On the Min-cost Traveling Salesman Problem with Drone

QUANG MINH HA¹ YVES DEVILLE¹ QUANG DUNG PHAM³ MINH HOÀNG HÀ²

ICTEAM, Université Catholique de Louvain, Belgium¹
CIRRELT, Ecole Polytechnique de Montréal, Canada²
SoICT, Hanoi University of Science and Technology, Vietnam³

{quang.ha, yves.deville}@uclouvain.be dungpq@soict.hust.edu.vn minhhoang.ha@cirrelt.net

Abstract

Once known to be used exclusively in military domain, unmanned aerial vehicles (drones) have stepped up to become a part of new logistic method in commercial sector called “last-mile delivery”. In this novel approach, small unmanned aerial vehicles (UAV), also known as drones, are deployed alongside with trucks to deliver goods to customers in order to improve the service quality or reduce the transportation cost. It gives rise to a new variant of the traveling salesman problem (TSP), of which we call TSP with drone (TSP-D). In this article, we consider a variant of TSP-D where the main objective is to minimize the total transportation cost. We also propose two heuristics: “Drone First, Truck Second” (DFTS) and “Truck First, Drone Second” (TFDS), to effectively solve the problem. The former constructs route for drone first while the latter constructs route for truck first. We solve a TSP to generate route for truck and propose a mixed integer programming (MIP) formulation with different profit functions to build route for drone. Numerical results obtained on many instances with different sizes and characteristics are presented. Recommendations on promising algorithm choices are also provided.

I. Introduction

Over the years, companies always look for methods to find the most cost efficient way to distribute goods across the logistic network [10]. Traditionally, a truck is used to handle these tasks and the corresponding transportation problem is modeled as a traveling salesman problem (TSP). However, with the emergence of technologies, a new distribution method arises where small unmanned aerial vehicles (UAV), also known as drones, are deployed to support parcels delivery. On one hand, there are 4 advantages of using a drone for delivery: (1) it can be operated without human as pilot, (2) it can avoid the congestion of traditional road networks by flying over them, (3) it is faster than trucks, and (4) it has much lower transportation cost (per kilometer) [13]. On the other hand, as the drones are operated using batteries, their flight endurance is limited, resulted in maximum travel distance and restricted size of parcels. Likewise, a truck has its advantage of long range traveling, large cargo and diversity of parcels’ size but it is also heavy, slow and cost inefficient.

Evidently, the pros of the truck resolve the cons of the drone and conversely, the pros of drone fix the cons of the truck. This is the foundation of a novel method named “last-mile delivery with drone” [3] that transports the drone closer to customer locations by the truck in order to service customers within its flight range and effectively increase its usability and schedule flexibility. Specifically, the truck departs from the depot carrying the drone and all of customer parcels. As the truck makes deliveries, the drone is launched from the truck to service a customer with a parcel. While the drone is in service, the truck continues its schedule to new customer locations. The drone then returns to the truck in a location different from the launch point.

In the literature, we are aware of two previous works that investigated the routing problem related to the combined truck and drone delivery approach. Murray and Chu [7] introduce
the problem and call it "Flying Sidekick Traveling Salesman Problem" (FSTSP). A mixed integer programming (MIP) formulation and a simple heuristic are proposed. Basically, their heuristic is based on "Truck First, Drone Second" (TFDS) idea, that is to construct a route for truck first by solving a TSP problem then use greedy procedures to select nodes that are serviced by drone. Agatz et al., [1] study a slightly different problem - called "Traveling Salesman Problem with Drone" (TSP-D), in which the drone has to follow the same road network as the truck. Moreover, in the TSP-D, the drone is allowed to be launched and return to a same location while in FSTSP this is forbidden. In Agatz et al., [1], the TSP-D is formulated as a MIP formulation and solved by "Truck First, Drone Second" (TFDS) heuristic in which drone route construction is based on local search and dynamic programming. Both of their works aim to minimize the time returning to depot of the truck with instances of 10 customers.

The minimization in truck’s traveling time can improve the service quality [8]. However, in every logistics or delivery operations, transportation cost also plays an important role, helping to reduce the overall business cost [4] [11]. Hence, minimizing this cost by using a more cost-efficient vehicle is a vital objective of every companies with transport and logistic activities. Since we are not aware of any previous works on TSP-D that aims to minimize the total transportation cost, it brings a strong motivation to follow this approach.

This paper studies a novel variant of TSP-D with the following hypotheses: (1) The truck and the drone must go alongside with each other. (2) The drone must be launched and retrieved in two different customer locations. (3) The truck cannot return to any visited customer to pickup the drone (it avoids the subtour problem in TSP). Most importantly, the objective is to minimize the total transportation cost of both vehicles.

In this paper, we propose MIP-based heuristics to solve this min-cost TSP-D variant. Our heuristics are based on not only the existing approach "Truck First, Drone Second" proposed in the literature, but also a novel approach that we call "Drone First, Truck Second". The contributions of this paper are:

- We propose a new variant of TSP-D in which the objective is to minimize the total transportation cost of vehicles.
- We propose two heuristics of two opposite concepts: "Drone First, Truck Second" (DFTS) and "Truck First, Drone Second" (TFDS)
- We introduce a mixed integer programming formulation to solve the clustering step in our heuristic methods
- We provide various sets of instances in different sizes of customers and options for testing this problem

This paper is structured as follows: section II gives the introduction. Section III discusses problem definition and its mixed integer formulation. We describe our two heuristics in section IV. Section IV is about the design of the experiments including instance generations and settings. After that, we discuss the numerical results in section V with a variety of analysis. Finally, section VI concludes the paper and gives suggestion for future researches.

II. THE TRAVELING SALESMAN PROBLEM WITH DRONE

This section is divided into three parts. Firstly, we give the definition of the Traveling Salesman Problem with Drone (TSP-D) where we describe the context, constraints, and explain the terminologies to be used further in this paper. Consequently, we present the MIP formulation adapted
from the work of [7]. Lastly, improvements in constraints and drone deliveries set are proposed to enhance the formulation in terms of modeling and solving time.

1. Problem definition

In TSP-D, we consider a set of customers (nodes), each of whom must be served exactly once by either a truck or a drone. The truck and the drone must depart from, and return to the a single depot exactly once. During the tour of the truck, it may proceed multiple drone deliveries. Each of drone delivery is a tuple consists of three nodes: the launch node, the drone node, and the return node as follows:

- A launch node can be either a depot or a customer location (node).
- A drone node is a customer location that is served by drone.
- After service a customer, drone will either rendezvous with the truck at the next customer location, or return to the depot. These two vehicles must wait for each other at the rendezvous point to fulfill the movement-synchronization characteristic [5]. Furthermore, in order to guarantee the service quality, these two vehicles waiting time are limited by a constant called time span.

