The min-cost parallel drone scheduling vehicle routing problem

Minh Anh Nguyen[a], Giang Thi-Huong Dang[b], Minh Hoàng Hà[a,*], Minh-Trien Pham[c]

[a] *ORLab, Faculty of Computer Science, Phenikaa University, Hanoi, Vietnam*
[b] *University of Economic and Technical Industries, Hanoi, Vietnam*
[c] *VNU University of Engineering and Technology, Hanoi, Vietnam*

Abstract

Adopting unmanned aerial vehicles (UAV), also known as drones, into the last-mile-delivery sector and having them work alongside trucks with the aim of improving service quality and reducing the transportation cost gives rise to a new class of Vehicle Routing Problems (VRPs). In this paper, we introduce a new optimization problem called the min-cost Parallel Drone Scheduling Vehicle Routing Problem (PDSVRP). This problem is a variant of the well-known Parallel Drone Scheduling Traveling Salesman Problem (PDSTSP) recently introduced in the literature in which we allow multiple trucks and consider the objective of minimizing the total transportation costs. We formulate the problem as a Mixed Integer Linear Program and then develop a Ruin and Recreate ($\mathcal{R}\&\mathcal{R}$) algorithm. Exploiting PDSVRP solution characteristics in an effective manner, our heuristic manages to introduce "sufficient" rooms to a solution via new removal operators during the ruin phase. It is expected to enhance the possibilities for improving solutions later in the recreate phase. Multiple experiments on a new set of randomly generated instances confirm the performance of our approach. To explore the benefits of drone delivery as well as the insight into the impact of related factors on the contribution of drones' use to operational cost, a sensitivity analysis is conducted. We also adapt the proposed algorithm to solve the PDSTSP and validate it via benchmarks available in the literature. It is shown that our algorithm outperforms state-of-the-art algorithms in terms of solution quality. Out of 90 considered instances, it finds 26 new best known solutions.

Keywords: Transportation; vehicle routing problem; drone parallel scheduling; mixed integer linear programming; ruin and recreate metaheuristic

1. Introduction

The work of Murray & Chu (2015) pioneered a new routing problem in which a truck and a drone collaborate to make deliveries. From an operations research perspective, the authors present two new prototypical variants expanding from the traditional Traveling Salesman Problem (TSP) called the Flying Sidekick TSP (FSTSP) and the Parallel Drone Scheduling TSP (PDSTSP), respectively. Since then, these two variants have attracted considerable research interest, establishing a new research direction on routing with trucks and drones. In the two

*Corresponding author. The first and second authors share the same contribution.
Email address: hoang.haminh@phenikaa-uni.edu.vn (Minh Hoàng Hà)

variants, we are given a set of customers who require to receive parcels from a depot where a truck and a set of identical drones are located. A customer must be visited once, either by the truck or a drone. However, not all customers can be served by drones due to practical constraints such as overweight parcels or remote customers who are too far from the depot. The two problems are different in the way drones operate to make deliveries. The FSTSP considers only one drone working together with the truck. The drone is launched from the truck at a customer location, delivers parcels to another customer, and then returns to the truck at the location of a third customer. While the drone is on a trip, the truck delivers parcels to other customers as long as the drone has enough battery to hover and wait for the truck. Therefore, this way of operation requires a synchronization between two vehicles. In the PDSTSP, there are multiple drones working independently. Each drone can fly directly between the depot and its customers, and can perform multiple back-and-forth trips within the time horizon. The truck independently departs from the depot and serves a subset of customers before returning to the depot. Since the drones and the truck do not cooperate, there is no synchronization between them in this version of the problem. Both variants aim to find a Hamilton cycle for the truck and a set of trips for the drones with the objective function that minimizes the completion time required to serve all customers and return all vehicles (truck and drones) to the depot.

On the one hand, the FSTSP appears to attract more attention due to its intriguing concept and exquisite complexity. Based on the concept of the FSTSP, Carlsson & Song (2018) perform an asymptotic theoretical analysis in the Euclidean plane as well as a collection of computational experiments. The authors conclude that the improvement in efficiency is proportional to the square root of the ratio of the speeds of the truck and the drone. Because the preliminary formulations proposed in Murray & Chu (2015) and Agatz, Bouman & Schmidt (2018) appear to be hard to solve, multiple studies have developed more efficient solution methods (Poikonen, Golden & Wasil, 2019; Bouman, Agatz & Schmidt, 2018; Ponza, 2016; de Freitas & Penna, 2020). Other studies focus on extending the practice of the original model. Murray & Raj (2020) introduce the multiple FSTSP (mFSTSP) in which a truck operates in coordination with a heterogeneous fleet of drones. To reflect the reality of deploying multiple aircraft from a single truck, detailed queue scheduling for drone arrivals and departures is incorporated within a proposed Mixed Integer Linear Programming (MILP) formulation. Poikonen & Golden (2020b); Cheng, Adulyasak & Rousseau (2020) take into account the drone energy drain function and allow each drone to carry multiple packages. Unlike many papers in the current literature, their problem model not only allows for a drone to visit multiple customers during a flight (multi-visit drone mode) but also allows the specification of a drone energy drain function that depends on package weight. Savuran & Karakaya (2016); Poikonen & Golden (2020a) study the idea of using a big aircraft as a mobile depot for a fleet of drones that must visit a set of fixed targets. Rather than considering one single truck or carrier, Sacramento, Pisinger & Ropke (2019) extend the FSTSP by enabling multiple capacitated trucks, each of which is equipped with a single drone, to introduce a problem called the Vehicle Routing Problem with Drones (VRPD). Compared with traditional truck-only systems, the results show the clear advantage of synchronizing small drones with trucks for delivery with regard to operational cost.

On the other hand, the PDTSP appears to receive less attention despite its apparent practical

applications. The PDSTSP includes two decision layers. In the first layer, the customers are assigned to either a truck or a drone. The second layer relates to the routing optimization, which consists in solving two NP-hard problems: a TSP for the truck and a parallel machine scheduling (PMS) problem for the drones. Also in Murray & Chu (2015), a MILP formulation and a heuristic are introduced. With the aid of a MILP solver, the formulation can provide exact solutions, but it is time-consuming and not applicable for instances of practical size. The main principle of the heuristic is to partition the set of customers into two subsets (a truck set and a drone set). Then, a TSP tour is computed for the customers assigned to the truck and the PMS problem is solved to assign customer requests (jobs) to drones (machines). At the end, an improvement step is performed to reassign customers to either the drone set or the truck to better balance between the truck and drones' activities.

Mbiadou Saleu, Deroussi, Feillet, Grangeon & Quilliot (2018) propose the first metaheuristic composed of two steps. The first step (or coding step) transforms a feasible PDSTSP solution into a customer sequence, also called giant tour or TSP tour. The second step (or decoding step) decomposes the customer sequence into a tour for the truck and a set of trips for the drones. This decomposition must satisfy the requirement that the customers in the truck tour follow the same order as in the giant tour, and the customers that are not in the truck tour will be served by the drones. A dynamic programming-based algorithm is used to solve the decomposition problem. More recently, Dell'Amico, Montemanni & Novellani (2020) propose a matheuristic in which the authors use the same idea of coding and decoding steps as found in Mbiadou Saleu, Deroussi, Feillet, Grangeon & Quilliot (2018). However, instead of dynamic programming, a MILP approach is used to handle the decoding phase. This method is tested on instances with 48 to 229 customers, and the obtained results show that it is competitive with state-of-the-art methods. More remarkably, 28 new best known solutions out of the 90 instances have been found.

Several PDSTSP variants are also introduced and studied in the literature. Ham (2018) extends the PDSTSP by considering two different types of drone tasks: drop and pickup. After completing a drop, a drone can either return to the depot to deliver the next parcels or fly directly to another customer for pickup. The author proposes a multi-truck, multi-drone, and multi-depot scheduling problem constrained by time-window, drop-pickup, and multi-visit, with the objective function to minimize the maximum completion time of all tasks. Dayarian, Savelsbergh & Clarke (2020) focus on the delivery system where a truck makes deliveries and is regularly resupplied by a drone. The paper shows the potential benefits of using the drone to resupply the truck, in terms of both increasing the number of serviced orders and reducing the service time. Kim & Moon (2018) address the limitations of the drone flight range by proposing to build an additional drone station near customer areas for storing and launching drones, rather than at the depot. Aside from making deliveries to its customers, the truck is also responsible for supplying drone parcels at the drone station, allowing the activation of drone deliveries. Based on the same idea, Schermer, Moeini & Wendt (2019) consider opening drone stations as decision variables when assuming that some stations may be opened for drone deliveries and would incur a fixed cost. The authors model the problem with two popular objective function: minimizing the makespan and minimizing the overall cost. The numerical results show that

the utilization of drone stations can reduce the makespan significantly. Wang, Hu, Du, Zhou, Deng & Hu (2019) extend the PDSTSP by allowing multiple trucks to carry a drone working in tandem as in the FSTSP. They seek to coordinate truck-drone tandem deliveries, together with independent drones to transport parcels from the depot to the customers distributed in a wide region. Experimental results show that the proposed system model outperforms the existing models which employ either independent drones or truck-carried drones alone.

In the PDSTSP, one cares about the speed of servicing customers, e.g., the service quality of the transportation system. In logistics activities, another important factor that needs to be taken into account is the transportation cost. Several publications in the literature have investigated the objective function of minimizing the total transportation cost in routing problems using the truck-drone combination for delivery. Mathew, Smith & Waslander (2015) consider a related problem called the Heterogeneous Delivery Problem (HDP). Ha, Deville, Pham & Hà (2018); Sacramento, Pisinger & Ropke (2019) aim to minimize the operational cost for the FSTSP and VRPD, respectively. The similar objective function is also studied in the hybrid vehicle-drone routing problem (HVDRP) for pick-up and delivery services introduced in Karak & Abdelghany (2019).

In this research, we are interested in the cost aspect of the delivery system based on the PDSTSP problem by studying the objective function that minimizes the total operational cost incurred by the trucks and drones. In addition, we extend the PDSTSP by adding two new constraints and allowing multiple trucks. First, the total weight of parcels delivered to customers assigned to each truck does not exceed the truck capacity. Second, the total working time of each vehicle (both trucks and drones) must be less than or equal to a pre-defined value. As such, the studied problem is a generation of the well-known Vehicle Routing Problem (VRP) and called the Parallel Drone Scheduling Vehicle Routing Problem (or PDSVRP for abbreviation).

We are also aware of another paper that considers a quite similar truck-drone paradigm. Conceição (2019) extend the PDSTSP by allowing multiple truck routes. The paper also studies the objective function of minimizing the total transportation cost and proposes a MILP formulation based on three-index flows. Results from experiments on real data suggest that there is the potential for cost reduction in the proposed problem. Even though the idea seems similar, there are several major points that make our work different. First, the previous study does not consider truck capacity constraints without which it is rather a multiple Traveling Salesman Problem (mTSP). Second, the study has not taken into account any restraints on drone use (e.g., maximum working time). This could lead to solutions without the presence of multiple vehicle routes, especially in the min-cost variant where there is no cost difference between stacking all customers into one drone and separating them into multiple drone tours. Besides, drones have much lower transportation costs than trucks, making drone deliveries more favorable. Without the limit on the working duration, drones are extensively used while trucks could not be mobilized in the problem solution. As a consequence, no capacity constraint could make the problem easier to handle. Also in Conceição (2019), only a MILP model is proposed and small size problem instances are solved. This could limit insights about the impact of different factors on the problem. Last but not least, our MILP model adopts the two-index flow formulation, which is more efficient than the three-index flow formulation proposed in Conceição (2019), allowing

our model to handle larger instances.

Table 1 summarizes the main characteristics of drone-aided routing problems studied in the literature, from which the comparison between the characteristics of our new variant with those of other considered problems is described. The table classifies the papers into four major problem classes namely the FSTSP, the PDSTSP, the Drones Delivery Problem (DDP), and the Carrier-Vehicle Problem with Drones (CVP-D) Macrina, Pugliese, Guerriero & Laporte (2020). The DDP is basically a version of the VRP in which deliveries are made only by a fleet of drones. The CVP-D combines the other three variants by using drones to rapidly visit customers while using slow carriers (e.g., air ships or large ground vehicles) to support on the ground as supply bases for drones' multiple trips. Other features of the problems are also considered: allowing single or multiple drones (Column "# drones"), single or multiple trucks (Column "# carriers"), working time limit for vehicles (Column "Time limit"), vehicle capacity (Column "Carrier capacity"), parcel weight-dependent energy drain function (Column "Energy function"), ability of drones to visit multiple customers during a flight (Column "Multi-visit drone), objective function, and solution methods used. Interested readers are referred to the detailed survey by Macrina et al. (2020).

The objective function of minimizing the total transportation cost in the PDSVRP also creates specific challenges in designing algorithmic components for efficient heuristics. For instance, if a customer happens to be assigned to drones, the determination to which drone tour the customer is added in the customer insertion, the main operation in metaheuristics based on Large Neighborhood Search, is technically challenging. In the min-time objective function, one can insert a customer to the drone with current lowest completion time. However, in the min-cost objective function, the cost incurred by the customer insertion is identical among the drones, making the decision less apparent. In another example, let us take a Local Search (LS) operator called Swap, which exchanges two nodes visited by two different drones. This type of local search can be found extensively in LS-based metaheuristics such as Iterated Local Search (ILS) and Greedy Randomized Adaptive Search Procedure (GRASP). While this operator can be directly applied to the min-time variant (Murray & Chu, 2015), it is not applicable to the min-cost variant; swapping two nodes of two drones yields no cost change. In other words, without careful algorithmic design, the search performance could be adversely affected. That is not to mention the additional constraints in terms of capacity and maximum tour duration, which increase the complexity of the problem, thus extensive adaptations are required.

Our main contributions are as follows: First, we introduce a new variant of the vehicle routing problem in which trucks and drones are combined to deliver parcels to customers. Second, a mixed integer linear program is proposed to formulate mathematically the problem, from which small instances can be solved to optimality. Third, a metaheuristic based on the Ruin-and-Recreate principle is introduced to tackle medium and large instances. Our algorithm is inspired from the Slack Induction by String Removals of Christiaens & Berghe (2020) with problem-tailored components such as: sweep removal, local search, perturbation, and acceptance criteria. The algorithms are tested on randomly generated instances with up to 400 customers and the obtained results are analysed to investigate their performance and provide other insights on the design of the transportation system. Finally, we also adapt the metaheuristic to solve the

Table 1: Summary of the related works

Literature	Problem class	# drones	# carriers	Time limit	Carrier capacity	Energy function	Multi-visit drone	Objective function	Solution method
Murray & Chu (2015)	FSTSP	1	1	no	no	no	no	makespan	MILP, greedy heuristic
Murray & Chu (2015)	PDSTSP	n	1	no	no	no	no	makespan	MILP, greedy heuristic
Carlsson & Song (2018)	FSTSP	1	1	no	no	no	no	makespan	simulation
Agatz et al. (2018)	FSTSP	1	1	no	no	no	no	makespan	route-first, cluster-second heuristics based on LS and DP
Poikonen et al. (2019)	FSTSP	1	1	no	no	no	no	makespan	heuristics based branch-and-bound
Bouman et al. (2018)	FSTSP	1	1	no	no	no	no	makespan	dynamic programming
Ponza (2016)	FSTSP	1	1	no	no	no	no	makespan	simulated annealing
de Freitas & Penna (2020)	FSTSP	1	1	no	no	no	no	makespan	metaheuristic(HGVNS)
Murray & Raj (2020)	FSTSP	n	1	no	no	no	no	makespan	MILP, three-phased heuristic
Poikonen & Golden (2020b)	FSTSP	n	1	no	no	yes	yes	makespan	three-phased heuristic
Cheng et al. (2020)	DDP	n	0	no	no	yes	yes	operational costs	branch-and-cut
Savuran & Karakaya (2016)	CVP-D	n	1	no	no	no	no	# nodes visited	genetic algorithm
Poikonen & Golden (2020a)	CVP-D	1	1	no	no	no	yes	makespan	branch-and-bound, greedy heuristics
Sacramento et al. (2019)	FSTSP	n	n	yes	yes	no	no	operational costs	metaheuristic (ALNS)
Mbiadou Saleu et al. (2018)	PDSTSP	n	1	no	no	no	no	makespan	iterative two-step heuristic
Dell'Amico et al. (2020)	PDSTSP	n	1	no	no	no	no	makespan	simplified MILP, matheuristic
Ham (2018)	PDSTSP	n	n	no	no	no	no	makespan	constraint programming
Dayarian et al. (2020)	PDSTSP	1	1	no	no	no	no	# nodes visited	two-phase heuristic
Kim & Moon (2018)	PDSTSP	n	1	no	no	no	no	makespan	MILP, decomposition
Schermer et al. (2019)	PDSTSP	n	1	no	no	no	no	makespan	MILP
Wang et al. (2019)	PDSTSP/ FSTSP	n	n	no	yes	no	yes	makespan	three-step heuristic
Mathew et al. (2015)	CVP-D		1	no	no	no	no	total distance	decomposition, heuristic
Ha et al. (2018)	FSTSP	1	1	no	no	no	no	operational costs	MILP, metaheuristic (GRASP)
Karak & Abdelghany (2019)	FSTSP	n	1	no	no	no	yes	operational costs	extended C&W heuristic
Conceição (2019)	PDSTSP	n	m	no	no	no	no	operational costs	MILP
This paper	PDSTSP	n	m	yes	yes	no	no	operational costs	MILP, metaheuristic (SISSRs)

PDSTSP and assess its performance on the benchmark instances. It is shown that our algorithm outperforms existing approaches in terms of solution quality. More remarkably, it improves on 26 of the best known solutions. The remainder of this paper is structured as follows: Section 2 introduces the problem definition and a mixed integer linear programming model. The detailed description of our metaheuristic is provided in Section 3. Experimental results are reported in Section 4. And finally, we conclude our work in Section 5.

