An Exact Solution Method for the TSP with Drone Based on Decomposition

Sebastián A. Vásquez, Gustavo Angulo, Mathias A. Klapp
School of Engineering, Pontificia Universidad Católica de Chile, Santiago, Chile
savasquez at uc dot cl, gangulo, maklapp at ing.puc dot cl

Abstract

The Traveling Salesperson Problem with Drone (TSP–D) is a routing model in which a given set of customer locations must be visited in the least amount of time, either by a truck route starting and ending at a depot or by a drone dispatched from the truck en route. We study the TSP–D model and propose a mixed–integer programming formulation which exploits the model’s structure and decomposes it into two natural decision stages: (1) selecting and sequencing a subset of customers served by the truck and (2) planning where to execute drone direct dispatches from the truck to each of the remaining customer locations. We design a Benders–type decomposition algorithm, strengthened by valid inequalities arising from structural properties of optimal solutions, and improved optimality cuts stemming from the notions of $t$–shortcut and $t$–reduction, which might be of independent interest. Finally, our solution approach is empirically tested over an extensive family of computationally simulated instances, which show its effectiveness.

Keywords: traveling salesman problem, vehicle routing, drones, Benders’ decomposition.

1 Introduction

Logistics operators, such as Amazon and UPS, continuously search to reduce last-mile delivery costs and therefore continue to pilot drone delivery technologies [24, 1, 26, 19, 8]. One promising alternative is to operate a delivery system that jointly uses a truck and a drone to benefit from the advantages of each vehicle; a drone is faster, autonomous and does not rely on roads, but a truck has more capacity and autonomy.

The Traveling Salesperson Problem with Drone (TSP–D), introduced in [15], studies how to efficiently plan a joint truck and drone delivery operation. The input of this model is a set of geographical customer locations to visit, either by a truck route starting and ending at a single depot, or by a drone executing direct dispatches to customers from the truck while en route. Each time a drone visits a customer location, it must immediately flight back to rendezvous the truck at one location scheduled in its route. If the truck reaches earlier to the meeting point, then it waits until the drone arrives. The objective function of the TSP–D is to minimize the total duration of the delivery operation, which is defined as the sum of the truck travel and waiting times. Figure [1] illustrates an instance of the TSP–D and a feasible solution. The truck route is represented by solid arcs while drone dispatches are pictured by dashed arcs.
Recently, the TSP–D has received vast attention in the vehicle routing research community. Several articles propose exact algorithms [6] and heuristic methods [2, 10, 11, 15, 16] to solve it. Existing exact approaches can be classified into three categories which, along with the method proposed in this work, are summarized in Table 1 and explained further below.

The first class employs implicit enumeration via Mixed–Integer Programming (MIP) formulations for the TSP–D and Constraint Programming (CP). MIP formulations that are solved directly with off–the–shelf solvers include [15] and the strengthening in [9], which simultaneously select a truck and a drone route, and [2], which introduces the concept of operation, allowing to decide on truck and drone routes splittings. However, these formulations become ineffective when solving to optimality instances having more than 10 customer locations.

The work in [22] proposes a Branch–and–Cut (B&C) method based on the formulation of [2]. The authors extend classic TSP valid cuts to the TSP–D and assume a maximum number of nodes that the truck can visit without the drone aboard. This solution approach is capable of solving instances of up to 20 customer locations. A CP approach is presented in [20], which reports solutions to instances of up to 18 customer locations.

The second group is driven by a priori explicit enumeration. An exact solution method based on Dynamic Programming (DP) over exponentially many states is proposed by [6]. This method has three solution stages: computing shortest paths for all combinations of terminal and inner nodes; for each combination, computing an optimal drone dispatch connecting each inner node with the terminal node; and selecting the truck route and drone direct dispatches that minimize the total travel time. The authors manage to solve instances of up to 16 customer locations within a 2 hour solution time. In [25], authors solve the TSP–D by enumerating all feasible truck routes and sorting them by ascending travel time. For each route, the minimum truck delay generated by the drone is computed via MIP only if the truck route travel time does not exceed the incumbent duration; the incumbent is later updated accordingly. This method is able to solve instances of up to 12 customer locations.

The third category relies on dynamic enumeration. In [18], a DP is embedded within a Branch–and–Price (B&P) method. Authors use ng–route relaxation, dual stabilization, and branching strategies to improve computing times. They report solutions to instances of up to 29 customer locations within 1 hour.

To the best of our knowledge, no research effort has studied decomposition approaches for the TSP–D, such as Benders’ decomposition [4], which has been proven successful in solving related combinatorial optimization problems; see [17]. In this work, we make a first step in this direction by leveraging the decision structure of the TSP–D to build and decompose a two-stage model for this problem. Below we summarize our contributions:

- We propose a novel MIP formulation for the TSP–D which exploits the model’s structure and decomposes it into two natural decision stages. In the first stage, we select a subset of customers served by the truck and sequence its visiting route. In the second stage, we design the drone’s operation for a given truck route, choosing where to execute direct dispatches to each of the remaining customer locations from the vehicle en route.

- We provide structural characterizations of optimal solutions, which allow us to simplify the formulations, derive valid inequalities, and provide lower bounds on the second–stage cost.

- We design a Benders–type algorithm to solve this two–stage model implementing a customized version for the TSP–D of the Integer L–shaped Method proposed in [12] and enhanced in [3]. In particular, we define t–shortcuts and t–reductions as a means of improving the effectiveness of optimality cuts, which we think might be of independent interest.

- We empirically assess the performance of our solution method by executing it on a series of instances with up to 25 nodes. Our results shows that the combination of the proposed strategies improves running times
and optimality gaps compared to a standard implementation.

Table 1: Summary of existing exact solution methods and the approach in this work

<table>
<thead>
<tr>
<th>Solution approach</th>
<th>Authors</th>
<th>Method description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit enumeration</td>
<td>[15, 9]</td>
<td>Route truck and drone simultaneously</td>
</tr>
<tr>
<td></td>
<td>[2]</td>
<td>Enumeration of truck and drone routes splittings</td>
</tr>
<tr>
<td></td>
<td>[22]</td>
<td>Enumeration of truck and drone routes splittings, B&amp;C</td>
</tr>
<tr>
<td></td>
<td>[20]</td>
<td>CP</td>
</tr>
<tr>
<td>Explicit enumeration</td>
<td>[6]</td>
<td>3–stage DP</td>
</tr>
<tr>
<td></td>
<td>[25]</td>
<td>Route explicit enumeration</td>
</tr>
<tr>
<td>Decomposition</td>
<td>This work</td>
<td>Truck and drone route decomposition</td>
</tr>
</tbody>
</table>

The remainder of this article is organized as follows. In Section 2, we formally define the TSP–D. Our proposed formulation and decomposition algorithm are presented in Section 3. In Section 4, we present several properties of the TSP–D which are used in Section 5 to strengthen our formulation. We present a computational study in Section 6. Finally, we conclude our work and discuss some extensions as future research in Section 7.

2 Problem Statement

The TSP–D is defined over a complete digraph \( G = (N, A) \), where \( N \) and \( A \) are the sets of nodes and arcs, respectively. Node 1 is the depot and customer locations are indexed as nodes 2 to \( n \equiv |N| \). Also, define \( \delta^+(S) \) and \( \delta^-(S) \) as the sets of outbound and inbound arcs for each cut-set \( S \subset N \), respectively. Vectors \( c, d: N^2 \rightarrow \mathbb{R}_+ \) represent truck and drone arc travel times, respectively; we assume they are proportional to each other, i.e., \( c = \alpha \cdot d \), and metric distance measures.