Once launched, the drone must visit a customer and return to a truck or to the depot within the drone’s flight endurance or time span limit. A tuple is only selected if and only if these two time constraints are met. There is also a time needed for the truck’s driver to launch and retrieve the drone. The objective is to minimize the total transportation cost, given their different rates per kilometer.

A visual explanation is shown in figure 1, 2, 3. Starting from the truck-only route in figure 1, figure 2 interprets the scenario where customers are far from the depot and outside the flight endurance of the drone.

![Figure 1: A truck-only tour](image)

On the other hand, figure 3 illustrates the case where some customers are within the drone flight range, therefore, the drone can start directly from the depot, service a customer, then rendezvous with the truck to continue the delivery task. At the end, the drone, after being
launched from the truck, service a customer, can directly flight to the depot as it meets the flight endurance requirement.

As can be seen from the example above, a solution of TSP-D is an assemblage of smaller routes, each of which may or may not contain the delivery by drone. In the case it has a drone, the route contains not only an drone delivery, but also a set of customers that the truck serve during the time the drone performs a delivery. We define this kind of route an drone route which is described below.

As the generalization of TSP, this is a NP-Hard problem. It reduces to a TSP when the endurance of the drone is 0, hence no drone is used during the computation.

1.1 Drone route

A drone route is defined as $(i, j, k, O)$ where:
- \((i, j, k)\) is a drone delivery: a tuple of three locations: launch point \(i\), service point \(j\), rendezvous point \(k\) as explained above

- \(\{O\}\) is a set of in-between nodes that the truck may travel between \(i\) and \(k\) (during the drone performs a sortie). This set may be empty, stating that the truck and the drone will travel a triangle (figure 5).

**Figure 4:** A sample drone route with intermediate customers between launch point and rendezvous point

**Figure 5:** A sample drone route with no in-between customers

An example is shown in Figure 4 set \(O\) has 3 customers. Another example is in Figure 5 where there is no customer in-between launch point and rendezvous point, hence the truck and the drone will travel a triangle. The definition of drone route is a central element in our cluster first formulation which is shown in the section below.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drone delivery</td>
<td>A tuple of three nodes (\langle i, j, k \rangle) represents drone’s launch point, service point and rendezvous point, respects the endurance and timespan constraints</td>
</tr>
<tr>
<td>In-between nodes</td>
<td>List of nodes that the truck visits during a drone delivery</td>
</tr>
<tr>
<td>Drone node</td>
<td>The service node (j) in a drone delivery (\langle i, j, k \rangle)</td>
</tr>
<tr>
<td>Drone route</td>
<td>A complete subroute of truck and drone operation</td>
</tr>
</tbody>
</table>

**Table 1:** Problem's terminologies

2. Notations, parameters and mathematical formulation

The following notations and parameters are applied for all the mixed integer programming (MIP) formulations in this paper. Let \(N = \{1, \ldots, n\}\) denote the set of customers. Let \(0\) and \(n + 1\) denote single physical depot. The set of all nodes in the graph is denoted by \(V = \{0, \ldots, n + 1\}\). We also denote \(V_0 = \{0, \ldots, n\}\) as the set of nodes that a drone can launch from,

\[
V_+ = \{1, \ldots, n + 1\}
\]

as the set of nodes that a drone can return. We denote \(d_{ij}\) the Manhattan distance between customer \(i\) and \(j\) and \(d'_{ij}\) is its Euclidean distance. Let \(\tau_{ij}\) and \(\tau'_{ij}\) denote the time travel for the
truck and the drone respectively, with \( i \in V_0 \) and \( j \in V_+ \). The travel time of each type of vehicle is calculated by taking into account their unique speed. Additionally, as the truck and drone are not allowed to revisit any node, \( \tau_{ij} \) and \( \tau'_{ij} \) are undefined for all \( i \in V \). Let \( s_L \) and \( s_R \) be the time taken for the drone to be launched from and recovered by, upon the rendezvous point. The flight endurance of the drone is denoted by \( e \). Let \( P \) be the set of all drone deliveries. We denote \( C_1 \), \( C_2 \) the transportation cost rates (per kilometer) of truck and drone respectively. The time span is denoted as \( T \).

We now define the decision variables. Let \( x_{ij} \in \{0, 1\} \) equal one if the truck goes from node \( i \) to node \( j \) with \( i \in V_0 \) and \( j \in V_+ \) and \( i \neq j \). Let \( y_{ijk} \in \{0, 1\} \) equal one if the drone serves the delivery \( (i, j, k) \). Let \( t_j, t'_j \geq 0, t_0 = t'_0 = 0 \) represent the earliest departure time of both truck and drone. Let \( p_{ij} \in \{0, 1\} \) equals one if the customer \( i \in N \) is visited at some moments before customer \( j \in N, j \neq i \) in the truck’s path. This variable ensures that consecutive drone deliveries are consistent with the ordering of the truck’s visitation sequence. As if both \( i \) and \( j \) are visited by the drone, the value of \( p_{ij} \) will be inconsequential in the constraints. We also have \( p_{0j} = 0 \) for all \( j \in N \), to indicate that the truck will always start the tour from the depot. As a standard TSP subtour elimination constraints, we denote \( 1 \leq u \leq n + 2 \) the position of the node \( i, i \in V_+ \) in the truck’s path.

