Exact Methods for the Traveling Salesman Problem with Drone

Roberto Roberti*[1] and Mario Ruthmair†[2]

[1]Department of Supply Chain Analytics, Vrije Universiteit Amsterdam, 1081HV Amsterdam, the Netherlands
[2]Department of Statistics and Operations Research, University of Vienna, Vienna, Austria

November 1, 2019

Abstract

Efficiently handling last-mile deliveries becomes more and more important nowadays. Using drones to support classical vehicles allows improving delivery schedules as long as efficient solution methods to plan last-mile deliveries with drones are available. We study exact solution approaches for some variants of the traveling salesman problem with drone (TSP-D) in which a truck and a drone are teamed up to serve a set of customers. This combination of truck and drone can exploit the benefits of both vehicle types: the truck has a large capacity but usually low travel speed in urban areas; the drone is faster and not restricted to street networks, but its range and carrying capacity are limited. We propose a compact mixed-integer linear program (MILP) for several TSP-D variants that is based on timely synchronizing truck and drone flows; such a MILP is easy to implement but nevertheless leads to competitive results compared to the state-of-the-art MILPs. Furthermore, we introduce dynamic programming recursions to model several TSP-D variants. We show how these dynamic programming recursions can be exploited in an exact branch-and-price approach based on a set partitioning formulation and using ng-route relaxation, dual variable stabilization, and a three-level hierarchical branching. The proposed branch-and-price can solve instances with up to 39 customers to optimality outperforming the state-of-the-art by more than doubling the manageable instance size.

1 Introduction

In the last decade, delivery companies increasingly aim at reducing the time for transporting goods from source to destination to improve customer satisfaction, decrease drivers' travel times, and optimize companies' resources. Also non-profit organizations, e.g., in the humanitarian sector, naturally strive for faster delivery of supporting goods to beneficiaries. An important line of research investigates the potential of optimizing last-mile logistics, which is considered one of the bottlenecks in supply chains, see Savelsbergh and Van Woensel (2016) for an overview. A large variety of alternative last-mile transportation modes and concepts appear in the literature and (mostly) also in practice, e.g., (electric) bicycles, autonomous robot vehicles, and unmanned aerial vehicles (UAVs) aka drones. Especially drones seem to be an attractive way of accelerating last-mile package delivery since their speed is in general higher than the average speed of ground vehicles, they are not limited to the street network, and thus are not affected by (urban) congestion. For example, Amazon (Vincent and Gartenberg, 2019) and UPS (Peters, 2019) have been testing their drone prototypes for several years, but a wide deployment in practice requires appropriate safety regulations and governmental support.

A downside of current drone technology applicable in urban logistics is the limited carriage capacity and flying range which makes it difficult or in some cases even impossible to launch drones directly from central warehouses. To mitigate these disadvantages drones can be teamed up with ground delivery vehicles, e.g., classical trucks, carrying a large number of parcels and replacement batteries or charging facilities for the drones. The idea is that the truck serves as a base station for a single (or multiple) drone(s), delivers

*r.roberti@vu.nl
†mario.ruthmair@univie.ac.at

parcels on its own, and—whenever it is possible and economically viable—equips some drone with one (or multiple) package(s), sends it to one (or multiple) customer(s), and meets the drone again after finishing the delivery and before the drone's range limit is reached. Such a setting naturally leads to optimization problems asking when and for which customers to deploy drones to minimize some cost or time measure such that all customers are satisfactorily served.

1.1 Scientific Contributions and Overview

Many different optimization problems on the combined usage of trucks and drones can be found in the literature, see, e.g., the survey by Otto et al. (2018). In this work, we focus on the exact solution of several variants of the traveling salesman problem with drone (TSP-D) which aims at finding a route for a single truck hosting a single drone such that the total completion time needed to visit all customers (either by truck or drone) and return to the depot is minimized. It turns out that already such a simplified problem is extremely difficult to solve—state-of-the-art exact approaches can only handle instances with usually much less than 20 customers—even though the TSP-D is tightly related to the classical TSP for which large instances with thousands of nodes are manageable. Therefore, it is of paramount importance to investigate simple truck-and-drone problems with the prospect of applying gained knowledge and extending efficient solution approaches to more complicated problem variants. Our main scientific contributions are:

- We propose exact solution approaches for a basic variant of the TSP-D as defined in Section 2. Additionally, we extend the basic problem with several features commonly used in the literature: (*i*) drone launch and landing may happen at the same node (loops), (*ii*) a subset of customers must not be served by drone, (*iii*) the drone's flying range is limited by time or depends on the parcels' weights, (*iv*) the drone is not allowed to wait (for the truck) on the ground, (*v*) launch and rendezvous times for the drone have to be considered, and (*vi*) the maximum number of customers served by the truck alone while the drone is airborne is limited.

- We introduce a compact mixed-integer linear programming (MILP) formulation for all considered TSP-D variants. The formulation is intentionally kept simple without need for advanced computational techniques and thus is easy to implement within currently available solver frameworks. Despite its simplicity, it is already competitive with many exact methods from the literature. The main modeling idea is based on synchronizing two flows in time—the truck and the drone flow.

- We present a dynamic programming approach for all considered TSP-D variants based on a view somehow contrary to the compact formulation. Here, truck and drone are seen as a single entity that can be temporarily separated (when the drone services a customer alone). Additional resources in the state description keep track of a potential separation until truck and drone rejoin.

- A set partitioning formulation exploiting the same combined view is introduced and solved by a branch-and-price (BP) approach. The associated pricing problem is solved by the dynamic program mentioned above, accelerated by advanced techniques such as ng-route relaxation and dual variable stabilization. A three-level hierarchical branching strategy is applied to ensure a well-balanced branch-and-bound tree.

- Extensive computational experiments are performed comparing the compact formulation and the BP method on all considered TSP-D variants demonstrating the flexibility and performance of our approaches. The BP method can solve instances with up to 39 customers to proven optimality and thus outperforms the state-of-the-art for particular variants by more than doubling the manageable instance size.

The paper is organized as follows. In the remainder of this section, we provide an overview on relevant literature. Section 2 defines the considered basic TSP-D variant. Section 3 states our compact MILP formulation for the basic TSP-D and its extensions. In Section 4, the dynamic programming recursion is described. In Section 5, we introduce our branch-and-price method. Section 6 computationally compares our solution methods for several TSP-D variants and relates them to the state-of-the-art. Finally, Section 7 concludes our work and gives a short outlook into future work.

1.2 Literature Review

The number of recent publications dedicated to drone-related research is growing extremely fast, so we mostly discuss papers that are relevant to our work, i.e., involving exact optimization approaches or related interesting problem features. Since the synchronization of truck and drone is an important (and difficult) aspect in those problems, we also refer to a survey on this topic in Drexl (2012).

Murray and Chu (2015) introduce the flying sidekick TSP (FSTSP) in which a single truck and a single drone have to serve a set of customers. The drone can carry only a single parcel at a time and is able to serve only a subset of the customers. It can start and end a delivery either at the depot or on the truck at some customer node, but start and end point must not be identical. Before start and after end, there is some predefined constant service time for loading the parcel and recharging or changing the battery, and the time between start and end is limited (also waiting for the truck depletes the drone's battery). Travel times are different for truck and drone but there is no predefined relation. Both truck and drone must not revisit nodes. The completion time to serve all customers and return to the depot is minimized, including potential waiting times for the synchronization of truck and drone. An MILP is proposed with Miller-Tucker-Zemlin-type sub-tour elimination constraints and continuous arrival time variables for both truck and drone. Kundu and Matis (2017) use a more detailed energy consumption model for the drone in the FSTSP by considering parcel weight and wind conditions and show results based on heuristics and simulation. Instead of the completion time, Ha et al. (2018) minimize operational costs related to transportation expenses and waiting times. Their compact MILP formulation is similar to the one in Murray and Chu (2015), only adapted to the alternative objective function. Murray and Raj (2019) extend the FSTSP by considering multiple drones on the truck, customer- and drone-dependent service times, and weight- and time-dependent range limits for the drones. A large but compact MILP formulation is proposed to handle multiple drones which can, however, only solve 8-customer instances to optimality. Jeong et al. (2019) put focus on two additional features, i.e., weight-dependent range limits for the drone and circular no-fly zones in which drones are banned. Dell'Amico et al. (2019) propose branch-and-cut approaches based on improved MILP formulations for the FSTSP by dynamically adding further valid inequalities, e.g., subtour elimination and tournament constraints, to strengthen the dual bounds. They also computationally compare the two cases when the drone is allowed to wait on ground after service or not. Especially, a two-index formulation leads to rather good results on the 10-node instances from Murray and Chu (2015), but unfortunately no larger instances are considered.

Agatz et al. (2018) study a variant of the FSTSP in which no drone-related service times are considered but the truck is allowed to revisit nodes and a drone delivery can start and end at the same node. The authors propose an MILP based on the concept of operations which are concatenated to a tour. A single operation contains exactly two nodes in which truck and drone are together but between those nodes they might serve different customers. There are exponentially many potential operations and in the proposed solution approach all of them are enumerated and added to the model a priori. This might be the reason why the solution of their formulation is limited to instances with up to 10 nodes. In Bouman et al. (2018), the same authors propose a dynamic programming approach for their TSP-D variant again based on the concept of operations, solving instances with up to 16 nodes. When restricting the number of nodes the truck is allowed to serve alone to a maximum of 2 while the drone is airborne, the computational limit is pushed to 20 nodes. The previous two papers are the only ones in our review which allow nodes to be revisited. The cost savings of revisits in practical problem instances are, however, not analyzed. Poikonen et al. (2019) propose a branch-and-bound method for solving the TSP-D variant above but do not allow the truck to revisit nodes. Each tree node corresponds to a partial customer sequence which is evaluated by exactly partitioning the sequence to truck and drone re-using the algorithm from Agatz et al. (2018). Their method can solve instances with up to 10 nodes to optimality.

Poikonen and Golden (2020) suggest heuristic methods for an extended TSP-D variant: (*i*) multiple parcels with individual weights may be carried by a capacitated drone leading to multiple customer visits in a row, (*ii*) the drone has limited energy capacity and its energy consumption is weight-dependent, and (*iii*) a set of predefined (not necessarily customer) locations may serve as drone take-off and landing positions. However, the truck is not allowed to serve customers while the drone is airborne and directly heads to the rendezvous point. In another variant, multiple drones may be deployed by the truck, but one or more drones can only be launched at a location if all drones are on-board. Optimizing the drone's speed is considered in

a further variant with speed-dependent energy consumption. Masone et al. (2019) allow arbitrary take-off and landing locations along the truck route. In a first approach, they iteratively discretize an edge to extend the set of possible positions, while in a more advanced approach they use a mixed-integer second order cone program to determine optimal take-off and landing positions for a given truck route and a set of drone deliveries to minimize the truck's waiting time.

Boysen et al. (2018) assume that the sequence of locations visited by the truck is fixed and aim to find the take-off and landing positions for multiple drones (without range limit) to serve a predefined set of customers. The problem is only solvable in polynomial time if there is a single drone which is forced to return to its take-off or the next location along the truck tour, while other more general variants are NP-hard. Next to the complexity results, two different MILP formulations for this problem are proposed. Carlsson and Song (2017) and Campbell et al. (2017) discuss some continuous approximation results for a TSP-D variant in the Euclidean plane and investigate potential benefits of using a drone to support a truck.

In Wang et al. (2017) and Poikonen et al. (2017) the same authors consider the vehicle routing problem with drones (VRP-D) and show some worst-case results for the benefits of additionally using drones. The VRP-D generalizes the TSP-D by allowing multiple capacitated trucks each equipped with multiple drones with the restriction that drones are forced to take off and land at the same truck.

Wang and Sheu (2019) study a different variant of the VRP-D: Drones may take-off from the depot, trucks, or so-called docking hubs, while landing is only allowed at the depot and the hubs where they are re-charged and re-loaded. At those hubs, trucks can re-load parcels and pickup a limited number of drones to bring them near to customers due to their limited flying range. Trucks and drones have load capacities and drones may carry more than one parcel if not exceeding their capacity (and thus may visit several customers in a row). Fixed truck acquisition costs and traveling costs for trucks and drones are minimized. Waiting times do not occur since it is assumed that enough drones are available at the depot and hubs. The authors propose (*i*) an arc-based formulation with vehicle- and drone-indexed variables, and (*ii*) a set partitioning formulation with path variables for each truck and drone tour which are linked in the master program. The latter is solved by branch-and-price obtaining optimal solutions for instances with up to 15 nodes.

Table 1 gives an overview on the problem features appearing in the papers discussed above: customer nodes which are incompatible to be served by a drone, explicit consideration of customer service times (not included in the flight times), number of trucks used, allowed node revisits by the truck, limit on the number of customers served by the truck while the drone is airborne, number of drones per truck, range limit for the drone (constant or weight-dependent), number of parcels a drone can carry, loops (drone serves a customer while the truck is waiting), required constant launch and rendezvous times for drones, drone is able to freely wait for the truck without affecting its range (e.g., on ground), and the set of potential take-off and landing sites (customer or specific nodes, hubs, or any points in the plane).

Plenty of other drone-related optimization problems appear in the literature where truck and drone are not necessarily coupled in a team, e.g., in the parallel drone scheduling TSP (Murray and Chu, 2015), the truck serves some of the customers while multiple drones start deliveries from the depot directly to a customer within range and immediately come back. In the drone routing problem (Cheng et al., 2018; Dorling et al., 2017), multiple drones with capacity restrictions and weight-dependent energy consumption are scheduled directly from a depot and serve (potentially multiple) customers before returning to the depot where they are re-charged and re-loaded and start their next trip. The latter problem can be seen (and solved) as a multi-trip VRP (Paradiso et al., 2019). The carrier-vehicle TSP, see, e.g., Gambella et al. (2017), is a related problem in which a slow carrier vehicle hosts a faster but range-limited vehicle which needs to visit a set of locations. Here, the take-off and landing points of the fast vehicle have to be determined in the continuous Euclidean plane.