An operation \((i, k, j) \in N^3\) over \( G \) is a drone dispatch taking off from the truck at node \( i \), visiting node \( k \) and returning to the truck at node \( j \) while the truck executes a route from \( i \) to \( j \). The set of all feasible operations is \( O = \{(i, k, j) \in N^3 : i \neq k \neq j, k \neq 1 \} \cup \{(1, k, 1) : k \in N \setminus \{1\}\} \); also, let \( \delta^+_O(i) \) and \( \delta^-_O(i) \) be the sets of operations starting and ending at node \( i \in N \), respectively.

Given a truck route \( R = \{r_0 = 1, r_1, \ldots, r_{|R|−1}, r_{|R|} = 1\} \), the duration of operation \((i, k, j)\) is equal to \( \max(\hat{c}_{ij}(R), d_{ik} + d_{kj}) \), i.e., the maximum between the truck’s partial route from node \( i = r_p \) to node \( j = r_q \), defined as \( \hat{c}_{ij}(R) \equiv \sum_{r_p < t < r_q} c_{r_t,r_{t+1}} \), and the drone dispatch duration. Therefore, the waiting time incurred by a truck executing route \( R \) during operation \((i, k, j)\) is equal to

\[
\omega_{ikj}(R) = (d_{ik} + d_{kj} - \hat{c}_{ij}(R))_+ \triangleq \max\{d_{ik} + d_{kj} - \hat{c}_{ij}(R), 0\}.
\]

In Figure 2a we provide an example of an operation \((r_2, k, r_6)\) planned for a route \( \{1, r_1, \ldots, r_7\} \) in a graph where customer nodes are depicted by circles and the depot by a square. The truck’s route is depicted with solid arcs, while the drone dispatch is drawn with dashed arcs.
Figure 2: Example of a truck route visiting all customer nodes but \( k \), which is visited by a drone dispatch defined by operation \( (r_2, k, r_6) \).

The total duration of any truck route \( R \) is the sum of the travel and waiting time of all the operations executed along the route. The objective of the TSP–D is to plan a minimum duration truck route \( R \) with an arbitrary number of operations visiting all customer nodes, either by the truck or the drone.

As [6], we assume that the truck is no faster than the drone, i.e., \( \alpha \geq 1 \), negligible service and drone charging times, unit drone capacity, and unlimited flight autonomy. Other important assumptions are that operations start and finish at different customer nodes, all customer nodes can be visited by either the truck or the drone, and that both vehicles can wait en route.

3 Model Formulation

Now, we present a TSP–D model based on MIP.

3.1 Truck route model

The truck’s route is encoded by a vector of binary variables \( x \in \{0, 1\}^{|A|} \), where \( x_a = x_{ij} \) takes value 1 if and only if the truck traverses arc \( a = (i, j) \in A \). Also, let \( \gamma \in \{0, 1\}^n \) be a vector of auxiliary binary variables, where \( \gamma_i \) takes value 1 if and only if node \( i \in N \) is visited by the truck. Thus, we define \( X \) as the feasible set of truck routes defined by all vectors \((x, \gamma)\) satisfying

\[
\sum_{a \in \delta^-(i)} x_a = \sum_{a \in \delta^+(i)} x_a = \gamma_i \quad i \in N \tag{1a}
\]

\[
\sum_{a \in \delta^+(S)} x_a \geq \max_{i \in S} \{\gamma_i\} \quad \emptyset \neq S \subset N : \{1\} \notin S \tag{1b}
\]

\[
x_{ij} \leq \sum_{(i,1) \in \delta^-(1)} x_{i,1} \quad (1, j) \in \delta^+(1) \tag{1c}
\]

\[
x_{i1} \leq \sum_{(i,1) \in \delta^-(1)} x_{1,j} \quad (i, 1) \in \delta^-(1) \tag{1d}
\]

\[
\sum_{i \in N} \gamma_i \geq \left\lceil \frac{n}{2} \right\rceil \tag{1e}
\]

\[
\gamma_1 = 1 \tag{1f}
\]

\[
x \in \{0, 1\}^{|A|} \tag{1g}
\]

\[
\gamma \in \{0, 1\}^{|N|} \tag{1h}
\]

Constraints (1a) define \( \gamma \) and force that the truck enters and leaves any visited node exactly once. Constraints (1b) enforce subtour elimination as modeled by [3], while (1c)–(1d) are symmetry breaking constraints. Constraint (1e) establishes that at least half of the nodes must be visited by the truck, since otherwise there exists no feasible drone solution. Finally, constraint (1f) guarantees that the depot is always visited. For simplicity, any feasible vector \((x, \gamma)\) is referred as a “truck route”.
3.2 Drone model

For each truck route \((x, \gamma)\), we define the set \(\mathcal{Y}(x)\) containing all feasible drone plans. Specifically, we define variables \(y_{ij} \in \{0, 1\}\) taking value 1 if and only if the drone traverses arc \((i, j)\) on board the truck; \(u_{ij} \in \{0, 1\}\) taking value 1 if and only if operation \((i, k, j)\) is executed; and \(w_{ikj} \in \mathbb{R}_+^+\) that registers the truck’s delay in operation \((i, k, j)\). Then, \(\mathcal{Y}(x)\) is formally defined as the set of vectors \((f, o, w, \gamma, u, v, c)\) for which there exists a vector \((\gamma, y, u, v, c)\) such that

\[
y_i = \sum_{a \in \partial^+(i)} x_a \quad i \in N \tag{2a}
\]

\[
u_i - u_{ij} + 1 \leq n \cdot (1 - y_{ij}) \quad (i, j) \in A : j \neq 1 \tag{2b}
\]

\[
u_1 = 1 \tag{2c}
\]

\[
2y_i \leq u_i \leq n \cdot y_i \quad i \in N \setminus \{1\} \tag{2d}
\]

\[
\sum_{i \in N \setminus \{j\}} y_{ij} = u_j - 1 \quad j \in N \tag{2e}
\]

\[
y_{ij} + y_{ji} \leq \min\{y_i, y_j\} \quad i, j \in N : i \neq j \tag{2f}
\]

\[
y_{ij} + y_{ji} \leq y_i + y_j \quad i, j \in N : i \neq j \tag{2g}
\]

\[
v_i - v_j + c_{ij} \leq M \cdot (1 - y_{ij}) \quad (i, j) \in E : j \neq 1 \tag{2h}
\]

\[
v_i \leq \sum_{a \in A} x_a c_a - \sum_{a \in \partial^-(i)} x_a c_a \quad i \in N \setminus \{1\} \tag{2i}
\]

\[
u_1 = 0 \tag{2j}
\]

\[
\hat{c}_{ij} \leq v_j - v_i + M \cdot (1 - y_{ij}) \quad i, j \in N : i \neq j \tag{2k}
\]

\[
\hat{c}_{ij} \leq M \cdot y_{ij} \quad i, j \in N : i \neq j \tag{2l}
\]

\[
f_{ij} \leq x_{ij} \quad (i, j) \in A \tag{2m}
\]

\[
\sum_{k \in N \setminus \{i, j\} \in O} o_{ikj} \leq y_{ij} \quad i, j \in N : i \neq j, j \neq 1 \tag{2n}
\]

\[
\sum_{(i, k, j) \in O} o_{kj} = 1 - y_k \quad k \in N \tag{2o}
\]

\[
\sum_{e \in \partial^+(i)} o_e \sum_{a \in \partial^+(i)} f_a = \sum_{e \in \partial^-(i)} o_e \sum_{a \in \partial^+(i)} f_a \quad i \in N \tag{2p}
\]

\[
\sum_{e \in \partial^+(i)} o_e \sum_{a \in \partial^+(i)} f_a = 1 \tag{2q}
\]

\[
w_{ikj} \geq (d_{ik} + d_{kj}) o_{ikj} - \hat{c}_{ij} \quad (i, k, j) \in O : (i, j) \neq (1, 1) \tag{2r}
\]

\[
w_{ikj} \geq (d_{ik} + d_{kj}) o_{ikj} - \sum_{a \in A} c_a x_a \quad k \in N \setminus \{1\} \tag{2s}
\]