2. The min-cost parallel drone scheduling vehicle routing problem (PDSVRP)

As mentioned earlier, in the PDSVRP, a fleet of homogeneous trucks and a flock of drones are coordinated to deliver packages to a set of customers, each of whom must be served only once. A truck performs its task within a single trip starting from the depot, passing through all assigned customers, and returning to the depot. The drones operate back-and-forth trips between the depot and customers; one trip for each delivery. It should be noted that not all deliveries can be managed by drones and some of them must be handled by driver-operated trucks due to practical requirements (e.g., oversized parcels, signature requirement). Indeed, drones are only able to carry a limited payload unit within a certain maximum flying endurance. The trucks also have a limited capacity that must be respected. In addition, the total working time of each vehicle (both trucks and drones) must not exceed a certain limit. The objective is to minimize the overall transportation cost when using the mixed fleet of vehicles to meet the customers' demand while also respecting capacity and time constraints. A visual illustration of a PDSVRP solution is depicted in Figure 1. The solid lines indicate the truck routes, while the red arrows indicate the back-and-forth trips of the drones. Filled circle nodes represent customers that can only be visited by the truck, whereas the remaining nodes represent customers that can be visited by either a truck or a drone. Triangle nodes correspond to drone visits.

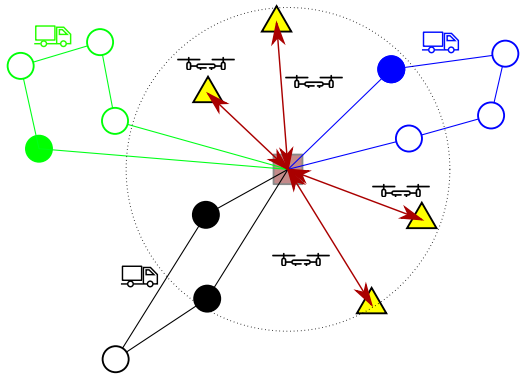


Figure 1: An example of a PDSVRP solution

2.1. Problem definition and notations

The problem can be represented on a complete directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where the node set $\mathcal{N} = \{0, 1, ..., n\}$ represents the depot (node 0) and the set of customers $\mathcal{C} = \{1, 2, ..., n\}$. As mentioned above, there might be a subset of *truck-only* customers $\mathcal{C}_\mathcal{T} \subseteq \mathcal{C}$ that can only be served by trucks. As such, let $\mathcal{C}_\mathcal{F} = \mathcal{C} \backslash \mathcal{C}_\mathcal{T}$ represent *mode-free* customers that are eligible to

both truck and drone delivery. The delivery request from customer $i \in \mathcal{C}$ is associated with a parcel weight w_i.

At the depot, there is a fleet of h trucks and a flock $\mathcal{D}$ of m drones available for delivery tasks. The distance required for a truck to traverse from node $i \in \mathcal{N}$ to node $j \in \mathcal{N}(j \neq i)$ is given by d_{ij}. Likewise, d'_{ij} denotes the flying distance from node i to node j of a drone. Given the fact that the drones only operate back-and-forth trips between the depot and a customer, let $d'_i = d'_{0i} + d'_{i0}$ particularly represent the total flying distance incurred by a drone to serve the customer $i \in \mathcal{C}_\mathcal{F}$. In the same manner, t_{ij} denotes the time spent by a truck to travel on edge $(i, j) \in \mathcal{E}$, and t'_i is the total time incurred as a drone serves a customer $i \in \mathcal{C}_\mathcal{F}$, which can include the time for operations such as: two-way flying between the depot and the customer, loading and unloading a parcel, and swapping the battery (if any) of the drone. Let C and C' be the transportation costs of the truck and drone, respectively, per unit of distance. During the operation, each drone is subject to a limited flying endurance e that does not allow it to operate further without swapping or recharging the battery. A maximum capacity Q' is also applied to restrain each drone from carrying overweight parcels. These two constraints are basically used to define whether a customer is drone-eligible in the preprocessing step. The trucks also have to respect the capacity constraint bounded by Q. Finally, there are upper bounds T and T' on the maximum working time for each truck and drone, respectively. Throughout our study, we make the following assumptions:

- A drone carries only one parcel at a time and has to return to the depot after each delivery.
- Input data, for example, driving and flight times of trucks and drones, is assumed to be deterministic.

The objective of the PDSVRP is to find a set of routes for the mixed fleet of trucks and drones minimizing the total transportation cost including traveling costs incurred by all vehicles and satisfying the following constraints:

- A drone serves one customer at a time by performing back-and-forth flights within limited endurance.
- A truck carries a total load that does not exceed its capacity.
- Each vehicle (both truck and drone) is subject to a maximum working time.

2.2. Mathematical formulation for the PDSVRP

The formulation makes use of the following variables:

- x_{ij}: binary variables equal to 1 if node $j \in \mathcal{N}$ is reached from $i \in \mathcal{N}$ $(i \neq j)$ by the trucks, 0 otherwise;
- y_i^k: binary variables equal to 1 if node $i \in \mathcal{C}_\mathcal{F}$ is visited - served by drone $k \in \mathcal{D}$, 0 otherwise;

- $u_i \in \mathbb{R}$: continuous variables specifying the weight that a truck is carrying up to node $i \in \mathcal{C}$ along the truck's path;
- $z_{ij} \in \mathbb{R}$: non-negative variables representing the cumulative duration of the truck tour at node $j \in \mathcal{N}$ after passing through node $i \in \mathcal{N}$ $(i \neq j)$.

The PDSVRP can be formulated as follows:

$$\text{minimize} \quad C \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}} d_{ij} x_{ij} + C' \sum_{k \in \mathcal{D}} \sum_{i \in \mathcal{C}_\mathcal{F}} d'_i y_i^k \tag{1}$$

$$\text{subject to} \quad \sum_{i \in \mathcal{C}} x_{0i} \leq h \tag{2}$$

$$\sum_{\substack{i \in \mathcal{N} \\ i \neq j}} x_{ij} - \sum_{\substack{i \in \mathcal{N} \\ i \neq j}} x_{ji} = 0 \quad \forall j \in \mathcal{N} \tag{3}$$

$$\sum_{\substack{i \in \mathcal{N} \\ i \neq j}} x_{ij} + \sum_{k \in \mathcal{D}} y_j^k = 1 \quad \forall j \in \mathcal{C}_\mathcal{F} \tag{4}$$

$$\sum_{\substack{i \in \mathcal{N} \\ i \neq j}} x_{ij} = 1 \quad \forall j \in \mathcal{C}_\mathcal{T} \tag{5}$$

$$u_i - u_j + Q x_{ij} \leq Q - w_j \quad \forall i \in \mathcal{C}, j \in \{\mathcal{C} : j \neq i\} \tag{6}$$

$$\sum_{j \in \mathcal{C}_\mathcal{F}} y_j^k t'_j \leq T' \quad \forall k \in \mathcal{D} \tag{7}$$

$$\sum_{\substack{l \in \mathcal{N} \\ l \neq i}} z_{li} + \sum_{\substack{j \in \mathcal{N} \\ j \neq i}} t_{ij} x_{ij} = \sum_{\substack{j \in \mathcal{N} \\ j \neq i}} z_{ij} \quad \forall i \in \mathcal{C} \tag{8}$$

$$z_{0i} = t_{0i} x_{0i} \quad \forall i \in \mathcal{C} \tag{9}$$

$$z_{i0} \leq T x_{i0} \quad \forall i \in \mathcal{C} \tag{10}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \in \mathcal{N}, i \neq j \tag{11}$$

$$y_i^k \in \{0, 1\} \quad \forall i \in \mathcal{C}_\mathcal{F}, k \in \mathcal{D} \tag{12}$$

$$0 \leq u_i \leq Q \quad \forall i \in \mathcal{C} \tag{13}$$

$$0 \leq z_{ij} \leq T \quad \forall i, j \in \mathcal{N}, i \neq j \tag{14}$$

The objective function (1) seeks to minimize the operational cost. Constraint (2) ensures the number of trucks departing from the depot is no greater than a given value. Constraints (3) are the equations of flow conservation, which ensure the continuity of a truck tour by specifying that the truck must leave a customer node after visiting for a delivery. Constraints (4) imply each mode-free customer is visited exactly once, either by a truck or by a drone. Constraints (5) guarantee the truck-only customers must be visited by the trucks. Constraints (6) impose the Miller-Tucker-Zemlin subtour elimination and capacity constraints (Kara, Laporte & Bektas, 2004). The total operating time of each drone is restricted in constraints (7), while constraints (8)-(10), adapted from the formulation of Kara (2011), together impose and make sure that the time traveled to the depot by each truck does not exceed the predetermined value T. Finally,

constraints (11)-(14) specify the decision variable domains.

3. A Slack Induction by String and Sweep Removals for the PDSVRP

Given the complexity of the problem, using MILP solvers can only solve small-sized problem instances within an acceptable computational time. The fact that the volume of shipping parcels keeps expanding encourages us to design a method to handle large-scale instances. As a contribution of this paper, we propose a simple, yet powerful heuristic method named Slack Induction by String and Sweep Removals (SISSRs), adapted from Slack Induction by Sweep Removals (SISRs) recently introduced to efficiently solve the VRPs in Christiaens & Berghe (2020).

In what follows, we first briefly describe the basic rationale and general implementation of the SISSRs algorithm (subsection 3.1), and then explain the detailed SISSRs tailored for the PDSVRP (subsections 3.2-3.4). Finally, we discuss modifications in the algorithm to address the min-time variant (subsection 3.5).

3.1. The general structure of the Ruin&Recreate with Slack Induction

Our method uses the same solution strategy as the SISRs introduced in Christiaens & Berghe (2020). In brief, it applies the Slack Induction concept into the Ruin-and-Recreate method. Starting from an initial solution, the method iteratively performs the two following phases: *ruin ($\mathcal{R}^-$) phase* where a part of the current solution is deconstructed by a set of ruin methods, and *recreate ($\mathcal{R}^+$) phase* where the incomplete solution is fully rebuilt by a set of construction methods.

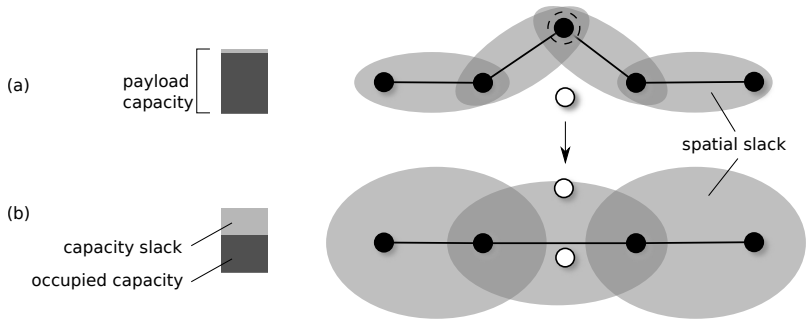


Figure 2: An illustration of capacity slack and spatial slack in the context of truck tour

Above all, SISRs rather focuses on the $\mathcal{R}^-$ method in a new perspective. It embraces the concepts of *capacity slack* and *spatial slack* which refer to the remaining resources of a tour in terms of load and space (time, distance, etc.) that are available to accommodate new upcoming customers. While it is easy to link the capacity slack to the unoccupied space in the truck, the spatial slack can be expressed by the vehicle's *reachable area*. This area appears as an elliptical region around each edge whose end points are the focus points for its corresponding ellipse. This implies that only customers inside this area can be reached and served by the truck. As visualized in Figure 2(a), the reachable area of the tour is shaded gray, and the customers selected for removal are outlined by dashed lines. The unvisited white node cannot be inserted into the truck tour as it is outside the reachable area and the payload capacity is fully occupied.

In other words, the insufficient capacity slack and spatial slack disallow the insertion of the white node into the tour. Naturally, the reachable area expands each time there are customers removed from the tour and shrinks as the tour accommodates new customers. As shown in Figure 2(b), the slack induction obtained by the removal of the white node expands the spatial and capacity slacks of the tour. It then enables the white node to be reached by the truck.

Observations on multiple ruined states of solutions show that insufficient slack induced during $\mathcal{R}^-$ phase is the major cause for the potential failures later in $\mathcal{R}^+$ phase. Accordingly, it is important that an effective $\mathcal{R}^-$ method should introduce enough "slack", both capacity and spatial slacks, into the solution so that the possibilities for improving the solution are greatly enhanced (Christiaens & Berghe, 2020). The general structure of SISRs guided by simulated annealing is shown in Algorithm 1.

Algorithm 1 SISRs for VRP

Input: VRP problem instance ins
Output: best solution s^{best}
function SISRs(ins)
  $s_0 \leftarrow$ `SolutionConstruction`
  $s^* \leftarrow s_0$, $s^{best} \leftarrow s_0$
  repeat
    $s =$ `Ruin&Recreate`(s^*)
    if $cost(s) < cost(s^*) - \Delta \times \ln(U(0,1))$ **then**
      $s^* \leftarrow s$
      if $cost(s^*) < cost(s^{best})$ **then**
        $s^{best} \leftarrow s^*$
    $\Delta \leftarrow \Delta \times \epsilon$
  until Terminal condition met

We exploit the same rationale described above to develop the proposed SISSRs algorithm for the PDSVRP. Compared to the original one, the versatility of the SISSRs in this paper is pushed further to fit the context of the PDSVRP at: (i) the design of its basic building blocks of the $\mathcal{R}^-$ and $\mathcal{R}^+$ operators to consider both trucks and drones, (ii) additional improvements in terms of local search moves, and the incorporation of different acceptance criteria and perturbation mechanisms to avoid the premature convergence.