The MIP formulation is presented as follows:

\[
\begin{align*}
\text{Min} & \quad C_1 \sum_{i \in V_0} \sum_{j \in V_+} d_{ij} x_{ij} + C_2 \sum_{i \in V_0} \sum_{j \in V_+} \sum_{k \in V_+} \sum_{i \neq j} (d'_{ij} + d'_{jk}) y_{ijk} \\
\text{s.t.} & \quad \sum_{i \in V_0} x_{ij} + \sum_{i \in V_0 \atop i \neq j} \sum_{k \in V_+} y_{ijk} = 1 \quad \forall j \in N \tag{1} \\
& \quad \sum_{j \in V_+} x_{0j} = 1 \tag{2} \\
& \quad \sum_{i \in V_0} x_{i, n+1} = 1 \tag{3} \\
& \quad u_i - u_j + 1 \leq (n + 2)(1 - x_{ij}) \quad \forall i \in N, j \in \{V_+ : j \neq i\} \tag{4} \\
& \quad \sum_{i \in V_0} x_{ij} = \sum_{k \in V_+} x_{jk} \quad \forall j \in N \tag{5} \\
& \quad \sum_{j \in N} \sum_{k \in V_+} y_{ijk} \leq 1 \quad \forall i \in V_0 \tag{6} \\
& \quad \sum_{i \in V_0 \atop i \neq k} \sum_{j \in N} y_{ijk} \leq 1 \quad \forall k \in V_+ \tag{7} \\
& \quad 2y_{ijk} \leq \sum_{h \in V_0 \atop h \neq i} x_{hi} + \sum_{l \in N \atop l \neq k} x_{lk} \quad \forall i \in N, j \in \{N : j \neq i\}, k \in \{V_+ : (i, j, k) \in P\} \tag{8} \\
& \quad y_{0jk} \leq \sum_{h \in V_0 \atop h \neq k} x_{hk} \quad \forall j \in N, k \in \{V_+ : \} \tag{9}
\end{align*}
\]
\begin{align}
  &u_k - u_i \geq 1 - (n + 2) \left(1 - \sum_{j \in N} \sum_{k \in V_+} y_{ijk} \right) \quad \forall i \in N, k \in \{V_+: k \neq i\} \\
  &t_i' \geq t_i - M \left(1 - \sum_{j \in N} \sum_{k \in V_+} y_{ijk} \right) \quad \forall i \in N \\
  &t_i' \leq t_i + M \left(1 - \sum_{j \in N} \sum_{k \in V_+} y_{ijk} \right) \quad \forall i \in N \\
  &|t_k - t_i'| \leq T \left(\sum_{i \in V_0} \sum_{j \in N} y_{ijk} \right) \quad \forall k \in V_+ \\
  &t_k \geq t_h + \tau_{bk} + SL \left(\sum_{i \in N} \sum_{m \in V_+} y_{klm} \right) + SR \left(\sum_{i \in V_0} \sum_{j \in N} y_{ijk} \right) - M(1 - x_{bk}) \quad \forall h \in V_0, k \in \{V_+: k \neq h\} \\
  &t_i' \geq t_i' + \tau_{ij}' - M \left(1 - \sum_{k \in V_+} y_{ijk} \right) \quad \forall j \in N', i \in \{N_0: i \neq j\} \\
  &t_i' \geq t_i' + \tau_{jk}' - M \left(1 - \sum_{i \in V_0} y_{ijk} \right) \quad \forall j \in N', k \in \{V_+: k \neq j\} \\
  &t_k' - (t_k' - t_i') \leq e + M(1 - y_{ijk}) \quad \forall k \in V_+, j \in \{N: j \neq k\}, i \in \{N_0: (i, j, k) \in P\} \\
  &u_i - u_j \geq 1 - (n + 2)p_{ij} \quad \forall i \in N, j \in \{N: j \neq i\} \\
  &u_i - u_j \leq -1 + (n + 2)(1 - p_{ij}) \quad \forall i \in N, j \in \{N: j \neq i\} \\
  &p_{ij} + p_{ji} = 1 \quad \forall i \in N, j \in \{N: j \neq i\} \\
  &t_i' \geq t_k' - M \left(3 - \sum_{j \in N} \sum_{m \in V_+} y_{ijk} - \sum_{n \in V_+} n_{ijk} - p_{il} \right) \quad \forall i \in V_0, k \in \{V_+: k \neq i\}, l \in \{N: l \neq i, l \neq k\} \\
  &t_0 = 0 \\
  &t_0' = 0 \\
  &p_{0j} = 1 \quad \forall j \in N \\
  &x_{ij} \in \{0, 1\} \quad \forall i \in V_0, j \in \{V_+: j \neq i\} \\
  &y_{ijk} \in \{0, 1\} \quad \forall i \in V_0, j \in \{N: j \neq i\}, k \in \{V_+: (i, j, k) \in P\}
\end{align}
The objective function (1) is to minimize the total transportation cost of both vehicles. The cost includes two parts: (i) the first part is the cost of the truck going in Manhattan distances; (ii) the second part is the cost of the drone, going two arcs in each drone delivery, in Euclidean distances.

For the constraints, each customer is served exactly once by either truck or drone as defined in constraint (2). Constraint (3) ensures that the truck departs from depot exactly once and constraint (4) says it must return to depot exactly once also. Truck’s subtour elimination is provided in (5) and (29). Constraint (6) states that the truck visits at node \( j \) must also depart from \( j \). Constraint (7) and (8) indicate that the drone can be launched (rendezvous) from (at) any customer location or the starting (ending) depot at most once. Constraint (9) associates the drone delivery with the truck route. It states that if the drone executes a delivery to \( j \) then the truck’s route must be assigned to both \( i \) and \( k \) in order for it to launch and retrieve the drone. In the special case where the drone is launched directly from the depot, constraint (10) ensures that the truck will pickup the drone at \( k \). Furthermore, in constraint (11), the truck must visit \( i \) before \( k \) if the drone executes a delivery \( (i, j, k) \).

Constraint (12) and (13) define that if the drone executes a delivery \( (i, j, k) \), then it must be synchronized in time with the truck at \( i \) for launching. In constraint (14), we ensure that both vehicles only wait for each other at the rendezvous point no later than \( T \) in a drone route. Constraint (15) incorporates the arrival time of the truck at one point \( k \) with the previous arrival time \( h \) and its travel time, and possibly, launch and/or retrieval time if there’s a drone delivery. Similarly, constraints (16) and (17) associate the arrival time of the drone with its travel time and retrieval time when there’s a drone delivery. Constraint (18) ensures drone’s flight endurance requirement. Constraint (19)-(22) define that if the drone is launched from \( i \) and return to \( k \) then during this delivery it cannot be launched again, given that \( i \) precedes \( l \) in the truck’s route. Finally, constraints (23)-(31) define the decision variables and their initial values. The value of big \( M \) from constraints (12)-(21) is determined by taking the total travel time using nearest neighbor heuristic.