Constraints \((2a)\) recover \(\gamma\) from an \(x\) vector. Constraints \((2b) - (2d)\) register in variable \(u_i\) the position of each node visited by the truck in its route relative to the depot and set to zero all \(u_i\) values of nodes visited by the drone; these constraints are proposed in the seminal work of [13]. Constraints \((2e) - (2g)\) record in each binary variable \(y_{ij} \in \{0, 1\}\) whether node \(j\) is visited by the truck after node \(i\); otherwise, the variable takes the value 0.
Constraints (2h–2l) compute in each variable \( x_{ij} \) the cumulative duration of the truck’s route from node 1 to node \( i \). Constraints (2k–2l) compute in each variable \( x_{ij} \) the duration of the partial truck route from node \( i \) to \( j \); \( x_{ij} \) set to 0 when \( y_{ij} = 0 \). In the formulation, we use \( M \equiv \min \{ \sum_{e \in A} c_{e} x_{e} : y = 1, (x, y) \in X \} \).

Constraints (2m) allow the drone to travel on board arc \((i, j)\), registered in binary variable \( f_{ij} \), only if this arc is traversed by the truck. Constraints (2n) allow operation \( e = (i, k, j) \) to be executed, registered in binary variable \( o_{e} = o_{ikj} \), only if node \( i \) is visited by the truck before node \( j \). Constraints (2o) enforce that all customers left unvisited by the truck are covered by the drone. Constraints (2p–2q) enforce a single drone flow conservation over the set of nodes nodes. Finally, constraints (2r–2s) store the truck’s delay in operation \((i, k, j)\) in variable \( w_{ikj} \), whose value is no smaller than both the drone’s direct dispatch travel time to \( k \) and the duration of the truck’s partial route from \( i \) to \( j \).

Although formulation (3) might seem somehow complicated, it is important to note that constraints (2a–2l) fix the values of \((y, y, u, v, \ell)\) for any vector \( x \in [0,1]^{A} \) defining a truck route, and thus the only variables are precisely \((f, o, w)\).

### 3.3 Two–stage model

The proposed decomposition is a two–stage MIP model. In the first–stage model (5), we design a truck route \((x, y)\) visiting a subset \( C \subseteq N \) of nodes. The objective function in (3) is to minimize the total truck travel time \( c^T x \) plus its total delay \( Q(x) \) due to drone synchronizations restrictions.

\[
\min \{ c^T x + Q(x) : (x, y) \in X \}.
\] (3)

The second–stage model (4) takes \((x, y)\) as an input and looks for a drone plan having minimum truck delay and visiting all nodes in \( N \setminus C \).

\[ Q(x) = \min \{ w : (f, o, w) \in Y(x) \} \] (4)

### 3.4 Overall solution strategy

We solve the TSP–D implementing an Integer L-Shaped Method (see [12]) adapted to our setting. Specifically, we relax model (3) and replace \( Q(x) \) with an underestimate \( \Theta \geq 0 \), obtaining

\[
\min \{ c^T x + \Theta : (x, y) \in X, \theta \geq 0 \}.
\] (5)

An optimal solution \((x, y, \Theta)\) to model (5) can be leveraged to obtain a feasible solution \((\hat{x}, \hat{y}, Q(\hat{x}))\) and produce a valid upper bound \( c^T \hat{x} + Q(\hat{x}) \) to the optimal value of model (3). If \( \hat{\Theta} < Q(\hat{x}) \), then solution \((\hat{x}, \hat{y}, \hat{\Theta})\) produces an optimality cut defined as

\[
\Theta \geq -Q(\hat{x}) \left[ \sum_{a \in A: x_a \neq 1} (1 - x_a) + \sum_{a \in A: x_a = 0} x_a \right] + Q(\hat{x}),
\] (6)

which is used to refine the relaxed model (5). When \((x, y) = (\hat{x}, \hat{y})\), then (6) reduces to \( \Theta \geq 0 \). In any other case, \( \sum_{a \in A: x_a = 1} (1 - x_a) + \sum_{a \in A: x_a = 0} x_a = m \geq 1 \) and (6) reduces to \( \Theta \geq (1 - m) \cdot Q(\hat{x}) \), where the right hand side is non–positive. Therefore, (6) is valid for each route \((x, y) \in X \).

Although (6) could be complemented with standard Benders’ optimality cuts obtained from the linear relaxation of the second–stage problem (4), our initial computational experience indicated that this approach was inefficient in our setting and was left out of further consideration.

A pseudo–code of the solution approach is presented as follows. Algorithm 1 details the related B&C procedure. Given a solution \((\hat{x}, \hat{y}, \hat{\Theta})\) to the linear relaxation of (5), if it is integer and improves the incumbent \((\hat{x}, \hat{y}, \hat{\Theta})\), i.e., \( c^T \hat{x} + \hat{\Theta} > c^T \hat{x} + \hat{\Theta} \), then the Optimality cut function is invoked. Algorithm 2 details the Optimality cut function, which checks if adding an optimality cut (6) to the linear relaxation of (5) is required, every time an integer candidate solution is found. To avoid redundant computation, we keep a list \( V \) of candidate solutions \((x, y)\) for which \( Q(x) \) has already been computed.

### 4 Properties

We now present several properties satisfied by at least one optimal solution to the TSP–D, which will prove useful in strengthening the above two–stage formulation.

**Proposition 1.** There exists an optimal solution for the TSP–D in which the drone does not traverse two consecutive arcs on board the truck.
Algorithm 1: Decomposition method to solve the TSP–D

Input: \( N,A,c,d, \) upper bound \( UB > 0, \) list of tree nodes \( T = \{1\}, V = \emptyset \)

Output: Optimal solution \((x^*, y^*)\) of \((3)\)

1. while \( T \neq \emptyset \) do
2. 2. Pull node \( \sigma \) from \( T \) and solve the nodal linear relaxation to model \((5)\);
3. 3. if \( \sigma \) is infeasible then
4. 4. Prune node \( \sigma \);
5. else
6. 6. Retrieve solution \((\hat{x}, \hat{y}, \hat{\theta})\);
7. 7. if \( c^T \hat{x} + \hat{\theta} \geq UB \) then
8. 8. Prune node \( \sigma \);
9. else if \((\hat{x}, \hat{y})\) is fractional then
10. 10. Branch and add two new nodes to \( T \);
11. 11. Remove node \( \sigma \) from \( T \);
12. else
13. 13. Feasible ← OptCut();
14. if Feasible then
15. 15. \((x^*, y^*, \theta^*) ← (\hat{x}, \hat{y}, \hat{\theta})\);
16. 16. \( UB ← c^T \hat{x} + \hat{\theta} \);
17. 17. Prune node \( \sigma \);
18. return \((x^*, y^*)\);

Algorithm 2: Optimality cut function

Input: Candidate integer solution \((\hat{x}, \hat{y}, \hat{\theta})\), list of known candidates solutions \( V \).

Output: \( \text{Feasible} \) if solution is feasible in \((3)\), \( \text{Infeasible} \) in other case.

