-23.51
B10	117.74	119.13	98.90	101.35	-16.00	85.56	88.29	-27.33
C1	215.00	218.87	192.95	197.41	-10.26	183.92	187.04	-14.46
C2	209.69	210.47	193.35	195.37	-7.79	183.56	185.67	-12.46
C3	212.02	214.38	206.23	208.02	-2.73	201.93	207.00	-4.76
C4	213.45	217.67	196.07	200.94	-8.14	194.27	197.68	-8.99
C5	220.50	226.23	215.89	218.81	-2.09	209.63	215.25	-4.93
C6	233.67	237.38	223.84	229.11	-4.21	216.16	221.89	-7.49
C7	222.81	227.99	205.26	210.97	-7.88	203.03	207.41	-8.88
C8	233.71	238.45	219.87	224.88	-5.92	218.28	222.94	-6.60
C9	226.02	233.10	210.66	218.44	-6.80	208.28	215.80	-7.85
C10	225.93	229.74	213.29	217.62	-5.59	209.62	213.83	-7.22
D1	304.73	313.18	296.61	306.26	-2.66	294.28	302.25	-3.43
D2	311.80	317.17	306.00	310.96	-1.86	305.06	311.37	-2.16
D3	294.23	308.78	282.93	292.91	-3.84	279.99	297.09	-4.84
D4	323.42	329.17	309.91	317.20	-4.18	307.57	317.73	-4.90
D5	319.17	320.89	313.52	315.07	-1.77	310.69	314.60	-2.66
D6	313.11	314.13	300.81	303.26	-3.93	296.23	300.01	-5.39
D7	319.92	323.78	304.14	315.14	-4.93	304.14	313.56	-4.93
D8	289.48	292.39	281.49	286.27	-2.76	279.74	282.51	-3.36
D9	316.04	322.55	305.87	311.70	-3.22	305.61	311.70	-3.30
D10	303.09	308.70	293.61	301.23	-3.13	293.24	301.01	-3.25
Mean					**-7.62**			**-12.46**

completion time is improved up to 34.40%. This is because on these instances, allowing the drone to visit more customer locations per sortie helps us to save time for launching and retrieving the drone. This amount of time in high-density instances has a major impact to the savings since the drone travel times between locations are relatively small (see the last column in Table 1).

6. CONCLUSIONS

The Traveling Salesman Problem with Multi-Visit Drone (TSP-MVD) presents a new extension of the Traveling Salesman Problem with Drone (TSP-D), in which the drone is able to fly with multiple cargo compartments to serve several customers during each flight. We considered the TSP-MVD whose objective is to minimize the time required to serve all customers and return both the truck and the drone to the depot. We proposed a Mixed Integer Linear Programming formulation and a heuristic method based on Iterated Local Search to solve the TSP-MVD. The numerical experiments showed the validity of our MILP

Table 8: Impact of parameter q on solutions of 100-customer instances

	$q = 1$ (TSP-D)		$q = 2$			$q = 5$		
Instance	obj^1_{best}	obj^1_{ave}	obj^2_{best}	obj^2_{ave}	$improve^2$(%)	obj^3_{best}	obj^3_{ave}	$improve^3$(%)
E1	188.46	189.89	158.57	161.74	-15.86	124.17	129.75	-34.11
E2	187.59	189.62	160.27	162.88	-14.56	128.88	131.40	-31.30
E3	188.54	190.26	162.15	165.24	-14.00	129.13	133.21	-31.51
E4	187.32	188.78	158.68	162.48	-15.29	126.46	130.61	-32.49
E5	188.30	190.07	160.78	165.72	-14.61	124.11	132.58	-34.09
E6	189.83	192.11	162.11	165.39	-14.60	124.52	129.48	-34.40
E7	190.68	192.16	162.71	166.82	-14.67	129.98	133.57	-31.83
E8	189.46	190.85	160.24	163.85	-15.42	127.11	131.09	-32.91
E9	189.07	190.55	162.87	165.93	-13.86	129.11	132.25	-31.71
E10	188.96	190.00	161.71	165.99	-14.42	131.61	135.83	-30.35
F1	328.19	337.49	296.88	307.74	-9.54	265.29	278.42	-19.17
F2	311.50	319.35	280.36	287.73	-10.00	249.02	258.20	-20.06
F3	317.51	330.40	285.97	298.40	-9.93	259.30	272.94	-18.33
F4	316.86	323.86	289.14	300.23	-8.75	262.17	274.45	-17.26
F5	318.52	332.07	283.28	300.55	-11.06	258.13	273.39	-18.96
F6	296.65	313.77	266.77	282.07	-10.07	240.35	256.45	-18.98
F7	316.94	329.39	281.95	293.99	-11.04	241.84	260.38	-23.70
F8	329.22	336.00	289.94	302.16	-11.93	256.98	273.17	-21.94
F9	316.69	326.71	289.68	304.37	-8.53	268.05	280.59	-15.36
F10	321.89	327.84	290.71	302.72	-9.69	272.58	283.89	-15.32
G1	416.70	437.84	385.54	406.77	-7.48	376.14	395.68	-9.73
G2	394.82	405.97	364.60	383.02	-7.65	354.94	368.73	-10.10
G3	418.44	433.66	385.45	402.92	-7.88	376.08	390.89	-10.12
G4	443.99	457.26	404.57	419.62	-8.88	385.14	409.94	-13.25
G5	423.99	435.99	395.81	411.49	-6.65	375.33	398.02	-11.48
G6	420.91	436.48	388.54	410.04	-7.69	384.99	410.01	-8.53
G7	414.10	433.60	378.50	396.75	-8.60	366.11	389.42	-11.59
G8	411.63	426.55	380.84	399.89	-7.48	371.69	389.64	-9.70
G9	434.75	453.15	402.11	420.14	-7.51	386.68	405.79	-11.06
G10	437.87	453.32	406.16	422.72	-7.24	390.66	415.60	-10.78
Mean					**-10.83**			**-20.67**