### III. Heuristics

This section presents two heuristics based on two adverse approaches. In "Drone First, Truck Second" (DFTS), we aim to find the clusters - the set of drone routes that are selected in the final solution. After that, we build the tour of truck given the existing drone routes. The underlying idea is to find as many drone routes as possible first since the cost of using drone is much cheaper, hence, more drone used, more cost-efficient. Alternatively, in "Truck First, Drone Second" (TFDS), we build the truck-only tour first, then apply an adapted version of Murray and Chu [7] heuristic to find the set of drone routes by replacing some nodes served by the truck with the drone nodes and possibly rearrange some nodes in truck’s tour. This approach considers truck-only tour as the initial solution (when drone endurance is 0), then apply multiple iteration with the hope of improving it.
1. Drone First, Truck Second (DFTS)

In this section, we introduce a mixed integer programming formulation to solve the cluster step in our heuristics. Before presenting the formulation, we first define additional notations. We also explain the method to find the drone routes $\Omega$ and two different profit functions: one considers the factor of time span and the other does not.

1.1 Additional notations

We use the existing notations and parameters of the formulation above and define additional ones as follows: let $\Omega$ represents the set of possible drone routes given the set of nodes $V$. We want to restate that a tuple of a drone route is selected if and only if it meets the flight endurance or time span requirement. Furthermore, a tuple (drone delivery) may be appear in one or many drone routes due to the various possibilities of in-between nodes $\{O\}$. Hence the size of $\Omega$ is at least the number of drone deliveries.

We also define some binary coefficients as follows: let $u_{ir} \in \{0, 1\}$ equals one if customer $i \in N$ appears in set $\{O\}$ of drone route $r \in \Omega$ in the final solution. Similarly, $d_{ir} \in \{0, 1\}$ equals one if customer $i \in N$ is a drone node of route $r \in \Omega$. Additionally, $t_{ir} \in \{0, 1\}$ equals one if customer $i \in V$ appears either at the launch point $i$ or rendezvous point $k$ of a drone delivery of route $r \in \Omega$.

We have the binary decision variables $\lambda_r \in \{0, 1\}$ equals one if drone route $r \in \Omega$ is selected in the final solution. Let $c_r$ denote the profit of choosing route $r$. This profit is calculated using various functions that are shown in the experiment settings section.

1.2 Drone first formulation

Given the additional notations, parameters and variables, we now present our mixed integer programming (MILP) formulation:

$$\text{Max } \sum_{r \in \Omega} c_r \lambda_r \quad (32)$$

subject to:

$$\sum_{r \in \Omega} u_{ir} \lambda_r \leq 1 \quad \forall i \in N \quad (33)$$

$$\sum_{r \in \Omega} d_{ir} \lambda_r \leq 1 \quad \forall i \in N \quad (34)$$

$$\sum_{r \in \Omega} t_{ir} \lambda_r + \sum_{r \in \Omega} d_{ir} \lambda_r \leq 1 - \sum_{r \in \Omega} u_{ir} \lambda_r \quad \forall i \in V \quad (35)$$

$$\sum_{r \in \Omega} t_{ir} \lambda_r + \sum_{r \in \Omega} u_{ir} \lambda_r \leq 1 - \sum_{r \in \Omega} d_{ir} \lambda_r \quad \forall i \in V \quad (36)$$

$$\sum_{r \in \Omega} d_{ir} \lambda_r + \sum_{r \in \Omega} u_{ir} \lambda_r \leq 1 \quad \forall i \in V \quad (37)$$

$$\sum_{r \in \Omega} t_{ir} \lambda_r \leq 2 - 2 \left( \sum_{r \in \Omega} d_{ir} \lambda_r + \sum_{r \in \Omega} u_{ir} \lambda_r \right) \quad \forall i \in V \quad (38)$$

The objective is to maximize the profit of choosing drone routes among $\Omega$. Constraint (33) ensures that each node appears in-between a drone route at most once. Similarly, constraint (34) guarantees that each node appears as drone node at most once. In constraint (35), if a node appears as an in-between nodes in a drone route, it cannot appear as terminal node or drone node.
in any other drone deliveries. Constraint (36) is similar to (35) but consider if a node appears as a drone node, it cannot be in any other drone deliveries. Constraint (37) and (38) states that a node is only play one role, either terminal node or drone node, or in-between node.

1.3 Finding the drone routes $\Omega$

From the formulation above, there remains the method to find the set of drone routes $\Omega$. We describe the algorithm as below:

**Algorithm 1: Find drone routes**

**Data:** Drone deliveries $P$, limit $L$  
**Result:** Set of drone routes $\Omega$

1. $\Omega \leftarrow \emptyset$
2. **foreach** $(i, j, k)$ in $P$ **do**
   3. $S \leftarrow$ all possible paths from $i$ to $k$ that goes to less than $L$ nodes ;
   4. **foreach** $s$ in $S$ **do**
   5. $\Omega \leftarrow \Omega \cup ((i, j, k), s)$
5. return $\Omega$

In details, with each drone delivery $(i, j, k)$, we collect all the paths that truck can go from $i$ to $k$. However, finding this set itself is a difficult task, therefore, we limit the number of nodes in the path $L$. In our experiment, we have $L = 1$, meaning that between $i$ and $k$ the truck can go to one more location at most. Additionally, as the objective of is to minimize the transportation cost, and the truck’s cost is much higher than the drone’s, it is better for the truck not to travel to many nodes while the drone is in delivery.

1.4 Profit functions

In the MIP formulation above, the profit $c_r, r \in \Omega$ is calculated by two different functions with various options. They are:

**Function 1:** $c_r = p_r$  

**Function 2:** $c_r = \frac{p_r}{w_r + 1}$  

Where $w_r$ is the actual waiting time between 2 vehicles at the rendezvous point and $p_r$ has 2 options:

1. $p_r$ is calculated by taking into account two elements: $(d_1)$ Euclidean distance from drone node of a drone delivery tuple $(i, j, k)$ to the center of the customers and $(d_2)$ Manhattan distance of the truck going from $i$ to $k$. Then we have:
   $$p_r = \frac{d_1}{d_2}$$  

2. Similar to (1), $p_r$ is calculated by replacing $(d_1)$ with $(d_3)$ which is the Euclidean distance from the drone node to the depot, so we have:
   $$p_r = \frac{d_3}{d_2}$$
Both options of \( p_r \) share the same logics. They aim to choose the drone routes that: (1) minimize distances of truck's path \( (d_2) \); (2) have drone nodes are ones in the boundary of the customer region \( (d_1) \), or the furthest from the depot \( (d_3) \). Since the model always maximize this total distance, it allows the drone to serve the far-most customers, hence, save more cost as the drone's is much cheaper than truck's.