1. if \((\hat{x}, \hat{y})\) ∈ \( V \) then
2. 2. return Feasible;
3. else
4. 4. \( V ← V \cup \{(\hat{x}, \hat{y})\} \);
5. 5. Compute \( Q(\hat{x}) \);
6. 6. if \( \hat{\theta} < Q(\hat{x}) \) then
7. 7. Add optimality cut \((6)\) to the nodal linear relaxation of model \((5)\);
8. 8. return \( \text{Infeasible} \);
9. else
10. 10. return \( \text{Feasible} \);

Proof. Consider any partial truck route of three consecutive node visits \( \{i,k,j\} \). We have \( d \leq c \) and \( c \), being metric, satisfies the triangle inequality. Therefore, we obtain

\[
\max\{d_{ik} + d_{kj}, \ c_{ij}\} \leq \max\{c_{ik} + c_{kj}, \ c_{ij}\} = c_{ik} + c_{kj}, \quad (7)
\]

and conclude that the truck’s total travel and waiting time incurred by a partial truck subroute \( \{i,j\} \) while executing operation \((i,k,j)\) does not exceeds the travel time of partial route \( \{i,k,j\} \).

Corollary 1. For the complete digraph \( G = (N,A) \) with \( n \geq 3 \), there exists an optimal solution to the TSP–D executing at least one drone dispatch.

Proposition 2 below strengthens Proposition 1 and proves pivotal in simplifying our formulation.

Proposition 2. There exists an optimal solution to the TSP–D in which all operations are planned consecutively over the truck’s route.
Proof. Consider the set of optimal solutions having a pair of non–consecutive operations. If this set is empty, we are done. Otherwise, choose one such solution in which the drone traverses an arc \((i,j)\) on board the truck between two operations, as depicted in Figure 3. Note that Proposition 1 guarantees the existence of such an optimal solution.

The waiting time incurred by the truck in the first operation is
\[
    w_1 = (d_1 + d_2 - \hat{c})_+,
\]
where \(d_1 + d_2\) is the duration of the drone dispatch to a node \(k\) and \(\hat{c}\) is the travel time of the related truck partial route. If \(d_3\) is the drone travel time from node \(k\) to \(j\), then
\[
    w_1 = (d_1 + d_2 - \hat{c})_+ \\
    \geq (d_1 + d_2 + d_{ij} - (\hat{c} + c_{ij}))_+ \\
    \geq (d_1 + d_3 - (\hat{c} + c_{ij}))_+ \\
    = w_2,
\]
where \(w_2\) is the waiting time incurred by the truck if the first operation is extended to node \(j\). Extending all non–consecutive operations this way, we obtain an optimal solution with the desired structure. 

Corollary 2. There exists an optimal solution to the TSP–D in which the drone takes off and lands at the depot (in potentially different operations).

In view of Proposition 2, we can assume without loss of optimality that the drone does not traverse any arc on board the truck, which implies that variables \(f\) can be set equal to zero in system (2) defining \(\Upsilon(x)\). In this case, constraint (2q) implies that at least one operation departing from the depot is executed, which is consistent with Corollaries 1 and 2. In turn, this implies that routes that cover all nodes in the first–stage problem can be removed from the search space, which can be accomplished by including the constraint
\[
    \sum_{i \in N} \gamma_i \leq n - 1
\]
in formulation 4 of \(X\).

4.1 Lower bound on waiting time

Now, consider a truck route \(R_x = [r_0 = 1, r_1, \ldots, r_{|R| - 1}, r_{|R|} = 1]\) defined by \((x, y) \in X\). Let \(D_x \subset A\) be the set of arcs in a drone plan minimizing the waiting time with respect to \(R_x\), and let \(O_x \equiv \{(i, k, j) \in O : (i, k), (k, j) \in D_x\}\) be the set of operations defined by \(D_x\). Proposition 3 presents a lower bound to the optimal value of the TSP–D.

Proposition 3. The total travel and waiting time duration of a truck route \((x, y) \in X\) is bounded below by both the travel time duration of \(R_x\) and \(D_x\), i.e.,
\[
    \max \left\{ \sum_{a \in A} c_a x_a, \sum_{a \in D_x} d_a \right\} \leq \sum_{a \in A} c_a x_a + Q(x). \tag{9}
\]
\[\begin{align*}
\max \left\{ \sum_{a \in A} c_a x_a, \sum_{a \in D_x} d_a \right\} &= \sum_{a \in A} c_a x_a + \left( \sum_{a \in D_x} d_a - \sum_{a \in A} c_a x_a \right) \\
&= \sum_{a \in A} c_a x_a + \left( \sum_{(i,k,j) \in O_x} (d_{ik} + d_{kj} - \xi_{ij}(R_x)) \right) \\
&\leq \sum_{a \in A} c_a x_a + \left( \sum_{(i,k,j) \in O_x} (d_{ik} + d_{kj} - \xi_{ij}(R_x)) \right) \\
&= \sum_{a \in A} c_a x_a + \sum_{(i,k,j) \in O_x} (1) \\
&= \sum_{a \in A} c_a x_a + Q(x),
\end{align*}\]

where the second equality follows from Proposition 2 \[
\text{and the inequality from the subadditivity of the function } (1_+) \]

Proposition 3 also provides a lower bound to the total truck waiting time. Given \((x, y) \in X, D_x\) defines a drone cycle covering the depot and alternating customer nodes in \(R_x\) and \(N \setminus R_x\). Then, a minimum duration cycle satisfying these constraints using drone travel times is less restrictive than a drone plan and, thus, is a lower bound for \(\sum_{a \in D_x} d_a\) in Proposition 3.

We can model all drone cycles dispatched from the depot and alternating customer nodes in \(R_x\) and \(N \setminus R_x\) via MIP by defining a variable \(s_a \in [0, 1]\) for each \(a \in A\), which is equal to 1 if and only if arc \(a\) is part of the drone cycle. Moreover, let \(\lambda_a \in [0, 1]\), which is equal to 1 if and only if \(s_a = 1\) and \(a\) has its tail at a node in \(R_x\), and let \(\mu_a \in [0, 1]\), which is equal to 1 if and only if \(s_a = 1\) and \(a\) has its head at a node in \(R_x\). Then, the set \(\gamma(y)\) defined by (10) defines all feasible vectors \(s\).

\[\begin{align*}
s_a &= \lambda_a + \mu_a & a \in A \tag{10a} \\
\sum_{a \in \delta^-(i)} \lambda_a &= \sum_{a \in \delta^+(i)} \mu_a &= 1 - y_i & i \in N \tag{10b} \\
\sum_{a \in \delta^-(i)} \lambda_a &= \sum_{a \in \delta^+(i)} \mu_a & i \in N \tag{10c} \\
\sum_{a \in \delta^+(i)} \lambda_a &\leq y_i & i \in N \tag{10d} \\
\sum_{a \in \delta^+(1)} \lambda_a &= 1 \tag{10e} \\
y'_i &= \sum_{a \in \delta^+(i)} s_a & i \in N \tag{10f} \\
\sum_{a \in \delta^+(S)} s_a &\geq \max_{i \in S} \{y'_i\} & \emptyset \neq S : 1 \notin S \subset N \tag{10g} \\
s_{ij} &\leq \sum_{(i,1) \in \delta^+(1): i \leq j} s_{i1} & (1, j) \in \delta^+(1) \tag{10h} \\
s_{ij} &\leq \sum_{(1,j) \in \delta^-(1): i \leq j} s_{1j} & (i, 1) \in \delta^-(1) \tag{10i} \\
s_a, \lambda_a, \mu_a &\in [0, 1] & a \in A \tag{10j} \\
y'_i &\in [0, 1] & i \in N \tag{10k}
\end{align*}\]

