Computers & Operations Research 192 (2026) 107477

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

A MILP and a genetic algorithm for the flying sidekick TSP with variable drone speeds

Federico Michelotto [a], Roberto Roberti [b,*]

[a] Department of Electrical, Electronic, and Information Engineering "Guglielmo Marconi", University of Bologna, Viale del Risorgimento 2, 40136 Bologna, Italy
[b] Department of Information Engineering, University of Padova, Via Gradenigo 6/B, 35131 Padova, Italy

ARTICLE INFO

Keywords:
TSP
Drone
Variable speed
MILP
Genetic algorithms
Exact methods

ABSTRACT

Thanks to technological advances, many last-mile delivery companies have launched projects that combine unmanned aerial vehicles, commonly known as drones, with classic ground vehicles to reduce delivery times and gas emissions and improve customer satisfaction. However, most drone-aided optimization problems in the literature assume a non-realistic scenario in which the drone speed is constant and the drone endurance is independent of its speed and payload. This work addresses this gap by considering a realistic non-linear non-convex drone power consumption model with variable drone speeds. We propose an exact and heuristic approach for the *Flying Sidekick Traveling Salesman Problem with Variable Drone Speeds* (FSTSP-VDS), an extension of the flying sidekick traveling salesman problem in which a truck is combined with a drone that can fly at variable speeds to deliver parcels to customers to minimize the makespan. If the drone energy-per-meter function is convex, the FSTSP-VDS can be formulated as a *Mixed Integer Linear Programming* (MILP) problem. We propose a MILP model that can solve to optimality instances with up to ten customers. We also show that, when a realistic drone energy model is used, fixing the drone speed can increase the average makespan by 32%. We also propose a genetic algorithm that outperforms the current state-of-the-art by 7% on average and finds optimal solutions for constant-speed instances with up to 99 customers.

1. Introduction

Over the past few years, there has been a growing interest in delivery systems that include *Unmanned Aerial Vehicles* (UAV) such as drones. For example, large delivery companies (e.g., UPS and Amazon) started several projects about using drones in last-mile delivery to reduce delivery times and gas emissions and improve customer satisfaction (see, e.g., Banker (2022)). For these reasons, the research community is studying different routing problems in which conventional vehicles are supported by one or more drones. The first problem of this category is the *Flying Sidekick Traveling Salesman Problem* (FSTSP) introduced by Murray and Chu (2015), an extension of the popular *Traveling Salesman Problem* (TSP) in which a truck is coupled with a drone to serve a set of customers while minimizing the makespan, i.e., the overall completion time.

A common assumption of most drone-aided routing problems, as in the FSTSP, is that the drone endurance is constant regardless of its speed and payload (see Macrina et al. (2020) and Liang and Luo (2022) for an overview). This hypothesis wrongly suggests that increasing the drone flying speed to the maximum allowable speed has only beneficial effects, e.g., increasing the drone range. In reality, because of the air resistance (drag), the power consumption is a non-linear function with respect to the vehicle speed. Therefore, when dealing with more realistic drone power consumption models, changing (not only reducing) the drone flying speed could increase the drone range and endurance and serve more customers with the drone, or serve the customers faster, reducing the makespan.

In this paper, we study the *Flying Sidekick Traveling Salesman Problem with Variable Drone Speeds* (FSTSP-VDS) proposed by Raj and Murray (2020) in which, unlike the FSTSP, the drone speed is not fixed a priori but can be selected from a continuous set for each arc traversed by the drone. However, Raj and Murray (2020) consider the general case with one or more drones, whereas we focus on the single-drone case. Our main scientific contributions are the following:

- we show that if the drone energy-per-meter function is convex with respect to the drone speed, the minimum and maximum travel time to launch the drone, serve a customer, and retrieve the drone at a rendezvous location, can be computed by solving a convex optimization problem;
- we present a compact *Mixed Integer Linear Programming* (MILP) formulation for the FSTSP-VDS under the condition that the drone

* Corresponding author.
E-mail addresses: federico.michelotto2@unibo.it (F. Michelotto), roberto.roberti@unipd.it (R. Roberti).

https://doi.org/10.1016/j.cor.2026.107477
Received 9 June 2025; Received in revised form 20 March 2026; Accepted 21 March 2026
Available online 25 March 2026

energy-per-meter function is convex. This formulation can solve instances with ten customers when cast within a general-purpose solver;
- we show that, when a realistic drone energy model is used, fixing the drone speed produces a substantial increase in the makespan up to 32% on average;
- we propose a genetic algorithm for the FSTSP-VDS that outperforms the state-of-the-art heuristic by a significant margin and when the drone speed is constant, finds optimal or near-optimal solutions (on average within 0.26% from the optimal ones) on instances with up to 99 customers (the largest instances tested).