Publication	Customers		Truck			Drone						
	drone incompatible	service times	# trucks	node revisits	node limit alone	# drones	range limit	# parcels	loops allowed	launch / rendezvous times	free waiting	take-off / landing
Agatz et al. (2018)	✓		1	✓		1	constant	1	✓		✓	customer
Bouman et al. (2018)			1	✓	✓	1	constant	1	✓		✓	customer
Boysen et al. (2018)	fixed			fixed route		≥ 1	–	1	✓			specific
Campbell et al. (2017)			1			≥ 1	–	1	✓			customer
Carlsson and Song (2017)			1			1	–	1	✓			any
Dell'Amico et al. (2019)	✓		1			1	constant	1		✓		customer
Ha et al. (2018)	✓		1			1	constant	1		✓		customer
Jeong et al. (2019)	✓		1			1	weight-dep.	1		✓		customer
Kundu and Matis (2017)			1			1	constant	1		✓		customer
Masone et al. (2019)			1		0	1	weight-dep.	≥ 1	✓			any
Murray and Chu (2015)	✓		1			1	constant	1		✓		customer
Murray and Raj (2019)	✓	✓	1			≥ 1	weight-dep.	1		✓		customer
Poikonen et al. (2017)			≥ 1			≥ 1	constant	1	✓			customer
Poikonen and Golden (2020)			1		0	≥ 1	weight-dep.	≥ 1	✓			specific
Poikonen et al. (2019)	✓		1			1	constant	1	✓			customer
Wang et al. (2017)			≥ 1			≥ 1	–	1	✓			customer
Wang and Sheu (2019)			≥ 1			≥ 1	constant	≥ 1	✓			customer / hubs

Table 1: Overview of problem properties and side constraints commonly used in the literature

In the light of the existing work discussed above, we first focus on a basic variant of the TSP-D that is a special case of many problems addressed in the literature because methodological contributions on this basic TSP-D can be instrumental to solve more general cases. In particular, we consider a single truck hosting a single drone with the following features and restrictions. All customers can potentially be served by the drone, and there are no service times at the customer. The truck is not allowed to revisit nodes, and there is no limit on the number of customers served while the drone is airborne. The drone has no range limit but is able to carry only one parcel at a time and thus serve only one customer in each trip. Launch and landing positions, restricted to the depot and the set of customers, cannot be the same for one drone trip, and corresponding launch and rendezvous times are not considered. Finally, the drone can freely wait for the truck on ground.

2 Problem Description of the TSP-D

The Traveling Salesman Problem with Drone (TSP-D) can be formally described as follows. A complete directed graph $\mathcal{G} = (V, \mathcal{A})$ is given. The vertex set V is defined as $V = \{0, 0'\} \cup N$, where both 0 and $0'$ represent a single depot and N a set of n customers to serve—in the following, we also use the notation $N_0 = N \cup \{0\}$ and $N_0' = N \cup \{0'\}$. The arc set $\mathcal{A}$ is defined as $\mathcal{A} = \{(0, j) \,|\, j \in N\} \cup \{(i, j) \,|\, i, j \in N : i \neq j\} \cup \{(i, 0') \,|\, i \in N\}$. A single truck, equipped with a single drone, is located at the depot. The truck and the drone have to be used to serve all customers N. The time for the truck and the drone to traverse arc $(i, j) \in \mathcal{A}$ is indicated with t_{ij}^T and t_{ij}^D, respectively. In general, the time to traverse an arc with the drone is not greater than the time to traverse the same arc with the truck, i.e., $t_{ij}^D \leq t_{ij}^T$, $\forall (i, j) \in \mathcal{A}$, however, we do not need this assumption for our solution methods. Each customer can be served by the truck alone, by the drone alone, or by both the truck and the drone teamed up. The drone has a limited capacity in the sense that it can serve at most one customer alone before returning to the truck to possibly start another service. The drone can leave and return to the truck at customer locations or at the depot only, and it cannot take off and then land at the same node. The goal of the TSP-D is to find a tour with minimum completion time serving all customers either by truck or drone and considering potential waiting times due to the synchronization of truck and drone.

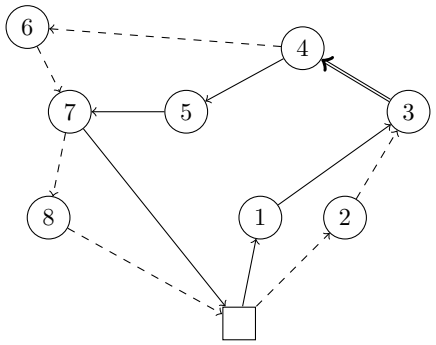

Figure 1: A feasible solution of a TSP-D instance with eight customers

Figure 1 illustrates a feasible solution of a TSP-D with eight customers (the circles) and the depot, i.e., nodes 0 and $0'$ (the rectangle). The truck leaves the depot to serve customer 1 while the drone is launched from the depot to serve customer 2. Then, the truck travels from customer 1 to 3, and the drone travels from customer 2 to 3. At customer 3, the truck and the drone have to be synchronized, so depending on the travel times t_{01}^T, t_{13}^T, t_{02}^D, t_{23}^D, either the trucks waits for the drone (if $t_{01}^T + t_{13}^T < t_{02}^D + t_{23}^D$) or the drone waits for the truck (if $t_{01}^T + t_{13}^T > t_{02}^D + t_{23}^D$). The truck and the drone move together to serve customer 4. The drone is launched toward customer 6, and, in the meantime, the truck travels toward customer 5 and then to customer 7. At customer 7, the truck and the drone have to wait on each other to rejoin. From customer 7, the truck travels back to the depot while the drone serves customer 8 before returning to the depot. The total completion time t of this solution is computed as:

$$t = \max\{t_{01}^T + t_{13}^T,\ t_{02}^D + t_{23}^D\} + t_{34}^T + \max\{t_{45}^T + t_{57}^T,\ t_{46}^D + t_{67}^D\} + \max\{t_{70'}^T,\ t_{78}^D + t_{80'}^D\}$$

In the remainder of the paper, we will use the following concepts. A *truck customer* is a customer visited by the truck alone. Similarly, a *drone customer* is a customer visited by the drone alone. A *combined customer* is a customer visited by both the truck and the drone. In the solution of Figure 1, customers 1 and 5 are truck customers, 2, 6, and 8 are drone customers, and 3, 4, and 7 are combined customers. A *truck arc* (*drone arc*, respectively) is an arc traversed by the truck (drone, respectively) alone. A *combined arc* is an arc traversed by the truck and the drone together. The solution of Figure 1 consists of five truck arcs (i.e., $(0,1)$, $(1,3)$, $(4,5)$, $(5,7)$, $(7,0')$), six drone arcs (i.e., $(0,2)$, $(2,3)$, $(4,6)$, $(6,7)$, $(7,8)$, and $(8,0')$), and one combined arc (i.e., $(3,4)$). A *truck leg* is a concatenation of truck arcs traversed by the truck alone in between two consecutive combined customers. A *drone leg* is a sequence of exactly two consecutive drone arcs traversed by the drone alone in between two consecutive combined customers. A *combined leg* is a concatenation of combined arcs traversed by the truck and the drone together that consists of combined customers only. The solution of Figure 1 features three truck legs (i.e., $0 \longrightarrow 1 \longrightarrow 3$, $4 \longrightarrow 5 \longrightarrow 7$, and $7 \longrightarrow 0'$), three drone legs (i.e., $0 \dashrightarrow 2 \dashrightarrow 3$, $4 \dashrightarrow 6 \dashrightarrow 7$, $7 \dashrightarrow 8 \dashrightarrow 0'$), and a single combined leg (i.e., $3 \Longrightarrow 4$). An *operation* is a synchronized pair of a truck leg and a drone leg in between the same pair of combined customers. Figure 1 shows three operations: the first consisting of truck leg $0 \longrightarrow 1 \longrightarrow 3$ and drone leg $0 \dashrightarrow 2 \dashrightarrow 3$ (we represent such operation as $[0 \longrightarrow 1 \longrightarrow 3, 0 \dashrightarrow 2 \dashrightarrow 3]$), the second consisting of $4 \longrightarrow 5 \longrightarrow 7$ and $4 \dashrightarrow 6 \dashrightarrow 7$, and the third consisting of $7 \longrightarrow 0'$ and $7 \dashrightarrow 8 \dashrightarrow 0'$. A TSP-D solution can be seen as a concatenation of operations and combined legs; for example, the solution of Figure 1 can be represented as $([0 \longrightarrow 1 \longrightarrow 3, 0 \dashrightarrow 2 \dashrightarrow 3], 3 \Longrightarrow 4, [4 \longrightarrow 5 \longrightarrow 7, 4 \dashrightarrow 6 \dashrightarrow 7], [7 \longrightarrow 0', 7 \dashrightarrow 8 \dashrightarrow 0'])$. Notice that a TSP-D solution can consist of operations only or a single combined leg.

3 Compact Formulation

In this section, we describe the compact formulation for the TSP-D first and then extend it to model a variety of side constraints. As we would like to propose formulations that can be easily implemented, we do not use any large sets of valid inequalities that require coding difficult separation procedures. In spite of the simplicity of the formulations we describe, we will show in Section 6 that they can solve larger instances than the formulations available from the literature.

3.1 Compact Formulation for the TSP-D

Let $x_{ij}^T \in \{0,1\}$ be a binary variable equal to 1 if the truck traverses arc $(i,j) \in \mathcal{A}$ (no matter if the drone is on-board or airborne), and let $x_{ij}^D \in \{0,1\}$ be a binary variable equal to 1 if the drone traverses arc $(i,j) \in \mathcal{A}$ (no matter if it is on-board or airborne). Let $y_i^T \in \{0,1\}$ ($y_i^D \in \{0,1\}$, resp.) be a binary variable equal to 1 if $i \in N$ is a truck customer (drone customer, resp.). Moreover, let $y_i^C \in \{0,1\}$ be a binary variable equal to 1 if $i \in N$ is a combined customer. Finally, let $a_i \in \mathbb{R}_+$ be the arrival time of the truck or the drone (or both) at node $i \in V$. The TSP-D can be formulated as:

$$t^* = \min \quad a_{0'} \tag{1a}$$

$$\text{s.t.} \sum_{(i,j)\in\mathcal{A}} x_{ij}^T = \sum_{(j,i)\in\mathcal{A}} x_{ji}^T \qquad i \in N \tag{1b}$$

$$\sum_{(i,j)\in\mathcal{A}} x_{ij}^T = y_i^T + y_i^C \qquad i \in N \tag{1c}$$

$$\sum_{(0,j)\in\mathcal{A}} x_{0j}^T = \sum_{(i,0')\in\mathcal{A}} x_{i0'}^T = 1 \tag{1d}$$

$$\sum_{(i,j)\in\mathcal{A}} x_{ij}^D = \sum_{(j,i)\in\mathcal{A}} x_{ji}^D \qquad i \in N \tag{1e}$$

$$\sum_{(i,j)\in\mathcal{A}} x_{ij}^D = y_i^D + y_i^C \qquad i \in N \tag{1f}$$

$$\sum_{(0,j)\in\mathcal{A}} x_{0j}^D = \sum_{(i,0')\in\mathcal{A}} x_{i0'}^D = 1 \tag{1g}$$

$$y_i^T + y_i^D + y_i^C = 1 \qquad i \in N \tag{1h}$$
$$a_i + t_{ij}^T \leq a_j + M(1 - x_{ij}^T) \qquad (i,j) \in \mathcal{A} \tag{1i}$$
$$a_i + t_{ij}^D \leq a_j + M(1 - x_{ij}^D) \qquad (i,j) \in \mathcal{A} \tag{1j}$$
$$x_{ij}^D + x_{ji}^D \leq y_i^C + y_j^C \qquad i,j \in N : i < j \tag{1k}$$
$$x_{0i}^D + x_{i0'}^D \leq 1 \qquad i \in N \tag{1l}$$
$$x_{ij}^T, x_{ij}^D \in \{0,1\} \qquad (i,j) \in \mathcal{A} \tag{1m}$$
$$y_i^T, y_i^D, y_i^C \in \{0,1\} \qquad i \in N \tag{1n}$$
$$a_i \in \mathbb{R}_+ \qquad i \in V \tag{1o}$$

The objective function (1a) aims at minimizing the total tour duration to serve all customers. Constraints (1b) are flow conservation constraints for the truck. Constraints (1c) link the x_{ij}^T variables with the y_i^T and y_i^C variables. Constraints (1d) ensure that the truck leaves and returns to the depot exactly once. Constraints (1e)-(1g) correspond to constraints (1b)-(1d) but are defined for the drone. Constraints (1h) ensure that each customer is visited exactly once. Constraints (1i)-(1j) act as sub-tour elimination constraints and set the arrival time at each node of the truck/drone. Constraints (1k) ensure that the drone travels from $i \in N$ to $j \in N$ if and only if the truck visits at least one of the two customers i and j, thus ensuring that each drone leg consists of a single drone customer. Constraints (1l) rule out drone legs of type $0 \dashrightarrow i \dashrightarrow 0'$ for any customer $i \in N$, which would correspond to loops rooted at the depot. Constraints (1m)-(1o) model the range of the decision variables.

Notice that constraints (1l) alongside constraints (1g) make the set of feasible solutions of (1) empty for instances with a single customer (i.e., $n = 1$). As instances with a single customer can be solved trivially, we assume that at least two customers must be served, i.e., $n \geq 2$.

Valid Inequalities. The computational time taken by formulation (1) to solve a TSP-D instance can be significantly decreased by adding the following valid inequalities

$$\sum_{(i,j)\in\mathcal{A}} t_{ij}^T x_{ij}^T \leq a_{0'} \qquad \sum_{(i,j)\in\mathcal{A}} t_{ij}^D x_{ij}^D \leq a_{0'}$$

which state that the duration of the tour cannot be lower than the maximum of the sum of the travel times of the arcs traversed by the truck and the sum of the travel times of the arcs traversed by the drone.

3.2 Compact Formulation for Commonly Used Side Constraints

We show how a variety of side constraints commonly investigated in the literature can be modeled based on formulation (1). For the sake of conciseness, we focus on the constraints that are more meaningful when solving a TSP with a single drone.

3.2.1 Loops.

The truck is allowed to launch the drone to serve a customer and wait for it to return to the same node where it was launched. In other words, drone legs starting and ending at the same combined customer are allowed. Such drone legs are called *loops*.

Let $z_{ij} \in \{0,1\}$ be a binary variable equal to 1 if the drone performs loop $i \dashrightarrow j \dashrightarrow i$ to serve customer $j \in N$ while the truck waits at node $i \in N_0$, $i \neq j$. To embed loops into formulation (1), the objective function (1a) is changed to

$$t^* = \min\ a_{0'} + \sum_{j\in N}(t_{0j}^D + t_{j0'}^D)z_{0j} + \sum_{i,j\in N\,:\,i\neq j}(t_{ij}^D + t_{ji}^D)z_{ij} \tag{2}$$

in order to consider the duration of the loops in the total tour duration, and constraints (1h) are replaced by

$$y_i^T + y_i^D + y_i^C + \sum_{j\in N_0} z_{ji} = 1 \qquad i \in N \tag{3a}$$

$$z_{ij} \leq y_i^C \qquad i,j \in N : i \neq j \tag{3b}$$

where constraints (3a) state that each customer can be visited by the truck alone, by the truck and the drone together, or by the drone alone (either in a drone leg or in a loop), and constraints (3b) allow the drone to perform a loop $i \dashrightarrow j \dashrightarrow i$ $(i,j \in N : i \neq j)$ only if i is a combined customer.