Constraints (10a) set \(s\) equal to the sum of \(\lambda\) and \(\mu\). Constraints (10b)–(10c) guarantee a drone route alternating between truck route nodes and drone nodes in \(N \setminus R_x\), while guaranteeing that all drone nodes are visited as well. Constraints (10d) enforce to leave the truck route via a \(\lambda\) variable and (10e) guarantees to visit the depot. Constraints (10f) compute a \(y'\) variable vector analogously to \(y\) in (1a) for this drone route. Finally, constraints (10g) work as sub-tour elimination constraints and (10h)–(10i) remove some symmetric solutions.
Figure 4 illustrates how this approximation works for a particular instance. An optimal solution to the TSP–D solution is depicted in Figure 4a, where the truck route covers $C = \{2, 3, 4, 5, 6, 7, 12, 13, 15\}$ and the optimal drone route covers $N \setminus C$. Figure 4b displays a drone route approximation feasible to $\mathcal{W}(\hat{\gamma})$ as defined in (10) having minimum travel time. Compared to the original drone route, this route does not follow the truck route sequence.

Figure 4: Example of the approximation proposed in (11)

Define

$$\bar{s}(\hat{\gamma}) \in \arg\min_{s \in \mathcal{W}(\hat{\gamma})} \left\{ \sum_{a \in A} d_a s_a \right\}$$

(11)

as a minimum duration drone route approximation given the set of customers visited in a truck route $(\hat{x}, \hat{\gamma}) \in X$. Proposition 4 defines a lower bound on the truck’s optimal waiting time, which is independent of the route sequence in $x$.

**Proposition 4.** For any given set of customers visited in a truck route $(\hat{x}, \hat{\gamma})$ encoded in vector $\hat{\gamma}$, the inequality

$$\left( \sum_{a \in A} d_a s_a (\hat{\gamma}) - c_a x_a \right) \geq Q(x),$$

(12)

is valid for at least one optimal solution to the TSP–D.

**Proof.** Any $(\hat{x}, \hat{\gamma})$ satisfies $\sum_{a \in A} d_a s_a (\hat{\gamma}) \leq \sum_{a \in D_x} d_a$. Then, we use Proposition 3 to obtain

$$\left( \sum_{a \in A} d_a s_a (\hat{\gamma}) - c_a x_a \right) = \max \left\{ \sum_{a \in A} c_a x_a, \sum_{a \in A} d_a s_a (\hat{\gamma}) \right\} - \sum_{a \in A} c_a x_a$$

$$\leq \max \left\{ \sum_{a \in A} c_a x_a, \sum_{a \in D_x} d_a \right\} - \sum_{a \in A} c_a x_a$$

$$\leq Q(x).$$

4.2 $t$–reduction

Now, we lower bound the waiting time of a specific neighbourhood of routes for a given truck route $(\hat{x}, \hat{\gamma}) \in X$ called $t$–reduction. We start by defining an additional concept referred as $t$–shortcut.

**Definition 1.** $t$–shortcut

Given a truck route $(\hat{x}, \hat{\gamma}) \in X$ and $t \in \{1, \ldots, \hat{\gamma}^{\max}\}$, where $\hat{\gamma}^{\max} \equiv \sum_{i \in N} \hat{\gamma}_i - \left\lceil \frac{n}{2} \right\rceil$, we define the $t$–shortcut of $(\hat{x}, \hat{\gamma})$, denoted $S_t(\hat{x}, \hat{\gamma})$, as the set of arcs $(i, j) \in A$ such that $i$ and $j$ are non–consecutive nodes in the truck route, node $i$ is visited by the truck before node $j$, and there are up to $t$ other nodes in the truck route between $i$ and $j$. Formally,

$$S_t(\hat{x}, \hat{\gamma}) \triangleq \left\{ (r_i, r_j) \in \mathcal{R}_x^2 : (r_i, r_j) \in A, 1 \leq j - i - 1 \leq t \right\}.$$
Figure 5 illustrates all elements in a 2–shortcut with dashed and dotted arcs for a specific truck route painted by solid arcs.

![Figure 5: Arcs in a 2–shortcut](image)

Proposition 5 presents an inequality that any t–shortcut satisfies.

**Proposition 5.** Given a route $(\hat{x}, \hat{y}) \in X$ and $t \in \{1, \ldots, t_{\text{max}}^\hat{y}\}$, any route $(x, y) \in X$ satisfies

$$\sum_{a \in A: \hat{x}_a = 1} (1 - x_a) - \sum_{a \in S(\hat{x}, t)} x_a - \sum_{i \in N: \hat{y}_i = 1} (1 - y_i) \geq 0.$$  \hfill (14)

**Proof.** Let $(x, y) \in X$. Inequality (14) is equivalent to

$$\sum_{a \in A: \hat{x}_a = 1} x_a + \sum_{a \in S(\hat{x}, t)} x_a \leq \sum_{i \in N: \hat{y}_i = 1} y_i.$$  \hfill (15)

The left hand side of (15) registers the number of arcs common to $(\hat{x}, \hat{y})$ and $(x, y)$ plus the number of arcs common to the t–shortcut $S(\hat{x}, t)$ and $(x, y)$. The right hand side registers the number of customers common to $R_x$ and $R_{\hat{x}}$.

Let $U \subseteq N$ and define $\Omega(U) \equiv \{(i, j) \in E : i, j \in U\}$ as the set of arcs with head and tail in $U$. Since $x$ defines a cycle, we have $\sum_{a \in \Omega(U)} x_a \leq |U|$ for all $U \subseteq N$. In particular, if $U = R_x \cap R_{\hat{x}}$, we have

$$\sum_{a \in \Omega(R_x \cap R_{\hat{x}})} x_a \leq |R_x \cap R_{\hat{x}}| = \sum_{i \in N: \hat{y}_i = 1} y_i.$$  \hfill (16)

Also, we have

$$\Omega(R_x \cap R_{\hat{x}}) = \{(i, j) \in E : \hat{x}_{ij} = 1, i, j \in R_x\}$$

$$\cup \{(i, j) \in S(\hat{x}, t) : i, j \in R_x\}$$

$$\cup \{(i, j) \in A \setminus S(\hat{x}, t) : \hat{x}_{ij} = 0, i, j \in R_x \cap R_{\hat{x}}\},$$

and thus we obtain

$$\sum_{a \in A: \hat{x}_a = 1} x_a + \sum_{a \in S(\hat{x}, t)} x_a \leq \sum_{a \in \Omega(R_x \cap R_{\hat{x}})} x_a.$$  \hfill (17)

The result follows by combining (16) and (17).

**Definition 2.** t–reduction

*Given a truck route $(\hat{x}, \hat{y}) \in X$ and $t \in \{1, \ldots, t_{\text{max}}^\hat{y}\}$, we define the t–reduction of $(\hat{x}, \hat{y})$, denoted $T(\hat{x}, t)$, as the set of routes built by joining partial sub–routes of $R_{\hat{x}}$ in order. In other words, a route $(x, y)$ is in $T(\hat{x}, t)$ if and only if covers $R_x \subseteq R_{\hat{x}}$ by only using arcs in $\{a \in A : \hat{x}_a = 1\} \cup S(\hat{x}, t)$.*

Figure 6 presents routes within the 2–reduction of the route example presented in Figure 5.
Figure 6: Example of routes in a 2-reduction

For each route \((x, \gamma)\) in the \(t\)-reduction of \((\hat{x}, \hat{\gamma})\), we have that \(\sum_{a \in A : \hat{x}_a = 1} (1 - x_a)\) registers the number of removed solid arcs from the original route \((\hat{x}, \hat{\gamma})\); \(\sum_{a \in S(t)} x_a\) computes the number of arcs selected from the \(t\)-shortcut of the original route; and \(\sum_{i \in N : \hat{\gamma}_i = 1} (1 - \gamma_i)\) registers the number of customer visits skipped from the original route. Proposition 6 relates these quantities.