The assumption of a convex drone energy-per-meter function is consistent with widely used drone energy models in the literature, such as those in Liu et al. (2017), Stolaroff et al. (2018) and Zhang et al. (2021), suggesting that our findings are applicable across a broad range of existing power consumption frameworks.

The remainder of the paper is organized as follows. Section 2 reviews the literature about drone-aided routing problems related to our work. Section 3 formally defines the FSTSP-VDS. Section 4 investigates the properties of convex energy-per-meter functions. Section 5 presents a compact MILP formulation for the FSTSP-VDS. Section 6 describes our genetic algorithm. Section 7 reports the computational results of the compact MILP formulation and the genetic algorithm. Section 8 discusses some conclusions.

2. Literature review

The recent growing interest of the research community in drone-aided routing problems does not allow a complete review of all scientific papers on this topic. Thus, we limit our review to papers about the FSTSP or the *Traveling Salesman Problem with Drone* (TSP-D). For an exhaustive review of drone-aided routing problems, we refer the reader to the surveys of Macrina et al. (2020) and Liang and Luo (2022). In the next sections, we summarize the relevant works on the FSTSP and the TSP-D, respectively, following the definitions of Murray and Chu (2015) for the FSTSP and Agatz et al. (2018) for the TSP-D, who first defined the two problems. The FSTSP is often referred to as the basic TSP-D.

2.1. The flying sidekick traveling salesman problem

The FSTSP is the first drone-aided routing problem studied in the literature (Murray and Chu, 2015). The FSTSP is an extension of the classical TSP in which a truck is coupled with a fixed-speed drone to serve a set of customers and return both vehicles to the depot in the minimum time, including potential waiting times related to the truck and drone synchronization. In particular, the drone can be launched from the truck at a customer location or the depot only, serve a customer, and then return to the truck at a different location, with the constraint that if the drone arrives before the truck at the rendezvous location, it cannot land and wait, but it must hover until the arrival of the truck, hence, consuming energy. Some customers may be unsuitable for drone delivery, for example, because the parcel is too heavy or because the customer location does not allow a safe landing, so some customers can only be served by truck. In addition, each node must be visited exactly once, and the time to launch and retrieve the drone and the time to deliver the packages must be considered.

Murray and Chu (2015) propose a MILP formulation and a heuristic for the FSTSP. Ha et al. (2018) propose a greedy randomized adaptive search procedure based on a procedure that optimally splits a TSP solution into an FSTSP solution, given the relative order between customers. Es Yurek and Ozmutlu (2018) propose an iterative decomposition-based approach. de Freitas and Penna (2020) propose a variable neighborhood search heuristic algorithm. Ha et al. (2020) propose a hybrid genetic heuristic. Several exact methods based on MILP formulations have been proposed, e.g., branch-and-bound (Dell'Amico et al. (2021a), Dell'Amico et al. (2021b), Dell'Amico et al. (2022)), column-and-row generation (Boccia et al., 2021), branch-and-price (Roberti and Ruthmair, 2021), branch-and-cut (Boccia et al., 2023) and column-generation-based (Blufstein et al., 2024).

2.2. The traveling salesman problem with drone

The TSP-D proposed by Agatz et al. (2018) differs from the FSTSP. Indeed, a node can be visited more than once, the drone can be launched and retrieved in the same location, the drone can land and wait, and drone launches, retrieval times, and parcel delivery times are neglected. Examples of heuristic approaches are those of Agatz et al. (2018), based on a split procedure similar to the one of Ha et al. (2018), and the variable neighborhood search of El-Adle et al. (2023). Exact approaches for the TSP-D are based on dynamic programming (Bouman et al., 2018), branch-and-bound ((Agatz et al., 2018), Poikonen et al. (2019)), branch-and-cut (Schermer et al., 2020), branch-and-price (Roberti and Ruthmair, 2021), and Benders decomposition (Vásquez et al., 2021).

2.3. Non-constant drone speed

In this section, we discuss the works related to drone-aided routing problems in which the drone speed is not fixed a priori. Raj and Murray (2020) introduce the FSTSP-VDS with multiple drones, an extension of the FSTSP in which drones are allowed to fly at different speeds, selected from a continuous set; the drone energy consumption is characterized by the model of Liu et al. (2017), and the authors propose a three-phase heuristic. Campuzano et al. (2023) introduce the drone-assisted variable speed asymmetric traveling salesman problem in which flight times depend not only on the payload weight but also on the drone speed and the weather conditions. The drone energy consumption is computed using the model of Liu et al. (2017), and the drone speed can be selected from a discrete set of three speed levels. The authors propose a MILP formulation and two heuristics based on variable neighborhood search and tabu search. Tamke and Buscher (2023) introduces the vehicle routing problem with drones and drone speed selection, a variant of the classical vehicle routing problem in which trucks are equipped with multiple drones to minimize the overall operational cost. The drone speeds are not fixed but have to be selected from a discrete set of five speed levels, and the drone energy consumption is modeled as in Stolaroff et al. (2018). The authors propose a MILP formulation.