Algorithm 2 SISSRs for the PDSVRP

Input: PDSVRP problem instance ins
function SISSRs(ins)
  $s_0 \leftarrow$ `SolutionConstruction`
  $s^* \leftarrow s_0$, $s^{best} \leftarrow s_0$
  repeat
    $s \leftarrow$ `Ruin&Recreate`(s^*)
    if $cost(s) < cost(s^*) \times (1 + \Delta)$ **then**
      $s^* \leftarrow$ `LocalSearch`(s)
      if $cost(s^*) < cost(s^{best})$ **then**
        $s^{best} \leftarrow s^*$
    if $Iter_{imp}$ iterations without improving s^{best} **then**
      `Perturbate`(s^*)
    $\Delta \leftarrow \Delta \times \epsilon$
  until Terminal condition met
return s^{best}

The general SISSRs algorithm, as shown in Algorithm 2, starts with building the first solution s_0 via the `SolutionConstruction` operator (line 2), then initializing the current solution s^* and the best solution s^{best} (line 3). The initial solution is passed to the iterative process for improvements (lines 4-12). During each iteration, the neighbor solution s is generated from the current solution s^* via `Ruin&Recreate` procedure (line 5). Only those neighbor solutions which satisfy the *Threshold Acceptance* condition (line 6) are selected into further improvement via the `LocalSearch` procedure (line 7). Here, Δ is a coefficient initially set to a given value Δ_{ini} and its value is adaptively changed during the search. The resulting solution is saved as the current solution s^* for the next iterations. It is saved to s^{best} if it improves the current best solution (lines 8-9). There is also a counter for the number of iterations without improving s^{best}. If the counter reaches $Iter_{imp}$, a perturbation phase is performed on s^{best} via `Perturbate`(s^*) procedure (lines 10-11). Finally, the value of Δ is decreased via cooling rate ϵ before starting the next iteration (line 12) if the stopping condition is not met. The main components of the algorithm are described in what follows.

3.2. Slack induction via Sweep and String Removals

As the PDSVRP considers both the trucks and drones, it is expected to stimulate the slack induction not only for truck tours or drone trips separately, but also for the combination of both vehicle types to enhance opportunities for promising vehicle assignment changes. We now propose two new premises to handle slack inductions adapted to the PDSVRP. These premises working alongside three proposed in Christiaens & Berghe (2020), making a total of five, serve as the basis for our $\mathcal{R}^-$ method. Based on these premises, we introduce a *Sweep removal* to handle slack induction for drone tours (subsection 3.2.2). An extension of the adjacent string removal in Christiaens & Berghe (2020) is also presented to create slack induction for truck tours in subsection 3.2.4.

3.2.1. The proposed premises

We now discuss several characteristics of PDSVRP solutions from which we design the proposed $\mathcal{R}^-$ method. We characterize a PDSVRP solution s by a set of absent customers $\mathcal{A}$ and a set of vehicle tours $\mathcal{T}$. Depending on the state of solution s (i.e., either partial or complete), set $\mathcal{A}$ and $\mathcal{T}$ will vary accordingly. For example, set $\mathcal{A}$ initially contains all customers and gradually becomes empty as solution s transforms from partial to complete solution in the process to assign customers into available vehicle tours. From now on, we refer to *truck nodes* (and *drone nodes*) as the customers who are served by the trucks (and drones respectively) in a PDSVRP solution. Formally, $\mathcal{C}_s^{truck}$ and $\mathcal{C}_s^{drone}$ denote the sets of customers who are visited by the trucks and the drones, respectively.

First, a drone tour can be expressed as a set of drone-eligible nodes, indicating the set of delivery trips of the drone. As the drone performs back-and-forth trips to reload a parcel for each customer, a drone tour is only subject to a maximum endurance, which can be converted to a limited travel distance. The requirement can be represented by the drone's *reachable area* given the amount of working time left. This area of a drone appears as a circular region around the depot, which is also the central point. Only drone-eligible customers ($c \in \mathcal{C}_{\mathcal{F}}$) that are inside

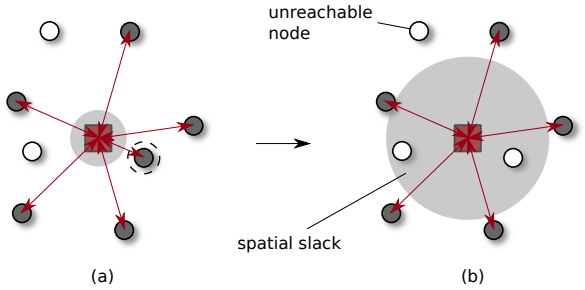


Figure 3: An illustration of the spatial slack in the context of drone tours

this area can be reached by the drones. Illustrated as the shaded gray area in Figure 3, the reachable area of the drone expands each time a customer is removed from a tour and shrinks as its tour accommodates new customers. From left to right, (a) given a set of trips from the depot to the gray nodes already visited by a drone, the trip to the node outlined by dashed line is chosen for removal; (b) the reachable area expands as the selected node is removed, which then enables the white node on the left to be reachable by the drone.

Second, the described moving pattern allows a drone to make its delivery trips to customers, independently in any order, without affecting the overall cost. In addition, the spatial distribution of visited nodes in a drone tour does not necessarily follow a distance-adjacent pattern like in the truck tour, but rather spreads out in random directions.

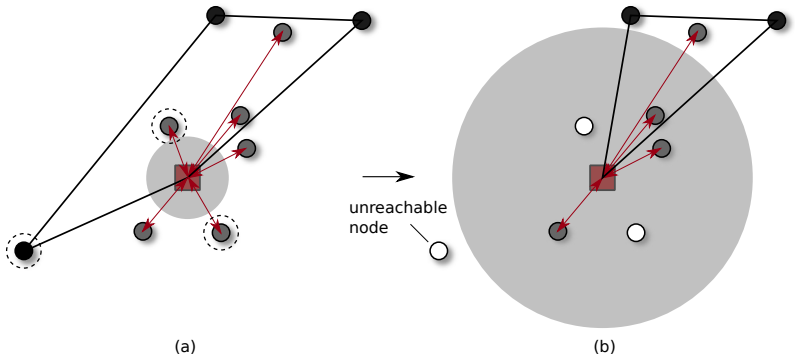


Figure 4: An example of ruin method that introduces insufficient slack to drone tours

Exploiting the characteristics of PDSVRP solutions described above, we propose the following two new premises:

- **New premise 1** (*Remove a "sufficient" number of drone customers*). Removing a small number of customers may fail to introduce enough spatial slack for the drones to serve customers far from the depot. Although it could be visually clear by which vehicles the removed customers should be served, it may be impossible if the vehicles gained insufficient spatial slack.

- **New premise 2** (*Sweep a cluster of drone customers*). Removing drone nodes simply based on distant adjacency would fail to make contact with further nodes. Conceptually, sweeping across the plane over the node set could create significant chances for such nodes to be selected for removal. In addition, it tends to incur less extra costs to insert into truck

tours a cluster of nodes, of which angular coordinates are close to each other. Therefore, one can expect to reassign nodes, which are misplaced in drone tours, into truck tours later in $\mathcal{R}^+$ phase.

On the one hand, Figure 4 shows an example of mentioned drawbacks that the two proposed premises aim to address. In Figure 4(a), three white nodes are selected for removal: two drone nodes and one truck node. In Figure 4(b), the drone's reachable area, rendered as the shaded circular area, expands after the removal of the two drone nodes. However, the additional spatial slack appears insufficient for the drone to handle the bottom left node as it stays outside the shaded area. As a result, the truck is still the only option to visit this node, which unfavorably incurs a considerable amount of cost savings.

On the other hand, Figure 5 illustrates an example of an effective $\mathcal{R}^-$ method. A cluster of three drone nodes, which are outlined, is swept in Figure 5(a), resulting in the ruined state of the solution in Figure 5(b). Notice in Figure 5(b) that the spatial slack of the drone is greatly introduced as it can now cover the entire area, including the remote customer at the bottom left. Ideally, the ruined state obtained by the $\mathcal{R}^-$ method is recreated by $\mathcal{R}^+$ in the solution illustrated in Figure 5(c), which appears to be significantly improved.

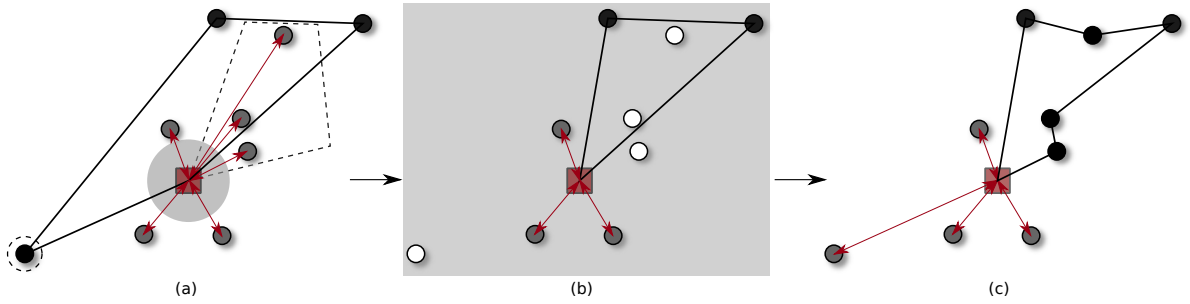


Figure 5: An example shows the effectiveness of the sweep removal

3.2.2. Sweep removal

In the first removal operator, we focus on the drone customers at the beginning of the $\mathcal{R}^-$ method. The two described premises support the implementation of such an operator, named *Sweep removal*, which aims to induce sufficient spatial slack to drone tours via removing a set of drone customers. Its idea is straightforward: removing a cluster of drone customers that are adjacent in the polar coordinate.

Algorithm 3 Sweep removal operator

Input: Solution $s = \{\mathcal{T}, \mathcal{A}\}$, where $|\mathcal{C}_s^{drone}| > 0$

function SweepRemoval(s)
  $nbSwept \leftarrow U(1, \delta|\mathcal{C}_s^{drone}|)$
  $\theta_{seed} \leftarrow U(0°, 360°)$
  Sort($\mathcal{C}_s^{drone}, \theta_{seed}$)
  for $c \in \mathcal{C}_s^{drone}$ **and** $|\mathcal{A}| < nbSwept$ **do**
    Remove c from s
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{c\}$
return Ruined solution s

Let s be a solution to be ruined. Given the set of drone customers $|\mathcal{C}_s^{drone}| > 0$, the procedure works as follows (lines 2-7 in Algorithm 3). In the beginning, the number of drone nodes to be removed, $nbSwept$, is randomly selected in the range of $(1, \delta \left|\mathcal{C}_s^{drone}\right|)$ where δ is a pre-defined parameter indicating the swept rate $(0 < \delta < 1)$ (line 2). In this removal, we assume a standard angular coordinate, in which the depot is taken as the pole and the polar axis directs to the right of the pole. A seed angle θ_{seed}, directed counter-clockwise from the polar axis, is randomly chosen from the continuous uniform distribution $U(0°, 360°)$ (line 3). The chosen seed angle draws a directed line l from the depot. The set of drone customers $\mathcal{C}_s^{drone}$ is then sorted according to how close each node is to line l, measured by the polar angle φ_c between line l and the straight line to node c from the pole (line 4). Subsequently, the main loop (lines 5-7) iterates over the $\mathcal{C}_s^{drone}$ set, removes from the current drone tours and adds to set $\mathcal{A}$ the closest node, i.e., the one with the smallest φ_c. The operator sweeps through $nbSwept$ nodes before termination. Figure 6 illustrates an example of the sweep removal.

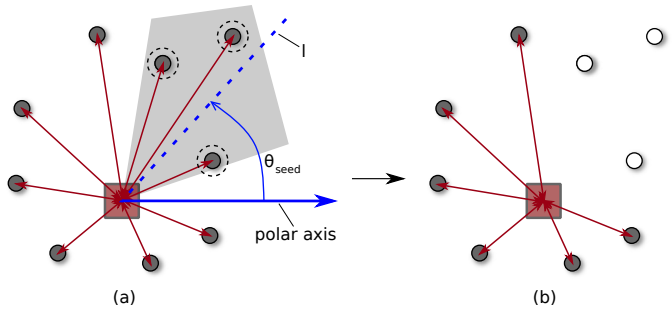


Figure 6: An example of Sweep Removal

3.2.3. Random drone-customer removal

This random removal operator simply selects q drone nodes at random and removes them from the solution. Parameter q is randomly selected in the range of $(1, \delta \left|\mathcal{C}_s^{drone}\right|)$ where δ equals the pre-defined swept rate used in the described sweep removal. The objective of this removal is to provide diversification to the search algorithm via diverging the structure of drone tours. As a part of the early attempt of $\mathcal{R}^-$ method, either random removal or sweep removal is randomly picked and applied to introduce slack to drone tours.

3.2.4. Adjacent string removal for the PDSVRP

The next progress of $\mathcal{R}^-$ method involves introducing slack to the truck tours via the Adjacent string removal for the PDSVRP, which is an extension to the original one proposed in Christiaens & Berghe (2020) for the CVRP. The extension lies in the fact that it considers removing both truck and drone nodes close to a seed node. This enables additional slack induction to drone tours and opens up opportunities for rearranging customer assignments within the area near the seed node. Algorithm 4 describes the procedural steps of the proposed removal operator.

Let s be the solution to be ruined. The numbers of removed customers and removed strings are controlled by two pre-determined parameters: the average number of removed customers $\bar{c}$

Algorithm 4 Adjacent string removal operator for the PDSVRP

Input: Solution $s = \{\mathcal{T}, \mathcal{A}\}$

1: **function** StringRemoval(s)
2: Determine the number of strings to be removed k_s
3: $c_s^{seed} \leftarrow$ select a random customer s
4: $R \leftarrow \emptyset$
5: **for** $c \in adj(c_s^{seed})$ **and** $|R| < k_s$ **do**
6: **if** $c \in \mathcal{C}_s^{truck}$ **and** $c \notin \mathcal{A}$ **and** $\tau_c \notin R$ **then** ▷ remove strings of truck nodes close to the seed customer
7: Determine the length of string l to be removed in tour τ_c
8: $\mathcal{A} \leftarrow \mathcal{A} \cup$ `RemovedString`(τ_c, l, c)
9: $R \leftarrow \{\tau_c\}$
10: **else if** $c \in \mathcal{C}_s^{drone}$ **and** $c \notin \mathcal{A}$ **then** ▷ remove any drone nodes close to the seed customer
11: Remove c from s
12: $\mathcal{A} \leftarrow \mathcal{A} \cup \{c\}$
13: **return** Ruined solution s

and the maximum length of each removed string L_{max}. The procedure begins with the calculation of the number k_s of strings to be removed out of solution s, which is derived from equations in Christiaens & Berghe (2020). To ensure that the capacity slack and spatial slack are spread across multiple tours, it is assumed that each tour can only be ruined at most once, and removing k_s strings actually implies ruining k_s tours. Next, a random seed customer c_s^{seed} is selected, from which the string removal operator is initiated (line 3). The set of ruined truck tours R is initially empty (line 4). The adjacent list containing all customers ordered by increasing distance from c_s^{seed}, called $adj(c_s^{seed})$, is iterated until set R contains k_s ruined tours, implying k_s strings have been removed (lines 5-12). In each iteration, for each customer $c \in adj(c_s^{seed})$ in the PDSVRP, two cases are considered:

1. Customer c belongs to set of the truck customers $\mathcal{C}_s^{truck}$ ($c \in \mathcal{C}_s^{truck}$), indicating c is served by the trucks. If c has not been removed ($c \notin A$) and its corresponding truck tour τ_c has not been ruined ($\tau_c \notin R$), a string of length l is removed from the tour τ_c (lines 6-9). The string is removed via the "string" or "split string" procedure based on c and l. In either case, l customers are selected and removed from τ_c before adding them to set A (line 8), and the tour τ_c is marked as ruined and being added to set R (line 9).
2. Customer c belongs to a set of the drone customers $\mathcal{C}_s^{drone}$ ($c \in \mathcal{C}_s^{drone}$), indicating c is served by the drones. If c has not been removed ($c \notin A$), it is removed from the solution s (lines 10-12). The global objective here is to create possibilities for customer exchanges between trucks and drones within the area around customer c_s^{seed}.