model when it provides first ever optimal solutions for a number of benchmark instances. Our new ILS performs better than existing methods on several classes of TSP-D instances in terms of solution quality and speed. We also demonstrated that the drone capacity has a significant impact on the completion time of the system. The extension of the proposed methods to solve problems with multiple vehicles and multi-visit drones would be an interesting topic for next researches.

ACKNOWLEDGMENT

This paper is dedicated to the memory of brilliant Vietnamese computer scientist, Professor Dinh Dieu Phan. We would also send this work to our first author, Dr. Quang Minh Ha, who passed away while the paper was being prepared. He took part in the algorithm design and implementation, experiment, and writing.

APPENDIX

Table 9: Performance of methods for solving TSP-D instances of small size taken from [1]

Instance	e	$MILP_o$	$BEST$	$MILP_n$	Gap_M	$time_M$	HGA	ILS	$\overline{ILS}$
37v1	20	56.468	56.468	56.468	0.00%	6275	56.468	56.468	56.468
37v1	40	52.096	50.573	50.573	0.00%	6073	50.573	50.573	50.573
37v2	20	53.207	53.207	53.207	0.00%	5744	53.207	53.207	53.207
37v2	40	47.311	47.311	47.311	0.00%	3053	47.311	47.311	47.311
37v3	20	53.687	53.687	53.687	0.00%	3673	53.687	53.687	53.687
37v3	40	53.687	53.687	53.687	0.00%	18515	53.687	53.687	53.687
37v4	20	67.464	67.464	67.464	0.00%	22699	67.464	67.464	67.464
37v4	40	66.487	66.487	66.487	15.78%	57033	66.487	66.487	66.487
37v5	20	50.551	50.551	50.551	0.00%	9292	50.551	50.551	50.551
37v5	40	45.835	45.835	44.835	0.00%	3935	44.835	44.835	44.835
37v6	20	47.601	45.176	47.311	0.00%	3179	47.311	47.311	47.311
37v6	40	47.601	45.863	43.602	0.00%	6605	43.602	43.602	43.602
37v7	20	51.887	49.581	49.581	0.00%	5684	49.581	49.581	49.581
37v7	40	46.621	46.621	46.621	0.00%	5206	46.621	46.621	46.621
37v8	20	64.687	62.381	62.381	18.19%	57449	62.381	62.381	62.381
37v8	40	59.776	59.776	59.416	20.09%	57528	59.416	59.416	59.416
37v9	20	45.985	45.985	42.416	0.00%	5838	42.416	42.416	42.416
37v9	40	42.416	42.416	42.416	0.00%	6463	42.416	42.416	42.416
37v10	20	43.093	42.416	41.729	0.00%	5785	41.729	41.729	41.729
37v10	40	41.729	41.729	41.729	0.00%	5655	41.729	41.729	41.729
37v11	20	48.215	42.896	42.896	0.00%	5166	42.896	42.896	42.896
37v11	40	42.896	42.896	42.896	0.00%	5318	42.896	42.896	42.896
37v12	20	61.569	56.696	56.273	0.00%	29817	56.273	56.273	56.273
37v12	40	55.696	55.696	55.696	13.98%	38071	55.696	55.696	55.696
40v1	20	49.430	49.430	49.430	0.00%	4793	49.430	49.430	49.430
40v1	40	48.723	46.886	46.886	0.00%	3525	46.886	46.886	46.886
40v2	20	50.708	50.708	50.708	0.00%	3167	50.708	50.708	50.708
40v2	40	46.423	46.423	46.423	0.00%	2967	46.423	46.423	46.423
40v3	20	56.102	56.102	56.102	0.00%	4365	56.102	56.102	56.102