Notice that the resulting formulation cuts off the feasible solution where the truck does not leave the depot and the drone performs all deliveries with n loops. Such a solution has a total duration of $\sum_{i \in N}(t_{0i}^D + t_{i0'}^D)$. It is easy to check if such a solution is better than the optimal solution of (1) plus (3b) with objective function (2) instead of (1a).

3.2.2 Incompatible Customers.

Let us call *incompatible customer* a customer that must be served by the truck and cannot therefore be a drone customer. Let $N^D \subset N$ be the set of customers that can be drone customers (i.e., $N \setminus N^D$ is the set of incompatible customers). To take incompatible customers into account, the following constraints are added to formulation (1):

$$y_i^D = 0 \qquad i \in N \setminus N^D \tag{4a}$$

$$z_{ji} = 0 \quad i \in N \setminus N^D,\ j \in N_0 : i \neq j \tag{4b}$$

Constraints (4a) state that customers of the set $N \setminus N^D$ cannot be drone customers. Constraints (4b) rule out all loops that serve customers of the set $N \setminus N^D$.

3.2.3 Drone Flying Range.

If the drone has a maximum flying range e, i.e., the drone cannot fly more than e units of time consecutively, then the following constraints are added to formulation (1):

$$x_{ij}^D \leq x_{ij}^T \qquad (i,j) \in \mathcal{A} : t_{ij}^D > e \tag{5a}$$

$$z_{0j} = 0 \qquad j \in N : t_{0j}^D + t_{j0'}^D > e \tag{5b}$$

$$z_{ij} = 0 \qquad i,j \in N : i \neq j,\ t_{ij}^D + t_{ji}^D > e \tag{5c}$$

$$\sum_{(j,i) \in \mathcal{A}} t_{ji}^D x_{ji}^D + \sum_{(i,j) \in \mathcal{A}} t_{ij}^D x_{ij}^D + (t_{0i}^D + t_{i0'}^D) z_{0i} + \sum_{j \in N} (t_{ji}^D + t_{ij}^D) z_{ji} \leq e + M(1 - y_i^D) \qquad i \in N \tag{5d}$$

Constraints (5a) guarantee that if the drone traverses an arc $(i,j) \in \mathcal{A}$ whose travel time exceeds the drone flying range, the drone is on-board. Constraints (5b)-(5c) prevent the drone from performing loops that exceed its flying range. Constraints (5d) ensure that if the drone visits a customer alone (i.e., $y_i^D = 1$ for some $i \in N$), then the total drone travel time of the arcs traversed by the drone and incident to i does not exceed the drone flying range—note that adding the z variables to the left-hand side is not necessary but allows to strengthen the constraint.

3.2.4 Weight-Dependent Drone Flying Range.

Let us assume the drone has a limited battery capacity $\bar{b}$, and there is a weight-dependent and arc-dependent energy consumption function $en_{ij}(w)$ that returns the battery consumed by the drone when traversing arc $(i,j) \in \mathcal{A}$ while transporting a load equal to w. Under the assumption that the drone can deliver only a single package in each drone leg, it is possible to pre-compute two values b_{ij}^{on} and b_{ij}^{off} that indicate the energy consumption while traversing arc $(i,j) \in \mathcal{A}$ when the package of customer j is on-board and when the drone travels empty from i to j, respectively. Under this assumption, constraints (5) are changed as follows to take the weight-dependent drone flying range into account:

$$x_{ij}^D \leq x_{ij}^T \qquad (i,j) \in \mathcal{A} : b_{ij}^{\text{off}} > \bar{b} \tag{6a}$$

$$z_{0j} = 0 \qquad j \in N : b_{0j}^{\text{on}} + b_{j0'}^{\text{off}} > \bar{b} \tag{6b}$$

$$z_{ij} = 0 \qquad i, j \in N : i \neq j,\ b^{\text{on}}_{ij} + b^{\text{off}}_{ji} > \bar{b} \tag{6c}$$

$$\sum_{(j,i)\in\mathcal{A}} b^{\text{on}}_{ji} x^D_{ji} + \sum_{(i,j)\in\mathcal{A}} b^{\text{off}}_{ij} x^D_{ij} + (b^{\text{on}}_{0i} + b^{\text{off}}_{i0'}) z_{0i} + \sum_{j\in N} (b^{\text{on}}_{ji} + b^{\text{off}}_{ij}) z_{ji}$$
$$\leq \bar{b} + M(1 - y^D_i) \qquad i \in N \tag{6d}$$

3.2.5 Drone Cannot Land and Wait.

If the drone cannot land and wait for the truck to arrive, but it can only land on the truck, meaning that the rendezvous must take place before the drone runs out of battery, then the following constraints can be added:

$$a_k - a_j \leq e + M(2 - x^D_{ji} - x^D_{ik}) + M(1 - y^D_i) \quad j \in N_0,\ \ i \in N,\ \ k \in N'_0 : i \neq j \neq k,\ t^D_{ji} + t^D_{ik} \leq e \tag{7a}$$

These constraints state that, for each triplet of nodes i, j, and k ($j \in N_0$, $i \in N$, $k \in N'_0$), if the drone performs leg $j \dashrightarrow i \dashrightarrow k$, then the arrival time at k minus the arrival time at j cannot exceed the drone flying range. Note that this feature that the drone cannot land and wait for the truck makes only sense if the drone has a limited flying range.

3.2.6 Launch and Rendezvous Times.

The launch time lt is defined as the time to prepare the drone before it can be launched. In the literature, see, e.g., Murray and Chu (2015), the launch time is incurred at any node that is not the depot. We assume that the truck and the drone do not move while the drone is being prepared for launching.

To model launch time, for each node $i \in N$ we add binary variable $\ell_i \in \{0, 1\}$, which is equal to 1 if customer i is a drone customer and i is not visited by the drone directly from the depot. Moreover, the objective function (2) is replaced by

$$t^* = \min\ a_{0'} + \sum_{j\in N} (t^D_{0j} + t^D_{j0'}) z_{0j} + \sum_{i,j\in N\,:\,i\neq j} (t^D_{ij} + t^D_{ji} + lt) z_{ij} + \sum_{i\in N} lt\ell_i, \tag{8}$$

and we add constraint

$$\ell_i \geq y^D_i - x^D_{0i} \quad i \in N. \tag{9}$$

The objective function (8) takes into account the launch time in the loops not rooted at the depot (see the third term) and in all drone legs not rooted at the depot (see the last term). Constraints (9) guarantee that variable ℓ_i equals 1 whenever customer i is a drone customer (i.e., $y^D_i = 1$) and i is not visited by the drone directly from the depot (i.e., $x^D_{0i} = 0$).

Similarly to the launch time, if it takes time rt (the so-called rendezvous time) to retrieve the drone at a node and we assume that the truck and the drone do not move in the meanwhile, the objective function (2) is replaced by

$$t^* = \min\ a_{0'} + \sum_{j\in N} (t^D_{0j} + t^D_{j0'} + rt) z_{0j} + \sum_{i,j\in N\,:\,i\neq j} (t^D_{ij} + t^D_{ji} + rt) z_{ij} + \sum_{i\in N} rt y^D_i$$

to increase the duration of loops (second and third term) and drone legs (last term) by rt.

3.2.7 Maximum Number of Customers per Truck Leg.

Let us assume that there can be at most $\overline{n}$ truck customers in each truck leg. To model such a limit $\overline{n}$, we consider three cases:

1) If $\overline{n} = 0$, we add constraints

$$y^T_i = 0 \quad i \in N$$

which ensure that there are no truck customers.

2) If $\overline{n} = 1$, we add constraints

$$x_{ij}^T + x_{ji}^T \leq y_i^C + y_j^C \quad i, j \in N : i < j \tag{10}$$

which are similar to constraints (1k) and guarantee that, for each pair of customers $i, j \in N : i < j$, if one of the two arcs $(i, j), (j, i) \in \mathcal{A}$ is traversed by the truck, then either i or j (or both) are combined nodes.

3) If $\overline{n} \geq 2$, we use variable $v_i \in \mathbb{Z}_+$ to count the number of customers at node $i \in N$ (including i) visited by the truck alone since the drone has been launched last, and add constraints

$$y_i^T \leq v_i \leq \overline{n} y_i^T \qquad i \in N \tag{11a}$$

$$v_i + 1 \leq v_j + (\overline{n} - 1)(2 - x_{ij}^T - y_j^T) \quad i, j \in N : i \neq j. \tag{11b}$$

Constraints (11a) guarantee that variable v_i takes a value between 1 and $\overline{n}$ for every truck customer $i \in N$. Constraints (11b) set $v_j = v_i + 1$ whenever the truck traverses arc $(i, j) \in \mathcal{A}$ and j is a truck customer.

4 Dynamic Programming Recursions

In this section, we formulate the TSP-D and its extensions introduced in Section 3 by using *Dynamic Programming* (DP). These recursions will be used to solve the pricing problem in the branch-and-price algorithm we present in Section 5.

4.1 DP Recursion for the TSP-D

As mentioned in Section 2, a solution of the TSP-D can be broken down into a concatenation of operations and combined legs. By definition, each operation consists of a truck leg and a drone leg; each truck leg is a sequence of truck arcs, and each drone leg is a sequence of exactly two drone arcs. Moreover, each combined leg is a sequence of combined arcs. The solution illustrated in Figure 1 can be represented as $([0 \longrightarrow 1 \longrightarrow 3, 0 \dashrightarrow 2 \dashrightarrow 3], 3 \Longrightarrow 4, [4 \longrightarrow 5 \longrightarrow 7, 4 \dashrightarrow 6 \dashrightarrow 7], [7 \longrightarrow 0', 7 \dashrightarrow 8 \dashrightarrow 0'])$ and can therefore be recursively generated by first building operation $[0 \longrightarrow 1 \longrightarrow 3, 0 \dashrightarrow 2 \dashrightarrow 3]$, then adding combined leg $3 \Longrightarrow 4$, adding operation $[4 \longrightarrow 5 \longrightarrow 7, 4 \dashrightarrow 6 \dashrightarrow 7]$, and finally operation $[7 \longrightarrow 0', 7 \dashrightarrow 8 \dashrightarrow 0']$. Each operation can be recursively generated by first generating the truck leg—one truck arc at a time—and then adding the drone leg. The DP recursions we describe in this section are based on the idea that each TSP-D solution can be decomposed into a set of truck arcs, drone legs, and combined arcs. Therefore, a complete solution can be recursively generated by iteratively adding a truck arc, a drone leg, or a combined arc at a time to a *partial solution* which is a concatenation of operations and combined legs possibly followed by a concatenation of truck arcs.

Given the set of partial TSP-D solutions, we define the function

$$f(S, i^T, i^D, \tau) \tag{12}$$

where (a) $S \subseteq N$ indicates the set of customers visited in a partial TSP-D solution; (b) $i^T \in N_0$ indicates the last node visited by the truck in a partial TSP-D solution; (c) $i^D \in N_0$ indicates the last combined node visited; and (d) $\tau \in \mathbb{R}_+$ represents the time spent by the truck traveling alone since i^D has been visited. Value $f(S, i^T, i^D, \tau)$ is the minimum duration of any partial TSP-D solution that (i) starts from the depot, (ii) visits the set of customers S, (iii) where the last vertex visited by the truck is i^T, (iv) the last vertex visited by the drone is i^D, and (v) τ is the time spent since the truck visited i^D.

Function (12) can be used to compute the duration t^* of the fastest TSP-D tour as follows. Function f is first initialized with $f(\varnothing, 0, 0, 0) = 0$. Then, a recursive step is applied n times. At iteration k ($k = 1, \ldots, n$) of the recursive step, values $f(S, i^T, i^D, \tau)$ with $|S| = k$ are computed by propagating values $f(S, i^T, i^D, \tau)$ with $|S| = k - 1$. In particular, each value $f(S, i^T, i^D, \tau)$ with $|S| = k - 1$ is propagated at most $2n$ times by adding a truck arc, a combined arc, or a drone leg at a time as follows:

1. *Add truck arc.* Function $f(S, i^T, i^D, \tau)$ is propagated toward each customer $j \in N \setminus S$ to compute $f(S \cup \{j\}, j, i^D, \tau + t_{i^T j}^T) = f(S, i^T, i^D, \tau)$, capturing the incomplete truck leg in the τ parameter.

2. *Add combined arc.* If $i^T = i^D$ (and consequently $\tau = 0$), function $f(S, i^T, i^D, \tau)$ is propagated toward each customer $j \in N \setminus S$ to compute $f(S \cup \{j\}, j, j, 0) = f(S, i^T, i^D, \tau) + t^T_{i^T j}$.

3. *Add drone leg.* If $i^T \neq i^D$, function $f(S, i^T, i^D, \tau)$ is propagated toward each customer $j \in N \setminus S$ to compute $f(S \cup \{j\}, i^T, i^T, 0) = f(S, i^T, i^D, \tau) + \max\{\tau, t^D_{i^D j} + t^D_{j i^T}\}$.

The optimal solution value t^* can then be computed as $t^* = \min\{t_1, t_2\}$, where

- t_1 is the minimum duration of any TSP-D solution where the last traversed arc is a combined arc, i.e.

$$t_1 = \min_{i \in N} \{f(N, i, i, 0) + t^T_{i0'}\}, \tag{13}$$

- t_2 is the minimum duration of any TSP-D solution that ends with an operation, i.e.

$$t_2 = \min_{\substack{j \in N \\ i^T, i^D \in N \setminus \{j\}}} \{f(N \setminus \{j\}, i^T, i^D, \tau) + \max\{\tau + t^T_{i^T 0'}, t^D_{i^D j} + t^D_{j0'}\}\}. \tag{14}$$

Figure 2 shows how the TSP-D solution represented in Figure 1 can be iteratively generated by adding, to partial solutions, a truck arc, a drone leg, or a combined arc at a time as indicated in the DP recursion previously described. The initialization of the solution corresponds to $f(\varnothing, 0, 0, 0) = 0$. In the first two recursive steps (see Figure 2a and 2b), truck arcs $(0, 1)$ and $(1, 3)$ are added to generate $f(\{1\}, 1, 0, t^T_{01}) = 0$ and $f(\{1, 3\}, 3, 0, t^T_{01} + t^T_{13}) = 0$. Then, the first operation is completed by adding drone leg $0 \dashrightarrow 2 \dashrightarrow 3$ to $f(\{1, 3\}, 3, 0, t^T_{01} + t^T_{13}) = 0$ to generate $f(\{1, 2, 3\}, 3, 3, 0) = \max\{t^T_{01} + t^T_{13}, t^D_{02} + t^D_{23}\}$ (see Figure 2c). Combined arc $(3, 4)$ is added next to generate $f(\{1, 2, 3, 4\}, 4, 4, 0) = \max\{t^T_{01} + t^T_{13}, t^D_{02} + t^D_{23}\} + t^T_{34}$. The second operation consisting of truck arcs $(4, 5)$ and $(5, 7)$ and drone leg $4 \dashrightarrow 6 \dashrightarrow 7$, is added in the next three steps (see Figure 2e, 2f, and 2g), and the partial solution corresponding to $f(\{1, 2, 3, 4, 5, 6, 7\}, 7, 7, 0) = \max\{t^T_{01} + t^T_{13}, t^D_{02} + t^D_{23}\} + t^T_{34} + \max\{t^T_{45} + t^T_{57}, t^D_{46} + t^D_{67}\}$ is computed. The final operation consisting of truck arc $(7, 0')$ and drone leg $7 \dashrightarrow 8 \dashrightarrow 0'$, is added in the end when computing t_2 according to (14) to obtain $f(N, 0', 0', 0) = \max\{t^T_{01} + t^T_{13}, t^D_{02} + t^D_{23}\} + t^T_{34} + \max\{t^T_{45} + t^T_{57}, t^D_{46} + t^D_{67}\} + \max\{t^T_{70'}, t^D_{78} + t^D_{80'}\}$ (see Figure 2h).