To the best of our knowledge, existing literature lacks exact solution approaches for drone-aided routing problems that incorporate both non-linear drone energy consumption and continuous drone speed ranges.

3. Problem description

The FSTSP-VDS can be formally described as follows. A complete directed graph $\mathcal{G} = (V, \mathcal{A})$ is given. The vertex set V is defined as $V = \{0, 0'\} \cup N$, where both 0 and $0'$ represent a single depot (respectively the starting and final location of truck-and-drone route) and $N = \{1, 2, \dots, n\}$ a set of n customers to serve. A single truck, equipped with a single drone, is located at the depot. Each customer $i \in N$ must be served exactly once, either by the truck or the drone. The arc set $\mathcal{A}$ is defined as $\mathcal{A} = \{(i, j) \mid i, j \in N : i \neq j\} \cup \{(0, j) \mid j \in N\} \cup \{(i, 0') \mid i \in N\}$. In the following, we use the notation $N_0 = N \cup \{0\}$ and $N_0' = N \cup \{0'\}$.

A parcel of weight w_i must be delivered to customer $i \in N$. The drone can carry only one parcel at a time and has a payload capacity K, so parcels of weight greater than K must be delivered by the truck. The discrete set of payloads that can be carried by the drone is defined as $W = \{0\} \cup \{w_i \leq K, i \in N\}$. The drone can leave and return to

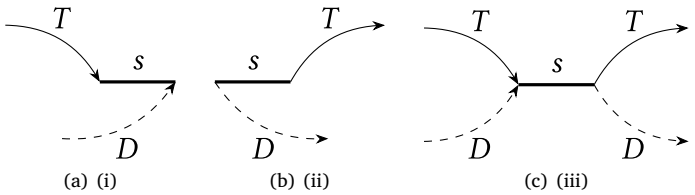

Fig. 1. Timelines of the three possible ways of serving a combined customer, where s denotes the delivery operation, T the truck arrival, and D the drone arrival: from left to right, before the truck and drone synchronization, after the drone launch, and after the truck and drone synchronization but before the drone launch.

Table 1
Coefficient values for the power consumption model of Liu et al. (2017).

Coefficient	k_1	k_2	c_1	c_2	c_4	c_5
Value	0.8554	0.3051	2.8037	0.3177	0.0296	0.0279
Unit	–	$\sqrt{\text{kg/m}}$	$\sqrt{\text{m/kg}}$	$\sqrt{\text{m/kg}}$	kg/m	N s/m

the truck at customer locations or the depot only, but it cannot be launched and retrieved at the same node. While the drone is flying or serving a customer, the truck can serve other customers. The drone can traverse different arcs at different flying speeds, but the flying speed must remain constant during the flight and be in the interval $[v^D_{min}, v^D_{max}]$. The distance that the truck (drone, resp.) must cover to traverse arc $(i,j) \in \mathcal{A}$ is indicated with d^T_{ij} (d^D_{ij}, resp.) – typically, $d^T_{ij} \neq d^D_{ij}$ as the truck and the drone travel on different pathways. The time for the truck to traverse arc $(i,j) \in \mathcal{A}$ is indicated with t^T_{ij}. The minimum and maximum feasible drone travel times to takeoff from node $i \in N_0$, deliver the parcel at node $j \in N$, and land at node $k \in N_0'$ (*drone leg* in the following), including potential hovering time over the landing node k, are denoted respectively with t^D_{ijk} and T^D_{ijk}, and are set to $+\infty$ in case of infeasibility. The set of all feasible drone legs is defined by $Z = \{(i,j,k) \mid i \in N_0, j \in N, k \in N_0' : t^D_{ijk} \in \mathbb{R}\}$.