1.5 Heuristic

The pseudo code is shown in Algorithm 2. It begins at line 1 by solving the clustering procedure in function \( \text{solveMIP} \) to get the set of selected drone routes. This set, along with the remaining nodes that are not appeared in any selected drone routes, is then used to construct a symmetry TSPLIB file in line 2. However, if we leave the drone route as it, during the process of building the truck’s tour, its origin may be altered as all elements of the drone route will be treated as normal nodes in a TSP graph. In order to keep it unchanged, we need to fix these edges in the TSPLIB file. This is the work of \( \text{constructTSPInput} \) function which is introduced in Algorithm 3.

After line 2, the "drone first" part of the heuristic is completed. The tour of the truck with given drone routes is then constructed by using a TSP solver in line 3. Finally we process the drone routes to get the solution in line 4. Figure 6 demonstrates a sample process of DFTS heuristic.

Algorithm 2: DFTS heuristic

Data: A TSP-D instance
Result: A TSP-D Solution

1. droneRoutes \( \leftarrow \text{solveMIP}(\text{instance}) \);
2. TSPInput \( \leftarrow \text{constructTSPInput} (\text{droneRoutes}, \text{instance}) \);
3. TSPSolution \( \leftarrow \text{solveTSP}(\text{TSPInput}) \);
4. Solution \( \leftarrow \text{createSolutionFromTSP} (\text{TSPSolution}, \text{instance}) \);
5. return Solution;

Constructing TSP with fixed edges

In Algorithm 3 we want to construct a full matrix TSPLIB file from a list of nodes, \( tspNodes \). Firstly, we consider each drone route in the droneRoutes to save all the edges from the launch node \( i \) to the rendezvous node \( k \) in line 6. We also add all the truck nodes (\( i, k \), and in-between nodes) to the tspNodes. This list is initialized with all the remaining nodes that are not in any drone routes. In line 10, we create a full weight matrix from tspNodes. In this matrix, all the edges that are found in fixedEdges are set to weight of -9999, so they are fixed and must be traversed in the final TSP solution.

Figure 6: DFTS sample process
Algorithm 3: constructTSPLIB function that create a TSPLIB file, as input for TSP solver

**Data:** droneRoutes

**Result:** outputFile

1. remainNodes ← all nodes not in droneRoutes;
2. tspNodes ← remainNodes;
3. fixedEdges ← [];
4. foreach route ∈ droneRoutes do
   5. ⟨i, j, k⟩ ← drone delivery of route;
   6. fixedEdges ← edges in the path from i to k;
   7. tspNodes ← i;
   8. tspNodes ← k;
   9. tspNodes ← in-between nodes of i and k;
10. fullMatrix = build a full weight matrix from tspNodes, with all the edges’ values in fixedEdges are set to -9999;
11. generate a TSPLIB file outputFile with given fullMatrix;
12. return outputFile;

2. Truck First, Drone Second (TFDS)

In this section, we present an adapted version of Murray and Chu’s FSTSP heuristic so that it incorporates the factor of time span and consider the savings of transportation cost instead of traveling time. The main principle of this algorithm is to run multiple iterations, each of which determine every truck-served customers to see whether they can be: (i) rearranged to another position to have better cost saving, (ii) removed from the truck’s route and become a drone node. During the iterations, a global maxSavings value is maintained and the algorithm will be terminated until no more savings can be achieved (maxSavings = 0).

Specifically, in algorithm 4 it firstly solves the truck-only TSP using the same solver as DFTS, then calculate the arrival times of the truck to each node, and the cumulative cost at each node (line 2). Initially, the original truck route is the only "subroute" in the truckSubRoutes (line 3). As the algorithm proceeds, it will divide the truck route into smaller subroutes and store in this list. Each of them may be a drone route or truck-only route. Line 5 starts the iterations and line 6 considers each customer j, calculating its possible savings in algorithm 5.

In algorithm 5, if j does not belong to a drone route, then we simply get the cost savings if we remove j and the truck goes directly from i to k (line 3). On the other hand, when j belongs to a drone route, we also need to compare the savings above with the transportation cost of the drone delivery (line 9).

We go back to algorithm 4 in line 8, where each subroute is processed to see if current considering customer j can be associate with it: either relocate j or make j to become its drone node. If the subroute is a drone route, then we call the function calcCostTruck (algorithm 6) otherwise calcCostDrone is called (algorithm 7).

In calcCostTruck function (algorithm 6), we iterate through each pair of adjacent nodes i and k in the subroute and evaluate the possibility to insert j between i and k to save the cost. Line 4 calculates the cost that can be saved, line 5 calculates the time cost when inserting j. We check the cost saving in line 6 and the time requirement in line 7. If this saving is better than maxSavings then we store these positions i, j, k (line 10).

For the calcCostDrone function (algorithm 7), we consider a truck-only subroute. The algorithm will find a possibility to create a new drone route with j being the drone node, or examine whether
Algorithm 4: TFDS heuristic with time span and min-cost objective

Data: truck-only sequence truckRoute
array t of arrival times at each node

Result: A TSP-D solution represented by a list of truckSubRoutes

1. Customers = N;
2. [truckRoute, t] = solveTSP(N);
3. truckSubRoutes = {truckRoute};
4. maxSavings = 0;
5. i* ← -1;
6. j* ← -1;
7. k* ← -1;
8. servedByDrone ← null;
9. repeat
10.     forall the j ∈ Customers do
11.         savings ← calcSavings(j);
12.     forall the subroute in truckSubRoutes do
13.         if there is a drone associate with this subroute then
14.             Call the calcCostTruck(j, t, subroute, savings) function
15.         else
16.             Call the calcCostDrone(j, t, subroute, savings) function
17.         if maxSavings ≥ 0 then
18.             Call the performUpdate function;
19.             Reset maxSavings = 0;
20.         else
21.             STOP
22. until Stop;
23. return truckSubRoutes;

Algorithm 5: calcSavings(j)

Data: j : a customer currently assigned to the truck

Result: Solution

1. Find i, the node immediately preceding j in the truck’s route;
2. Find k, the immediate successor node to j in the truck’s route;
3. savings = (d_{i,j} + d_{j,k} - d_{i,k}) C_1;
4. return savings;

j can be removed from the truck’s route for drone delivery (line 1-4). If the answer is yes, then we check the cost saving and store the locations of i, j, k and update the maxSavings (line 5-8).