**Proposition 6.** Given \((\hat{x}, \hat{\gamma}) \in \mathcal{X}\) and \(t \in \{1, \ldots, t_{\text{max}}\}\), a route \((x, \gamma) \in \mathcal{X}\) belongs to \(T_{\hat{x}}(t)\) if and only if it satisfies

\[
\sum_{a \in A : \hat{x}_a = 1} (1 - x_a) - \sum_{a \in S(t)} x_a - \sum_{i \in N : \hat{\gamma}_i = 1} (1 - \gamma_i) = 0. \tag{18}
\]

**Proof.** We first prove that if \((x, \gamma) \in T_{\hat{x}}(t)\), then it satisfies (18). Let \(U \subseteq \mathcal{R}_x\). We know that \(\sum_{a \in \Omega(U)} x_a = |U|\) if and only if \(U = \mathcal{R}_x\). We also know that \(S_{\hat{x}}(t) \subseteq \{a \in A : \hat{x}_a = 0\}\). By definition, if \((x, \gamma) \in T_{\hat{x}}(t)\), then it satisfies

\[
\sum_{a \in A : \hat{x}_a = 0} = 0.
\]

Thus, we have that

\[
\sum_{a \in A : \hat{x}_a = 1} x_a + \sum_{a \in S(t)} x_a + \sum_{a \in A \setminus S(t) : \hat{x}_a = 0} x_a = \sum_{a \in A} x_a = \sum_{a \in \Omega(\mathcal{R}_x)} x_a = |\mathcal{R}_x| = |\mathcal{R}_x \cap \mathcal{R}_{\hat{x}}| = \sum_{i \in N : \hat{\gamma}_i = 1} \gamma_i,
\]

and thus \((x, \gamma)\) satisfies (18).

We now prove that if \((x, \gamma)\) satisfies (18), then it belongs to \(T_{\hat{x}}(t)\). Equivalently, we prove that if \((x, \gamma) \in \mathcal{X} \setminus T_{\hat{x}}(t)\), then it does not satisfy (18). If \((x, \gamma) \in \mathcal{X} \setminus T_{\hat{x}}(t)\), then we have

\[
\sum_{a \in A \setminus S(t) : \hat{x}_a = 0} x_a \geq 1.
\]
Moreover, we have
\[ \sum_{a \in \delta^+(R_x \cap R_k)} x_a + \sum_{a \in \Omega(R_x \cap R_k) \cap S_x(t)} x_a \geq 1. \]

Hence, considering Proposition 5 and the fact that
\[ \sum_{a \in A: x_a = 1} x_a + \sum_{a \in \delta^+(R_x \cap R_k)} x_a + \sum_{a \in \Omega(R_x \cap R_k) \cap S_x(t)} x_a = \sum_{i \in N: \hat{y}_i = 1} \sum_{a \in \delta^+(i)} x_a , \]
any route \((x, y) \in \mathcal{X} \setminus T_x(t)\) satisfies
\[ \sum_{a \in A: x_a = 1} x_a + \sum_{a \in \delta^+(R_x \cap R_k)} x_a \leq \sum_{i \in N: \hat{y}_i = 1} (1 - \gamma_i) - 1. \quad (19) \]

Re-arranging (19), we obtain
\[ \sum_{a \in A: x_a = 1} (1 - x_a) - \sum_{a \in \delta^+(R_x \cap R_k)} x_a = \sum_{i \in N: \hat{y}_i = 1} (1 - \gamma_i) \geq 1. \]

As a consequence of Proposition 6, equivalently, we can define the \(t\)-reduction of \((x, \hat{y})\) as
\[ T_x(t) \triangleq \left\{ (x, y) \in \mathcal{X} : \sum_{a \in A: x_a = 1} (1 - x_a) - \sum_{a \in \delta^+(R_x \cap R_k)} x_a - \sum_{i \in N: \hat{y}_i = 1} (1 - \gamma_i) = 0 \right\}. \]

The following key result states that all routes in the \(t\)-reduction of \((x, \hat{y}) \in \mathcal{X}\) produce no less truck waiting time than \((x, \hat{y})\).

**Proposition 7.** Given a truck route \((x, \hat{y}) \in \mathcal{X}\) and \(t \in \{1, \ldots, t^{\text{max}}_y\}\), the inequality
\[ Q(x) \geq Q(\hat{x}) \quad (20) \]
holds for any route \((x, y) \in T_x(t)\).

**Proof.** The result is trivial when \((\hat{x}, \hat{y}) = (x, y)\), thus we assume that \((\hat{x}, \hat{y}) \neq (x, y)\). Any route \((x, y) \in T_x(t) \setminus \{ (x, \hat{y}) \}\) is built by removing, at least, one customer node in \(R_x\). Therefore, we have \(\hat{c}_{ij}(\hat{R}_x) \leq \hat{c}_{ij}(R_x)\) for all \(i, j \in R_x\). This implies that \(w_{ikj}(R_x) \geq w_{ikj}(\hat{R}_x)\) for all \(i, j \in R_x\) and \(k \in N \setminus R_x\). We obtain
\[
Q(x) = \sum_{(i, k, j) \in \Omega_x} w_{ikj}(R_x) \\
= \sum_{(i, k, j) \in \Omega_x: k \in R_x} w_{ikj}(R_x) + \sum_{(i, k, j) \in \Omega_x: k \in N \setminus R_x} w_{ikj}(R_x) \\
\geq \sum_{(i, k, j) \in \Omega_x: k \in N \setminus R_x} w_{ikj}(R_x) \\
\geq \sum_{(i, k, j) \in \Omega_x: k \in N \setminus R_x} w_{ikj}(\hat{R}_x) \\
\geq Q(\hat{x}).
\]

Thus, \((x, y)\) satisfies (20). \(\Box\)

### 5 Method improvements

We now present two improvements on the first-stage estimate of the total truck waiting derived from the properties discussed in the previous section.
5.1 Desynchronization bound

For any truck route \((x, \gamma) \in \mathcal{X}\), Proposition \[3\] guarantees that there exists a drone route approximation \(s \in \mathcal{V}(\gamma)\) such that

\[
\sum_{a \in A} d_s x_a - c_a x_a \leq Q(x). \tag{21}
\]

In particular, we can add such a constraint and variable \(s \in \mathcal{V}(\gamma)\) to the relaxed first-stage model \[5\], obtaining the improved relaxation

\[
\min \left\{ c^T x + \theta : x \in \mathcal{X}, \theta \geq 0, \theta \geq \sum_{a \in A} d_s x_a - c_a x_a, s \in \mathcal{V}(\gamma) \right\} \tag{22}
\]

which is used in Algorithms \[1\] and \[2\] instead of model \[5\]. Without loss of optimality, we can add the corrective cut

\[
\sum_{a \in A \setminus \hat{g}} (1 - s_a) \leq \Delta(\gamma, \hat{\gamma}) \cdot \sum_{a \in A} g_a(\hat{\gamma}), \tag{23}
\]

to the relaxed model \[22\], where \(\Delta(\gamma, \hat{\gamma}) \triangleq \sum_{i \in N} \hat{\gamma}_i (1 - \gamma_i) + \sum_{i \in N : \gamma_i = 0} \gamma_i\). This is a direct result of Proposition \[4\].