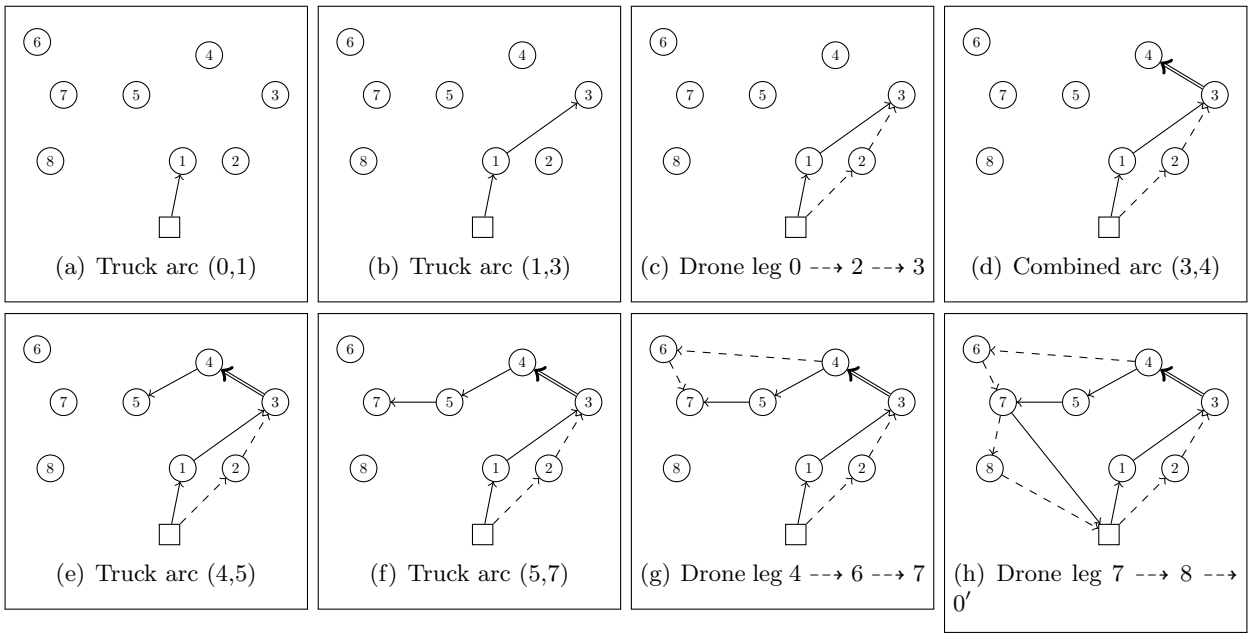

Figure 2: Step-by-step description of how the TSP-D solution in Figure 1 is iteratively generated by recursively adding truck arcs, drone legs, and combined arcs to partial solutions

4.2 DP Recursion to Handle Side Constraints

In this section, we show how the DP recursion presented in Section 4.1 can be adjusted to handle the side constraints investigated in Section 3.2 for the compact formulation.

4.2.1 Loops.

Loops can be handled by adding the following type of propagation to function $f(S, i^T, i^D, \tau)$ if $i^T = i^D$ (and consequently $\tau = 0$):

4. *Add loop.* If $i^T = i^D$, function $f(S, i^T, i^D, \tau)$ is propagated toward each customer $j \in N \setminus S$ to compute $f(S \cup \{j\}, i^T, i^D, 0) = f(S, i^T, i^D, \tau) + t^D_{i^T j} + t^D_{j i^T}$.

4.2.2 Incompatible Customers.

When the drone can serve a subset of customers only, $N^D \subset N$, the two propagations *3. Add drone leg* and *4. Add loops* (if loops are allowed) in the recursive step change as follows:

3. *Add drone leg.* If $i^T \neq i^D$, function $f(S, i^T, i^D, \tau)$ is propagated toward each customer $j \in N^D \setminus S$ to compute $f(S \cup \{j\}, i^T, i^T, 0) = f(S, i^T, i^D, \tau) + \max\{\tau, t^D_{i^D j} + t^D_{j i^T}\}$.

4. *Add loop.* If $i^T = i^D$, function $f(S, i^T, i^D, \tau)$ is propagated toward each customer $j \in N^D \setminus S$ to compute $f(S \cup \{j\}, i^T, i^D, 0) = f(S, i^T, i^D, \tau) + t^D_{i^T j} + t^D_{j i^T}$.

Moreover, equation (14) to compute t_2 is changed as follows:

$$t_2 = \min_{\substack{j \in N^D \\ i^T, i^D \in N \setminus \{j\}}} \{f(N \setminus \{j\}, i^T, i^D, \tau) + \max\{\tau + t^T_{i^T 0'}, t^D_{i^D j} + t^D_{j 0'}\}\}.$$

4.2.3 Drone Flying Range.

Similarly to the previous case where some customers cannot be drone customers, when the drone has a flying range e, the two propagations *3. Add drone leg* and *4. Add loops* (if loops are allowed) change as follows:

3. *Add drone leg.* If $i^T \neq i^D$, function $f(S, i^T, i^D, \tau)$ is propagated toward each customer $j \in N \setminus S$ if $t^D_{i^T j} + t^D_{j i^T} \leq e$ to compute $f(S \cup \{j\}, i^T, i^T, 0) = f(S, i^T, i^D, \tau) + \max\{\tau, t^D_{i^D j} + t^D_{j i^T}\}$.

4. *Add loop.* If $i^T = i^D$, function $f(S, i^T, i^D, \tau)$ is propagated toward each customer $j \in N \setminus S$ if $t^D_{i^T j} + t^D_{j i^T} \leq e$ to compute $f(S \cup \{j\}, i^T, i^D, 0) = f(S, i^T, i^D, \tau) + t^D_{i^T j} + t^D_{j i^T}$.

Moreover, equation (14) to compute t_2 is changed as follows:

$$t_2 = \min_{\substack{j \in N,\, i^T, i^D \in N \setminus \{j\}: \\ t^D_{i^D j} + t^D_{j 0'} \leq e}} \{f(N \setminus \{j\}, i^T, i^D, \tau) + \max\{\tau + t^T_{i^T 0'}, t^D_{i^D j} + t^D_{j 0'}\}\}.$$

4.2.4 Weight-Dependent Drone Flying Range.

The weight-dependent drone flying range can be integrated in the DP recursion by replacing the conditions of type $t^D_{ij} + t^D_{jk} \leq e$ described above for the case with constant range limit with $b^{\text{on}}_{ij} + b^{\text{off}}_{jk} \leq \bar{b}$.

4.2.5 Drone Cannot Land and Wait.

If the drone cannot land and wait for the truck, the propagation *1. Add truck arc* has to be adjusted to prevent truck legs to last longer than the drone flying range because this would imply that, in the corresponding operation, the drone would have to land and wait for the truck:

1. *Add truck arc.* Function $f(S, i^T, i^D, \tau)$ is propagated toward each customer $j \in N \setminus S$ only if $\tau + t^T_{i^T j} \leq e$ to compute $f(S \cup \{j\}, j, i^D, \tau + t^T_{i^T j}) = f(S, i^T, i^D, \tau)$.

Moreover, also equation (14) to compute t_2 must prevent the truck legs to last longer than the drone flying range and are changed as follows:

$$t_2 = \min_{\substack{j \in N,\, i^T, i^D \in N \setminus \{j\}: \\ \max\{\tau + t^T_{i^T 0'}, t^D_{i^D j} + t^D_{j 0'}\} \leq e}} \{f(N \setminus \{j\}, i^T, i^D, \tau) + \max\{\tau + t^T_{i^T 0'}, t^D_{i^D j} + t^D_{j 0'}\}\}.$$

4.2.6 Launch and Rendezvous Times.

To take the launch time lt and the rendezvous time rt into account, the propagations of type *1. Add truck arc*, *3. Add drone leg*, and *4. Add loop* are changed as follows:

1. *Add truck arc.* Function $f(S, i^T, i^D, \tau)$ is propagated toward each customer $j \in N \setminus S$ to compute $f(S \cup \{j\}, j, i^D, \tau + t^T_{i^T j}) = f(S, i^T, i^D, \tau) + lt$. Notice that lt is added only if $i^T \neq 0$.

3. *Add drone leg.* If $i^T \neq i^D$, function $f(S, i^T, i^D, \tau)$ is propagated toward each customer $j \in N \setminus S$ to compute $f(S \cup \{j\}, i^T, i^T, 0) = f(S, i^T, i^D, \tau) + \max\{\tau, t^D_{i^D j} + t^D_{j i^T}\} + rt$.

4. *Add loop.* If $i^T = i^D$, function $f(S, i^T, i^D, \tau)$ is propagated toward each customer $j \in N \setminus S$ to compute $f(S \cup \{j\}, i^T, i^D, 0) = f(S, i^T, i^D, \tau) + t^D_{i^T j} + t^D_{j i^T} + lt + rt$. Notice that lt is added only if $i^T \neq 0$.

Moreover, equation (14) to compute t_2 is replaced with $t_2 = \min\{t'_2, t''_2\}$, where:

$$t'_2 = \min_{j \in N,\, i \in N \setminus \{j\}} \{f(N \setminus \{j\}, i, i, 0) + \max\{t^T_{i0'}, t^D_{ij} + t^D_{j0'}\}\} + lt + rt \tag{15}$$

$$t''_2 = \min_{\substack{j \in N,\, i^T \in N \setminus \{j\} \\ i^D \in N \setminus \{i^T, j\}}} \{f(N \setminus \{j\}, i^T, i^D, \tau) + \max\{\tau + t^T_{i^T 0'}, t^D_{i^D j} + t^D_{j0'}\}\} + rt \tag{16}$$

4.2.7 Maximum Number of Customers per Truck Leg.

In Section 3.2.7, we examined three cases related to the maximum number of customers per drone leg $\overline{n}$: $\overline{n} = 0$, $\overline{n} = 1$, and $\overline{n} \geq 2$. Handling the two latter cases (i.e., when $\overline{n} \geq 1$) is not trivial in a DP recursion. Indeed, it requires adding an additional state variable/resource in function $f(S, i^T, i^D, \tau)$. To handle the case where $\overline{n} = 0$ in the DP described in Section 4.1, two changes are needed. The propagation *1. Add truck arc* is applied only to $f(S, i^T, i^D, \tau)$ if $i^T = i^D$ because adding a truck arc to a partial solution corresponding to a state with $i^T \neq i^D$ implies having truck customers in the set of feasible solutions. The second change is related to the equation (14) to compute t_2, which is replaced by:

$$t_2 = \min_{j \in N,\, i \in N \setminus \{j\}} \{f(N \setminus \{j\}, i, i, 0) + \max\{t^T_{i0'}, t^D_{ij} + t^D_{j0'}\}\},$$

5 An Exact Branch-and-Price Method

Solving the TSP-D with or without the side constraints previously discussed by using the DP recursions described in Section 4 is prohibitive from a computational point of view. Therefore, in this section, we describe an exact branch-and-price method that relies on a set partitioning formulation and where the pricing problem is solved with DP starting from the recursions presented in Section 4. The main components of the exact branch-and-price for the TSP-D are *ng*-route relaxation (see Section 5.2), dual variable stabilization (see Section 5.3), and a three-level hierarchical branching (see Section 5.4).

5.1 Set Partitioning Formulation

Let us define a *route* in graph $\mathcal{G}$ as an ordered sequence of operations and combined legs that start from the depot, end at the depot, and such that the final node of each operation/combined leg coincides with the initial node of the subsequent operation/combined leg. Let $\mathcal{R}$ be the set of all routes in graph $\mathcal{G}$. Each route $r \in \mathcal{R}$ is defined by coefficients a_{ir} that indicate the number of times customer $i \in N$ is visited by route r and by d_r that indicate the duration of route $r \in \mathcal{R}$, including the time to perform loops (if allowed). Moreover, let $\xi_r \in \{0, 1\}$ be a binary variable equal to 1 if route $r \in \mathcal{R}$ is selected (0 otherwise). The TSP-D can be formulated as the following set partitioning model:

$$t^* = \min \sum_{r \in \mathcal{R}} d_r \xi_r \tag{17a}$$

$$\text{s.t.} \sum_{r \in \mathcal{R}} \xi_r = 1 \tag{17b}$$

$$\sum_{r \in \mathcal{R}} a_{ir} \xi_r = 1 \quad i \in N \tag{17c}$$

$$\xi_r \in \{0,1\} \qquad r \in \mathcal{R} \tag{17d}$$

The objective function (17a) aims at minimizing the duration of the selected route. Constraint (17b) guarantees that exactly one route is selected. Constraints (17c) ensure that each customer is visited exactly once. Constraints (17d) are integrality constraints.

Formulation (17) is valid not only for the TSP-D, but also for any generalization of the TSP-D that includes any of the side constraints considered in Section 3 as all such constraints can be implied by the definition of *feasible* route.

As the set $\mathcal{R}$ contains an exponential number of elements, formulation (17) can be used to solve the TSP-D only if column generation/branch-and-price is applied to dynamically generate $\boldsymbol{\xi}$ variables. Any solution of (17) corresponds to a Hamiltonian route of $\mathcal{R}$, so, in principle, any column-generation-based algorithm relying on (17) could search for Hamiltonian routes only. Nevertheless, in this case, solving the pricing problem is as complicated as solving the TSP-D itself, so there would not be any benefit in using formulation (17) to solve the TSP-D. To efficiently use (17) to solve the TSP-D, a route relaxation is needed, so that the set $\mathcal{R}$ does not only contain Hamiltonian routes but the pricing problem is easier to solve. One of the most efficient route relaxations proposed in the literature is the *ng*-route relaxation proposed in Baldacci et al. (2011). The branch-and-price algorithm we propose is based on formulation (17), where the set of routes $\mathcal{R}$ contains *ng*-routes visiting exactly n customers. At each node of the branch-and-price tree, the master problem corresponds to the linear relaxation of (17) (and possibly additional constraints dictated by branching decisions), and the pricing problem is solved by dynamic programming as described in Section 5.2. The strategies to stabilize the dual variables and to branch on fractional solutions are described in Sections 5.3 and 5.4, respectively.