40v3	40	53.933	53.933	53.933	0.00%	11527	53.933	53.933	53.933
40v4	20	69.902	69.902	69.902	17.98%	34708	69.902	69.902	69.902
40v4	40	68.397	68.397	68.397	25.17%	56770	68.397	68.397	68.397
40v5	20	45.358	43.533	43.533	0.00%	2042	43.533	43.533	43.533
40v5	40	46.590	43.533	43.533	0.00%	4094	43.533	43.533	43.533
40v6	20	44.076	44.076	43.949	0.00%	2644	43.949	43.949	43.949
40v6	40	44.076	44.076	43.810	0.00%	3295	43.810	43.810	43.877
40v7	20	51.922	49.996	49.422	0.00%	5026	49.422	49.422	49.422
40v7	40	49.204	49.204	49.204	0.00%	8766	49.204	49.204	49.204
40v8	20	65.624	62.796	63.222	26.76%	56724	62.576	62.222	62.222
40v8	40	62.270	62.270	62.004	28.75%	57377	62.004	62.004	62.004
40v9	20	44.253	42.799	42.533	0.00%	2772	42.533	42.533	42.533
40v9	40	44.253	42.799	42.533	0.00%	2995	42.533	42.533	42.533
40v10	20	43.076	43.076	43.076	0.00%	3655	43.076	43.076	43.076
40v10	40	43.076	43.076	43.076	0.00%	2507	43.076	43.076	43.076
40v11	20	49.204	49.204	49.204	0.00%	5930	49.204	49.204	49.204
40v11	40	49.204	49.204	49.204	0.00%	8194	49.204	49.204	49.204
40v12	20	62.004	62.004	62.004	27.19%	57451	62.004	62.004	62.004
40v12	40	62.004	62.004	62.004	26.88%	57362	62.004	62.004	62.004
43v1	20	69.586	69.586	69.586	0.00%	7220	69.586	69.586	69.587
43v1	40	57.251	55.493	55.493	0.00%	4362	55.493	55.493	55.493
43v2	20	72.146	72.146	72.146	0.00%	9307	72.146	72.146	72.147
43v2	40	58.053	58.053	58.053	0.00%	3567	58.053	58.053	58.053
43v3	20	77.344	77.344	77.344	0.00%	8960	77.344	77.344	77.344
43v3	40	69.175	69.175	68.431	0.00%	5482	68.431	68.431	68.431
43v4	20	90.144	90.144	90.144	0.00%	21012	90.144	90.144	90.144
43v4	40	82.700	82.700	83.653	22.22%	57508	82.700	82.700	82.700
43v5	20	63.247	55.493	54.973	0.00%	2861	54.973	54.973	55.214
43v5	40	53.447	53.447	51.929	0.00%	5779	51.929	51.929	51.929
43v6	20	64.702	58.053	55.209	0.00%	5219	55.209	55.209	55.209
43v6	40	52.329	52.329	52.329	0.00%	4598	52.329	52.329	52.329
43v7	20	67.770	64.409	65.523	0.00%	5029	65.523	65.523	65.523
43v7	40	60.743	60.743	60.743	0.00%	14749	60.743	60.743	60.743
43v8	20	83.700	77.209	78.323	0.00%	49554	78.323	78.323	78.323
43v8	40	74.686	73.967	72.967	17.87%	57418	72.967	72.967	72.967
43v9	20	59.321	49.049	45.931	0.00%	3706	45.931	45.931	45.931
43v9	40	47.250	47.250	45.931	0.00%	5558	45.931	45.931	45.931
43v10	20	61.240	47.935	46.935	0.00%	2831	46.935	46.935	46.935
43v10	40	48.865	47.935	46.935	0.00%	7715	46.935	46.935	46.935
43v11	20	67.435	57.382	56.395	0.00%	8877	56.395	56.395	56.395
43v11	40	56.395	56.395	56.395	0.00%	13055	56.395	56.395	56.395
43v12	20	83.700	69.195	69.195	8.56%	57499	69.195	69.195	69.195
43v12	40	69.195	69.195	69.195	26.93%	57378	69.195	69.195	69.195

Tab.10: Performance comparison between ILS and HGA [7] on 50-customer instances of [3] with $q = 1$