This cut fixes values in \(s\) for a given \(\gamma\) vector. We present the pseudo-code implementation of this function in Algorithm \[3\] which must be invoked before the Optimality cut function in Algorithm \[1\].

Algorithm 3: Corrective cut function

\begin{algorithm}
\caption{Corrective cut function}
\begin{algorithmic}
\IF {\((x, \gamma, \hat{\gamma})\) candidate integer solution, \(V\) list of known candidates solutions.}
\IF {\((x, \gamma, \hat{\gamma}) \in V\) such that \(\gamma = \hat{\gamma}\)}
\STATE Compute \(g(\hat{\gamma})\);
\STATE Add \[23\] to the nodal linear relaxation of model \[22\];
\ENDIF
\ENDIF
\end{algorithmic}
\end{algorithm}

5.2 Improved Optimality Cut

Given a truck route \((x, \gamma) \in \mathcal{X}\), we have that any other route \((x, \gamma) \in \mathcal{X}\) satisfies

\[
\sum_{a \in A : x_a = 1} (1 - x_a) \geq 1.
\]

Therefore, we may lift the optimality cut defined in \[6\] obtaining

\[
\theta \geq -Q(\hat{x}) \sum_{a \in A : x_a = 1} (1 - x_a) + Q(\hat{x}). \tag{24}
\]

Moreover, Propositions \[5\] and \[7\] directly imply that

\[
\theta \geq -Q(\hat{x}) \left[ \sum_{a \in A : x_a = 1} (1 - x_a) - \sum_{a \in \hat{S}_i} x_a - \sum_{i \in N : \hat{\gamma}_i = 0} (1 - \gamma_i) \right] + Q(\hat{x}) \tag{25}
\]

is also a valid inequality for a fixed \(t \in \{1, \ldots, t_{\hat{\gamma}}^{\max}\}\). The cut defined in \[25\] reduces to \(\theta \geq Q(\hat{x})\) when \((x, \gamma) \in T_{\hat{\gamma}}(t)\), which is valid due to \[20\]. Also, its right hand side is non-positive for any truck route \((x, \gamma) \in \mathcal{X} \setminus T_{\hat{\gamma}}(t)\).

6 Numerical experiments

We empirically validate our proposed method and improvements by solving a family of computationally simulated instances. Our algorithm is implemented in Python 3.7 calling the Gurobi 8.1.1 solver each time a MIP was solved, and executed in a computer cluster with an 8-core Intel–E5–2470 processor, 2.5 GHz CPU and 32 GB of RAM. The stopping criterion is optimality within the solver’s default tolerance or a time limit of 7200 seconds. We do not report results on the extended formulation \[3\] since it was clearly outperformed by the decomposition approach in our initial experiments.
6.1 Computational results

We designed a set of 20 network topologies, each having a set of 25 uniformly distributed customer nodes over a 100x100 square service region. Truck travel time \( c \) is set as the Euclidean distance vector truncated to one decimal and multiplied by 10 to obtain integer values, while relative drone speeds are set to \( \alpha \in \{1, 2, 4\} \). Each instance are produced by selecting a subset of nodes with cardinality \( n \in \{15, \ldots, 25\} \) for each graph.

To solve model (5), we use a multicommodity flow representation of \( X \); see [7]. Alternatively, we solve it by relaxing all constraints (1b) and adding them dynamically if violated. Our results for this last formulation are skipped in this article, since the multicommodity flow formulation produced the best results for all instances tested.

Table 2 presents results over all instances with \( \alpha = 2 \) and different sizes of \( n \). We include the initial algorithm proposed in Section 3.4, an improved version with the Desynchronization Bound (DB), one equipped with Improved Optimality Cuts (IOC), and one with both (DB + IOC). Column \#Opt presents the number of instances solved to optimality within 2 hours, column Runtime the average CPU time over all instances optimally solved, and column Gap the average optimality gap over all unsolved instances computed as

\[
\text{Gap} = 100 \cdot \frac{v_{UB} - v_{LB}}{v_{UB}},
\]

where \( v_{UB} \) is the objective value of the best solution found and \( v_{LB} \) is its best lower bound obtained. Running times are shown in format “hours : minutes : seconds”. Empirically, we observe that the method’s overall performance decreases as the instance size \( n \) increases. The number of optimally solved instances decreases and the computation time increases as \( n \) increases. This may be explained due to a higher number of feasible solutions and generated cuts, as shown in Tables 3 and 4. The execution of the decomposition method without further improvement runs out of memory for at least one instance of sizes 18, 20, 21 and 22. When compared to the original algorithm, both the DB and IOC improvements produce a performance increase, solving more instances to optimality and requiring less computation time. The increase is higher when both improvements are used.

Table 2: Computational results for \( \alpha = 2 \)

<table>
<thead>
<tr>
<th>( n )</th>
<th>No improvement</th>
<th>DB</th>
<th>IOC</th>
<th>DB + IOC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Opt</td>
<td>Runtime</td>
<td>Gap (%)</td>
<td>#Opt</td>
</tr>
<tr>
<td>15</td>
<td>20/20</td>
<td>00:13:12</td>
<td>–</td>
<td>20/20</td>
</tr>
<tr>
<td>16</td>
<td>16/20</td>
<td>00:27:38</td>
<td>12.88</td>
<td>20/20</td>
</tr>
<tr>
<td>17</td>
<td>15/20</td>
<td>00:24:24</td>
<td>14.52</td>
<td>20/20</td>
</tr>
<tr>
<td>18</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>19/20</td>
</tr>
<tr>
<td>19</td>
<td>5/20</td>
<td>00:42:33</td>
<td>11.56</td>
<td>19/20</td>
</tr>
<tr>
<td>20</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>18/20</td>
</tr>
<tr>
<td>21</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>12/20</td>
</tr>
<tr>
<td>22</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>6/20</td>
</tr>
<tr>
<td>23</td>
<td>0/20</td>
<td>–</td>
<td>22.23</td>
<td>2/20</td>
</tr>
<tr>
<td>24</td>
<td>0/20</td>
<td>–</td>
<td>25.96</td>
<td>1/20</td>
</tr>
<tr>
<td>25</td>
<td>0/20</td>
<td>–</td>
<td>25.54</td>
<td>0/20</td>
</tr>
</tbody>
</table>

*At least one instance runs out of memory

Tables 3 and 4 present the average generation of optimality and corrective cuts over all optimally solved instances. Column Cuts displays the average number of cuts added to the first–stage, and column Time displays the average computation time spent generating all these cuts over each instance. We empirically conclude that the use of DB considerably reduce the number of optimality cuts and the time spent generating them. Also, both DB and IOC, produce a synergy in the instances tested and reduce the number of generated cuts when compared to the other settings. As expected, when all instances are optimally solved, the average number of cuts increases as \( n \) increases.
Table 3: Optimality cut generation for $\alpha = 2$