The drone has a fixed battery capacity of B Joules, and the drone endurance, expressed in seconds, is a function of the parcel weight and the drone's speed. A fully charged battery is installed in the drone before every launch. The parameter h, expressed in meters, defines the drone cruise altitude. Drone takeoff and landing operations consist of vertical flights at a constant speed, respectively equal to v^D_t and v^D_l. We refer the reader to Section 3.1 for a complete description of the drone power consumption model employed in this work. The drone must rejoin the truck at a landing location before it runs out of battery. If the drone arrives before the truck at the rendezvous location, the drone cannot land and wait, but it must hover above the rendezvous location until the truck arrives, consuming energy. Vice versa, if the truck arrives before the drone, the truck must wait for the drone. Before the takeoff, the drone must be prepared, and this operation takes s_L seconds. When the drone lands at the rendezvous location, it takes s_R seconds to retrieve the drone. Hence, for each drone takeoff and landing, the additional time required to prepare and retrieve the drone must be considered. The time to deliver a parcel, once the vehicle arrives at a customer location $i \in N$, equals σ^T_i seconds for the truck and σ^D_i seconds for the drone. Without loss of generality, we define $\sigma^T_0 = \sigma^T_{0'} = \sigma^D_0 = \sigma^D_{0'} = 0$. Parcel delivery at rendezvous locations can occur before or after the truck and drone synchronization. Similarly, parcel delivery at takeoff locations can occur before or after the drone launch. Hence, if a location is both a rendezvous and a takeoff location, the parcel delivery can be done in three different ways: (i) before the truck and drone synchronization, (ii) after the drone launch, or (iii) after the truck and drone synchronization but before the drone launch. An illustrative example of the three possible delivery scenarios is depicted in Fig. 1. In all cases, customers visited by the truck and the drone are always served by the former.

The goal of the FSTSP-VDS is to find a truck-drone tour of minimum makespan serving all customers either by truck or drone, taking into account potential waiting times due to the synchronization between the two vehicles.

In the following, we will use the following definitions. A *truck customer* (*drone customer*, resp.) is a customer visited by the truck (drone, resp.) alone. On the other hand, if a customer is visited by both the truck and the drone, we call it *combined customer*. A *truck arc* (*drone arc*, resp.) is traversed by the truck (drone, resp.) alone. A *combined arc* is traversed by the truck while the drone is onboard. A *truck leg* is a concatenation of truck arcs traversed by the truck alone in between two consecutive combined customers. A *drone leg* is a concatenation of exactly two consecutive drone arcs traversed by the drone alone between two consecutive combined customers. A *combined leg* is a concatenation of combined arcs traversed by the truck while the drone is onboard that consists of combined customers only. An *operation* consists of a truck leg and a drone leg in between the same pair of combined customers. Notice that a FSTSP-VDS solution is a concatenation of operations and combined legs.

3.1. Drone endurance

In this work, the drone endurance is characterized by the power consumption model of Liu et al. (2017), developed for small multi-rotor unmanned aircraft systems — the same model employed in the work of Raj and Murray (2020). The model characterizes the drone power consumption with respect to speed v (in m/s) and payload weight w (in kg), during the three flight stages of the drone, i.e., during the vertical takeoff or landing (1a), the horizontal cruising (1b), and the stationary hovering (1c):

$$P^{TL}(v,w) = k_1(D+w)g\left[\frac{v}{2} + \sqrt{\left(\frac{v}{2}\right)^2 + \frac{(D+w)g}{k_2^2}}\right] + c_2\left((D+w)\,g\right)^{3/2} \tag{1a}$$

$$P^{C}(v,w) = (c_1+c_2)\left[\left((D+w)g - c_5\,(v\cos\theta)^2\right)^2 + \left(c_4 v^2\right)^2\right]^{3/4} + c_4 v^3 \tag{1b}$$

$$P^{H}(w) = (c_1+c_2)\left((D+w)\,g\right)^{3/2} \tag{1c}$$

The coefficients k_1, k_2, c_1, c_2, c_4, c_5 are the parameters of the power consumption model of Liu et al. (2017), and their values can be found in Table 1. The other parameters, D and θ, define the physical characteristics of the drone. In particular, D is the drone frame weight, which is assumed to be 1.5 kg, and θ is the angle of attack of the drone, which equals 10 degrees. Finally, g is the gravitational acceleration (9.8 m/s^2).

The Liu et al. (2017) drone power consumption during horizontal cruising and the corresponding drone range, with respect to different speeds and parcel weights, are illustrated in Figs. 2(i) and 2(ii). Fig. 2(i) shows that the drone power consumption $P^C(v,w)$ is non-linear and non-convex; indeed, consider the speeds 1.0 m/s and 6.0 m/s and a payload of 0.0 kg, we have $P^C(\frac{1}{2}(1.0+6.0), 0.0) = 171.36\ W > \frac{1}{2}(P^C(1.0,0.0) + P^C(6.0,0.0)) = 170.66\ W$.