3.3. Recreate method - greedy insertion with blinks for the PDSVRP

In $\mathcal{R}^+$ phase, a recreate method rebuilds a ruined solution s by inserting consecutively ejected customers in set $\mathcal{A}$ into s until all customers are served. In the context of PDSVRP, each insertion requires the $\mathcal{R}^+$ method to examine all possible positions in both truck and drone tours for the cheapest option. Before going further, it is important to clarify the definition of "*position*" in truck and drone tours. As mentioned above, a drone can make its delivery trips to customers independently in any order without affecting the overall cost. Therefore, "position"

in the context of truck literally refers to a position in a tour, whereas in the context of drone "position" it simply encapsulates to which drone the customer is assigned without considering the relative position within the tour. The proposed $\mathcal{R}^+$ for the PDSVRP works as follows (see Algorithm 5).

Given the solution $s = \{\mathcal{T}, \mathcal{A}\}$, in line 2, the process first starts by sorting set $\mathcal{A}$ according to one of the criteria presented in Table 2. A weighted random selection is called to decide by which criterion set $\mathcal{A}$ would be sorted. The weights are initially set to $\{w_1, w_2, w_3, w_4, w_5\}$ for each sorting criterion. Then, following the sorted set, each customer $c \in \mathcal{A}$ is inserted into solution s at the *best* position, called pos^{best} (lines 3-15). The value of pos^{best} is initially set to `null`, indicating no positions in trucks and drones for c are selected (line 4).

Table 2: Five sorting criteria

Criteria	Description	Weight
random	Sorting customers in an *arbitrary* order	w_1
near-dp	Sorting customers according to the rule "closest to the depot first"	w_2
far-dp	Sorting customers according to the rule "farthest to the depot first"	w_3
near-tr	Sorting customers according to the rule "closest to the nearest truck tour first"	w_4
far-tr	Sorting customers according to the rule "farthest to the nearest truck tour first"	w_5

Two insertion possibilities are considered for each customer c, to *truck tours* or to *drone tours*:

1. (**To truck tours**). All feasible positions pos_τ for the insertion into truck tours $\tau \in \mathcal{T}$ are iterated over (lines 5-7). There is a probability γ representing the blink rate. Each position is evaluated with a probability of $1 - \gamma$, otherwise position pos_τ is skipped as if the algorithm blinks (line 6). If a position pos_τ is found for which the cost of inserting c is cheaper than the current best position pos^{best}, the position pos_τ becomes the new best position (lines 7).
2. (**To drone tours**). Positioning customer c into drone tours is then considered if c is a drone-eligible customer and there is sufficient time slack among the drones to serve c, tested by function FitDrone(s, c) (line 8). Again, if the algorithm does not blink ($U(0,1) \leq 1 - \gamma$) and the cost to serve c by the drones is cheaper than the current best cost, it is the best option if c is served by the drones. Position pos^{best} is updated to the drone with the largest spatial slack (lines 9-10).

Note that during each customer insertion, we must take into account the constraints related to the truck's capacity and the working time of both vehicles. The feasibility of an insertion can be easily determined in constant time by simply comparing the extra weight and the additional travel time caused by the insertion with the remaining truck's capacity and the vehicle's slack times, respectively.

Algorithm 5 SISSR's Recreate Method

Input: Ruined-solution $s = \{\mathcal{T}, \mathcal{A}\}$

1: **function** RECREATE(s)
2: SORT($\mathcal{A}$)
3: **for** $c \in \mathcal{A}$ **do**
4: $pos^{best} \leftarrow$ `null`
5: **for** each feasible position p_τ in truck tour $\tau \in \mathcal{T}$ **do**
6: **if** $cost(p_\tau) < cost(pos^{best})$ **and** $U(0,1) > 1-\gamma$ **and** $pos^{best} \neq$ `null` **then**
7: $pos^{best} \leftarrow p_\tau$
8: **if** c is drone-eligible **and** FITDRONE(s, c) **then**
9: **if** $cost'(c) < cost(pos^{best})$ **and** $U(0,1) > 1-\gamma$ **and** $pos^{best} \neq$ `null` **then**
10: $pos^{best} \leftarrow$ the drone with the largest spatial slack
11: **if** $pos^{best} =$ `null` **then**
12: $\mathcal{T} \leftarrow \mathcal{T} \cup \{\text{new tour } \tau\}$
13: $pos^{best} \leftarrow$ position in τ
14: Insert c to its best position at pos^{best}
15: $\mathcal{A} \leftarrow \mathcal{A} \setminus \{c\}$
16: **return** Recreated-solution s

The blink concept from Christiaens & Berghe (2020) is applied with the aim to add randomization to the search. However, it should be noted that the algorithm only blinks if $pos^{best} \neq$ `null` to avoid it from running into extreme cases where it is unable to insert c into solution s as all feasible positions are blinked. In the case that no positions are found in existing tours, a new empty truck tour is created to serve c (lines 11-13). Finally, c is inserted into the best position and removed from $\mathcal{A}$ (lines 14-15).

3.4. Other components

3.4.1. Initial solution

The procedure of constructing the initial solution s_0 makes use of the $\mathcal{R}^+$ procedure followed by local search. We consider set $\mathcal{A}$ of absent customers to initially contain all customers, indicating no customers have been served by any tours. Following Algorithm 5, the operator repeatedly assigns each customer into tours at its best possible position until all customers are served. Local search procedure is then called to improve the overall quality the solution. The resulting solution is then used as the input to feed the main loop of the improvement stage of Algorithm 2 (lines 3-11).

3.4.2. Local search

A PDSVRP solution is separately comprised of a set of drone tours and truck tours. The drone's back-and-forth moving scheme and the min-cost objective function make intra-route and inter-route neighborhoods among the drone tours irrelevant to obtain any cost reductions. On the other hand, the truck tours can be explicitly improved via existing VRP local search operators. Therefore, we apply four well-known neighborhoods, namely RELOCATE, SWAP, 2-OPT and 2-OPT*, as algorithmic components to improve the quality of truck tours.

In addition, the early flaws in customer-vehicle assignments can further be improved via exchanging nodes between a truck tour and a drone tour. To explore better vehicle assignments,

we use four new inter-route neighborhood structures, namely Swap', Swap*, Shift-t, and Shift-d:

- Swap'. The operator seeks possible cost reductions by swapping the current positions of a pair of one drone customer and one truck customer who must be drone-eligible.
- Swap*. The neighborhood consists in taking two customers: one truck customer v_T and one drone customer v_D from truck route r_T and drone tour r_D, respectively. v_T is then assigned to r_D while v_D is inserted to a position of r_T that incurs the cheapest cost.
- Shift-t. The neighborhood of the operator contains all solutions, in which one truck customer is removed from its truck tour and reinserted into a drone tour.
- Shift-d. In contrary to Shift-t, the neighborhood contains all solutions, in which one drone customer is removed from its tour and reinserted into a truck tour.

To efficiently search and reduce the computation time, the pruning technique in Toth & Vigo (2003) and Mester & Bräysy (2007) is adopted. Rather than trying to iteratively evaluate every move, only moves involving a fixed number of nearest neighbors $n_{nearest}$ for each node are considered.

Each iteration of the local search evaluates every feasible move which satisfies the problem constraints. We adopt the first-improvement fashion, i.e., immediately perform the first detected moves that lead to better solutions. The process starts again from that improved solution and repeats until the solution can no longer be improved.

3.4.3. Perturbation mechanisms

We also apply a perturbation procedure in the SISSRs to overcome the problem of premature convergence in local optima. The function `Perturbation` (line 11, Algorithm 2) is triggered after a certain number of iterations $Iter_{imp}$ without any improvements. When the function `Perturbation` is called, the number of random swaps p is randomly selected in range $[p_{min}, p_{max}]$. Then an iterative process is started by randomly picking a pair of distinct customer nodes (i, j) to perform swapping. There are three swapping possibilities:

- Truck-truck swap move: i and j are both truck nodes;
- Truck-drone swap move: i is truck node and j is drone node;
- Drone-drone swap move: i and j are both drone nodes.

We note that only feasible swap moves, which respect the capacity and route duration constraints, are allowed. The process is repeated until p swaps are executed or no feasible swap is found after a certain number of attempts.

3.5. SISSRs adaptation for the min-time PDSVRP

As the *min-time* PDSVRP variant seeks to minimize the makespan (i.e., the maximum completion time), three considerable modifications in the algorithm are required to adapt the

method to the problem at hand. *The first modification* involves the way to compute the objective value. Considering the insertion of customer c into solution s during $\mathcal{R}^+$ procedure, the best position pos^{best} is determined by:

$$pos^{best} = \text{argmin}\,(t'_\tau - t_s) \tag{15}$$

where t'_τ is the completion time of a tour τ (both trucks and drones) after accommodating customer c, and t_s is the current makespan of s before the insertion of c.

The second modification revolves around the local search operators. Unlike the min-cost variant in which the focus is on reducing the overall cost, the min-time variant pays attention on reducing the cost of the longer tours (i.e., higher completion time). Taking this characteristic into account, all local search inter-route operators described in subsection 3.4.2 are slightly modified in the way they evaluate a move. In particular, suppose $t = \max(t_i, t_j)$ is the completion time of the longer tour, either tour i or j. A feasible move is considered improving if it leads to the neighbor in which completion time of resultant tours t'_i and t'_j satisfy: $(t'_i \leq t \wedge t'_j < t) \vee (t'_i < t \wedge t'_j \leq t)$. In other words, the resultant tours must have a shorter completion time compared to the current completion time of the longer one.

Finally, *the third modification* focuses on the acceptance criteria to accept a solution. As in the min-cost variant, the acceptance criteria tailored for the min-time variant apply Threshold Acceptance (TA) which allows "acceptable" up-hill neighborhoods. However, instead of taking the makespan as the only evaluation, the general quality of a solution, which is measured by *the total of completion time of all current tours*, is also considered. The idea is to consolidate the qualities of other tours besides the longest one, which might be beneficial later in the long term to lead to promising neighbors. Therefore, we add the following acceptance criterion: a solution s is accepted if its total completion time is less than that of the incumbent solution s^* ($\Gamma_s < \Gamma_{s^*}$). Note that this criterion only applies to solution s with makespan equal to s^* ($t_s = t_{s^*}$). This modification eventually results in a slight change in Algorithm 2, specifically, in lines 5-7 as follows:

Algorithm 6 SISSRs: min-time adaptation

```
1: function SISSRs-MINTIME(ins)
2:     ...
5:     if t_s < t_s* × (1 + Δ) or
       t_s = t_s* and Γ_s < Γ_s* then                ▷ additional acceptance rule
6:         s* ← LocalSearch(s)
7:         if t_s* < t_s^best or
           t_s* = t_s^best and Γ_s* < Γ_s^best then  ▷ additional acceptance rule
8:             s^best ← s*
9:     ...
```

4. Experimental results

The goals of our computational experiments are as follows. The first experiment investigates the impact of key algorithmic components on the performance. The second experiment aims

to validate the MILP formulation and to evaluate the ability of SISSRs to reach optimality over small instances. In the third experiment, we examine the capability of SISSRs in handling large-scale benchmark instances. The obtained results would be used as a reference to assess the performance of future approaches proposed to solve the PDSVRP. The fourth experiment attempts to compare the adapted SISSRs in solving the min-time PDSTSP with the state-of-the-art algorithms. Finally, we carry out a sensitivity study to analyze the effects of different problem parameters.

The parameters of our metaheuristic are chosen empirically. We have tested various parameter combinations, and the one shown in Table 3 gives the best performance in terms of both solution quality and computational time for our algorithm.

Table 3: Parameter setting of SISSRs

Notation	**Description**	**Value**
$\bar{c}$	The average number of removed customers	$0.15n$
L^{max}	The maximum cardinality of the removed strings	$0.15n$
α	The split rate	0.5
β	The split depth	0.95
δ	The swept rate	0.3
w_1, w_2, w_3, w_4, w_5	Distributed weight for biased selecting *random*, *near-dp*, *far-dp*, *near-tr*, and *far-tr*, respectively	{5,1,1,2,2}
γ	The blink rate	0.1
Δ_{ini}	The initial threshold of deterioration	0.1
ϵ	The cooling rate	0.999975
$Iter_{imp}$	The number of iterations without improvement to perform perturbation	10k
$Iter_{max}$	The total number of iterations	100k
p_{min}	The minimum number of swaps performed during perturbation	3
p_{max}	The maximum number of swaps performed during perturbation	$0.1n$
$n_{nearest}$	The size of restricted neighborhoods of each node considered in local search operators	20

Note: n is the number of customers.

Our algorithms are implemented in C++ compiled with GCC 9.3.0. We use CPLEX 12.10 to solve the MILPs. All tests are run on a desktop operating Xubuntu 20.04, with AMD Ryzen 3700X @4.0GHz CPU, 16GB RAM. The detailed benchmark instances and results of our experiments can be found at http://orlab.com.vn/home/download.

4.1. Instance set

Given no access to real-life instances and the fact that there are no benchmark instances that consider the similar model and objective function, randomly generated instances are chosen to test the proposed methods. Thus, based on various statistical data shared by logistics service providers and other reliable information sources, we will develop our own instance generator for this study. The objective is to capture as many operational scenarios as possible, which could occur in practice. Basically, we use data found in the literature or provided by companies who have introduced prototype models to set the values for the parameters of the instance generator.

First, the problem size represented by the number of customers n is successively picked from the set {30, 50} for small size, {100, 200} for medium size, and {400} for large size. The nodes

representing the customers are randomly generated in a square of $d \times d$ dimension. The value of grid size d is an important factor since it determines the size of the drone-eligible customer set, hence affecting potential vehicle assignment decisions. If the grid size is too large, the customers tend to distribute far from the depot and the problem is closer to the VRP. On the other hand, if d is too small, more customers will be served by the drones as they are in general much cheaper. As a consequence, the problem can be closer to the PMS. Given the presumed battery capacity of the drones, we vary the value of d between 20 to 25 km for benchmarking purposes. Three different settings for the depot position are considered: (i) Central (c) - in the center of the grid at point $(\frac{d}{2}, \frac{d}{2})$; (ii) Edge (e) - on the edges of the grid at points $(\frac{d}{2}, 0)$, $(\frac{d}{2}, d)$, $(0, \frac{d}{2})$, and $(d, \frac{d}{2})$; (iii) Random (r) - at a random point in the grid. Also, as proposed in Solomon (1987); Uchoa et al. (2017) for the VRP instances, customers are distributed in the grid following three fashions: Random (r), Clustered (c), or Random-Clusterd (rc).

From the data collected by e-commerce firms and logistics providers, the customer demands are created. Amazon stated that 86% of its deliveries are associated with items under 2.27 kg (Allain, 2013). Moreover, UPS disclosed that the maximum load for a package transported in the truck is 68 kg (UPS, 2020). Following the above limits, the customer demands are randomly generated according to a uniform distribution. Let p be a random number associated with customer i in the instance $(0 < p < 1)$. Then the demand (in kilograms) of customer i is given by Equation (16).

$$w_i = \begin{cases} q \in U(0, 2.27) & \text{if } p < 0.86 \\ q \in U(2.27, 68) & \text{otherwise} \end{cases} \tag{16}$$

Based on drone specifications gathered from DHL's and Amazon's website, we assume a drone can carry up to 2.27 kg and travel with an average speed of 40 km/h (DHL, 2016; Allain, 2013; Scott & Scott, 2019). In line with the same information source, we also assume that the drones can travel a maximum distance of 24 km. Thus, they can serve customers of at most 12 km from the distribution center. This assumption limits the battery endurance of the drones to a value equivalent to 0.6 hour of flying.