<table>
<thead>
<tr>
<th>$n$</th>
<th>No improvement</th>
<th>DB</th>
<th>IOC</th>
<th>DB + IOC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cuts</td>
<td>Time</td>
<td>Cuts</td>
<td>Time</td>
</tr>
<tr>
<td>15</td>
<td>7,231</td>
<td>00:08:43</td>
<td>569</td>
<td>00:00:43</td>
</tr>
<tr>
<td>16</td>
<td>14,817</td>
<td>00:21:39</td>
<td>823</td>
<td>00:00:59</td>
</tr>
<tr>
<td>17</td>
<td>9,291</td>
<td>00:13:55</td>
<td>2,212</td>
<td>00:03:35</td>
</tr>
<tr>
<td>18</td>
<td>*</td>
<td>*</td>
<td>1,572</td>
<td>00:02:49</td>
</tr>
<tr>
<td>19</td>
<td>10,240</td>
<td>00:21:31</td>
<td>2,441</td>
<td>00:04:43</td>
</tr>
<tr>
<td>20</td>
<td>*</td>
<td>*</td>
<td>3,166</td>
<td>00:07:37</td>
</tr>
<tr>
<td>21</td>
<td>*</td>
<td>*</td>
<td>3,330</td>
<td>00:08:34</td>
</tr>
<tr>
<td>22</td>
<td>*</td>
<td>*</td>
<td>1,606</td>
<td>00:05:21</td>
</tr>
<tr>
<td>23</td>
<td>–</td>
<td>–</td>
<td>1,284</td>
<td>00:04:39</td>
</tr>
<tr>
<td>24</td>
<td>–</td>
<td>–</td>
<td>2,254</td>
<td>00:12:01</td>
</tr>
<tr>
<td>25</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

*At least one instance runs out of memory

Table 4: Corrective cut generation for $\alpha = 2$

<table>
<thead>
<tr>
<th>$n$</th>
<th>No improvement</th>
<th>DB</th>
<th>IOC</th>
<th>DB + IOC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cuts</td>
<td>Time</td>
<td>Cuts</td>
<td>Time</td>
</tr>
<tr>
<td>15</td>
<td>–</td>
<td>–</td>
<td>437</td>
<td>00:00:34</td>
</tr>
<tr>
<td>16</td>
<td>–</td>
<td>–</td>
<td>630</td>
<td>00:00:39</td>
</tr>
<tr>
<td>17</td>
<td>–</td>
<td>–</td>
<td>1,950</td>
<td>00:01:06</td>
</tr>
<tr>
<td>18</td>
<td>*</td>
<td>*</td>
<td>1,102</td>
<td>00:01:54</td>
</tr>
<tr>
<td>19</td>
<td>–</td>
<td>–</td>
<td>1,907</td>
<td>00:02:32</td>
</tr>
<tr>
<td>20</td>
<td>*</td>
<td>*</td>
<td>2,266</td>
<td>00:04:49</td>
</tr>
<tr>
<td>21</td>
<td>*</td>
<td>*</td>
<td>2,184</td>
<td>00:05:58</td>
</tr>
<tr>
<td>22</td>
<td>*</td>
<td>*</td>
<td>822</td>
<td>00:04:51</td>
</tr>
<tr>
<td>23</td>
<td>–</td>
<td>–</td>
<td>666</td>
<td>00:03:55</td>
</tr>
<tr>
<td>24</td>
<td>–</td>
<td>–</td>
<td>1,401</td>
<td>00:09:24</td>
</tr>
<tr>
<td>25</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

*At least one instance runs out of memory

Table 5 presents average results of our best method for three different values of $\alpha$. When the drone is relatively slower ($\alpha = 1$), the number of instances solved to optimality decreases, and average computation times increase. This is consistent with the fact that the average initial optimality gap, i.e., the percentage difference between the objective cost of the linear relaxation in the Branch–&–Bound root node and the optimal value, increases as the drone gets slower (see Figure 7). This occurs because the minimum truck waiting time of each feasible solution reduces as a function of the drone relative speed and, therefore, the initial first–stage solution in the relaxed model becomes tighter.

Compared to [18], which offers the most promising results reported in the literature for the TSP–D, our approach is competitive and simpler to implement than a B&P setting in which the search tree must be manually controlled. Instead, our method is implemented in a commercial solver using its built–in functions.
Table 5: Computational results of enhanced method DB + IOC

<table>
<thead>
<tr>
<th>n</th>
<th>α = 1</th>
<th>α = 2</th>
<th>α = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Opt</td>
<td>Runtime</td>
<td>Gap (%)</td>
</tr>
<tr>
<td>15</td>
<td>20/20</td>
<td>00:02:39</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>20/20</td>
<td>00:04:07</td>
<td>-</td>
</tr>
<tr>
<td>17</td>
<td>20/20</td>
<td>00:07:42</td>
<td>-</td>
</tr>
<tr>
<td>18</td>
<td>20/20</td>
<td>00:26:47</td>
<td>-</td>
</tr>
<tr>
<td>19</td>
<td>20/20</td>
<td>00:31:34</td>
<td>-</td>
</tr>
<tr>
<td>20</td>
<td>14/20</td>
<td>00:47:39</td>
<td>1.9</td>
</tr>
<tr>
<td>21</td>
<td>7/20</td>
<td>00:46:12</td>
<td>3.75</td>
</tr>
<tr>
<td>22</td>
<td>4/20</td>
<td>01:11:03</td>
<td>5.30</td>
</tr>
<tr>
<td>23</td>
<td>2/20</td>
<td>00:39:12</td>
<td>5.13</td>
</tr>
<tr>
<td>24</td>
<td>1/20</td>
<td>00:50:03</td>
<td>6.77</td>
</tr>
<tr>
<td>25</td>
<td>0/20</td>
<td>-</td>
<td>7.13</td>
</tr>
</tbody>
</table>

Figure 7: Average initial optimality gap for all instances of size \( n \in \{15,\ldots,19\} \)

Finally, Figure 8 reports results regarding our algorithm's speed to discover optimal solutions. It presents the cumulative percentage of optimal solutions found as a function of the computation time. Times are registered as the elapsed instance computation time occurred until the last incumbent update. These experiments were performed over 100 instances for each problem size \( n \in \{15,\ldots,20\} \), setting \( \alpha = 2 \). We observe that all optimal solutions for instances with \( n < 20 \) and for more than 90% of instances with 20 customers are obtained before 100 minutes.
7 Concluding remarks

We present a novel two-stage MIP formulation and an exact solution method based on a Benders-type decomposition algorithm designed to solve the TSP–D model. Our formulation is strengthened with several improvements derived from multiple structural properties of an optimal solution for the TSP–D. The performance of the proposed algorithm is also validated in a series of computationally simulated experiments and improved via two sets of valid inequalities.

Our formulation and solution strategy can also be adapted to cover multiple extensions of the TSP–D. For example, we may allow loop operations of the form \((i,k,i)\) while a stopped truck waits for the drone at node \(i\); this setting is studied in [6] and becomes relevant for instances with relatively higher drone speeds or truck accessibility constraints. To work with this additional flexibility, we may delete constraint (1e), and repair the TSP–D problem to allow multiple drone dispatches from one customer location. Also, we may impose autonomy constraints on the drone’s flight time as studied in [18]. In this setting, we may re–define \(O\) by removing infeasible operations to the second–stage model via an improved feasibility cut of the form:

\[
\sum_{a \in A} x_a \left(1 - x_a \right) - \sum_{a \in S(t)} x_a - \sum_{i \in N} \gamma_i \left(1 - y_i \right) \geq 1. \tag{26}
\]

Cut (26) may also be used for incomplete digraphs. Moreover, our methodology can be extended to the setting in which the truck carries and deploys multiple independent drones. This last setting has only been addressed heuristically within the literature [14][21] and becomes relevant to model real problems such as [8]. Further research may involve solving the Vehicle Routing Problem with Drone, in which a fleet of trucks is available, each carrying potentially multiple drones [23].

In general, the proposed method offers a general solution approach to sequencing problems having a two-stage decision scheme, in particular for those whose second–stage cost function is monotone with respect to structures such as \(t\)-reductions. Therefore, further research also includes adapting the proposed method to address other sequencing problems with a decomposable decision structure.

References