However, since the FSTSP-VDS requires a constant drone flying speed during each flight, we are interested in studying how much energy is consumed by the drone to travel horizontally one meter at a

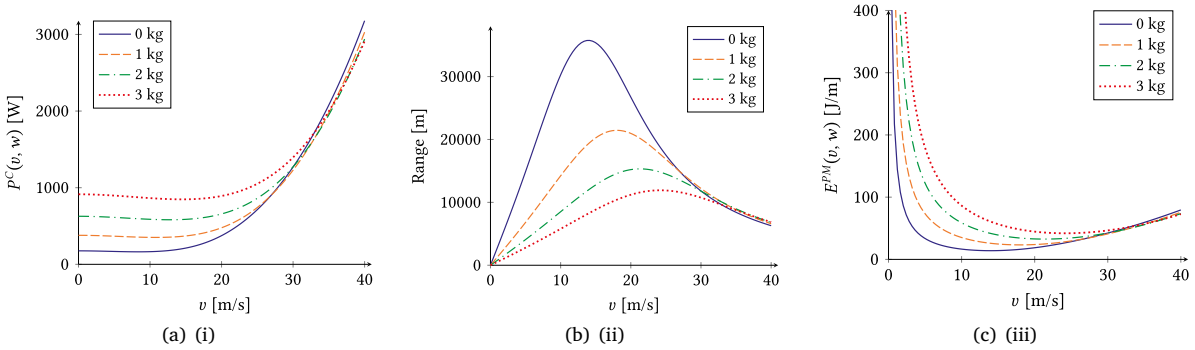

Fig. 2. Drone endurance features for different payload weights w (Liu et al., 2017): (i) drone power consumption w.r.t. the drone speed v; (ii) drone range in meters w.r.t. the drone speed v given a battery capacity of 500 kJ; (iii) energy-per-meter function $E^{PM}(v, w)$ w.r.t. the drone speed v.

given speed – energy-per-meter in the following – and how this quantity varies with respect to the drone speed. Fig. 2(iii) shows the drone energy-per-meter consumption, defined as $E^{PM}(v, w) = (1/v)P^C(v, w)$, with respect to speed v for different payload weights w.

In Sections 4 and 5, we show that, if the energy-per-meter function $E^{PM}(v, w)$ is convex for each parcel weight $w \in W$, the FSTSP-VDS can be formulated as a compact MILP.

3.2. Example

Fig. 3 illustrates a solution of a FSTSP-VDS instance with five customers (the circles) and the depot, that is, nodes 0 and $0'$ (the rectangle). In the example, truck arcs (drone arcs, resp.) are depicted by solid (dashed, resp.) lines and combined arcs by double lines. Initially, the truck moves with the drone onboard towards customer 3, taking time t^T_{03}. Then, the drone is launched to serve customer 2 while the truck moves towards customer 1. After serving customer 2, the drone moves towards customer 1 and rejoins with the truck. The drone leg travel time must be between t^D_{321} and T^D_{321}. The truck leg travel time $t^T_{3\to1}$ depends on when customers 3 and 1 are served. In particular, if the parcel delivery to customer 3 occurs before the drone launch and customer 1 is served after the drone and truck synchronization, $t^T_{3\to1}$ equals t^T_{31}. If instead customer 3 is served after the drone launch, σ^T_3 must be added to t^T_{31}. Moreover, if customer 1 is served before the drone and truck synchronization, σ^T_1 must also be added. After the truck and the drone synchronization, the drone is launched to serve customer 5 while the truck serves customer 4. Finally, the truck and drone return to the depot and rejoin. As before, the drone leg travel time must be between $t^D_{150'}$ and $T^D_{150'}$, and the truck leg travel time $t^T_{1\to0'}$ depends on when customer 1 is served. If customer 1 is served before the drone launch, $t^T_{1\to0'}$ equals $t^T_{14} + \sigma^T_4 + t^T_{40'}$. Otherwise, if customer 1 is served after the drone launch, $t^T_{1\to0'}$ equals $\sigma^T_1 + t^T_{14} + \sigma^T_4 + t^T_{40'}$.

Below we list the six possible makespans depending on when the combined customers 3 and 1 are served. In particular, T_1, T_2, T_3 (T_4, T_5, T_6, resp.) correspond to the makespans when customer 3 is served before (after) the drone launch, and customer 1 is served respectively, after synchronization but before the drone launch, before synchronization, and after the drone launch:

$$T_1 = t^T_{03} + \sigma^T_3 + \max\{t^D_{321}, t^T_{31}\} + \sigma^T_1 + \max\{t^D_{150'}, t^T_{14} + \sigma^T_4 + t^T_{40'}\} + 2(s_L + s_R)$$
$$T_2 = t^T_{03} + \sigma^T_3 + \max\{t^D_{321}, t^T_{31} + \sigma^T_1\} + \max\{t^D_{150'}, t^T_{14} + \sigma^T_4 + t^T_{40'}\} + 2(s_L + s_R)$$
$$T_3 = t^T_{03} + \sigma^T_3 + \max\{t^D_{321}, t^T_{31}\} + \max\{t^D_{150'}, \sigma^T_1 + t^T_{14} + \sigma^T_4 + t^T_{40'}\} + 2(s_L + s_R)$$
$$T_4 = t^T_{03} + \max\{t^D_{321}, \sigma^T_3 + t^T_{31}\} + \sigma^T_1 + \max\{t^D_{150'}, t^T_{14} + \sigma^T_4 + t^T_{40'}\} + 2(s_L + s_R)$$
$$T_5 = t^T_{03} + \max\{t^D_{321}, \sigma^T_3 + t^T_{31} + \sigma^T_1\} + \max\{t^D_{150'}, t^T_{14} + \sigma^T_4 + t^T_{40'}\} + 2(s_L + s_R)$$
$$T_6 = t^T_{03} + \max\{t^D_{321}, \sigma^T_3 + t^T_{31}\} + \max\{t^D_{150'}, \sigma^T_1 + t^T_{14} + \sigma^T_4 + t^T_{40'}\} + 2(s_L + s_R)$$