Next, we assume the truck fleet is a set of standard homogeneous delivery vans with a capacity limit of 1300 kg. According to the survey of Sanchez-Diaz, Palacios-Argüello, Levandi, Mardberg & Basso (2020) on medium-duty trucks delivering in urban areas, the average speed of a vehicle during a day was between 24 and 34 km/h. Hence, we assume that the trucks operate with a constant average speed of 30 km/h. It is also assumed that there are always enough trucks available at the depot. Manhattan distances are used for the truck and Euclidean distances for the drones. For simplicity, we suppose that the time spent on loading, unloading a parcel, and swapping the battery of the drone is negligible compared to the flying time, and thus can be removed from the service time of a customer.

The maximum route duration of each vehicle type (T and T') is scaled such that the solution requires a relatively large number of vehicle routes. In general, T and T' are set to the same value varying from 1.5 h to 3 h. However, some instances may contain customers that are too far from the depot that no vehicle can service, possibly leading to infeasible instances. To ensure

every customer is servable in such cases, the maximum truck route duration T is set to $2 \times t_{0i^*}$, where i^* is the farthest customer. Likewise, the number of drones in instances is selected so that it is neither too large nor too small, balancing the number of customers served by the drones and the trucks in the solution. Hence, we set the number of drones to 4 for the small instances, and to 5 for the medium and large instances. These might create more complex vehicle-assignment decisions, possibly leading to more challenging instances.

Followed by the research of Deutsche Bank (Kim, 2016; Lauryn, 2019), we would set one kilometer of drone delivery cost at \$0.03 while normal trucking price would be \$1.25 for the same mileage. Several cost unit settings will be considered for system analysis later in Section 4.4.

For a given grid setting with the number of customers (n) and the fixed type of customer distribution (cd), three different grids with known customer location are generated. For each obtained grid, the depot is then added following three settings "c", "e", and "r" as mentioned above. As such, for a given number of customers, the total number of generated instances is $3 \times 3 \times 3 = 27$. The instances are named as *n-cd-id-dp*, where *id* is the instance number from 0 to 2 and *dp* is the depot position. For example, 30-r-0-c represents the first instance of the class of grid with 30 customers whose locations are randomly distributed and the depot being positioned in the center. Finally, Table 4 summarizes the problem parameters and their values.

Table 4: The problem-parameter setting for the PDSVRP instance generator

Parameter	**Notation**	**Value**	**Reference**
Truck speed	v	30 km/h	Sanchez-Diaz et al. (2020)
Drone speed	v'	40 km/h	DHL (2016); Allain (2013); Scott & Scott (2019)
Endurance	e	0.6 h	See text
Truck payload capacity	Q	1300 kg	-
Drone payload capacity	Q'	2.27 kg	DHL (2016); Allain (2013); Scott & Scott (2019)
Truck cost per km	C	\$0.03/km	Kim (2016); Lauryn (2019)
Drone cost per km	C'	\$1.25/km	Kim (2016); Lauryn (2019)
Maximum route duration	T, T'	up to 3 h	-
Grid size	d	up to 25 km	-
Number of customers	n	{30, 50, 100, 200, 400}	-
Depot location	g	{c, e, r}	-
Customer location	g	{r, c, rc}	-

4.2. Solution approach evaluation on the min-cost PDSVRP

We evaluate the proposed methods on the aforementioned generated instances. For each instance, the running time of solving MILP by CPLEX is limited to one hour, while the meta-heuristic is repeatedly run 30 times.

4.2.1. The impact of key SISSRs components

This part reports the sensitivity analysis on the impact of the key components of the proposed SISSRs: Local Search (LS), Threshold Acceptance (TA), Perturbation (Pert), Sweep Removal (Sweep), and Random Removal (Rand). In particular, the full-featured algorithm (i.e., **Full** configuration) and following configurations obtained by removing one chosen operator are investigated:

- **No LS** - local search operators are disabled
- **No TA** - only better solutions are accepted (i.e., hill climbing moves are not allowed)
- **No Pert** - no perturbation is performed
- **No Sweep** - only random drone removals are performed during ruin stage
- **No Rand** - only sweep drone removals are performed during ruin stage

Preliminary results suggest a ceiling effect upon the small instances, in which the benchmarks are too easy so that no noticeable differences are found among all configurations of the algorithm. For a reasonable computational effort, we decide to select the collection of 54 medium instances with 100 customers and 200 customers for this experiment. Several aggregated indicators for dispersion and central tendency are reported in Table 5: the gap of best solutions (Column gap_{best}); the quantiles of gaps including Q1 quantile, Q2 quantile (median), and Q3 quantile (Columns gap^{Q1}, gap^{Q2}, and gap^{Q3}, respectively); the average gap ($\overline{gap}$), the number of best solutions found by the configuration (# bks), and the average run time in seconds ($\overline{\text{time}}$). Note that, to calculate the gap tolerance value, the best solution s^* for each instance across all configurations is recorded as the reference.

Table 5: Sensitivity analysis on the components of SISSRs

Instance set	Config	gap_{best} (%)	gap^{Q1} (%)	gap^{Q2} (%)	gap^{Q3} (%)	$\overline{gap}$ (%)	# bks	$\overline{\text{time}}$ (s)
100-customer	no LS	0.088	1.321	2.964	4.505	3.030	20	8.017
	no TA	0.026	0.492	0.813	0.845	0.617	24	41.837
	no Pert	0.029	0.192	0.273	0.499	0.344	24	40.828
	no Sweep	0.025	0.191	0.193	0.438	0.281	26	40.711
	no Rand	0.025	0.191	0.194	0.491	0.279	25	40.424
	Full	0.025	0.191	0.193	0.275	0.268	26	47.730
200-customer	no LS	1.186	3.122	4.322	5.423	4.465	4	30.311
	no TA	0.278	0.915	1.495	2.075	1.557	15	120.096
	no Pert	0.265	0.567	0.943	1.500	1.017	18	118.333
	no Sweep	0.373	0.554	0.968	1.084	0.935	17	100.419
	no Rand	0.284	0.805	0.866	1.176	0.962	17	105.721
	Full	0.059	0.541	0.916	1.085	0.919	19	100.184

The obtained results suggest that the **Full** configuration yields the best performance in terms of solution quality (shown in gap_{best} and # bks columns) and consistency (shown in $\overline{gap}$ and quantile-related gap columns). This shows the significant contribution of all the key components to the quality of the method in overall. However, some components have much larger impact than others. This pattern appears especially clearer on 200-customer instances. The local search component has the most positive impact on the performance of the proposed method, followed by the threshold acceptance. The remaining components have the relatively similar impact.

4.2.2. Experiments on small instances

The objective of this part is to validate the MILP model and compare the performance of the proposed methods for the min-cost PDSVRP. Because CPLEX cannot solve the medium and large instances in reasonable running time, we select a collection of 54 small instances with

30 and 50 customers for the experiment. The initial solutions, which are obtained from the first recreate phase of the SISSRs, are also taken into account to assess the ability of SISSRs compared with simple constructive heuristics.

Tables 6 and 7 report the obtained results for this experiment. They provide information regarding the MILP performance: the objective value of the best solutions found (Column "z"), the gap returned by CPLEX (Column "gap"), and the execution time in seconds (Column "$time$"). Additionally, the quality of initial solutions is reported in Columns "$\overline{z_{init}}$" (the average objective value) and "$\overline{gap_{init}}$" (the average gap in percentage to the best found solution). Likewise, the performance of the SISSRs is accessed via: the objective values of the best found solutions (Column "z_{best}"), the gap of the best solutions found over 30 runs (Column "gap_{best}"), the average gap over 30 runs (Column "$\overline{gap}$"), the average competation time in seconds (Column "$\overline{time}$"). It should be noted that the gap value of heuristic solutions in this experiment is calculated as: $gap = 100(\frac{z'}{z^*} - 1)$, where z' and z^* are the objective values of solutions found by heuristic methods and the best solutions found by all methods (including heuristics and MILP formulation), respectively. Finally, we also provide information to expose the characteristics of the best solutions found by all methods: the number of drone-eligible customers (Column # DE) in the instance, the number of customers served by the drones (Column # D), and the number of trucks being used (Column # trucks).

Table 6: Performance of the proposed methods on small instances with 30 customers.

Instance	**MILP**			**Init**		**SISSRs**				#	#	#
	z	gap	$time$	$\overline{z_{init}}$	$\overline{gap_{init}}$	z_{best}	gap_{best}	$\overline{gap}$	$\overline{time}$	DE	D	trucks
30-r-0-c	128.67	0.00	169.06	217.00	68.65	128.67	0.00	0.00	7.19	23	13	3
30-r-0-e	300.85	49.59	1817.10$^+$	350.05	16.35	300.85	0.00	0.00	6.89	14	5	5
30-r-0-r	199.76	32.55	1536.89$^+$	268.07	34.19	199.76	0.00	0.00	6.33	14	8	4
30-r-1-c	58.30	0.00	2.79	134.68	131.04	58.30	0.00	0.00	8.24	26	16	1
30-r-1-e	243.11	38.71	651.71$^+$	314.41	29.33	243.11	0.00	0.00	7.73	16	8	4
30-r-1-r	164.75	22.79	2310.15$^+$	206.85	25.55	164.75	0.00	0.00	7.20	21	11	3
30-r-2-c	91.57	0.00	41.60	168.09	83.56	91.57	0.00	0.00	9.04	29	15	2
30-r-2-e	235.25	41.34	956.03$^+$	306.05	30.09	235.25	0.00	0.00	6.60	17	9	4
30-r-2-r	211.26	41.46	789.46$^+$	263.32	24.65	211.26	0.00	0.00	7.06	17	9	4
30-c-0-c	82.15	0.00	38.87	112.27	36.67	82.15	0.00	1.17	9.22	24	11	2
30-c-0-e	150.44	0.00	41.87	189.12	25.71	150.44	0.00	0.00	8.61	11	3	3
30-c-0-r	112.72	0.00	326.63	135.67	20.36	112.72	0.00	0.00	8.22	21	5	3
30-c-1-c	39.98	0.00	4.07	79.56	98.99	39.98	0.00	0.00	9.09	26	16	1
30-c-1-e	172.49	40.75	3600.00	184.49	6.96	172.49	0.00	0.00	8.13	20	7	3
30-c-1-r	61.28	0.00	1.11	117.53	91.78	61.28	0.00	0.00	6.61	24	13	1
30-c-2-c	77.00	0.00	6.17	107.44	39.53	77.00	0.00	0.00	9.23	26	18	2
30-c-2-e	166.82	14.95	3600.00	177.35	6.31	166.82	0.00	0.00	7.10	19	3	3
30-c-2-r	88.81	0.00	31.42	128.55	44.74	88.81	0.00	0.00	8.54	25	10	2
30-rc-0-c	81.06	0.00	15.81	119.70	47.67	81.06	0.00	0.00	8.43	26	14	2
30-rc-0-e	181.89	20.77	3600.00	231.10	27.05	181.89	0.00	0.00	7.35	14	6	3
30-rc-0-r	146.73	21.13	2491.38$^+$	182.11	24.11	146.73	0.00	0.00	8.22	22	5	3
30-rc-1-c	56.48	0.00	7.51	125.07	121.43	56.48	0.00	5.62	7.16	27	16	1
30-rc-1-e	176.57	14.24	1280.95$^+$	234.68	32.91	176.57	0.00	0.00	7.35	16	2	3
30-rc-1-r	97.91	0.00	83.60	150.12	53.32	97.91	0.00	0.00	9.17	26	17	2
30-rc-2-c	77.00	0.00	4.01	137.39	78.43	77.00	0.00	0.00	8.49	25	14	2
30-rc-2-e	187.18	26.17	3600.00	230.92	23.37	187.18	0.00	0.00	6.63	16	2	3
30-rc-2-r	94.21	0.00	3.70	134.04	42.28	94.21	0.00	0.00	5.93	17	14	2

The results somewhat reveal the impact of different problem parameters on the hardness of an instance. Obviously, when the number of customers increases, the larger MILP model makes the problem harder to solve to optimality. From Table 6, we observe that CPLEX is able to find the optimal solutions for 15 over 27 instances with 30 customers in less than 3 minutes. As the

Table 7: Performance of the proposed methods on small instances with 50 customers.

Instance	MILP			Init		SISSRs				#	#	#
	z	gap	$time$	$\overline{z_{init}}$	$\overline{gap_{init}}$	z_{best}	gap_{best}	$\overline{gap}$	$\overline{time}$	DE	D	trucks
50-r-0-c	161.13	0.00	169.26	287.95	78.70	161.13	0.00	0.00	15.35	41	15	3
50-r-0-e	-	-	2636.25$^+$	435.21	40.16	**310.50**	0.00	0.00	14.33	22	8	5
50-r-0-r	-	-	2498.11$^+$	353.81	43.52	**246.53**	0.00	0.00	13.79	29	14	4
50-r-1-c	177.21	9.56	3600.00	280.84	58.48	177.21	0.00	0.00	16.28	41	15	4
50-r-1-e	216.97	13.89	3600.00	370.02	70.54	216.97	0.00	0.00	14.59	24	10	3
50-r-1-r	-	-	2572.35$^+$	355.04	55.64	**228.11**	0.00	0.00	15.3	33	13	4
50-r-2-c	182.09	8.60	3600.00	278.73	53.58	**181.49**	0.00	0.58	13.56	32	15	4
50-r-2-e	-	-	2387.22$^+$	500.12	33.35	**375.05**	0.00	0.00	12.75	25	4	6
50-r-2-r	-	-	2633.51$^+$	380.61	21.48	**313.32**	0.00	0.00	12.9	29	12	6
50-c-0-c	89.04	0.44	3600.00	151.73	70.41	89.04	0.00	0.00	16.43	44	15	2
50-c-0-e	183.26	32.04	3600.00	255.69	39.79	**182.91**	0.00	0.00	15.53	30	10	4
50-c-0-r	112.17	3.83	3563.53$^+$	153.15	36.53	112.17	0.00	0.00	15.59	36	17	2
50-c-1-c	81.76	0.00	859.09	116.83	42.90	81.76	0.00	0.00	17.12	42	18	2
50-c-1-e	133.57	17.58	3600.00	177.55	32.92	133.57	0.00	0.00	12.94	41	6	3
50-c-1-r	149.24	41.45	1512.49$^+$	196.76	32.18	**148.86**	0.00	0.00	15.92	41	17	3
50-c-2-c	77.16	0.00	73.52	154.70	100.50	77.16	0.00	0.00	19.45	40	15	2
50-c-2-e	214.25	38.75	2684.94$^+$	292.78	36.65	214.25	0.00	0.00	16.97	23	13	4
50-c-2-r	136.60	29.48	2288.27$^+$	186.17	36.38	**136.51**	0.00	0.00	17.37	34	15	3
50-rc-0-c	116.80	0.00	53.87	236.97	102.88	116.80	0.00	0.21	19.06	43	16	2
50-rc-0-e	-	-	2679.98$^+$	345.61	42.80	**242.02**	0.00	0.00	14.64	32	5	4
50-rc-0-r	217.87	40.76	3600.00	295.49	35.63	217.87	0.00	0.00	15.63	39	18	4
50-rc-1-c	110.44	0.00	63.76	227.81	106.27	110.44	0.00	0.14	20.72	42	15	2
50-rc-1-e	201.62	27.15	3600.00	324.39	60.90	201.62	0.00	0.00	15.74	27	11	3
50-rc-1-r	173.16	23.59	2907.96$^+$	299.11	72.73	173.16	0.00	0.00	16.69	37	14	3
50-rc-2-c	129.82	7.22	3600.00	229.04	76.66	**129.65**	0.00	0.03	18.14	41	16	3
50-rc-2-e	-	-	3032.47$^+$	487.22	27.05	**383.49**	0.00	0.00	12.95	26	8	7
50-rc-2-r	197.20	33.36	3600.00	272.75	46.39	**186.31**	0.00	0.00	15.72	29	16	4

number of customers increases to 50, Table 7 shows that CPLEX averagely takes much more time and closes the gaps for only 5 instances. In the unsuccessful cases, CPLEX is forced to terminate due to out-of-memory status on 12 instances (marked as "+" in Column *time*) and cannot provide any feasible solution on 7 instances (marked as "-").