5.2 ng-Route Relaxation

Let us define for each customer $i \in N$, a set of customers $N_i \subseteq N$ that contains the *neighborhood* of i, i.e., the so-called *ng-set*. The ng-set usually contains the customers that are closest to i. An *ng*-route is a not necessarily elementary route where a customer $i \in N$ is visited more than once if and only if there exists at least one customer j visited in between the two consecutive visits to customer i such that $i \notin N_j$. The size of the sets N_i determines the subtours allowed in the *ng*-routes, the quality of the lower bound returned by solving the linear relaxation of (17), and the complexity of the pricing problem. Indeed, the larger the set N_i, the fewer subtours are allowed in the *ng*-routes, the better the lower bound returned by solving the linear relaxation of (17), but also the more time-consuming the pricing problem to solve. In the extreme case where $N_i = N$ for each customer $i \in N$, *ng*-routes cannot contain any subtour, so the linear relaxation of (17) provides the best possible bound among all route relaxations, but the pricing problem would be as complicated as solving the TSP-D itself. In practice, defining *ng*-sets of cardinality eight or ten has proved to be a good trade-off between the quality of the bounds obtained and the complexity to solve the pricing problem. In our branch-and-price, we set each *ng*-set N_i equal to the five closest customers to $i \in N$; indeed, the number of functions to compute and the computing time to solve the pricing problem quickly increases with the cardinality of the *ng*-sets, and further increasing their cardinality would not pay off on average.

Let $u_0 \in \mathbb{R}$ and $u_i \in \mathbb{R}$ $(i \in N)$ be the dual variables associated with constraints (17b) and (17c), respectively, of the linear relaxation of (17). The DP recursion presented in Section 4 can be changed as follows to generate *ng*-routes of negative reduced costs w.r.t. the dual solution $\boldsymbol{u} \in \mathbb{R}^{n+1}$. Instead of $f(S, i^T, i^D, \tau)$, we use function $f(ng, k, i^T, i^D, \tau)$, where $ng \subseteq N_{i^T}$ is the subset of customers already visited that cannot be visited in the next propagation, k is the number of customers visited so far, and i^T, i^D, and τ, are defined as in Section 4. Function f is initialized with $f(\varnothing, 0, 0, 0, 0) = -u_0$. The propagations are defined as:

1. *Add truck arc.* Function $f(ng, k, i^T, i^D, \tau)$ is propagated toward each customer $j \in N \setminus (ng \cup \{i^T, i^D\})$ to compute reduced costs $f((ng \cup \{i^T\}) \cap N_j, k+1, j, i^D, \tau + t^T_{i^T j}) = f(ng, k, i^T, i^D, \tau) - u_j$.

2. *Add combined arc.* If $i^T = i^D$ (and consequently $\tau = 0$), function $f(ng, k, i^T, i^D, \tau)$ is propagated toward each customer $j \in N \setminus (ng \cup \{i^T\})$ to compute $f((ng \cup \{i^T\}) \cap N_j, k+1, j, j, 0) =$

$f(ng, k, i^T, i^D, \tau) + t^T_{i^T j} - u_j$.

3. *Add drone leg.* If $i^T \neq i^D$, function $f(ng, k, i^T, i^D, \tau)$ is propagated toward each customer $j \in N \setminus (ng \cup \{i^T, i^D\})$ to compute $f((ng \cup \{j\}) \cap N_{i^T}, k+1, i^T, i^T, 0) = f(ng, k, i^T, i^D, \tau) + \max\{\tau, t^D_{i^D j} + t^D_{j i^T}\} - u_j$.

Moreover, t_1 and t_2 are computed as:

$$t_1 = \min_{i \in N, ng \subseteq N_i} \{f(ng, n, i, i, 0) + t^T_{i0'}\}$$

$$t_2 = \min_{\substack{i^T, i^D \in N,\ ng \subseteq N_{i^T} \\ j \in N \setminus (ng \cup \{i^T, i^D\})}} \{f(ng, n-1, i^T, i^D, \tau) + \max\{\tau + t^T_{i^T 0'}, t^D_{i^D j} + t^D_{j0'}\} - u_j\}.$$

To speed up the solution process, the following three dominance rules can be used to limit the number of values $f(ng, k, i^T, i^D, \tau)$ to compute. The first dominance rule is intuitive, so we omit the corresponding proof. The second and the third dominance rules are more complicated, so we also provide, in the appendix, a proof of them.

Dominance Rule 1 *Let $f(ng_1, k, i^T, i^D, \tau_1) = f^1$ and $f(ng_2, k, i^T, i^D, \tau_2) = f^2$ such that (a) $f^1 < f^2$, (b) $i^T = i^D$, (c) $NG_1 \subseteq NG_2$, and (d) $\tau_1 \leq \tau_2$, then $f(ng_1, k, i^T, i^D, \tau_1)$ dominates $f(ng_2, k, i^T, i^D, \tau_2)$.*

Dominance Rule 2 *Let $f(ng_1, k, i^T, i^D, \tau_1) = f^1$ and $f(ng_2, k, i^T, i^D, \tau_2) = f^2$ be the function values for two states such that $i^T \neq i^D$. If the following conditions hold:*

$$ng_1 \subseteq ng_2 \tag{18a}$$
$$f^1 \leq f^2 \tag{18b}$$
$$f^1 + \tau_1 \leq f^2 + \tau_2 \tag{18c}$$

and at least one of them is strictly satisfied, then $f(ng_1, k, i^T, i^D, \tau_1)$ dominates $f(ng_2, k, i^T, i^D, \tau_2)$.

Dominance Rule 3 *Let $f(ng_1, k, i^T, i^D, \tau_1) = f^1$ and $f(ng_2, k, i^T, i^D, \tau_2) = f^2$ be the function values for two states such that $i^T \neq i^D$. If the following conditions hold:*

$$ng_1 \subseteq ng_2 \tag{19a}$$
$$f^1 \geq f^2 \tag{19b}$$
$$f^1 + \tau_1 \leq f^2 + \tau_2 \tag{19c}$$
$$f^1 + \max_{j \in N, s \in N'_0 : j \neq s} \{t^D_{i^D j} + t^D_{js}\} \leq f^2 + \tau_2 \tag{19d}$$

and at least one of them is strictly satisfied, then $f(ng_1, k, i^T, i^D, \tau_1)$ dominates $f(ng_2, k, i^T, i^D, \tau_2)$.

The side constraints discussed in Section 4.2 can similarly be handled when generating ng-routes, so we omit the details here. However, we have to mention that Dominance Rule 2 cannot be applied in case the drone cannot land and wait for the truck. Indeed, as there is no guarantee that $\tau_1 \leq \tau_2$, not all propagations of $f(ng_2, k, i^T, i^D, \tau_2)$ are feasible for $f(ng_1, k, i^T, i^D, \tau_1)$, so $f(ng_1, k, i^T, i^D, \tau_1)$ may not dominate $f(ng_2, k, i^T, i^D, \tau_2)$ under the conditions of Dominance Rule 2. Dominance Rules 1 and 3 remain valid and can be applied.

5.3 Stabilization of the Master Problem

The master problem corresponding to the linear relaxation of (17) is affected by degeneracy, so the dual variables $\boldsymbol{u} \in \mathbb{R}^{n+1}$ can be highly unstable when computing the lower bound at each node of the search tree. To stabilize the dual variables, we use a simple stabilization technique inspired by Pigatti et al. (2005). First, we replace the equality sign in constraint (17b) with the $\leq$ sign to halve the domain of the corresponding dual variable u_0 (i.e., $u_0 \in \mathbb{R}_-$). Second, we replace the equality sign in constraints (17c) with the $\geq$ sign, again to halve the domain of the corresponding dual variables u_i ($i \in N$) (i.e., $u_i \in \mathbb{R}_+$). Third, we

introduce a non-negative real variable φ_i for each customer $i \in N$ that allows customer $i \in N$ to be visited less than once, but, if this happens, an additional cost $\overline{u}_i$ is paid. Therefore, the stabilized master problem corresponds to the following linear program:

$$\min \sum_{r \in \mathcal{R}} d_r \xi_r + \sum_{i \in N} \overline{u}_i \varphi_i \tag{20a}$$

$$\text{s.t.} \sum_{r \in \mathcal{R}} \xi_r \leq 1 \tag{20b}$$

$$\sum_{r \in \mathcal{R}} a_{ir} \xi_r + \varphi_i \geq 1 \qquad i \in N \tag{20c}$$

$$\xi_r \geq 0 \qquad r \in \mathcal{R} \tag{20d}$$

$$\varphi_i \geq 0 \qquad i \in N \tag{20e}$$

Notice that the effect of introducing the φ variables is to create a box around the possible values of each dual variable u_i, i.e., $0 \leq u_i \leq \overline{u}_i$. If the optimal dual solution $\boldsymbol{u}^*$ corresponding to the lower bound computed at a given node, features $u_i^* = \overline{u}_i$ for some customer $i \in N$, the corresponding lower bound may be lower than the optimal solution cost of the corresponding non-stabilized master problem. Therefore, at each node, we apply an iterative procedure where we first initialize $\overline{u}_i = \min_{j \in N_0, k \in N_0' : i \neq j \neq k} \{\max\{t_{jk}^T, t_{ji}^D + t_{ik}^D\}\}$, and, once (20) is solved, we check if in the corresponding dual solution $u_i^* = \overline{u}_i$ for some $i \in N$ and, if this happens, (20) is solved again by setting $\overline{u}_i = \infty$ for all $i \in N$ such that $u_i^* = \overline{u}_i$.

5.4 Branching Strategy

At each node of the search tree, whenever the optimal fractional solution is not integer, we perform a binary branching based on three types of decisions. Given an optimal solution $\boldsymbol{\xi}^*$ of the master problem and the corresponding ng-routes, we compute (i) how many times each customer is served by the drone alone (let us call such a value y_i^{*D} for each customer $i \in N$), (ii) how many times each arc $(i,j) \in \mathcal{A}$ is traversed by the truck (let us call such a value x_{ij}^{*T} for each arc $(i,j) \in \mathcal{A}$), and (iii) how many times each arc $(i,j) \in \mathcal{A}$ is traversed by the drone alone (let us call such a value w_{ij}^* for each arc $(i,j) \in \mathcal{A}$).

A three-level hierarchical branching is performed. First, we check if y_i^{*D} is not zero or one for any customer $i \in N$, and we branch on the customer for which y_i^{*D} is closest to 0.5. Let i^* be such a customer, i.e., $i^* = \arg\min_{i \in N} |y_i^{*D} - 0.5|$. Two child nodes are generated by setting that i^* is a drone customer (i.e., $y_{i^*}^{*D} = 1$) or i^* cannot be a drone customer (i.e., $y_{i^*}^{*D} = 0$). If all values y_i^{*D} are either zero or one, then we branch on the arc $(i,j) \in \mathcal{A}$ for which x_{ij}^{*T} is closest to 0.5. Let (i^*, j^*) be such an arc, i.e., $(i^*, j^*) = \arg\min_{(i,j) \in \mathcal{A}} |x_{ij}^{*T} - 0.5|$. Two child nodes are generated by setting that (i^*, j^*) is traversed by the truck (i.e., $x_{ij}^{*T} = 1$) or not (i.e., $x_{ij}^{*T} = 0$). Finally, if all values x_{ij}^{*T} are equal to zero or one, we check if w_{ij}^* is not zero or one for any arc $(i,j) \in \mathcal{A}$, and we branch on the arc $(i,j) \in \mathcal{A}$ for which w_{ij}^* is closest to 0.5. Let (i^*, j^*) be such an arc, i.e., $(i^*, j^*) = \arg\min_{(i,j) \in \mathcal{A}} |w_{ij}^* - 0.5|$. Two child nodes are generated by setting that (i^*, j^*) is a drone arc (i.e., $w_{ij}^* = 1$) or cannot be a drone arc (i.e., $w_{ij}^* = 0$).

Notice that all these branching decisions are *robust* when it comes to handling them in the pricing problem. Indeed, they can be taken into account when propagating each function and do not require additional state variables/resources.

6 Computational Results

In this section, we report the computational results achieved by the compact formulation (hereafter CF) described in Section 3 and by the branch-and-price algorithm (hereafter BP) described in Section 5 on the TSP-D and its variants discussed above. This section is organized as follows. In Section 6.1, we describe the test instances used in the experiments. Section 6.2 summarizes the results on the TSP-D. Sections 6.3-6.8 describe the results achieved on the TSP-D with the considered side constraints. The results for some of the TSP-D variants from the literature are reported in Section 6.9.

Both CF and BP were coded in C and compiled with GNU C++ 7.4. We use CPLEX 12.9 to solve CF and to solve the restricted master problem in BP. The default parameter setting is used to solve CF except

for parameters CPX_PARAM_THREADS, set equal to 1 to use a single thread, and CPX_PARAM_EPINT, set equal to 1e-10 to decrease integrality tolerance—indeed, with the default value of 1e-05 a few optimal solutions cannot be found. As to BP, the default parameter setting is used except for parameters CPX_PARAM_THREADS (set equal to 1) and CPX_PARAM_LPMETHOD set equal to 1 to use the primal simplex when solving the restricted master problem at each node of the search tree. All experiments are conducted on a single thread of an Intel Xeon E5-2670v2 machine running at 2.5 GHz. We impose a time limit of one hour and a memory limit of 8 GB RAM to solve each instance.

6.1 Test Instances

We test CF and BP on the test instances introduced by Poikonen et al. (2019), which are publicly available at http://stefan-poikonen.net/tspd_instance_data.zip. In particular, we considered instances with 9, 19, 29, and 39 customers. For each size, 25 instances are available. These instances were created by randomly locating the depot and the n customers on a 50-by-50 grid, where the coordinates are distributed uniformly in each of the two dimensions. Given two locations i and j and the corresponding x-y coordinates (x_i, y_i) and (x_j, y_j), the truck travel time t^T_{ij}, and the drone travel time t^D_{ij}, are computed as $t^T_{ij} = \lfloor |x_i - x_j| + |y_i - y_j| \rfloor$ and $t^D_{ij} = \lfloor \frac{((x_i-x_j)^2+(y_i-y_j)^2)^{0.5}}{\alpha} \rfloor$, respectively. The truck follows the taxicab metric while the drone follows the Euclidean distance metric at a speed that is α times faster the truck. We test all instances with $\alpha = 1, 2, 3$ in order to assess the impact of the drone speed compared to the truck speed and to have a sufficiently large set of 300 instances.