Serving a combined customer with the drone on the ground is never more convenient than serving it while the drone is airborne, but when the drone's endurance is finite, the latter case could be infeasible, whereas the former case may still be feasible.

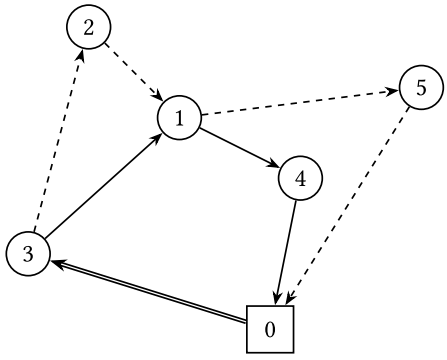

Fig. 3. A solution of a FSTSP-VDS instance with five customers.

4. Drone travel times computation

In this section, we show that if the energy-per-meter function is convex for each $w \in W$, the set of feasible drone leg travel times of each drone leg $(i, j, k) \in Z$ is also convex. Thereby, we can forget about drone speeds and bound the minimum and maximum feasible drone leg travel times; this allows us to formulate the FSTSP-VDS as a MILP. Then, we present two convex optimization problems to compute, respectively, the minimum and the maximum drone leg travel times of a generic drone leg $(i, j, k) \in Z$, respectively, t^D_{ijk} and T^D_{ijk}. From now on, we assume that the energy-per-meter function is convex for each $w \in W$. For the sake of clarity, we ignore w.l.o.g. the constant quantities related to the takeoff and landing operations (s_L and s_R), and the time required to serve the drone customer (σ^T_j).

Let us introduce the following set of constraints:

$$E^{PM}(v^D_{ij}, w_j)\, d^D_{ij} + E^{PM}(v^D_{jk}, 0)\, d^D_{jk} + P^H(0)\, t_h \le B \tag{2a}$$
$$t_h \ge 0 \tag{2b}$$
$$v^D_{ij}, v^D_{jk} \in [v^D_{min}, v^D_{max}] \tag{2c}$$

Constraint (2a) imposes that the drone energy consumption to traverse the drone leg (i, j, k) at speeds v^D_{ij} and v^D_{jk} plus the energy required to hover for a time t_h seconds must not exceed the battery capacity of B Joules. Constraints (2b) and (2c) model the feasible range of the variables v^D_{ij}, v^D_{jk}, and t_h. Since the left-hand side of (2a) is a sum of convex functions, the feasible region of (2) is convex.

Therefore, to compute t^D_{ijk} one can solve the convex optimization problem corresponding to minimizing $d^D_{ij}/v^D_{ij} + d^D_{jk}/v^D_{jk} + t_h$ subject to

(2). Furthermore, since the feasible region of (2) is convex, the set of feasible drone leg travel times of the above convex optimization problem is also convex. To compute T^D_{ijk}, we propose the following convex optimization problem. The total energy consumed by the drone to travel horizontally d meters in t seconds with a payload of w kg, $E^d(t,w)$ in the following, can be expressed as follows:

$$E^d(t,w) = t\,P^C(d/t,w) = d\,(t/d)\,P^C(d/t,w) = d\,E^{PM}(d/t,w) \tag{3}$$

Now observe that the function $g(x) = \frac{1}{x}, x > 0$ is a strict order reversing map (or monotone decreasing function), i.e., $\forall x_1, x_2 > 0 : x_1 < x_2 \implies g(x_1) > g(x_2)$. Therefore, if $E^{PM}(x,w)$ is convex in the interval $x \in [v^D_{min}, v^D_{max}]$ for any $w \in W$, $E^{PM}(d/x,w)$ is convex in the interval $x \in [d/v^D_{max}, d/v^D_{min}]$ for any $w \in W$. Consequently, $E^d(x,w)$ is convex in the interval $x \in [d/v^D_{max}, d/v^D_{min}]$ for any $w \in W$. From the above observations, we define the following convex optimization problem to compute T^D_{ijk}:

$$\max\ t^D_{ij} + t^D_{jk} + t_h \tag{4a}$$

$$E^{d^D_{ij}}(t^D_{ij}, w_j) + E^{d^D_{jk}}(t^D_{jk}, 0) + t_h\,P^H(0) \le B \tag{4b}$$

$$t^D_{ij} \in [d^D_{ij}/v^D_{max}, d^D_{ij}/v^D_{min}] \tag{4c}$$

$$t^D_{jk} \in [d^D_{jk}/v^D_{max}, d^D_{jk}/v^D_{min}] \tag{4d}$$

$$t_h \ge 0 \tag{4e}$$

The objective function (4a) aims at maximizing the sum of the travel times to traverse the arcs (i,j) and (j,k), in t^D_{ij} and t^D_{jk} seconds respectively, plus the potential hovering time of t_h seconds over node k. Constraint (4b) ensures that the energy consumption is not greater than B Joules. Constraints (4c)–(4e) model the range of the decision variables t^D_{ij}, t^D_{jk} and t_h.

Observe that in general, a drone leg $(i,j,k) \in Z$ can be completed in exactly T seconds (considering the potential hovering above the rendezvous node) by more than one drone speed configuration (v^D_{ij}, v^D_{jk}). To this extent, we propose the following convex optimization problem to find a drone speed configuration that minimizes the energy consumption to traverse a drone leg $(i,j,k) \in Z$ in a target travel time of exactly T seconds:

$$\min\ E^{d^D_{ij}}(t^D_{ij}, w_j) + E^{d^D_{jk}}(t^D_{jk}, w_j) + (T - t^D_{ij} - t^D_{jk})P^H(0) \tag{5a}$$

$$t^D_{ij} + t^D_{jk} \le T \tag{5b}$$

$$t^D_{ij} \ge d^D_{ij}/v^D_{max} \tag{5c}$$

$$t^D_{jk} \ge d^D_{jk}/v^D_{max} \tag{5d}$$

The objective function (5a) aims at minimizing the energy to traverse the arcs (i,j) and (j,k), in t^D_{ij} and t^D_{jk} seconds respectively, plus the energy required to hover over node k for $T - t^D_{ij} - t^D_{jk}$ seconds. Constraint (5b) ensures that the travel time to arrive in k must not be greater than T. Constraints (5c) and (5d) define the range of the decision variables t^D_{ij} and t^D_{jk}. Finally, observe that formulation (5) can be used to compute the convex set $[t^D_{ijk}, T^D_{ijk}]$ with the bisection method. The models outlined in this section allow determining in pre-processing a range $[t^D_{ijk}, T^D_{ijk}]$ of feasible drone leg travel times for each feasible drone leg $(i,j,k) \in Z$. In Section 5, we show how such ranges can be used to build a MILP model for the FSTSP-VDS when the drone energy-per-meter function is convex.

5. Compact formulation

In this section, we describe a compact formulation for the FSTSP-VDS, under the assumption that the energy-per-meter is convex. The resulting formulation is compact (i.e., polynomial number of constraints and variables) and can be solved with a generic MILP solver such as CPLEX or GuRoBi.

Let $x^T_{ij} \in \{0,1\}$ be a binary variable equal to 1 if the truck traverses the arc $(i,j) \in \mathcal{A}$, and let $x^D_{ij} \in \{0,1\}$ be a binary variable equal to 1 if the drone traverses the arc $(i,j) \in \mathcal{A}$ (no matter if it is onboard truck or airborne). Let $y^T_i, y^D_i, y^C_i \in \{0,1\}$ be three binary variables equal to 1 if $i \in N$ is a truck, drone, or combined customer, respectively. Let $z_{ijk} \in \{0,1\}$ be a binary variable equal to 1 if the drone traverses the drone leg $(i,j,k) \in Z$.

Let $\alpha_i \in \{0,1\}$ be a binary variable equal to 1 if the truck delivery at node $i \in V$ occurs right after the truck arrival and before the truck and drone synchronization if i is a rendezvous node (case 1(i)). Let $\beta_i \in \{0,1\}$ be a binary variable equal to 1 if the truck delivery at node $i \in V$ occurs after the drone launch – an operation must start from i – (case 1(ii)). Let $\gamma_i \in \{0,1\}$ be a binary variable equal to 1 if the truck delivery at node $i \in V$ occurs after the synchronization between the truck and drone and before the drone launch if i is a takeoff node – an operation must end in i – (case 1(iii)). Finally, let $a_i \in \mathbb{R}_+$ be a real variable denoting the latest arrival time between the truck and drone at node $i \in V$, including the time to serve i if $\alpha_i = 1$, without considering the time required to launch and retrieve the drone, and the parcel deliveries occurred in between successive operations ($\gamma_i = 1$). If i is a drone customer, a_i is not defined.