In addition, the distribution of customers appears to impact the hardness of the problem as well. The significant gap differences among three distribution groups ("r", "rc", "c") support this observation. In particular, a number of hard instances, on which CPLEX cannot reach to optimality, tend to concentrate more in random distribution instances and less in clustered ones. Optimal solutions are more frequently found in the "c" instance set, become scarce in the "rc" set, and very few in the "r" set. We also observe that the "c" instances require, in general, less running time to solve than the remaining instance sets.

The depot position also affects the problem hardness. Given the same customer position, we observe that the computation time required by CPLEX to solve instances to optimality is longer if the depot is located on the edge or randomly in the grid. In such instances, CPLEX frequently struggles to close the gaps and suffers the out-of-memory status, or even ends up with no feasible solution. As such, the instances with the depot locations fell into the "e" and "r" categories tend to be harder than the "c" one.

Another observation is that the quality of initial solutions appears inconsistent as their gaps broadly vary from 6.31% to 131.04% across 54 instances. This partly shows the hardness and diversity of the generated instances. The SISSRs has a good performance as it can find all the optimal solutions found by CPLEX. In addition, it finds better solutions than CPLEX does on 6 instances (marked in bold in Column z_{best}). The small gap values in Column $\overline{gap}$ on most cases

indicate the stability of the algorithm over multiple runs. In both tables, our metaheuristic is quite fast as its running time on average is less than 21 seconds.

Related to the characteristics of the min-cost PDSVRP solutions, we observe that depending on each instance setting, the ratio of customers served by the drones over the total drone-eligible customers varies in a wide range from 12.50% to 82.35%. This confirms again the diversity of the created instances and contributes to the perception that the allocation problem of customers to truck or drone would not be trivial to solve.

Other observations can be derived from the results. Locating the depot at the center of the grid creates the cheapest solutions while on the instances with the depot on the edge, the solutions are the most expensive. This can be explained by the fact that on the instances in which the depot is at the center, the customers are closer to the depot and the instances have more drone-eligible customers, leading to more customers being served by the drones and possibly less trucks being used. On the other hand, when the depot is located on the edges of the grid, the customers are further from the depot and more customers are served by the trucks, which are the more expensive transportation mode. As a result, the solutions in this scenario are in general more expensive. Finally, as expected, when the customers are located in clusters, the solutions are the cheapest, followed by the random-clustered and random distributions.

4.3. Experiments with medium and large instances

In this section, we expect to further analyze the performance of the proposed methods on a collection of medium and large instances containing 100, 200, and 400 customers. As the matter of fact, there are no other solution approaches proposed for the min-cost PDSVRP. In addition, preliminary results suggest that the MILP model is incapable of solving medium and large instances efficiently. As a result, we consider SISSRs to be the only subject of the experiment. Our main objectives of the experiment are to observe the behavior of the metaheuristic on large instances and provide comparison references for future approaches.

As in the previous experiments, we run 30 times on each instance and record multiple central tendency indicators. The obtained results are summarized in Tables 8, 9 and 10 in which the meaning of column headings can be derived from the previous result tables. As can be observed, the proposed method provides a consistent performance with the average gaps less than 5.59%. The consistency is also evident in the dispersion measured by the interquartile range $IQR = gap^{Q3} - gap^{Q1}$, which is less than 1% on most instances. Columns $\overline{gap_{init}}$ and $\overline{gap}$ show that the quality of the final solutions is substantially improved thanks to the SISSRs, compared to the corresponding initial solutions. The average gaps of the initial solutions varying from 41.51% to 206.72%, which are relatively high. Finally, the computation time of SISSRs is acceptable when its average values in Column $\overline{time}$ never exceed 10 minutes on the large instances with 400 customers.

4.4. Sensitivity analysis

In this experiment, we perform a sensitivity analysis of certain problem-related parameters of interest to expose some preliminary insights into their impacts on the contributions of drones to reduce the transportation cost. Specifically, we aim to answer the following questions: (1)

Table 8: Performance of the proposed heuristic method on 100-customer instances of the min-cost PDSVRP

Instance	z_{best}	gap^{Q1} (%)	gap^{Q2} (%)	gap^{Q3} (%)	$\overline{gap}$ (%)	$\overline{gap_{init}}$ (%)	$\overline{\text{time}}$ (s)	# DE	# D	# trucks
100-r-0-c	272.77	0.00	0.00	0.17	0.07	76.67	47.47	69	25	4
100-r-0-e	506.07	0.00	0.00	0.00	1.37	55.94	42.01	30	12	6
100-r-0-r	325.29	0.01	0.02	0.07	0.04	60.78	47.62	64	23	5
100-r-1-c	302.46	0.00	0.00	0.00	0.00	83.81	43.82	56	23	4
100-r-1-e	470.06	0.00	0.00	0.00	0.00	69.94	41.40	35	7	6
100-r-1-r	350.97	0.00	0.00	0.00	0.00	69.75	42.01	50	23	5
100-r-2-c	284.77	0.00	0.00	0.00	0.00	76.03	47.86	73	23	4
100-r-2-e	435.91	0.00	0.00	0.00	0.00	70.68	46.45	39	11	5
100-r-2-r	320.98	0.00	0.00	0.00	0.00	69.35	43.83	61	26	5
100-c-0-c	218.59	1.21	1.21	1.21	1.13	55.42	44.15	45	19	3
100-c-0-e	315.84	0.00	0.00	0.00	0.00	47.97	44.02	24	8	4
100-c-0-r	252.39	0.00	0.00	0.00	0.00	43.71	54.34	67	17	4
100-c-1-c	154.61	0.00	0.00	0.00	0.00	75.06	42.18	29	10	2
100-c-1-e	222.15	0.00	0.00	0.00	0.00	55.36	68.39	62	20	3
100-c-1-r	110.95	0.00	0.00	0.00	0.00	92.36	58.62	78	26	2
100-c-2-c	108.35	0.00	0.00	0.00	0.00	56.09	66.87	64	18	2
100-c-2-e	118.71	3.27	3.27	3.27	3.05	70.09	50.35	69	20	2
100-c-2-r	86.28	0.00	0.00	2.00	0.87	67.52	72.53	65	25	1
100-rc-0-c	252.37	0.00	0.03	0.03	0.02	95.51	45.62	74	25	4
100-rc-0-e	472.95	0.00	0.00	0.00	0.00	55.68	37.57	28	2	6
100-rc-0-r	337.61	0.00	0.00	0.00	0.00	61.86	39.56	54	23	5
100-rc-1-c	276.78	0.00	0.00	0.00	0.00	54.88	49.72	66	26	4
100-rc-1-e	432.44	0.00	0.00	0.00	0.00	55.75	39.55	18	2	5
100-rc-1-r	423.33	0.00	0.00	0.00	0.03	60.14	48.09	55	19	6
100-rc-2-c	260.45	0.00	0.00	0.00	0.00	82.78	41.91	63	24	4
100-rc-2-e	358.24	0.00	0.00	0.00	0.00	70.44	37.68	19	7	4
100-rc-2-r	331.97	0.00	0.00	0.00	0.00	41.51	45.08	57	21	5

"which parameters have the most important influences on the cost efficiency of the system?"; and (2) "what scenarios facilitate the use of drones to create a considerable amount of cost saving?". Answering these questions could play an important role in the decision to adopt the new truck-drone combination for delivery in practice, as well as during the investment process and system design.

4.4.1. Methodology

In recent research related to drone and truck routing problems, the one-factor-at-a-time (OFAT) approach has been widely used for similar sensitivity analysis due to its simplicity (Ha et al., 2018; Murray & Chu, 2015; Ha et al., 2020; Agatz et al., 2018; Sacramento et al., 2019). This approach varies the value of each independent variable one at a time, whereas the remaining ones are kept constant, to monitor how the considered variable affects the investigated outcome. However, this approach does not fully explore the entire value space, since it does not take into account the simultaneous variation of independent variables. This means the OFAT approach often fails to detect the presence of interactions among independent variables, hence it limits general insights.

In this research, we instead apply a powerful analytical method, named *Classification And Regression Trees (CART)* (Breiman, Friedman, Stone & Olshen, 1984), for our sensitivity study. Specifically, the method is able to reveal interactions among variables via building a regression model in the form of a decision-tree structure. In such a model, the problem-related parameters

Table 9: Performance of the proposed heuristic method on 200-customer instances of the min-cost PDSVRP

Instance	z_{best}	gap^{Q1} (%)	gap^{Q2} (%)	gap^{Q3} (%)	$\overline{gap}$ (%)	$\overline{gap_{init}}$ (%)	$\overline{\text{time}}$ (s)	# DE	# D	# trucks
200-r-0-c	430.42	0.00	0.00	0.03	0.02	106.34	99.22	117	27	6
200-r-0-e	728.00	0.00	0.00	0.00	0.00	87.30	95.74	62	9	9
200-r-0-r	528.29	5.01	5.22	5.33	4.92	92.95	96.22	77	29	6
200-r-1-c	434.66	0.37	1.18	4.55	2.31	110.86	99.89	124	26	6
200-r-1-e	601.06	0.00	0.00	0.03	0.08	73.12	98.45	67	15	7
200-r-1-r	481.83	0.09	0.09	0.09	0.26	113.69	100.05	89	29	6
200-r-2-c	446.15	2.67	2.78	2.89	2.74	105.06	97.84	120	26	6
200-r-2-e	606.05	0.10	0.10	0.10	0.11	91.49	92.80	60	14	7
200-r-2-r	497.72	0.05	0.16	0.43	0.25	93.11	98.17	102	28	7
200-c-0-c	234.40	0.00	0.00	0.00	0.00	127.36	92.52	130	21	4
200-c-0-e	442.74	0.00	0.71	0.71	0.38	52.05	85.54	67	3	6
200-c-0-r	270.76	0.28	0.28	0.28	0.26	57.28	104.52	94	34	4
200-c-1-c	225.55	0.00	0.00	0.00	0.00	92.37	124.65	129	26	3
200-c-1-e	461.57	0.00	0.00	0.00	0.00	51.88	88.96	93	15	6
200-c-1-r	239.91	0.00	0.00	0.00	0.00	106.41	108.73	122	22	4
200-c-2-c	263.26	0.99	0.99	0.99	0.95	108.99	96.69	120	21	4
200-c-2-e	486.05	0.04	7.62	7.66	5.27	74.15	104.88	79	15	6
200-c-2-r	322.61	0.00	0.00	0.00	0.00	69.71	98.28	113	30	5
200-rc-0-c	354.23	0.24	0.67	0.68	0.55	108.42	103.12	139	24	5
200-rc-0-e	575.94	0.00	0.00	0.00	0.21	92.28	90.85	51	8	7
200-rc-0-r	484.66	0.04	0.15	0.26	0.71	67.90	105.72	106	31	7
200-rc-1-c	370.01	0.00	0.00	0.17	0.06	99.75	94.90	86	24	5
200-rc-1-e	591.58	0.03	0.03	0.04	0.08	75.52	108.67	100	17	7
200-rc-1-r	411.34	0.00	0.00	0.00	0.02	66.50	97.83	90	32	6
200-rc-2-c	372.30	0.50	0.56	0.85	1.53	114.94	108.07	135	25	5
200-rc-2-e	534.45	0.25	0.25	0.25	0.24	111.56	110.21	77	23	6
200-rc-2-r	432.17	2.32	2.33	2.33	2.23	90.70	102.45	89	24	5

play the role of *predictor variables* (i.e., independent variables). The *outcome variable* (i.e, dependent variable) of the investigation is the cost-effectiveness of using drones in the delivery system, which is measured by the *cost saving* computed as $100\frac{(z-z')}{z}$, where z' and z are the transportation costs of the system with and without using drones, respectively.

CART constructs a decision tree using the reduction in a statistical criterion, referred to as "impurity" (i.e., variance), as the guideline. From the root node containing the entire sample, it performs binary recursive partitions (*splits*) the data is set into smaller groups at each level so as to produce child nodes that are as homogeneous as possible on the dependent variable (Ma, 2018; Breiman et al., 1984). In our regression tree, the impurity of a node is measured by the Mean Squared Error (*MSE*). To partition a parent node into two child nodes, the model searches for the best pair of predictor variable and split value that divides the data into two groups (S_1 and S_2) with the minimum overall MSE (Kuhn & Johnson, 2013):

$$\text{MSE} = \frac{1}{n_1}\sum_{i\in S_1}(y_i - \bar{y}_1)^2 + \frac{1}{n_2}\sum_{i\in S_2}(y_i - \bar{y}_2)^2 \tag{17}$$

where n_j and $\bar{y}_j$ are the number of samples and the mean of dependent variable of groups S_j ($j = 1, 2$). The corresponding y_i in each group S_j indicates the actual value of the dependent variable for case i ($i = 1, 2, 3, ...|S_j|$).

Table 10: Performance of the proposed SISSRs on 400-customer instances of the min-cost PDSVRP

Instance	z_{best}	gap^{Q1} (%)	gap^{Q2} (%)	gap^{Q3} (%)	$\overline{gap}$ (%)	$\overline{gap_{init}}$ (%)	$\overline{\text{time}}$ (s)	# DE	# D	# trucks
400-r-0-c	604.94	0.18	0.36	0.43	0.34	142.17	399.70	268	33	7
400-r-0-e	801.45	0.02	0.09	0.18	0.12	119.78	418.55	135	28	9
400-r-0-r	626.46	0.33	0.45	1.30	1.08	135.85	425.18	209	38	7
400-r-1-c	578.77	0.22	0.38	0.54	0.39	150.89	407.54	259	37	7
400-r-1-e	772.30	0.06	0.22	0.46	0.39	110.50	415.05	135	22	9
400-r-1-r	584.07	0.84	0.99	1.28	0.96	137.24	439.87	226	38	7
400-r-2-c	587.88	0.25	0.32	0.50	0.37	153.58	400.80	246	35	7
400-r-2-e	771.16	0.68	0.82	0.89	0.78	112.80	428.22	128	23	9
400-r-2-r	692.91	0.82	3.89	4.45	3.04	135.09	459.51	125	34	7
400-c-0-c	271.82	0.13	0.17	0.40	0.60	185.27	490.44	308	32	3
400-c-0-e	378.98	0.08	0.12	0.14	0.11	137.50	387.45	100	27	4
400-c-0-r	403.85	0.17	0.18	0.22	0.18	128.48	415.42	208	26	5
400-c-1-c	187.20	0.09	0.11	0.18	0.14	206.72	582.43	282	27	2
400-c-1-e	249.01	0.10	0.10	0.29	0.20	139.74	500.52	151	36	3
400-c-1-r	219.23	0.00	0.05	0.17	0.09	158.18	442.16	286	27	3
400-c-2-c	152.79	5.65	5.82	5.87	5.59	110.05	595.08	268	34	2
400-c-2-e	245.54	0.04	0.06	0.09	0.07	102.34	368.10	140	18	3
400-c-2-r	231.89	0.01	0.02	0.05	0.03	114.09	384.89	189	24	3
400-rc-0-c	497.90	0.87	0.93	1.12	0.81	147.94	371.11	236	34	6
400-rc-0-e	700.96	0.24	0.25	0.75	0.47	96.11	356.04	117	21	8
400-rc-0-r	604.38	0.19	1.15	2.95	1.48	109.23	424.86	194	37	7
400-rc-1-c	518.40	0.12	0.14	0.92	0.42	137.35	341.52	193	35	6
400-rc-1-e	784.02	0.02	0.02	0.02	0.06	100.85	334.95	73	11	9
400-rc-1-r	531.44	2.62	2.95	2.96	2.62	120.71	399.80	216	32	6
400-rc-2-c	510.64	0.17	0.27	0.39	0.29	140.09	410.27	278	39	6
400-rc-2-e	723.63	0.15	0.17	0.33	0.22	94.64	378.33	120	27	8
400-rc-2-r	625.31	0.23	0.39	0.51	0.40	100.77	434.71	139	29	6

The partitioning process stops when the resulting leaves are sufficiently homogeneous. A combination of useful stopping criteria (i.e., tree pruning) is necessary to avoid over-fitting (i.e., not working well on new data) and produce a readable graph for demonstration purposes. In this research, we limit the depth of the tree to 5 levels and the minimum number of samples per leaf node to 30.