	HGA_{best}	$time_{HGA}$	ILS_{best}	ILS_{avg}	ILS_{gap}	$time_{ILS}$	gap (%)
B1	115.65	0.76	115.72	118.45	2.36	0.38	0.06
B2	118.39	0.33	118.39	119.96	1.33	0.36	0.00
B3	116.21	0.57	116.21	118.79	2.22	0.47	0.00
B4	118.71	0.47	118.99	120.65	1.40	0.48	0.24
B5	115.78	0.58	115.78	118.48	2.33	0.38	0.00
B6	114.31	0.88	115.26	117.97	2.35	0.46	0.83
B7	115.52	0.62	115.52	116.63	0.95	0.41	0.00
B8	117.90	0.78	117.90	118.28	0.32	0.39	0.00
B9	117.64	0.39	117.72	118.69	0.83	0.37	0.07
B10	117.38	0.60	117.74	119.13	1.18	0.44	0.31
C1	215.07	0.60	**215.00**	218.87	1.80	0.43	**-0.03**
C2	209.23	0.53	209.69	210.47	0.37	0.35	0.22
C3	212.02	0.38	212.02	214.38	1.12	0.24	0.00
C4	212.08	0.60	213.45	217.67	1.98	0.44	0.65
C5	223.06	0.48	**220.50**	226.23	2.60	0.34	**-1.15**
C6	234.01	0.31	**233.67**	237.38	1.59	0.31	**-0.15**
C7	222.27	0.51	222.81	227.99	2.33	0.45	0.24
C8	234.26	0.46	**233.71**	238.45	2.03	0.39	**-0.23**
C9	226.01	0.68	226.01	233.10	3.14	0.42	0.00
C10	226.17	0.48	**225.93**	229.74	1.68	0.38	**-0.11**
D1	306.39	0.61	**304.73**	313.18	2.77	0.33	**-0.54**
D2	313.93	0.57	**311.80**	317.17	1.72	0.35	**-0.68**
D3	295.86	0.60	**294.23**	308.78	4.95	0.36	**-0.55**
D4	323.72	0.56	**323.42**	329.17	1.78	0.33	**-0.09**
D5	321.46	0.40	**319.17**	320.89	0.54	0.24	**-0.71**
D6	313.21	0.49	**313.11**	314.13	0.33	0.28	**-0.03**
D7	316.65	0.32	319.92	323.78	1.21	0.35	1.03
D8	293.76	0.58	**289.48**	292.39	1.01	0.33	**-1.46**
D9	317.85	0.41	**316.04**	322.55	2.06	0.34	**-0.57**
D10	305.51	0.41	**303.09**	308.70	1.85	0.33	**-0.79**
Mean		0.53			1.74	**0.37**	**-0.11**

Tab. 11: Performance comparison between ILS and HGA [7] on 100-customer instances of [3] with $q = 1$

	HGA_{best}	$time_{HGA}$	ILS_{best}	ILS_{avg}	ILS_{gap}	$time_{ILS}$	gap (%)
E1	187.67	3.60	188.46	189.89	0.76	2.89	0.42
E2	187.21	5.60	187.59	189.62	1.08	3.53	0.20
E3	188.09	4.58	188.54	190.26	0.91	2.75	0.24
E4	186.23	4.69	187.32	188.78	0.78	2.61	0.59
E5	187.71	4.06	188.30	190.07	0.94	2.13	0.31
E6	189.16	4.84	189.83	192.11	1.20	2.63	0.35
E7	190.39	3.84	190.68	192.16	0.77	3.01	0.15
E8	189.02	4.22	189.46	190.85	0.74	3.10	0.23
E9	189.76	4.00	**189.07**	190.55	0.78	2.29	**-0.36**
E10	189.45	3.40	**188.96**	190.00	0.55	1.96	**-0.26**
F1	322.94	5.73	328.19	337.49	2.83	3.08	1.63
F2	308.74	5.24	311.50	319.35	2.52	2.95	0.89
F3	309.67	5.61	317.51	330.40	4.06	2.84	2.53
F4	311.37	6.06	316.86	323.86	2.21	2.66	1.76
F5	314.82	6.57	318.52	332.07	4.25	2.84	1.18
F6	294.38	4.70	296.65	313.77	5.77	3.08	0.77
F7	311.41	4.92	316.94	329.39	3.93	2.70	1.78
F8	323.74	5.21	329.22	336.00	2.06	2.87	1.69
F9	315.56	4.66	316.69	326.71	3.17	3.18	0.36
F10	312.70	3.94	321.89	327.84	1.85	3.00	2.94
G1	417.92	4.45	**416.70**	437.84	5.07	2.43	**-0.29**
G2	389.64	2.40	394.82	405.97	2.82	2.85	1.33
G3	411.47	4.90	418.44	433.66	3.64	3.04	1.69
G4	433.09	4.67	443.99	457.26	2.99	2.71	2.52
G5	421.05	4.48	423.99	435.99	2.83	2.38	0.70
G6	415.46	5.51	420.91	436.48	3.70	2.87	1.31
G7	409.31	5.21	414.10	433.60	4.71	2.71	1.17
G8	406.51	5.08	411.63	426.55	3.62	2.84	1.26
G9	428.16	5.91	434.75	453.15	4.23	3.31	1.54
G10	426.82	5.40	437.87	453.32	3.53	2.92	2.59
Mean		4.78				**2.81**	1.04