6.2 Computational Results on the TSP-D

Table 2 reports the results achieved on the TSP-D. The table summarizes the results for each size (n = 9, 19, 29, 39) and each value of α ($\alpha = 1, 2, 3$). Column #inst reports the number of instances considered (i.e., 25). For CF and BP, column Opt indicates the number of instances solved to optimality within the time limit, and Cpu indicates the average computing time over the instances solved. For BP, column Gap_{root} indicates the average gap at the root node between the optimal solution value and the root lower bound (computed over the instances solved to optimality only). The last two columns compare the results achieved by the BP with the results achieved by the CF: Opt is the number of instances solved by both methods, and Rt_{Cpu} is the average ratio between the computing time of BP and the computing time of CF over the instances solved by both methods only. Tables 4, 6, 7, 9, 10, 11, 13, 15, 17, 19, 20 reported in the next sections have the same format of Table 2.

Table 2: Summary of the computational results on the TSP-D

			CF		BP			BP vs CF	
n	α	#inst	Opt	Cpu	Gap_{root}	Opt	Cpu	Opt	Rt_{Cpu}
9	1	25	25	7.7	1.35%	25	0.1	25	0.01
9	2	25	25	7.4	1.40%	25	0.1	25	0.01
9	3	25	25	5.1	0.62%	25	0.1	25	0.02
19	1	25	0		2.20%	25	52.5	0	
19	2	25	0		2.51%	25	15.1	0	
19	3	25	0		3.30%	25	10.9	0	
29	1	25	0		1.89%	19	1291.9	0	
29	2	25	0		3.00%	24	491.8	0	
29	3	25	0		4.28%	24	470.0	0	
39	1	25	0		1.66%	1	3438.2	0	
39	2	25	0		1.55%	3	2553.0	0	
39	3	25	0		2.00%	5	1986.2	0	
All		300	75	6.7	2.26%	226	312.5	75	0.01

Table 3: Comparison of the computational results on the TSP-D by varying the relative drone speed

		$\alpha = 2$ vs $\alpha = 1$		$\alpha = 3$ vs $\alpha = 1$	
n	Opt	Δ_{t^*}	Rt_{Cpu}	Δ_{t^*}	Rt_{Cpu}
9	25	-19.0%	1.2	-29.1%	0.9
19	25	-18.5%	0.6	-26.3%	0.5
29	19	-18.8%	0.6	-25.9%	0.7
39	0				

Table 2 shows that CF can solve all 75 instances with 9 customers, but none of the other instances. BP can solve 226 of the 300 instances: all instances with 9 and 19 customers, all but 8 instances with 29

customers, and 9 of the 75 instances with 39 customers. The lower bound at the root node of BP is of good quality (on average 2.26% from optimality on the instances solved). Moreover, the comparison between BP and CF shows that all 75 instances solved by CF can be solved by BP, but the computing time taken by BP is about 1% of the computing time required by CF. By looking at the results achieved by BP when varying α, we can observe that when α is equal to 1 (i.e., the drone is slower), instances seem to be more difficult to solve.

Table 3 sheds more light on the results reported in Table 2 by comparing the results achieved on the 25 instances of each size when $\alpha = 2$ versus $\alpha = 1$ and when $\alpha = 3$ versus $\alpha = 1$. Each row of the table indicates the size (n), the number of instances solved to optimality (Opt) for all values of α, the average difference between the optimal solution values of the instances solved to optimality (Δ_{t^*}), and the ratio between the computing times over the instances solved to optimality for any value of α ($\mathsf{Rt}_{\mathsf{Cpu}}$). In the next sections, Tables 5, 8, 12, 14, 16, and 18 have the same format as Table 3.

Table 3 shows that when $\alpha = 2$ (i.e., the drone is twice as fast as the truck), the optimal completion times decrease, on average, by 18-19% compared to when $\alpha = 1$ (i.e., the drone and the truck are equally fast). When $\alpha = 3$, the optimal completion times decrease further by 5-10%. We can also see that solving instances with $\alpha = 2$ or 3 is usually less time-consuming than solving instances with $\alpha = 1$.

6.3 Computational Results When Loops Are Allowed

Table 4 reports the results obtained on the 300 instances previously considered when loops are allowed. CF could solve all instances with 9 customers, but none of the other instances. BP could solve all instances with 9 and 19 customers, all but 12 instances with 29 customers, and 6 instances with 39 customers. The BP takes about 1% of the computing time of CF to solve the 9-customer instances, and it seems that instances with $\alpha = 1$ are more challenging than instances with $\alpha = 2$ or 3. All in all, Tables 2 and 4 suggest similar conclusions.

Table 4: Summary of the computational results when loops are allowed

			CF		BP			BP vs CF	
n	α	#inst	Opt	Cpu	$\mathsf{Gap}_{\mathsf{root}}$	Opt	Cpu	Opt	$\mathsf{Rt}_{\mathsf{Cpu}}$
9	1	25	25	12.5	1.38%	25	0.1	25	0.02
9	2	25	25	10.6	1.51%	25	0.1	25	0.01
9	3	25	25	6.2	0.25%	25	0.1	25	0.02
19	1	25	0		2.25%	25	55.3	0	
19	2	25	0		2.46%	25	21.5	0	
19	3	25	0		3.26%	25	16.7	0	
29	1	25	0		1.83%	18	1273.7	0	
29	2	25	0		2.86%	23	785.4	0	
29	3	25	0		4.53%	22	975.0	0	
39	1	25	0		1.30%	2	3160.5	0	
39	2	25	0			0		0	
39	3	25	0		1.48%	4	2286.3	0	
All		300	75	6.7	2.21%	219	366.4	75	0.01

Table 5: Comparison of the computational results between allowing and preventing loops

n	α	Opt	Δ_{t^*}	$\mathsf{Rt}_{\mathsf{Cpu}}$
9	1	25	0.0%	1.6
9	2	25	-1.3%	1.3
9	3	25	-3.8%	1.3
19	1	25	0.0%	1.1
19	2	25	-0.4%	1.2
19	3	25	-1.9%	1.9
29	1	18	0.0%	1.2
29	2	22	-0.7%	1.6
29	3	22	-2.2%	3.3
39	1	1	0.0%	0.9
39	2	0	-	
39	3	2	-3.9%	1.6

Table 5 compares the results achieved when loops are allowed vs. when loops are prevented. We can see that, when $\alpha = 1$, loops are not beneficial and do not allow decreasing the optimal completion times. When $\alpha = 2$ or $\alpha = 3$, the optimal completion times decrease on average by up to 3.9%. However, when loops are allowed, it is significantly more time-consuming to close the instances to optimality.

6.4 Computational Results with Incompatible Customers

To assess the impact of having incompatible customers, we test the 300 instances previously considered by preventing the drone to visit, first, 20% of the customers and then 40% of them. In particular, when 20% of the customers are incompatible, we prevent the drone from serving customers 1, 6, 11, etc. When 40% of

the customers are incompatible, the drone cannot serve alone customers 1, 2, 6, 7, 11, 12, etc. Tables 6 and 7 summarize the results achieved by CF and BP for those settings.

Table 6: Summary of the computational results when 20% of customers are incompatible

			CF		BP			BP vs CF	
n	α	#inst	Opt	Cpu	Gap_{root}	Opt	Cpu	Opt	Rt_{Cpu}
9	1	25	25	3.7	0.55%	25	0.1	25	0.03
9	2	25	25	3.6	0.35%	25	0.1	25	0.02
9	3	25	25	2.1	0.20%	25	0.0	25	0.04
19	1	25	1	2876.3	1.73%	25	29.4	1	0.00
19	2	25	1	3416.7	1.40%	25	7.3	1	0.00
19	3	25	3	1696.2	1.63%	25	9.1	3	0.01
29	1	25	0		1.52%	22	1039.7	0	
29	2	25	0		1.43%	23	551.6	0	
29	3	25	0		1.67%	25	568.8	0	
39	1	25	0			0		0	
39	2	25	0		0.69%	6	1706.9	0	
39	3	25	0		0.85%	4	2797.7	0	
All		300	80	145.2	1.14%	230	314.6	80	0.03

Table 6 shows that CF can solve all instances with 9 customers and 5 instances with 19 customers. BP can solve all instances with 9 and 19 customers, all but 5 instances with 29 customers, and 10 instances with 39 customers. Instances with $\alpha = 1$ are more challenging than the other instances. On average, the computing time taken by BP is 3% of the computing time taken by CF.

Table 7: Summary of the computational results when 40% of customers are incompatible

			CF		BP			BP vs CF	
n	α	#inst	Opt	Cpu	Gap_{root}	Opt	Cpu	Opt	Rt_{Cpu}
9	1	25	25	1.7	0.11%	25	0.1	25	0.06
9	2	25	25	0.8	0.13%	25	0.1	25	0.11
9	3	25	25	1.0	0.00%	25	0.1	25	0.17
19	1	25	9	1887.3	1.03%	25	23.1	9	0.01
19	2	25	14	1227.8	0.22%	25	7.8	14	0.02
19	3	25	19	1329.8	0.05%	24	13.5	18	0.02
29	1	25	0		0.84%	22	602.5	0	
29	2	25	0		0.66%	24	712.1	0	
29	3	25	0		0.52%	20	1185.0	0	
39	1	25	0		0.12%	1	1972.9	0	
39	2	25	0		0.37%	1	3517.2	0	
39	3	25	0		0.00%	1	3408.0	0	
All		300	117	508.8	0.38%	218	293.8	116	0.08

Table 8: Comparison of the computational results when some customers are incompatible

			20 vs 0		40 vs 0	
n	α	Opt	Δ_{t^*}	Rt_{Cpu}	Δ_{t^*}	Rt_{Cpu}
9	1	25	3.4%	1.0	8.5%	0.9
9	2	25	11.7%	1.0	30.0%	0.9
9	3	25	22.5%	0.9	49.6%	1.2
19	1	25	2.3%	0.9	6.5%	0.7
19	2	25	7.0%	1.1	20.5%	1.0
19	3	24	11.6%	2.5	31.5%	3.9
29	1	17	2.2%	1.0	7.0%	0.8
29	2	22	7.7%	2.3	20.8%	2.6
29	3	20	12.3%	2.5	31.4%	7.0
39	1	0	-	-	-	-
39	2	0	-	-	-	-
39	3	0	-	-	-	-

Table 7 shows that CF can solve all instances with 9 customers and many of the 19-customer instances (i.e., 42 of the 75 instances). BP can solve all instances with 9 customers, all but one instance with 19 customers, all but 9 instances with 29 customers, and 3 instances with 39 customers. On average, BP takes 8% of the time required by CF to close an instance to optimality.

By comparing the results of Tables 2, 6, and 7, we can see that the presence of incompatible customers does not change the complexity of the problem. Indeed, the number of instances solved to optimality by BP and the average computing time is similar in all three settings.

Table 8 compares the results achieved by BP when 20% and 40% of the customers are incompatible with the results of Section 6.2. We can observe that the presence of incompatible customers can significantly

increase the optimal completion times (by up to 49.6% on average) and the computing time.

6.5 Computational Results with Drone Flying Range

To assess the impact of the drone flying range on the optimal completion time and on the performance of CF and BP, we test the 300 instances of Section 3.1 by setting the drone flying range, e, equal to 30, 20, and 10. The corresponding results are summarized in Tables 9, 10, and 11.

Table 9: Summary of the computational results with drone flying range $e = 30$

			CF			BP		BP vs CF	
n	α	#inst	Opt	Cpu	Gap_{root}	Opt	Cpu	Opt	Rt_{Cpu}
9	1	25	25	4.6	0.11%	25	0.0	25	0.02
9	2	25	25	6.3	2.36%	25	0.1	25	0.01
9	3	25	25	6.4	0.70%	25	0.1	25	0.01
19	1	25	0		2.75%	25	11.6	0	
19	2	25	0		3.07%	25	13.4	0	
19	3	25	0		3.35%	25	9.9	0	
29	1	25	0		3.44%	22	605.1	0	
29	2	25	0		3.29%	25	353.9	0	
29	3	25	0		4.06%	23	382.1	0	
39	1	25	0		2.05%	9	962.4	0	
39	2	25	0		2.59%	12	1967.6	0	
39	3	25	0		1.99%	7	1494.7	0	
All		300	75	5.8	2.51%	248	300.7	75	0.01

Table 10: Summary of the computational results with drone flying range $e = 20$

			CF			BP		BP vs CF	
n	α	#inst	Opt	Cpu	Gap_{root}	Opt	Cpu	Opt	Rt_{Cpu}
9	1	25	25	1.2	0.11%	25	0.0	25	0.06
9	2	25	25	3.5	2.30%	25	0.1	25	0.03
9	3	25	25	3.8	2.85%	25	0.1	25	0.03
19	1	25	11	1361.1	0.52%	25	3.4	11	0.01
19	2	25	1	1318.7	4.01%	25	10.5	1	0.00
19	3	25	0		4.13%	25	8.3	0	
29	1	25	0		2.42%	24	282.7	0	
29	2	25	0		3.68%	25	246.7	0	
29	3	25	0		4.77%	25	423.4	0	
39	1	25	0		2.62%	5	1285.4	0	
39	2	25	0		2.96%	12	1872.9	0	
39	3	25	0		2.49%	11	1787.9	0	
All		300	87	189.7	2.75%	252	288.4	87	0.04

Table 9 shows that CF can solve all instances with 9 customers, but none of the other instances. BP can solve all instances with 9 and 19 customers, all but 5 instances with 29 customers, and 28 instances with 39 customers. BP can solve all instances solved by CF in about 1% of the computing time. Moreover, the drone speed does not have an impact on the complexity of the instances. If we compare the results of BP with $e = 30$ and without flying range (i.e., Tables 9 and 2), we can see that with $e = 30$, 22 additional instances can be solved to optimality.

Table 10 shows that CF can solve all instances with 9 customers and 12 instances with 19 customers. BP can solve all instances with 9 and 19 customers, all but one instance with 29 customers, and 28 instances with 39 customers. On average, BP takes about 4% of the computing time taken by CF. Moreover, we can see that the results obtained with $e = 20$ and with $e = 30$ are quite similar.

Table 11 summarizes the results achieved with $e = 10$. We can see that having such a tight flying range allows CF to solve all instances with 9 customers, most of the instances with 19 customers (59 out of 75 instances), and 9 instances with 29 customers. BP can solve 260 of the 300 instances, which is 8 more than with $e = 20$, 12 more than with $e = 30$, and 34 more than without drone flying range.