Figure 7 visualizes the overall procedure of the sensitivity analysis using the CART approach. The process starts with the selection of investigated parameters from which a number of instances are randomly generated. The generated instances are then solved by SISSRs and the results (i.e., outcome variable values) are recorded. Finally, the entire data set composed of values of the investigated parameters and corresponding results are used to build a CART decision tree. We employ `DecisionTreeRegressor` class in Python's Scikit-learn library to implement the CART method.

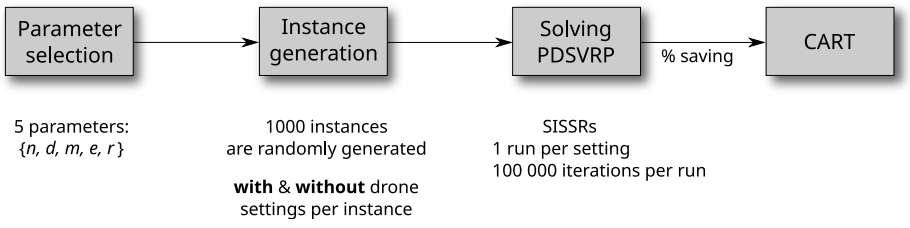


Figure 7: The process of sensitivity analysis using CART method

4.4.2. Selected parameters and instances

In the first step, we select five important parameters for investigation: (1) the number of customers n, (2) the grid size d, (3) the number of drones m, (4) the drone's battery endurance e, and (5) the truck-drone cost ratio r (referred to as cost ratio). To make the result more insightful, we conduct the analysis on generated instances that are capable of simulating real-life scenarios. Therefore, the selection of related parameters is based on the assumptions made in Subsection 4.1. The list of parameters and their values are detailed in Table 11. The instance generation begins by uniformly choosing values for five investigated parameters n, d, m, e, and r in certain ranges, then the remaining parameters are set at constant to fixed values. Following the process, 1000 random instances are generated. Notice that for each instance, two settings: *with* and *without* drones integrated in the delivery system, are then solved by SISSRs to determine how much cost saving the drones provide.

Table 11: Parameters controlled by the random generator

Indicators	**Parameters**	**Range**
n	Number of customers	$[50, 500]$
d	Grid size d (in km)	$[20, 40]$
m	Number of drones	$[1, 20]$
e	Drone's battery endurance (in hour)	$[0.25, 1.0]$
r	Ratio of truck unit cost over drone unit cost, $r = \frac{C}{C'}$	$[10.0, 50.0]$
constant	Truck speed (in km/h)	30
	Drone speed (in km/h)	40
	Truck payload capacity (in kg)	1300
	Drone payload capacity (in kg)	2.27
	Drone cost unit per km (in $)	0.03
	Work time limit (in hour)	8.0

4.4.3. Results and discussions

Parameter importance

For each parameter, we first report the relative importance computed as the normalized total reduction of impurity across all splits (Scikit-learn, 2020). The more important parameter tends to drive the higher impact on cost saving. This metric helps to answer the first question on what parameters have the biggest impacts on predictions. Figure 8 visualizes the ranked parameter importance of the five considered parameters.

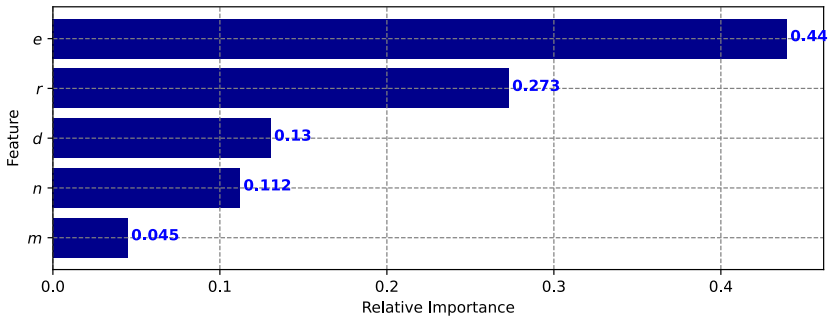


Figure 8: Parameter importance

As can be seen in the figure, battery endurance e would be the most essential parameter asso-

ciated with the cost saving. A possible explanation is that a wider coverage range of the drones would offer more opportunities for vehicle assignment alternatives resulting in cost-cutting. In several worse case scenarios where the endurance of the drone is low, the drones can only operate near the depot. Substituting truck deliveries in such areas with a drone alone would not provide significant impacts to reduce the overall truck cost as the trucks still have to serve the majority of the distant customers. This is not to mention other factors (e.g., the presence of truck-only requests and low cost ratios) which could reduce the drone's contribution within the area.

Cost ratio r follows in the second place. It is important to emphasize that the drone tends to suffer high miles-per-delivery due to its back-and-forth delivery discipline. Such a round trip would become costly if the customer is far from the depot. In this respect, a truck often requires less effort (i.e., traveled distance) to serve additional customers compared to proposing a new round-trip by a drone. Especially when the customers are close to each other in a given area, the truck even appears to be the more favorable option. This sheds light on the profound importance of the cost ratio. For possible cost saving, it is clear that the cost ratio r should be as high as possible to outweigh the cost of back-and-forth drone trips to stimulate drone use over truck use. The two parameters, grid size d and number of customers n, have relatively equal importance and less impacts on the cost saving. Finally, the least impact factor is the number of drones m.

Interpretation of the decision-tree and discussion

We now analyse the results based on the interpretation of the CART decision tree. From Figure 9, we can observe important split points and interactions among parameters. This helps to answer the second question of finding the scenarios which can take advantage of the drone use to achieve a considerable amount of cost saving.

We refer to the first node at the top of the tree as the root node, which contains the information of the entire sample data. The terminal nodes at the bottom of the tree are called leaves. Every node in between is referred to as an internal node. Each node in the tree is characterized by the following parameters: (1) "mse" is the mean squared error value; (2) "samples" is the number of samples; and (3) "value" is the average cost saving of data samples. At each node, we verify a condition on a variable. The left branch of a node corresponds to the samples that satisfy the condition (True statement) and the right branch represents the samples that do not meet the condition (False statement).

Overall, using drones could offer 20.42% cost saving on average over 1000 samples. Based on the results for the data range under investigation, it is clear that each parameter has different impacts in different scenarios:

- Unsurprisingly, the greatest impact among all five considered parameters, the battery endurance, is verified at the root of the tree. The results suggest that battery endurance e larger than 0.655 h is the essential criterion to secure significant cost savings, 29.709% on average compared to 12.533% of those with $e \leq 0.655$. Closer look on the left branch where the battery endurance is below the essential criterion ($e \leq 0.655$), further partitioning on e again emphasizes the importance of the battery capacity. For example, in the branch following Path 1, only satisfying $e \geq 0.435$ can still secure an appealing cost saving.

Otherwise, when $e < 0.435$, the limited flying range leaves drones little room to contribute significant cost savings.

- When the battery endurance is not favorable (e.g., $e \leq 0.655$ in our experiment), further partitioning on d in the left branch shows that the grid size should be the next important factor to consider. The results suggest that when e is greater than 0.505, decent cost savings of 18.001% can be obtained in cases with grid size d less than 33.5. We also observe that e should be at least 0.345 to cover efficiently a grid size d less than 22.5.

- On the other hand, when the battery endurance is favorable (e.g., $e > 0.655$ in our experiment), the number of customers n affects the contribution of drones. It is shown that using drones is more beneficial when less customers are served. A possible explanation is that the decrease in n could lower the customer density, which raises the insertion cost for a truck to accommodate additional customers into its current tour. In other words, dense customer distribution creates more chances to form good customer permutations for truck tours. Contrarily, drone trips tend to be unaffected by the customer density. As can be seen on the right branch, the results show a boost in cost savings (Path 2 and Path 3) of lower n compared to that of higher n.

- One could expect the higher cost ratio r provides better cost savings. In fact, the results suggest the cost ratio is particularly important when the battery endurance guarantees a sufficient flying range (e.g., $e > 0.655$). The cost ratio r greater than 22.65 is a favorable condition for a boost in cost saving (Path 3 and Path 4). Besides, the cost ratio still plays a relatively important role when the grid size is small. In such cases, it must reach to a given value ($r > 26.75$) to overshadow the truck cost which turns out to be competitive as customers are close to each other (Path 1).

- The number of drones appears important only when the drone use is encouraged (i.e., drone battery and cost ratio are favorable), and the number of customers is sufficiently large ($n > 245$). It is reasonable that a given number of drones are needed to fully exploit the incentive of drones for potential cost reductions. In such cases, as long as at least 6 drones are available, the system is ready to offer averagely 33.69% cost saving (Path 4), which is nearly double the cost savings achieved in cases suffering the lack of drone workforce (the branch corresponding to $m \leq 5$).

Several examples of scenarios, in which the drones can only contribute a small amount of cost saving, are visualized in Figure 10. Here, the dots are customer locations, the filled square in the middle is the depot, and the dashed circle indicates the drone's flight range (equivalent to the drone's battery endurance).

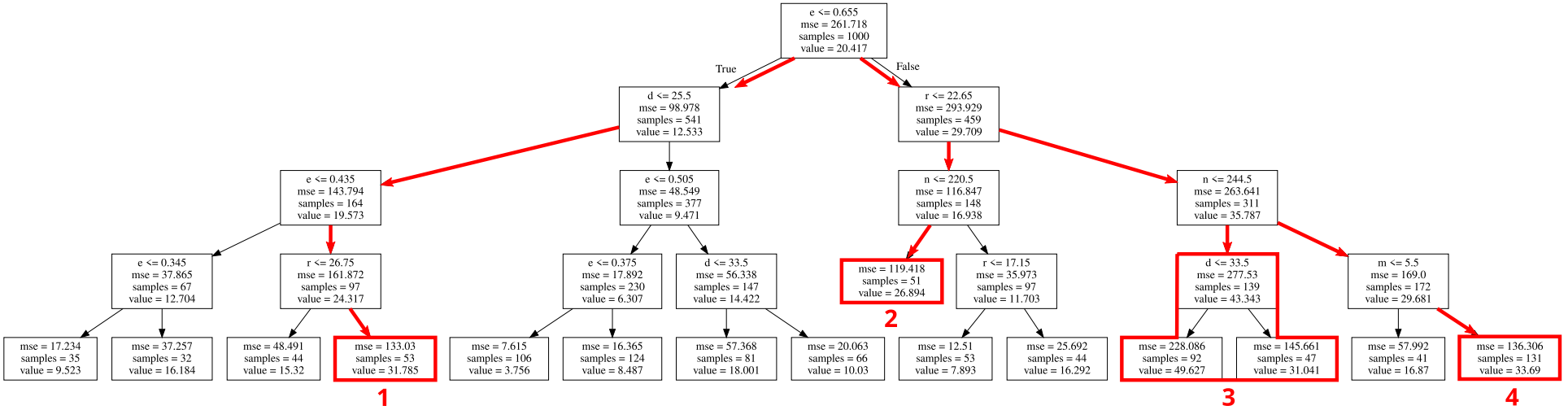
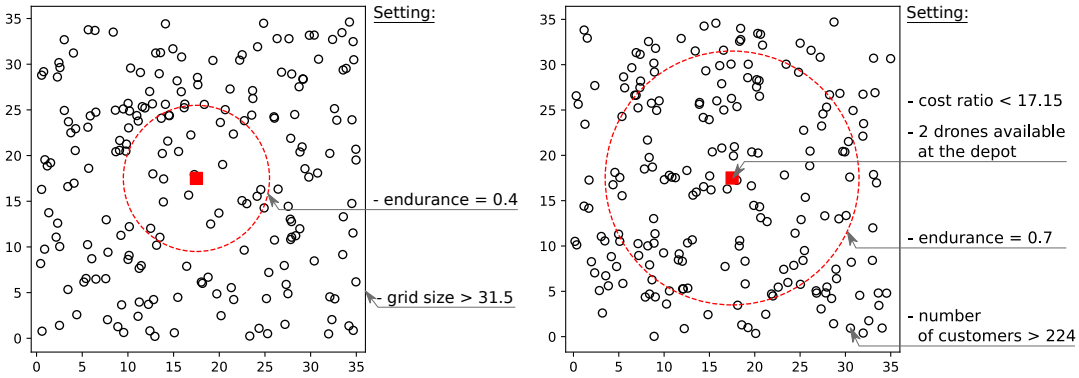


Figure 9: Illustration of the resulting regression tree



(a) Poor battery endurance with large grid size (b) Unattractive drone cost or lacking of drones available

Figure 10: Examples of scenarios in which drones show little contribution to reduce transportation cost

Finally, it is worth mentioning that this part does not intend to generalize the insights to a broader context. Depending on scenarios, we are aware that the results might change for different datasets. While the insights are derived from the tested instances, the majority of our data assumptions are based on real-world information. We believe the general theme might still be applied to relevant circumstances within a certain extent. In addition, the decision-tree structure is useful for practitioners to easily detect and interpret important factors for a specific case. Besides, the ability to identify split values, in which two distinct subgroups are clearly separated in terms of cost saving, is also valuable as it provides the very first intuition for the decision-making process. As a result, we encourage practitioners to adapt this analysis method to their own specific situations.

4.5. Performance of SISSRs on the min-time variant

In this subsection, we aim to analyze the performance of our algorithm on the min-time variant. We compare the SISSRs with other state-of-the-art algorithms on the PDSTSP instances generated in Mbiadou Saleu et al. (2018), which have been intensively used in the recent related works. It is worth mentioning again that the PDSTSP, which was originally introduced by Murray & Chu (2015), is the special case of the min-time PDSVRP where the number of trucks is set to 1 and the capacity constraints are relaxed. The PDSTSP instances introduced in Mbiadou Saleu et al. (2018) are based on classic TSPLIB instances (Reinelt, 1991) with the number of customers varying from 48 to 229. For each original TSP instance, several PDSTSP instances were generated by modifying the following parameters: the position of the depot, which is either at the center of the customers or at one corner of the customers' region; the percentage of drone-eligible customers, ranging from 0% to 100%; the speed of the drones, which is expressed as a factor of the vehicle speed, with values ranging from 1 to 5; the number of drones, that are between 1 and 5. The tables are organized in blocks to highlight the series of tests where a single construction parameter is modified from the reference instance (in the first row). We refer the interested readers to Mbiadou Saleu et al. (2018) for the detailed description and generator of these instances.

Two recent algorithms proposed in Mbiadou Saleu et al. (2018) and Dell'Amico et al. (2020) are selected for the comparison. We notice that the two referenced algorithms are tested only in a single run. In addition, they are run on an Intel core(TM) i5-6200U CPU at 2.30Ghz and an Intel(R) Xeon(R) E5-2620 v4 at 2.10GHz processors, respectively. These machines have a single-thread performance rating of approximately 1.6 times slower than our processor according to Passmark Software (https://www.cpubenchmark.net/). Since different CPU speed conversion techniques can provide very different results, we have decided to present the raw running times, letting the readers choose their preferred approach. The maximum computation time in Mbiadou Saleu et al. (2018) and Dell'Amico et al. (2020) is limited within 300 and 1200 seconds, respectively. Therefore, to have a fair comparison, we set the algorithm to stop after 300 seconds for each run. In addition, we make another 10 runs to evaluate the stability of the algorithm. Hence, SISSRs is repeatedly run 11 times in total for each instance.