Backward to TFDS heuristic in algorithm 4 after considering all customers and their possibilities for relocation or being a drone node (line 6-12), we check if the best changes can result in a positive maxSavings, meaning we can actually save more cost, then we call the performUpdate function to make the changes (line 14) and reset maxSavings afterward (line 15). In the case we cannot find any changes that can save the existing cost, the algorithm is stopped.
Algorithm 6: calcCostTruck(j, t, subroute, savings) - Calculates the cost of inserting customer j into a different position of the truck’s route

Data:
- j: current considering customer
- t: the vector of truck’s arrival time to each node
- subroute: current considering subroute
- savings: current savings if j is a drone node

Result: Updated $i^*, j^*, k^*, servedByDrone$

1. Find a, the first node in truck’s subroute;
2. Find b, the last node in truck’s subroute;
3. For all the adjacent i and k in subroute do
   4. cost = $(d_{i,j} + d_{j,k} - d_{i,k})C_1$;
   5. timeCost = $\tau_{i,j} + \tau_{j,k} - \tau_{i,k}$;
   6. If cost < savings then
      7. If $t[b] - t[a] + timeCost \leq \min\{e, timespan\}$ then
         8. If savings - cost > maxSavings then
            9. servedByDrone = False;
            10. $j^* = j; i^* = i; k^* = k$;
            11. maxSavings = savings - cost;

Algorithm 7: calcCostDrone(j, t, subroute, savings) - Calculates the cost of serving customer j by a drone (a drone node)

Data:
- j: current considering customer
- subroute: current considering subroute
- savings: current savings if j is a drone node

Result: Updated $i^*, j^*, k^*, servedByDrone$

1. For all the i and k in subroute, i precedes k do
   2. If $t[i] + t[j] + timeCost \leq \min\{e, timespan\}$ then
      3. Find $c'[k]$, the truck’s cost at node k if j was removed from the truck’s route;
      4. cost = $(d'_i + d'_j + d'_{j,k})C_2$;
      5. If savings - cost > maxSavings then
         6. servedByDrone = True;
         7. $j^* = j; i^* = i; k^* = k$;
         8. maxSavings = savings - cost;

This is a simple yet effective heuristic to find good drone routes in order to save the transportation cost. However, it has one weakness due to its greedy characteristics: once it chooses a node to be drone node, it will not try to relocate this node in the next iterations.
Algorithm 8: performUpdate function

**Data:** servedByDrone, \( i^*, j^*, k^* \)

**Result:** Updated truckRoute, truckSubRoutes, \( t \)

1. if servedByDrone == True then
   2. The Drone is now assigned to \( i^* \rightarrow j^* \rightarrow k^* \);
   3. Remove \( j^* \) from truckRoute and truckSubRoutes;
   4. Append a new truck subroute that starts at \( i^* \) and ends at \( k^* \);
   5. Remove \( i^*, j^*, k^* \) from Customers;
   6. Update \( t \);

7. else
   8. Remove \( j^* \) from its current truck subroute;
   9. Insert \( j^* \) between \( i^* \) and \( k^* \) in the new truck subroute;
10. Update truckRoute;
11. Update \( t \);

IV. Experiments

In this section, we describe our design of experiment in term of instances and settings. For the instances, we adapt the characteristics of depot locations from the work of [7], the customer locations generations from [1]. The detailed explanations are shown in subsections below.

1. Instances generation

In order to evaluate the impact of different aspects of depot and customer locations to the performance of our heuristics, 3 sets of instances were generated. Each of them has different number of customers along with the square region that they are distributed (see table 2). The customer locations respect the triangle inequality and are uniformly distributed across the area.

<table>
<thead>
<tr>
<th>Set</th>
<th>Number of customers</th>
<th>Square region (km²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Set 2</td>
<td>50</td>
<td>25</td>
</tr>
<tr>
<td>Set 3</td>
<td>100</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 2: List of instance sets and their characteristics

We design the first set (Set 1) similar to the parameters of Murray and Chu [7]. Set 2 has 5 times more customers in 2 times bigger than Set 1. The last set doubled in both customer and area size. We put customers in larger area so that the drone alone cannot make direct delivery to all customers from the depot, bringing up the meaning of using it alongside with truck to travel long distance, closer to customers’ regions.

For each instance, the depot is placed at one of the three locations: in \((0,0)\), the average of the x- and y-coordinates of the customers, the average x-coordinates with y-coordinates of zero. We call them 0-center, xy-center, x-center respectively. In every set, we generate 10 instances for each depot location setting. Therefore there are 90 instances generated in total.
2. Experiment settings

We create three combinations of speed between drone and truck and drone’s endurance. With the truck’s speed set to a fixed value, we diversify the drone’s speed and its endurance. The settings are shown in Table 3.

<table>
<thead>
<tr>
<th>Name</th>
<th>Speed of truck (km/h)</th>
<th>Speed of drone (km/h)</th>
<th>Endurance (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>40</td>
<td>25</td>
<td>40</td>
</tr>
<tr>
<td>S2</td>
<td>40</td>
<td>56</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 3: Speed and endurance settings

The time to launch and retrieve the drone is 1 minute each. The time span is set to 5 minutes. Transportation cost ratio between drone and truck is 1:25. We will also do an experiment with the ratio 1:50 and 1:10 with Set 2 to evaluate the impact of this value to our heuristics performance.

To solve the MIP model, we use CPLEX 12.6.2. The TSP is solved (in both heuristics) by the state-of-the-art TSP solver Concorde [2].

With 2 function types, 2 $p_r$ options, we have 4 settings of DFTS, labeled as: obj-1, obj-2, obj-1w, obj-2w. TFDS heuristic is labeled as tf.

V. Numerical results

This section describes experimental results on all settings of our heuristics and the one proposed by [7]. The algorithm is implemented in Scala language and the tests are run on a computer with 2.4-GHz Intel Core i7 5500U and 8GB of RAM. The rest of this section is organized as follows. Firstly, we analyze the performance of all heuristics with optimal solution of the MIP model (section II) in small instances. Secondly, we compare the results of all heuristics with truck-only value to evaluate their overall performance. Then we report the results of DFTS heuristic under all settings against TFDS. After that, we analyze which is the best setting for each instance options. The last two sections evaluate the impact of truck - drone speed ratios and cost ratios to the performance of our heuristics.

1. Comparison with optimal solution

In this section, we run 300 tests to evaluate the performance between the MIP model optimal solution [2] and two heuristics. The gap is calculated by:

$$\text{gap} = \frac{\text{heuristicObjective} - \text{optimalObjective}}{\text{optimalObjective}} \times 100$$

(43)

The smaller positive gap, the better performance of the heuristics, as it is close to the optimal solution. The results are shown in Table 4. As can be seen, DFTS with obj-1 and obj-2 delivers the best average gap comparing to TFDS. In details, both obj-1 and obj-2 has a similar average performance (28.54% and 32.86%) while it is about 60% worse in TFDS (49.45%). This is also true with the best gap and worst gap. While the best gap between obj-1, obj-2 and TFDS doesn’t not so different, the worst gap shows a significant distinction. TFDS’s worst gap is about 2.3 times larger than obj-1 and 1.4 times than obj-2.