Table 12 compares the results achieved with different drone flying ranges ($e = 30, 20, 10$) with the case without drone flying range. As expected, we can observe that the lower the drone flying range, the higher the average optimal completion time. When $e = 10$, the average completion time can increase by up to 57.1% compared with the case without drone flying range. We cannot observe any clear trend in the computing time required to close instances with drone flying range compared to instances without drone flying range.

For the sake of conciseness, we did not perform experiments for the TSP-D with weight- and arc-dependent drone flying range since this would involve an extensive discussion on instance generation and reasonable energy consumption functions.

6.6 Computational Results when the Drone Cannot Land and Wait

In this section, we want to assess the impact of preventing the drone from landing and waiting for the truck. We consider the instances of Section 6.5 with $e = 20$ and add the side constraint that the drone cannot land and wait. A summary of the results is provided in Table 13.

Table 11: Summary of the computational results with drone flying range $e = 10$

			CF			BP		BP vs CF	
n	α	#inst	Opt	Cpu	Gap_{root}	Opt	Cpu	Opt	Rt_{Cpu}
9	1	25	25	0.2	0.00%	25	0.0	25	0.30
9	2	25	25	1.0	0.18%	25	0.0	25	0.08
9	3	25	25	2.4	0.58%	25	0.1	25	0.03
19	1	25	25	97.4	0.25%	25	10.3	25	0.62
19	2	25	21	912.2	0.92%	25	3.4	21	0.02
19	3	25	13	1855.1	4.09%	25	9.5	13	0.00
29	1	25	9	583.7	0.42%	23	164.0	8	0.02
29	2	25	0		3.76%	25	351.2	0	
29	3	25	0		5.91%	25	256.3	0	
39	1	25	0		0.54%	20	404.7	0	
39	2	25	0		4.24%	11	1780.2	0	
39	3	25	0		3.35%	6	2075.8	0	
All		300	143	357.0	1.85%	260	229.5	142	0.19

Table 12: Comparison of the computational results with different drone flying ranges ($e = 30, 20, 10$)

			30 vs 0		20 vs 0		10 vs 0	
n	α	Opt	Δ_{t^*}	Rt_{Cpu}	Δ_{t^*}	Rt_{Cpu}	Δ_{t^*}	Rt_{Cpu}
9	1	25	17.6%	0.6	25.0%	0.5	30.1%	0.4
9	2	25	6.2%	1.0	25.5%	1.1	51.0%	0.6
9	3	25	0.2%	1.0	13.5%	1.5	57.2%	1.1
19	1	25	8.2%	0.4	17.5%	0.1	29.9%	0.4
19	2	25	1.0%	0.8	5.3%	0.8	33.6%	0.6
19	3	25	0.1%	1.1	2.3%	1.1	19.4%	2.1
29	1	16	3.9%	0.3	12.4%	0.2	29.7%	0.0
29	2	24	0.3%	0.6	1.6%	0.5	21.5%	1.5
29	3	23	0.0%	1.2	0.5%	1.2	8.4%	1.0
39	1	0	-	-	-	-	-	-
39	2	0	-	-	-	-	-	-
39	3	2	0.0%	1.0	0.0	0.5%	2.3%	1.7

Table 13: Summary of the computational results with drone flying range $e = 20$ if the drone cannot land and wait for the truck

			CF			BP		BP vs CF	
n	α	#inst	Opt	Cpu	Gap_{root}	Opt	Cpu	Opt	Rt_{Cpu}
9	1	25	25	1.7	0.00%	25	0.0	25	0.03
9	2	25	25	28.0	0.61%	25	0.0	25	0.00
9	3	25	25	41.9	1.60%	25	0.0	25	0.00
19	1	25	5	1975.0	0.31%	25	2.1	5	0.00
19	2	25	0		1.76%	25	1.4	0	
19	3	25	0		3.89%	25	4.0	0	
29	1	25	0		1.54%	25	44.9	0	
29	2	25	0		3.99%	25	123.2	0	
29	3	25	0		5.03%	25	167.4	0	
39	1	25	0		2.45%	16	879.2	0	
39	2	25	0		3.58%	19	1395.9	0	
39	3	25	0		2.81%	15	1034.6	0	
All		300	80	145.8	2.25%	275	235.2	80	0.01

Table 14: Comparison of the computational results on instances with $e = 20$ if the drone cannot or can land and wait for the truck

n	α	Opt	Δ_{t^*}	Rt_{Cpu}
9	1	25	1.3%	0.7
9	2	25	6.4%	0.4
9	3	25	13.9%	0.5
19	1	25	3.1%	0.8
19	2	25	5.5%	0.4
19	3	25	6.0%	0.6
29	1	24	2.5%	0.5
29	2	25	2.6%	0.5
29	3	25	1.4%	0.4
39	1	4	3.2%	1.2
39	2	11	0.5%	0.6
39	3	10	0.4%	0.6

Table 13 shows that CF can solve all instances with 9 customers and 5 instances with 19 customers. BP can solve all instances with up to 29 customers and 50 out of 75 instances with 39 customers. If we compare the results of Tables 10 and 13, we can see that 23 additional instances can be solved if the drone cannot land and wait and the average computing time is on average lower. This suggests that instances are in general easier to solve if the drone cannot land and wait.

Table 14 compares the results achieved by BP when the drone cannot land and wait for the truck with the case where the drone is allowed to land and wait. We can observe that, if the drone cannot land and wait, the average optimal completion time increases by up to 13.9%. Nevertheless, the average computing time required by BP to solve instances where the drone cannot land and wait is usually lower.

6.7 Computational Results with Launch and Rendezvous Time

In this section, we assess the impact of considering launch and rendezvous times. We consider the 300 instances of Section 6.2 with $lt = 1$ and $rt = 1$, as suggested, e.g., by Murray and Chu (2015). The results achieved by CF and BP are summarized in Table 15.

Table 15: Summary of the computational results with launch and rendezvous time equal to 1

			CF			BP		BP vs CF	
n	α	#inst	Opt	Cpu	Gap_{root}	Opt	Cpu	Opt	Rt_{Cpu}
9	1	25	25	9.8	1.29%	25	0.1	25	0.02
9	2	25	25	8.4	1.28%	25	0.1	25	0.01
9	3	25	25	5.4	0.66%	25	0.1	25	0.02
19	1	25	4	3009.5	1.39%	25	33.5	4	0.00
19	2	25	0		2.16%	25	11.5	0	
19	3	25	0		2.64%	25	8.9	0	
29	1	25	0		1.71%	22	1155.4	0	
29	2	25	0		2.56%	25	475.4	0	
29	3	25	0		3.45%	25	431.2	0	
39	1	25	0		0.68%	2	2446.9	0	
39	2	25	0		1.40%	2	2403.6	0	
39	3	25	0		1.51%	5	1615.1	0	
All		300	79	159.8	1.88%	231	291.0	79	0.02

Table 16: Comparison of the computational results on instances with $lt = rt = 1$ and $lt = rt = 0$

n	α	Opt	Δ_{t^*}	Rt_{Cpu}
9	1	25	3.1%	1.3
9	2	25	6.8%	1.0
9	3	25	8.7%	1.3
19	1	25	4.3%	0.8
19	2	25	10.4%	0.9
19	3	25	13.0%	1.1
29	1	18	6.4%	0.9
29	2	24	13.9%	0.9
29	3	24	17.4%	1.3
39	1	0	-	-
39	2	0	-	-
39	3	1	23.5%	1.1

Table 15 shows that CF can solve all 9-customer instances and four instances with 19 customers. BP can solve all instances with up to 19 customers, all but three instances with 29 customers, and nine instances with 39 customers. BP takes about 2% of the time taken by CF to solve instances to optimality. If we compare the results of Table 15 with the results of Table 2, we can see that a similar number of instances can be solved by BP in roughly the same computing time, so the launch and the rendezvous times do not affect the complexity of the problem.

Table 16 compares the results summarized in Table 2 with the results summarized in Table 15. As expected, we can see that the presence of launch and rendezvous times increases the optimal completion times, but there is no clear impact on the average computing time.

6.8 Computational Results without Truck Customers

As discussed in Section 4.2, handling the maximum number of truck customers per truck leg, $\overline{n}$, is not trivial in the DP recursion. Nevertheless, the special case where $\overline{n} = 0$ can easily be handled. In this section, we summarize the results in such a case, i.e., where truck customers are not allowed.

Table 17: Comparison of the computational results on instances without truck customers (i.e., $\overline{n} = 0$)

			CF			BP		BP vs CF	
n	α	#inst	Opt	Cpu	Gap_{root}	Opt	Cpu	Opt	Rt_{Cpu}
9	1	25	25	5.9	0.36%	25	0.0	25	0.02
9	2	25	25	12.2	1.37%	25	0.1	25	0.01
9	3	25	25	6.5	0.69%	25	0.0	25	0.01
19	1	25	10	1116.4	1.54%	25	3.0	10	0.00
19	2	25	0		2.97%	25	6.2	0	
19	3	25	0		3.66%	25	6.3	0	
29	1	25	0		1.21%	24	97.0	0	
29	2	25	0		3.08%	25	85.1	0	
29	3	25	0		4.52%	25	187.2	0	
39	1	25	0		1.07%	19	557.1	0	
39	2	25	0		2.89%	16	938.7	0	
39	3	25	0		3.43%	17	1318.0	0	
All		300	85	138.5	2.21%	276	208.5	85	0.01

Table 18: Comparison of the computational results on instances with $\overline{n} = 0$ and $\overline{n} = \infty$

n	α	Opt	Δ_{t^*}	Rt_{Cpu}
9	1	25	11.7%	0.5
9	2	25	3.6%	0.7
9	3	25	1.6%	0.7
19	1	25	12.7%	0.1
19	2	25	4.1%	0.4
19	3	25	1.6%	0.7
29	1	19	14.2%	0.1
29	2	24	3.3%	0.2
29	3	24	1.2%	0.3
39	1	1	15.2%	0.0
39	2	1	5.2%	0.2
39	3	4	1.8%	0.8

Table 17 summarizes the results achieved on the 300 instances of Section 6.2 with the additional constraint

that $\overline{n} = 0$. CF can solve all instances with nine customers and ten instances with 19 customers. BP can solve all but one instance with up to 29 customers and 52 instances with 39 customers. BP is clearly more efficient and takes about 1% of the computing time taken by CF to solve an instance. If we compare the results of Tables 2 and 17, we can see that if there are no truck customers, the problem is on average easier to solve and 50 instances more can be solved in less computing time.

A comparison of the results achieved by BP with $\overline{n} = 0$ (i.e., Table 17) and with $\overline{n} = \infty$ (i.e., Table 2) is provided in Table 18. We can see that preventing the truck from visiting customers alone results in higher optimal completion times, but the computing time needed by BP to find an optimal solution strongly decreases.

6.9 Computational Results on TSP-D from the Literature

The last set of computational results we present is aimed at comparing the computational performance of CF and BP with the best mathematical models from the literature.

Murray and Chu (2015), Ha et al. (2018), and Dell'Amico et al. (2019) consider—among others—a TSP-D where loops are not allowed, 20% of the customers are incompatible, $e = 20$, the drone cannot land and wait for the truck, $lt = rt = 1$, and $\overline{n} = \infty$. They test several MILP models on 10-customer instances. Table 19 summarizes the results achieved by CF and BP on the instances considered in this paper in a setting similar to Murray and Chu (2015), Ha et al. (2018), and Dell'Amico et al. (2019), in order to show the potential of our approaches on a TSP-D already addressed in the literature.

Table 19 shows that CF can solve all instances with nine customers and 14 instances with 19 customers. BP can solve all instances with up to 29 customers and 53 instances with 39 customers. When comparing the results of Tables 2 and 19, we can observe that the additional constraints considered in Table 19 make the resulting problem easier to solve than the basic TSP-D. Even though the formulations of Murray and Chu (2015), Ha et al. (2018), and Dell'Amico et al. (2019) were tested on a different set of test instances and within a different experimental environment, as they cannot solve consistently instances with 10 customers, we can say that BP is certainly superior to their models from a computational viewpoint and CF is also competitive.

Poikonen et al. (2019) consider a TSP-D where loops are allowed, all customers can be visited by the drone alone, $e = 20$, the drone cannot land and wait for the truck, $lt = rt = 0$, and $\overline{n} = \infty$. The proposed heuristic branch-and-bound can solve to optimality all instances with nine customers.

Table 20 summarizes the results achieved by CF and BP on a computational setting similar to Poikonen et al. (2019). We can see that CF can solve all instances with nine customers and three instances with 19 customers. BP can solve all but one instance with up to 29 customers and 37 instances with 39 customers. When comparing the results of Tables 2 and 20, we can observe that the additional constraints considered in Table 20 make the resulting problem easier to solve than the basic TSP-D.

7 Conclusions

We have introduced a compact MILP formulation and an exact branch-and-price (BP) approach for several variants of the TSP-D. The compact formulation is easy to use, i.e., there is no need to implement separation procedures for exponentially-sized sets of valid inequalities or advanced decomposition methods. Nevertheless, Cplex is able to solve with it instances with a similar size (10 to 20 customers) as the ones which can be handled with state-of-the-art methods from the literature. The BP approach, however, is much more efficient in the sense that it can solve instances with up to 39 customers, significantly pushing forward the computational limits. It relies on a sophisticated dynamic programming recursion and speed-up techniques as *ng*-route relaxation, dual variable stabilization, and three-level hierarchical branching. Regarding the results also with respect to the additional side constraints considered, we observe that allowing loops, considering drone-incompatible customers, launch and rendezvous times, and preventing the drone from waiting for the truck on the ground, has no significant effect on the computational complexity and solution times. On the other hand, heavily restricting the set of feasible solutions by limiting the drone flying range or the number of customers a truck can serve alone in a row seems to make the problem easier to solve for the BP. This is quite intuitive since usually fewer non-dominated states are generated in the pricing problem.