Tables 12-15 below show the results for the instances with 101-229 customers. Because the three algorithms provide solutions with the same objective values for the instances with less than 100 customers, these instances appear to be easy for metaheuristic methods. As a result, we do not report the results for them. Each instance is characterized by several settings, namely, the percentage of drone-eligible customers over the total (el), the drone speed (sp), the number of drones (#), and the depot location (dp). In Columns MDFGQ18 and DMN20, we report the objective value of the best solutions found by Mbiadou Saleu et al. (2018) and Dell'Amico et al. (2020), respectively. Regarding the SISSRs, we report the single-run result (Column z^1), the best result over 10-run (Column z_{best}), and the average gap (Column $\overline{gap}$) over 10 runs. The better results in single-run mode of SISSRs are underlined in column z^1, while the new best known solutions of SISSIRs in 10 executions are marked in bold in column z.

Table 12: Results of the algorithms on the TSPLIB instance *eil101* from Mbiadou Saleu et al. (2018)

Instance				**MDFGQ18**	**DMN20**	**SISSRs**		
el	**sp**	**#**	**dp**			z^1	z_{best}	$\overline{gap}(\%)$
80	2	1	1	564.00	564.00	564.00	564.00	0.02
80	2	1	2	650.00	648.98	648.98	648.98	0.03
0	2	1	1	819.00	819.00	819.00	819.00	0.00
20	2	1	1	738.00	736.00	736.00	736.00	0.00
40	2	1	1	646.00	646.00	646.00	646.00	0.00
60	2	1	1	578.00	578.00	578.00	578.00	0.00
100	2	1	1	561.41	560.00	560.00	560.00	0.08
80	1	1	1	650.00	650.00	650.00	650.00	0.00
80	3	1	1	504.00	504.00	503.19	**503.19**	0.00
80	4	1	1	456.00	456.00	456.00	456.00	0.00
80	5	1	1	420.83	421.00	420.83	420.83	0.00
80	2	2	1	456.00	456.00	456.00	456.00	0.02
80	2	3	1	395.00	395.00	395.00	395.00	0.10
80	2	4	1	346.68	346.00	346.00	346.00	0.00
80	2	5	1	319.74	318.00	318.00	318.00	0.00

Table 13: Results of the algorithms on the TSPLIB instance *gr120* from Mbiadou Saleu et al. (2018)

Instance				MDFGQ18	DMN20	SISSRs		
el	**sp**	**#**	**dp**			z^1	z_{best}	$\overline{gap}(\%)$
80	2	1	1	1414.00	1420.76	1414.00	1414.00	0.00
80	2	1	2	1730.00	1726.00	1726.00	1726.00	0.21
0	2	1	1	2006.00	2006.00	2006.00	2006.00	0.00
20	2	1	1	1736.00	1736.00	1736.00	1736.00	0.00
40	2	1	1	1624.00	1624.00	1624.00	1624.00	0.01
60	2	1	1	1494.00	1494.00	1494.00	1494.00	0.00
100	2	1	1	1414.80	1416.00	1414.00	**1414.00**	0.00
80	1	1	1	1592.00	1592.00	1592.00	1592.00	0.18
80	3	1	1	1289.27	1291.00	1284.74	**1284.74**	0.00
80	4	1	1	1189.71	1192.00	1186.00	**1186.00**	0.00
80	5	1	1	1112.00	1114.00	1112.00	1112.00	0.00
80	2	2	1	1188.51	1197.00	1186.00	**1186.00**	0.00
80	2	3	1	1044.65	1050.00	1044.00	**1044.00**	0.00
80	2	4	1	946.04	946.04	943.00	**943.00**	0.01
80	2	5	1	880.00	881.00	878.70	**878.69**	0.00

Table 14: Results of the algorithms on the TSPLIB instance *pr152* from Mbiadou Saleu et al. (2018)

Instance				MDFGQ18	DMN20	SISSRs		
el	**sp**	**#**	**dp**			z^1	z_{best}	$\overline{gap}(\%)$
80	2	1	1	76008.00	76008.00	76008.00	76008.00	0.00
80	2	1	2	76556.00	76556.00	76556.00	76556.00	0.00
0	2	1	1	86596.00	86596.00	86596.00	86596.00	0.00
20	2	1	1	82504.00	82504.00	82504.00	82504.00	0.00
40	2	1	1	77372.00	77316.00	77236.00	**77236.00**	0.00
60	2	1	1	76786.00	76786.00	76786.00	**76758.00**	0.01
100	2	1	1	74468.00	74302.00	74302.00	74302.00	0.00
80	1	1	1	80164.00	79952.00	79952.00	79952.00	0.00
80	3	1	1	72936.00	72936.00	72936.00	72936.00	0.00
80	4	1	1	70412.00	70328.00	70148.00	**70148.00**	0.00
80	5	1	1	67798.00	67798.00	*67858.00*	*67858.00*	0.00
80	2	2	1	70244.00	70405.45	70148.00	**70148.00**	0.00
80	2	3	1	65062.10	64720.30	64550.00	**64550.00**	1.20
80	2	4	1	60027.40	59772.00	59756.00	**59756.00**	0.00
80	2	5	1	56336.10	56262.00	56227.04	**56178.00**	0.08

Table 15: Results of the algorithms on the TSPLIB instance *gr229* from Mbiadou Saleu et al. (2018)

Instance				MDFGQ18	DMN20	SISSRs		
el	**sp**	**#**	**dp**			z^1	z_{best}	$\overline{gap}(\%)$
80	2	1	1	1794.84	1785.86	1781.42	**1780.86**	0.10
80	2	1	2	1913.74	1911.58	1911.58	1911.58	0.02
0	2	1	1	2020.16	2017.24	2017.24	2017.24	0.01
20	2	1	1	1862.76	1860.14	1860.14	1860.14	0.00
40	2	1	1	1828.02	1827.02	1825.44	**1824.80**	0.04
60	2	1	1	1807.50	1797.37	1796.62	**1796.62**	0.01
100	2	1	1	1498.05	1496.29	1496.29	1496.29	0.03
80	1	1	1	1865.00	1863.12	1861.98	**1861.98**	0.11
80	3	1	1	1735.16	1725.45	1716.74	**1716.74**	0.02
80	4	1	1	1679.33	1675.82	1666.40	**1664.78**	0.02
80	5	1	1	1642.04	1629.38	1621.28	**1620.24**	0.07
80	2	2	1	1686.75	1673.72	1664.78	**1664.78**	0.06
80	2	3	1	1603.90	1592.52	1581.00	**1580.88**	0.00
80	2	4	1	1518.62	1526.92	1511.74	**1511.74**	0.00
80	2	5	1	1483.68	1467.76	1458.74	**1458.74**	0.01

As can be observed, not only is our algorithm able to reproduce nearly all but also improve a significant amount of state-of-the-art solutions. Specifically, the single run of SISSRs finds better solutions on 26 out of 90 considered instances while providing only one worse solution (marked by italic numbers in Table 14) than the two other methods. A higher concentration of improvements is on the larger instances, especially those that need to make use of a higher number of drones for delivery. In all instances that have more than one drone in sets *gr120*, *pr152*, and *gr229*, our algorithm finds improved solutions.

In the 10-run test, the average gaps show that the SISSRs is able to provide consistent results as the maximum gap value is just 1.20%. Remarkably, in this test, we further improve 7 solutions previously found in the single-run test. And in total, 26 new best known solutions (numbers in bold in Column z_{best}) have been found in this research. Therefore, it would be fair to conclude that the SISSRs dominates the existing metaheuristics in terms of solution quality to become the new state-of-the-art method for the PDSTSP.

5. Conclusions

In this paper, we have studied a new variant of the Parallel Drone Scheduling Traveling Salesman Problem (PDSTSP), named the min-cost Parallel Drone Scheduling Vehicle Routing Problem (PDSVRP). The problem seeks to minimize the total transportation costs incurred by a mixed fleet of trucks and drones embedded in a delivery system. We propose a mixed integer linear programming formulation and a simple, yet efficient Ruin-and-Recreate ($\mathcal{R}\&\mathcal{R}$) heuristic method, called SISSRs, in an attempt to address the new problem. The SISSRs, in particular, exploits PDSVRP solution characteristics in an effective manner to address common drawbacks of regular ruin and recreate methods. By embracing the slack induction concept, the algorithm manages to introduce "sufficient" room to a solution via Adjacent String Removal and Sweep Removal during the ruin phase. It is expected to enhance the possibilities for improving solutions later in the recreate phase that greedily inserts with blinks removed customers. We also propose several adaptations of the algorithm to tackle the PDSTSP, the min-time variant of PDSVRP with a single truck.

Via multiple intensive experiments, we validate our model and demonstrate the performance of our SISSRs algorithm. Overall, the obtained results on a new set of randomly generated instances with up to 400 customers confirm that our metaheuristic can stably provide high-quality solutions in a short running time. In addition, we perform a sensitivity analysis using the Classification And Regression Trees (CART) to quantitatively understand the impact of drones on the total transportation cost of the delivery system under different scenarios. The results suggest that the battery capacity is the most important parameter for drones to guarantee attractive benefits, followed by the truck-drone cost ratio. The other three parameters, namely, the number of customers, number of drones, and grid size have less of an impact on cost savings. From the analysis, we also determine in which scenarios using drones could provide the most and the least cost savings. As for solving the min-time variant, it is indicative that the proposed algorithm dominates the state-of-the-art algorithms in terms of solution quality. More remarkably, we find 26 new best known solutions.

Acknowledgement

The authors would like to thank the anonymous reviewers for the valuable comments and suggestions which help us greatly improve the quality of the paper. This paper is dedicated to the memory of Quang Minh Ha, who sadly passed away in 2020. His works have laid the foundation for our research direction on routing problems with drones.

References

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

Allain, R. (2013). Physics of the Amazon Octocopter Drone. https://www.wired.com/2013/12/physics-of-the-amazon-prime-air-drone/. Accessed 12 August 2020.

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

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

Carlsson, J. G., & Song, S. (2018). Coordinated logistics with a truck and a drone. *Management Science*, *64*, 4052–4069.

Cheng, C., Adulyasak, Y., & Rousseau, L.-M. (2020). Drone routing with energy function: Formulation and exact algorithm. *Transportation Research Part B: Methodological*, *139*, 364–387.

Christiaens, J., & Berghe, G. V. (2020). Slack induction by string removals for vehicle routing problems. *Transportation Science*, *54*, 417–433.

Conceição, A. (2019). *Logistics Challenges in a New Distribution Paradigm : Drone Delivery*. Technical Report Instituto Superior Técnico.

Dayarian, I., Savelsbergh, M., & Clarke, J.-P. (2020). Same-day delivery with drone resupply. *Transportation Science*, *54*, 229–249.

Dell'Amico, M., Montemanni, R., & Novellani, S. (2020). Matheuristic algorithms for the parallel drone scheduling traveling salesman problem. *Annals of Operations Research*, *289*, 211–226.

DHL (2016). Successful Trial Integration of DHL Parcelcopter into Logistics Chain. https://www.dhl.com/en/press/releases/releases_2016/all/parcel_ecommerce/successful_trial_integration_dhl_parcelcopter_logistics_chain.html. Accessed 12 August 2020.

de Freitas, J. C., & Penna, P. H. V. (2020). A variable neighborhood search for flying sidekick traveling salesman problem. *International Transactions in Operational Research*, *27*, 267–290.

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

Ha, Q. M., Deville, Y., Pham, Q. D., & Hà, M. H. (2020). A hybrid genetic algorithm for the traveling salesman problem with drone. *Journal of Heuristics*, *26*, 219–247.

Ham, A. M. (2018). Integrated scheduling of m-truck, m-drone, and m-depot constrained by time-window, drop-pickup, and m-visit using constraint programming. *Transportation Research Part C: Emerging Technologies*, *91*, 1–14.

Kara, I. (2011). Arc based integer programming formulations for the distance constrained vehicle routing problem. In *3rd IEEE International Symposium on Logistics and Industrial informatics* (pp. 33–38). IEEE.

Kara, I., Laporte, G., & Bektas, T. (2004). A note on the lifted miller–tucker–zemlin subtour elimination constraints for the capacitated vehicle routing problem. *European Journal of Operational Research*, *158*, 793–795.

Karak, A., & Abdelghany, K. (2019). The hybrid vehicle-drone routing problem for pick-up and delivery services. *Transportation Research Part C: Emerging Technologies*, *102*, 427 – 449.

Kim, E. (2016). The most staggering part about Amazon's upcoming drone delivery service. https://www.businessinsider.com/cost-savings-from-amazon-drone-deliveries-2016-6. Accessed 12 August 2020.

Kim, S., & Moon, I. (2018). Traveling salesman problem with a drone station. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *49*, 42–52.

Kuhn, M., & Johnson, K. (2013). *Applied predictive modeling* volume 26. Springer.

Lauryn, E. (2019). Air drop - your next delivery may arrive by drone. https://360.here.com/the-potential-impact-of-drones-on-the-supply-chain. Accessed 12 August 2020.

Ma, X. (2018). *Using classification and regression trees: A practical primer*. IAP.

Macrina, G., Pugliese, L. D. P., Guerriero, F., & Laporte, G. (2020). Drone-aided routing: A literature review. *Transportation Research Part C: Emerging Technologies*, *120*, 102762.

Mathew, N., Smith, S. L., & Waslander, S. L. (2015). Planning paths for package delivery in heterogeneous multirobot teams. *IEEE Transactions on Automation Science and Engineering*, *12*, 1298–1308.

Mbiadou Saleu, R. G., Deroussi, L., Feillet, D., Grangeon, N., & Quilliot, A. (2018). An iterative two-step heuristic for the parallel drone scheduling traveling salesman problem. *Networks*, *72*, 459–474.

Mester, D., & Bräysy, O. (2007). Active-guided evolution strategies for large-scale capacitated vehicle routing problems. *Computers & Operations Research*, *34*, 2964–2975.

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

Murray, C. C., & Raj, R. (2020). The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones. *Transportation Research Part C: Emerging Technologies*, *110*, 368–398.

Poikonen, S., & Golden, B. (2020a). The mothership and drone routing problem. *INFORMS Journal on Computing*, *32*, 249–262.

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

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

Ponza, A. (2016). *Optimization of Drone-assisted parcel delivery*. Master's thesis Università Niversita Di Padova. URL: http://tesi.cab.unipd.it/51947/1/Andrea_Ponza_-_Analisi_e_ottimizzazione_nell'uso_di_droni_per_la_consegna_di_prodotti.pdf. Accessed May 12th, 2021.

Reinelt, G. (1991). TSPLIB - a traveling salesman problem library. *ORSA Journal on Computing*, *3*, 376–384.

Sacramento, D., Pisinger, D., & Ropke, S. (2019). An adaptive large neighborhood search metaheuristic for the vehicle routing problem with drones. *Transportation Research Part C: Emerging Technologies*, *102*, 289–315.

Sanchez-Diaz, I., Palacios-Argüello, L., Levandi, A., Mardberg, J., & Basso, R. (2020). A time-efficiency study of medium-duty trucks delivering in urban environments. *Sustainability*, *12*, 425.

Savuran, H., & Karakaya, M. (2016). Efficient route planning for an unmanned air vehicle deployed on a moving carrier. *Soft Computing*, *20*, 2905–2920.

Schermer, D., Moeini, M., & Wendt, O. (2019). The traveling salesman drone station location problem. In *World congress on global optimization* (pp. 1129–1138). Springer.

Scikit-learn (2020). User Guide. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html. Accessed 12 August 2020.

Scott, J. E., & Scott, C. H. (2019). Models for drone delivery of medications and other healthcare items. In *Unmanned Aerial Vehicles: Breakthroughs in Research and Practice* (pp. 376–392). IGI Global.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, *35*, 254–265.

Toth, P., & Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, *15*, 333–346.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., & Subramanian, A. (2017). New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, *257*, 845–858.

UPS (2020). Weight and Size. https://www.ups.com/us/en/help-center/packaging-and-supplies/weight-size.page. Accessed 12 August 2020.

Wang, D., Hu, P., Du, J., Zhou, P., Deng, T., & Hu, M. (2019). Routing and scheduling for hybrid truck-drone collaborative parcel delivery with independent and truck-carried drones. *IEEE Internet of Things Journal*, *6*, 10483–10495.