Also, DFTS with obj-1w and obj-2w have the worst performance among all gaps, of which we will analyze in the section below.
2. Comparison with truck-only TSP optimal solutions

In this result, we want to observe the gap in percentage between TFDS and 4 settings with truck-only TSP optimal solution. The results are demonstrated in Table 5, 6, 7. The gap is calculated by:

$$\text{gap} = \frac{\text{newObjective} - \text{tspObjective}}{\text{tspObjective}} \times 100$$ (44)

A larger negative gap means better performance against truck-only TSP solution. The average gaps in each table is the result of 300 tests run. From the tables, we can easily observe that DFTS with obj-1, obj-2 outperformed TFDS (tf) regardless of the customer size. Moreover, TFDS has a stable performance in terms of average gap, best and worst gap among all sets because of its greedy nature and initial solution (always start from an optimal TSP truck-only solution).

Furthermore, DFTS with time span factor in the profit function showed a poor performance with positive average gaps among 3 sets. It is because in order to minimize the time span while choosing drone routes, it may falls into a case like in Figure 7.

When looking at the best gap and worse gap among 3 sets, we can also notice the improvement in stability of obj-1 and obj-2 with the incremental number of customers. In details, positive gap appears with obj-2 in Set 1 (4.71), obj-1 in Set 2 (0.21), but not in Set 3 where they are quite similar (-10.13 and -10.03) and closer to the best gap (-24.48 and -23.66).

<table>
<thead>
<tr>
<th>Set 1</th>
<th>Avg gap (%)</th>
<th>Best gap (%)</th>
<th>Worst gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tf</td>
<td>-13.31</td>
<td>-32.57</td>
<td>0.00</td>
</tr>
<tr>
<td>obj-1</td>
<td>-24.04</td>
<td>-54.25</td>
<td>-0.09</td>
</tr>
<tr>
<td>obj-1w</td>
<td>-0.10</td>
<td>-37.67</td>
<td>48.04</td>
</tr>
<tr>
<td>obj-2</td>
<td>-21.42</td>
<td>-47.16</td>
<td>4.71</td>
</tr>
<tr>
<td>obj-2w</td>
<td>3.25</td>
<td>-27.87</td>
<td>32.91</td>
</tr>
</tbody>
</table>

Table 5: Compare with TSP - Set 1
3. Comparison between two heuristics

Because instances and results of Bouman et al. are not available by private contact, we are able to compare our heuristics with the one adapted from Murray and Chu only. We calculate the gap - the percentage that our models can outperform or being outperformed by TFDS heuristic. The gap is calculated by:

$$\text{gap} = \frac{\text{objective} - f_{\text{stsp}}}{f_{\text{stsp}}} \times 100$$  \hspace{1cm} (45)

A negative average gap means that the setting of DFTS is better than TFDS heuristic. Additionally, only a winning rate of more than 50% can result in an overall dominance. The results are shown in Table 6, 7, 8. The gaps and winning rates are calculated among 300 tests for each table. As can be seen, among all sets, DFTS with obj-1 and obj-2 shows better performance against TFDS in average gap (from -6.25% to -12.18%) and winning rate (mostly more than 80%). Furthermore,
obj-2 has a stable gap in every tests (-9.45, -10.53, -0.01) comparing to TFDS. Again, DFTS with time span profit function delivers poorest performance, with winning rate less than 10% in Set 1 and 0% in Set 2,3, strengthening our observation that it is not a good strategy to minimize waiting time when choosing clusters in DFTS.

<table>
<thead>
<tr>
<th>Set 1</th>
<th>Avg gap (%)</th>
<th>Winning rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj-1</td>
<td>-12.18</td>
<td>80.00</td>
</tr>
<tr>
<td>obj-1w</td>
<td>15.36</td>
<td>10.00</td>
</tr>
<tr>
<td>obj-2</td>
<td>-9.45</td>
<td>68.33</td>
</tr>
<tr>
<td>obj-2w</td>
<td>19.62</td>
<td>8.33</td>
</tr>
</tbody>
</table>

Table 8: Compare between two heuristics - Set 1

<table>
<thead>
<tr>
<th>Set 2</th>
<th>Avg gap (%)</th>
<th>Winning rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj-1</td>
<td>-8.00</td>
<td>81.67</td>
</tr>
<tr>
<td>obj-1w</td>
<td>24.07</td>
<td>0.00</td>
</tr>
<tr>
<td>obj-2</td>
<td>-10.53</td>
<td>86.67</td>
</tr>
<tr>
<td>obj-2w</td>
<td>21.50</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 9: Compare between two heuristics - Set 2

<table>
<thead>
<tr>
<th>Set 3</th>
<th>Avg gap (%)</th>
<th>Winning rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj-1</td>
<td>-6.25</td>
<td>66.67</td>
</tr>
<tr>
<td>obj-1w</td>
<td>15.87</td>
<td>0.00</td>
</tr>
<tr>
<td>obj-2</td>
<td>-9.01</td>
<td>75.00</td>
</tr>
<tr>
<td>obj-2w</td>
<td>14.95</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 10: Compare between two heuristics - Set 3

4. Comparison between instance options

In this section, we report the average gap in section 2 for each of the instance option (depot location) and the best setting in Table 11. This number is obtained by taking the rate among 300 tests for each depot location.

As can be seen, all heuristics perform best when the depot is located at the center of the customers. This strengthens our observations and hypotheses when designing \( p_t \) in profit function in section 1.4 with obj-1 and obj-2 performs best in xy-center instances and a distinctive average gap comparing to other instance options (obj-1: -25.41 vs. -17.08 and -20.44; obj-2: -26.26 vs. -17.92 and -19.31). We demonstrate this scenario in Figure 8 where the drones are sent to the far-most locations, allowing the truck to travel in an “inner-circle” of the customer region.