Table 19: Summary of the computational results on instances similar to the ones in Murray and Chu (2015), Ha et al. (2018), and Dell'Amico et al. (2019)

			CF		BP			BP vs CF	
n	α	#inst	Opt	Cpu	Gap_{root}	Opt	Cpu	Opt	Rt_{Cpu}
9	1	25	25	1.2	0.00%	25	0.0	25	0.04
9	2	25	25	11.6	0.00%	25	0.0	25	0.00
9	3	25	25	24.7	0.81%	25	0.0	25	0.00
19	1	25	13	765.1	0.23%	25	3.4	13	0.02
19	2	25	0		0.86%	25	1.5	0	
19	3	25	1	2923.4	1.09%	25	1.7	1	0.00
29	1	25	0		1.00%	25	40.0	0	
29	2	25	0		1.89%	25	156.3	0	
29	3	25	0		1.25%	25	157.7	0	
39	1	25	0		1.95%	21	880.0	0	
39	2	25	0		1.69%	21	800.4	0	
39	3	25	0		1.13%	11	1286.1	0	
All		300	89	155.1	0.96%	278	210.2	89	0.01

Table 20: Summary of the computational results on instances similar to Poikonen et al. (2019)'s instances

			CF		BP			BP vs CF	
n	α	#inst	Opt	Cpu	Gap_{root}	Opt	Cpu	Opt	Rt_{Cpu}
9	1	25	25	2.2	0.00%	25	0.0	25	0.02
9	2	25	25	19.9	1.01%	25	0.0	25	0.00
9	3	25	25	47.0	1.16%	25	0.0	25	0.00
19	1	25	3	1044.1	0.26%	25	1.9	3	0.00
19	2	25	0		1.93%	25	2.7	0	
19	3	25	0		3.33%	25	3.4	0	
29	1	25	0		1.48%	25	57.4	0	
29	2	25	0		4.02%	25	266.7	0	
29	3	25	0		4.77%	24	366.7	0	
39	1	25	0		2.42%	15	991.9	0	
39	2	25	0		2.85%	13	1390.1	0	
39	3	25	0		2.35%	9	1472.4	0	
All		300	78	62.3	2.06%	261	242.6	78	0.01

There might be some room for improvement in our BP approach, e.g., by involving more advanced stabilization techniques, strong branching, by combining forward and backward propagation, and by using completion bounds in the pricing problem. Potential future work could be to extend the TSP-D toward multiple trucks, multiple drones per truck, and/or multiple customers visited in a single drone trip. It is not obvious how to adapt our solution approaches to handle these additional features without introducing many variables/constraints and symmetry in the compact formulation and a large number of resources in the dynamic programming recursion within our BP.

References

N. Agatz, P. Bouman, and M. Schmidt. Optimization approaches for the traveling salesman problem with drone. *Transportation Science*, 52(4):965–981, 2018.

R. Baldacci, A. Mingozzi, and R. Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations research*, 59(5):1269–1283, 2011.

P. Bouman, N. Agatz, and M. Schmidt. Dynamic programming approaches for the traveling salesman problem with drone. *Networks*, 72(4):528–542, 2018.

N. Boysen, D. Briskorn, S. Fedtke, and S. Schwerdfeger. Drone delivery from trucks: Drone scheduling for given truck routes. *Networks*, 72(4):506–527, 2018.

J. F. Campbell, D. Sweeney, and J. Zhang. Strategic design for delivery with trucks and drones. Technical report, University of Missouri, 2017.

J. G. Carlsson and S. Song. Coordinated logistics with a truck and a drone. *Management Science*, 64(9): 4052–4069, 2017.

C. Cheng, Y. Adulyasak, and L.-M. Rousseau. Formulations and exact algorithms for drone routing problem. Technical Report CIRRELT-2018-31, CIRRELT, July 2018.

M. Dell'Amico, R. Montemanni, and S. Novellani. Models and algorithms for the flying sidekick traveling salesman problem. Technical report, Universita di Modena e Reggio Emilia, 2019.

K. Dorling, J. Heinrichs, G. G. Messier, and S. Magierowski. Vehicle routing problems for drone delivery. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(1):70–85, 2017.

M. Drexl. Synchronization in vehicle routing-a survey of vrps with multiple synchronization constraints. *Transportation Science*, 46(3):297–316, 2012.

C. Gambella, A. Lodi, and D. Vigo. Exact solutions for the carrier–vehicle traveling salesman problem. *Transportation Science*, 52(2):320–330, 2017.

Q. M. Ha, Y. Deville, Q. D. Pham, and M. H. Hà. On the min-cost traveling salesman problem with drone. *Transportation Research Part C: Emerging Technologies*, 86:597–621, 2018.

H. Y. Jeong, B. D. Song, and S. Lee. Truck-drone hybrid delivery routing: Payload-energy dependency and no-fly zones. *International Journal of Production Economics*, 214:220–233, 2019.

A. Kundu and T. I. Matis. A delivery time reduction heuristic using drones under windy conditions. In K. Coperich, E. Cudney, and H. Nembhard, editors, *Proceedings of the 2017 Industrial and Systems Engineering Conference*, pages 1864–1869, 2017.

A. Masone, S. Poikonen, and B. Golden. A continuous solution method for the multi-visit drone routing problem. Presentation at INFORMS Transportation Science and Logistics Conference 2019, 2019.

C. Murray and R. Raj. The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones. Technical Report SSRN 3338436, University of Buffalo, NY, USA, 2019.

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

A. Otto, N. Agatz, J. Campbell, B. Golden, and E. Pesch. Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey. *Networks*, 72(4):411–458, 2018.

R. Paradiso, R. Roberti, D. Lagan, and W. Dullaert. An exact solution framework for multi-trip vehicle routing problems with time windows. *Operations Research*, 2019. (accepted).

J. Peters. UPS just won FAA approval to fly as many delivery drones as it wants. The Verge, Oct. 2019. URL `https://www.theverge.com/2019/10/1/20893655/ups-faa-approval-delivery-drones-airline-amazon-air-uber-eats-alphabet-wing`. Accessed on 2019-10-31.

A. Pigatti, M. P. de Arago, and E. Uchoa. Stabilized branch-and-cut-and-price for the generalized assignment problem. *Electronic Notes in Discrete Mathematics*, 19:389 – 395, 2005. doi: https://doi.org/10.1016/j.endm.2005.05.052. Proceedings of GRACO2005.

S. Poikonen and B. Golden. Multi-visit drone routing problem. *Computers & Operations Research*, 113: 104802, 2020.

S. Poikonen, X. Wang, and B. Golden. The vehicle routing problem with drones: Extended models and connections. *Networks*, 70(1):34–43, 2017.

S. Poikonen, B. Golden, and E. A. Wasil. A branch-and-bound approach to the traveling salesman problem with a drone. *INFORMS Journal on Computing*, 31(2):335–346, 2019.

M. Savelsbergh and T. Van Woensel. 50th anniversary invited articlecity logistics: Challenges and opportunities. *Transportation Science*, 50(2):579–590, 2016.

J. Vincent and C. Gartenberg. Here's Amazon's new transforming Prime Air delivery drone. The Verge, June 2019. URL `https://www.theverge.com/2019/6/5/18654044/amazon-prime-air-delivery-drone-new-design-safety-transforming-flight-video`. Accessed on 2019-10-31.

X. Wang, S. Poikonen, and B. Golden. The vehicle routing problem with drones: several worst-case results. *Optimization Letters*, 11(4):679–697, 2017.

Z. Wang and J.-B. Sheu. Vehicle routing problem with drones. *Transportation Research Part B: Methodological*, 122:350–364, 2019.

Proof of Dominance Rule 2.

Because of condition (18a), every propagation of $f(ng_2, k, i^T, i^D, \tau_2)$ is also a feasible propagation of $f(ng_1, k, i^T, i^D, \tau_1)$ (but not vice versa). This indicates that the set of feasible propagations of $f(ng_2, k, i^T, i^D, \tau_2)$ is a subset of the feasible propagations of $f(ng_1, k, i^T, i^D, \tau_1)$. In particular, any ng-route that can be obtained by propagating $f(ng_2, k, i^T, i^D, \tau_2)$ can be obtained by combining $f(ng_2, k, i^T, i^D, \tau_2)$ with a function of type $f(ng_b, n-k, i^T, i_b^D, \tau_b) = b$, such that $ng_2 \cap ng_b = \varnothing$ and $i_b^D \notin ng_2$.

Let $\tilde{d}_1$ be the reduced cost of the ng-route obtained by combining $f(ng_1, k, i^T, i^D, \tau_1)$ with $f(ng_b, n-k, i^T, i_b^D, \tau_b)$ where the drone performs drone leg $i^D \dashrightarrow j \dashrightarrow i_b^D$. Reduced cost $\tilde{d}_1$ is computed as

$$\tilde{d}_1 = f_1 + b + u_{i^T} - u_j + \max\{\tau_1 + \tau_b, t^D_{i^D j} + t^D_{j i_b^D}\} \tag{21}$$

Similarly, let $\tilde{d}_2$ be the reduced cost of the ng-route obtained by combining $f(ng_2, k, i^T, i^D, \tau_2)$ with $f(ng_b, n-k, i^T, i_b^D, \tau_b)$ where the drone performs drone leg $i^D \dashrightarrow j \dashrightarrow i_b^D$. Reduced cost $\tilde{d}_2$ is computed as

$$\tilde{d}_2 = f_2 + b + u_{i^T} - u_j + \max\{\tau_2 + \tau_b, t^D_{i^D j} + t^D_{j i_b^D}\} \tag{22}$$

To prove the dominance rule, we have to prove that $\tilde{d}_1$ cannot be higher than $\tilde{d}_2$ if $f(ng_1, k, i^T, i^D, \tau_1)$ and $f(ng_2, k, i^T, i^D, \tau_2)$ satisfy (18a)-(18c). There are six cases to consider:

1. If $\tau_1+\tau_b \le \tau_2+\tau_b \le t^D_{i^D j}+t^D_{j i_b^D}$, then $\tilde{d}_1 = f_1+b+u_{i^T}-u_j+t^D_{i^D j}+t^D_{j i_b^D}$ and $\tilde{d}_2 = f_2+b+u_{i^T}-u_j+t^D_{i^D j}+t^D_{j i_b^D}$. Because of (18b), $\tilde{d}_1 \le \tilde{d}_2$.

2. If $\tau_2+\tau_b \le \tau_1+\tau_b \le t^D_{i^D j}+t^D_{j i_b^D}$, then $\tilde{d}_1 = f_1+b+u_{i^T}-u_j+t^D_{i^D j}+t^D_{j i_b^D}$ and $\tilde{d}_2 = f_2+b+u_{i^T}-u_j+t^D_{i^D j}+t^D_{j i_b^D}$. Because of (18b), $\tilde{d}_1 \le \tilde{d}_2$.

3. If $\tau_1+\tau_b \le t^D_{i^D j}+t^D_{j i_b^D} \le \tau_2+\tau_b$, then $\tilde{d}_1 = f_1+b+u_{i^T}-u_j+t^D_{i^D j}+t^D_{j i_b^D}$ and $\tilde{d}_2 = f_2+b+u_{i^T}-u_j+\tau_2+\tau_b$, which implies $\tilde{d}_1 \le \tilde{d}_2$ because of (18b) and $t^D_{i^D j} + t^D_{j i_b^D} \le \tau_2 + \tau_b$.

4. If $t^D_{i^D j}+t^D_{j i_b^D} \le \tau_1+\tau_b \le \tau_2+\tau_b$, then $\tilde{d}_1 = f_1+b+u_{i^T}-u_j+\tau_1+\tau_b$ and $\tilde{d}_2 = f_2+b+u_{i^T}-u_j+\tau_2+\tau_b$, which implies $\tilde{d}_1 \le \tilde{d}_2$ because of condition (18b) and $\tau_1 + \tau_b \le \tau_2 + \tau_b$.

5. If $\tau_2+\tau_b \le t^D_{i^D j}+t^D_{j i_b^D} \le \tau_1+\tau_b$, then $\tilde{d}_1 = f_1+b+u_{i^T}-u_j+\tau_1+\tau_b$ and $\tilde{d}_2 = f_2+b+u_{i^T}-u_j+t^D_{i^D j}+t^D_{j i_b^D}$. Due to (18c) and because of condition $\tau_2 + \tau_b \le t^D_{i^D j} + t^D_{j i_b^D}$, we have $\tilde{d}_1 \le f_2 + b + u_{i^T} - u_j + \tau_2 + \tau_b \le f_2 + b + u_{i^T} - u_j + t^D_{i^D j} + t^D_{j i_b^D} = \tilde{d}_2$, so $\tilde{d}_1 \le \tilde{d}_2$.

6. $t^D_{i^D j}+t^D_{j i_b^D} \le \tau_2+\tau_b \le \tau_1+\tau_b$, then $\tilde{d}_1 = f_1+b+u_{i^T}-u_j+\tau_1+\tau_b$ and $\tilde{d}_2 = f_2+b+u_{i^T}-u_j+\tau_2+\tau_b$. Because of (18c), $\tilde{d}_1 \le \tilde{d}_2$.

As in all six cases $\tilde{d}_1 \le \tilde{d}_2$, so the dominance rule holds. □

Proof of Dominance Rule 3.

To simplify the notation of the proof, let us call $\sigma = \max_{j \in N, s \in N'_0 : j \neq s}\{t^D_{i^D j} + t^D_{js}\}$. Because of (19b), (19c), and (19d), we have

$$0 \le f^1 - f^2 \le \tau_2 - \tau_1 \qquad \text{and} \qquad 0 \le f^1 - f^2 \le \tau_2 - \sigma$$

As indicated in the proof of Dominance Rule 2, let $f(ng_b, n-k, i^T, i_b^D, \tau_b) = b$ be any function that can be combined with $f(ng_2, k, i^T, i^D, \tau_2)$ ($f(ng_1, k, i^T, i^D, \tau_1)$, respectively) to generate an ng-route, and let $\tilde{d}_2$ ($\tilde{d}_1$, respectively) be the reduced cost of the corresponding ng-route computed with (22) ((21), respectively).

As $\tau_2 - \sigma \ge 0$ and $\tau_b \ge 0$, then $\tau_2 \ge \sigma$ and $\tau_2 + \tau_b \ge \sigma$. Therefore, $\tilde{d}_2 = f_2 + b + u_{i^T} - u_j + \max\{\tau_2 + \tau_b, t^D_{i^D j} + t^D_{j i_b^D}\} = f_2 + b + u_{i^T} - u_j + \tau_2 + \tau_b$.

Because of (19c), we have

$$f^1 + b + u_{i^T} - u_j + \tau_1 + \tau_b \leq f^2 + b + u_{i^T} - u_j + \tau_2 + \tau_b = \tilde{d}_2$$

and because of (19d), we have

$$f^1 + b + u_{i^T} - u_j + t^D_{i^D j} + t^D_{js} \leq f^1 + b + u_{i^T} - u_j + \sigma \leq c(f_2 \oplus b \odot j) = f^2 + b + u_{i^T} - u_j + \tau_2 + \tau_b = \tilde{d}_2$$

that show that the reduced cost of the ng-route generated by combining $f(ng_1, k, i^T, i^D, \tau_1)$ and $f(ng_b, n-k, i^T, i^D_b, \tau_b)$ cannot be higher than the reduced cost of the ng-route generated by combining $f(ng_2, k, i^T, i^D, \tau_2)$ and $f(ng_b, n-k, i^T, i^D_b, \tau_b)$. Therefore, $f(ng_1, k, i^T, i^D, \tau_1)$ dominates $f(ng_2, k, i^T, i^D, \tau_2)$ if conditions (19a)-(19d) hold. □