REFERENCES

[1] C. Murray and A. Chu, "The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery," *Transportation Research Part C: Emerging Technologies*, vol. 54, pp. 86–109, 2015.

[2] N. Agatz, P. Bouman, and M. Schmidt, "Optimization approaches for the traveling salesman problem with drone," *Transportation Science*, vol. 52, no. 4, pp. 965–981, 2018.

[3] Q. Ha, Y. Deville, Q. Pham, and M. Ha, "On the min-cost traveling salesman problem with drone," *Transportation Research Part C: Emerging Technologies*, vol. 86, pp. 597–621, 2018.

[4] J. Carlsson and S. Song, "Coordinated logistics with a truck and a drone," *Management Science*, vol. 64, no. 9, pp. 3971–4470, 2017.

[5] J. Freitas and P. Penna, "A variable neighborhood search for flying sidekick traveling salesman problem," *International Transactions in Operational Research*, vol. 27, no. 1, pp. 267–290, 2018.

[6] A. Ponza, "Optimization of drone-assisted parcel delivery," Master's thesis, Universita Degli Studi Di Padova, 2016.

[7] Q. Ha, Y. Deville, Q. Pham, and M. Ha, "A hybrid genetic algorithm for the traveling salesman problem with drone," *Journal of Heuristics*, vol. 26, pp. 219–247, 2020.

[8] R. Roberti and M. Ruthmair, "Exact methods for the traveling salesman problem with drone," *Transportation Science*, vol. 55, no. 2, pp. 315–335, 2021.

[9] M. DellAmico, R. Montemanni, and S. Novellani, "Exact models for the flying sidekick traveling salesman problem," *International Transactions in Operational Research*, 2021.

[10] S. Vasquez, G. Angulo, and M. Klapp, "An exact solution method for the tsp with drone based on decomposition," *Computers & Operations Research*, vol. 127, 2021.

[11] C. Murray and R. Raj, "The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones," *Transportation Research Part C: Emerging Technologies*, vol. 110, pp. 368–398, 2020.

[12] P. Kitjacharoenchai, M. Ventresca, M. Javadi, S. Lee, J. Tanchoco, and P. Brunese, "Multiple traveling salesman problem with drones: Mathematical model and heuristic approach," *Computers & Industrial Engineering*, vol. 129, pp. 14–30, 2019.

[13] X. Wang, S. Poikonen, and B. Golden, "The vehicle routing problem with drones: Several worst-case results," *Optimization Letters*, vol. 11, no. 4, pp. 679–697, 2017.

[14] S. Poikonen, X. Wang, and B. Golden, "The vehicle routing problem with drones: Extended models and connections," *Networks*, vol. 70, no. 1, pp. 34–43, 2017.

[15] Z. Wang and J. Sheu, "Vehicle routing problem with drones," *Transportation Research Part B: Methodological*, vol. 122, no. 4, pp. 350–364, 2019.

[16] S. Poikonen and B. Golden, "Multi-visit drone routing problem," *Computers & Operations Research*, vol. 113, p. 104802, 2020.

[17] Y. Liu, Z. Liu, J. Shi, G. Wu, and W. Pedrycz, "Two-echelon routing problem for parcel delivery by cooperated truck and drone," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2020.

[18] M. Gendreau and J. Potvin, *Handbook of Metaheuristics (third edition)*. Springer, 2019.

[19] P. Toth and D. Vigo, "The granular tabu search and its application to the vehicle-routing problem," *INFORMS Journal on Computing*, vol. 15, no. 4, pp. 333–346, 2003.

[20] D. Goldberg and R. Lingle, "Alleles, loci, and the traveling salesman problem," in *Proceedings of the 1st International Conference on Genetic Algorithms*, 1985, pp. 154–159.

Received on June 18, 2021
Accepted on September 10, 2021