It is also logical when the best setting of xy-center is obj-1 (DFTS). It is also the best in x-center
Table 11: Instance options and its best setting performance

<table>
<thead>
<tr>
<th>Depot location</th>
<th>fstsp</th>
<th>obj-1</th>
<th>obj-1w</th>
<th>obj-2</th>
<th>obj-2w</th>
<th>Best</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-center</td>
<td>-16.27</td>
<td>-17.08</td>
<td>-1.86</td>
<td>-17.92</td>
<td>-2.80</td>
<td>obj-2</td>
</tr>
<tr>
<td>x-center</td>
<td>-18.00</td>
<td>-20.44</td>
<td>-4.36</td>
<td>-19.31</td>
<td>-3.90</td>
<td>obj-1</td>
</tr>
</tbody>
</table>

Figure 8: An example of xy-center depot location. The red lines are truck’s routes, the dashed green lines are drone’s routes

5. Impacts of truck - drone speed ratios to the heuristics’ performances

In this section, we analyze the impact of different speed ratio between truck and drone to the performance of the heuristics and settings. We conduct the test again only on Set 2 (50 customers). Table 12 and Table 13 reports the result for speed-endurance setting S1, S2 in Table 3. In details, drone is slower than truck but has more endurance in S1, and faster than truck but less endurance in S2.

When drone is slower, TFDS performs badly with the average gap is significantly worse than DFTS with obj-1 and obj-2 (-4.62% vs. -18.78% and -20.04%). Furthermore, the best gap of obj-1 and obj-2 are about two times better than TFDS. On the other hand, when drone is faster, TFDS’s performance is much increased and close to DFTS with obj-1 and obj-2. This phenomena can be explained as follows: with a higher speed, the drone is able to serve more and further customers, therefore allows TFDS to choose more drone nodes for relocation, making it performs better.

In addition, this change also affects the results of DFTS with time span in profit function (obj-1w and obj-2w), in details, from 13.99% → -4.87% with obj-1w; 13.39% → -4.24% with obj-2w. Also, their best gaps are also close to TFDS. It means higher drone’s speed not only allows DFTS to save more cost, but also minimize the waiting time of both vehicles.
6. Impacts of truck - drone cost ratios

To assess the impact of truck - drone cost ratio to the performance of our heuristics, we conduct 600 more tests on Set 2 with two more ratios - 1:50 and 1:10. The average gap between each ratio is shown in Table 14 where detailed gaps are reported in Table 15. The results are obvious: the higher truck - drone cost ratio, the more cost-efficient we can achieve, regardless of the heuristics. However, this effect is more clearly in DFTS with obj-1 and obj-2. In details, it jumps from -12.59% → -19.17% → 21.39% and -14.93% → -21.45% → -23.66% for obj-1 and obj-2, while this number is slightly changed in TFDS: -9.29% → -11.54% → -11.99%.

Therefore, we can either reduce the cost of the drone or truck in order to lengthen this ratio. For example: a higher durability battery for the drone, and a better fuel efficient truck. Futhermore, hybrid solution is also an option where we can combine between solar energy and battery for the drone, or using hybrid electric trucks.

<table>
<thead>
<tr>
<th>Set 2</th>
<th>Avg gap (%)</th>
<th>Best gap (%)</th>
<th>Worst gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tf</td>
<td>-4.62</td>
<td>-26.28</td>
<td>0.00</td>
</tr>
<tr>
<td>obj-1</td>
<td>-18.78</td>
<td>-54.25</td>
<td>0.21</td>
</tr>
<tr>
<td>obj-1w</td>
<td>13.99</td>
<td>-24.48</td>
<td>48.04</td>
</tr>
<tr>
<td>obj-2</td>
<td>-20.04</td>
<td>-47.16</td>
<td>4.71</td>
</tr>
<tr>
<td>obj-2w</td>
<td>13.39</td>
<td>-15.04</td>
<td>32.91</td>
</tr>
</tbody>
</table>

Table 12: Compare with TSP with speed-endurance setting S1 - Set 2

<table>
<thead>
<tr>
<th>Set 2</th>
<th>Avg gap (%)</th>
<th>Best gap (%)</th>
<th>Worst gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tf</td>
<td>-18.64</td>
<td>-32.57</td>
<td>-5.08</td>
</tr>
<tr>
<td>obj-1</td>
<td>-20.98</td>
<td>-46.35</td>
<td>-1.43</td>
</tr>
<tr>
<td>obj-1w</td>
<td>-4.87</td>
<td>-37.67</td>
<td>9.63</td>
</tr>
<tr>
<td>obj-2</td>
<td>-21.16</td>
<td>-46.16</td>
<td>1.99</td>
</tr>
<tr>
<td>obj-2w</td>
<td>-4.24</td>
<td>-27.87</td>
<td>11.55</td>
</tr>
</tbody>
</table>

Table 13: Compare with TSP with speed-endurance setting S2 - Set 2

<table>
<thead>
<tr>
<th>Avg gap</th>
<th>1:10 (%)</th>
<th>1:25 (%)</th>
<th>1:50 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tf</td>
<td>-9.29</td>
<td>-11.54</td>
<td>-11.99</td>
</tr>
<tr>
<td>obj-1w</td>
<td>9.50</td>
<td>9.76</td>
<td>4.80</td>
</tr>
<tr>
<td>obj-2</td>
<td>-14.93</td>
<td>-21.45</td>
<td>-23.66</td>
</tr>
<tr>
<td>obj-2w</td>
<td>7.97</td>
<td>7.33</td>
<td>3.59</td>
</tr>
</tbody>
</table>

Table 14: Comparison of average gap with TSP in different cost ratios - Set 2
VI. Conclusion

This paper introduces a new variant of the Traveling Salesman Problem with Drone (TSP-D) with the objective is to minimize the total transportation cost. We present two heuristic methods - DFTS and TFDS - to solve the problem based on two opposing approaches. In DFTS, we propose a mixed integer programming formulation to solve the clustering step. An experiment is conducted with a variety of parameters, settings and instance options. The numerical analysis shows that DFTS without the time span in profit function performs better than TFDS in term of average results. However, TFDS has a good stability while DFTS is less stable (positive gaps).

As nature of a generalized version of a NP-hard problem (TSP), larger instances of this problem cannot be solved by using a MIP solver for clustering, in a reasonable time. It resulted in the tuning of time span parameter to reduce the $\Omega$ size in MIP model. Hence, local search techniques should be developed. That is the next step in our developments for this problem. Additionally, we want to have more numerical analysis on the relation between different instance options, speed and cost ratio to the performance of heuristics. They will clearly give us closer look into the nature of this problem. It is also obviously clear to extend this problem to a multiple trucks and multiple drones problem. Also an online and dynamic version is also a logical direction.

Acknowledgement

The research of this paper is supported by National Foundation for Science and Technology (NAFOSTED), project ID FWO.102.2013.04

References