For clarity, we summarize here the model parameters introduced in Section 3: the truck travel time t^T_{ij} to traverse arc $(i,j) \in \mathcal{A}$; the minimum and maximum drone leg travel times t^D_{ijk}, T^D_{ijk}, for each feasible drone leg $(i,j,k) \in Z$ (including takeoff at node i, landing at node j, serving customer j, takeoff at node j and landing at node k); the minimum and maximum drone flying speed v^D_{min}, v^D_{max}; the time required to launch and retrieve the drone s_L, s_R; and the time required to serve a customer $i \in N$ with the truck and with the drone σ^T_i, σ^D_i. Finally, the parameter M (set equal to the minimum truck-only makespan) is used to disable a constraint if a binary decision variable is not triggered. The FSTSP-VDS can be formulated as follows:

$$t^* = \min\quad a_{0'} + (s_L + s_R)\sum_{i\in N} y^D_i + \sigma^T_i \sum_{i\in N}\gamma_i, \tag{6a}$$

$$\text{s.t.} \sum_{(i,j)\in\mathcal{A}} x^T_{ij} = \sum_{(j,i)\in\mathcal{A}} x^T_{ji} = y^T_i + y^C_i \qquad i \in N \tag{6b}$$

$$\sum_{(0,j)\in\mathcal{A}} x^T_{0j} = \sum_{(i,0')\in\mathcal{A}} x^T_{i0'} = 1 \tag{6c}$$

$$\sum_{(i,j)\in\mathcal{A}} x^D_{ij} = \sum_{(j,i)\in\mathcal{A}} x^D_{ji} = y^D_i + y^C_i \qquad i \in N \tag{6d}$$

$$\sum_{(0,j)\in\mathcal{A}} x^D_{0j} = \sum_{(i,0')\in\mathcal{A}} x^D_{i0'} = 1 \tag{6e}$$

$$y^T_i + y^D_i + y^C_i = 1 \qquad i \in N \tag{6f}$$

$$y^C_i = \alpha_i + \beta_i + \gamma_i \qquad i \in N \tag{6g}$$

$$\beta_i \le \sum_{(i,j,k)\in Z} z_{ijk} \qquad i \in N_0 \tag{6h}$$

$$\gamma_k \le \sum_{(i,j,k)\in Z} z_{ijk} \qquad k \in N'_0 \tag{6i}$$

$$a_i + t^T_{ij} + \sigma^T_i\beta_i + \sigma^T_j(\alpha_j + y^T_j) \le a_j + M(1 - x^T_{ij}) \qquad (i,j) \in \mathcal{A} \tag{6j}$$

$$a_i + \sum_{(i,j,k)\in Z} (t^D_{ijk} + M)z_{ijk} \le a_k + M \qquad i \in N_0, k \in N'_0 \tag{6k}$$

$$a_i + \sum_{(i,j,k)\in Z} (T^D_{ijk} - M) z_{ijk} \geq a_k - M \qquad i \in N_0, k \in N'_0 \tag{6l}$$

$$\sum_{(i,j,k)\in Z} z_{ijk} = y^D_j \qquad j \in N \tag{6m}$$

$$\sum_{(k,i,j)\in Z} z_{kij} + \sum_{(i,j,k)\in Z} z_{ijk} \leq x^D_{ij} \qquad (i,j) \in \mathcal{A} \tag{6n}$$

$$M(1 - x^D_{ij}) + a_j \geq a_i + y^D_j \left(\frac{d^D_{ij}}{v^D_{max}} \right) \qquad (i,j) \in \mathcal{A} \tag{6o}$$

$$x^T_{ij} \geq x^D_{ij} + y^C_i + y^C_j - 2 \qquad (i,j) \in \mathcal{A} \tag{6p}$$

$$x^T_{ij}, x^D_{ij} \in \{0,1\} \qquad (i,j) \in \mathcal{A} \tag{6q}$$

$$y^T_i, y^D_i, y^C_i \in \{0,1\} \qquad i \in N \tag{6r}$$

$$z_{ijk} \in \{0,1\} \qquad (i,j,k) \in Z \tag{6s}$$

$$\alpha_i, \beta_i, \gamma_i \in \{0,1\} \qquad i \in V \tag{6t}$$

$$a_i \in \mathbb{R}_+ \qquad i \in V \tag{6u}$$

The objective function (6a) aims at minimizing the total tour duration to serve all customers and return to the depot. Precisely, the objective function is equal to the variable $a_{0'}$, plus the time required to launch and retrieve the drone $s_L + s_R$ multiplied by the number of drone legs in solution $\sum_{i\in N} y^D_i$, plus the time required to serve the customers in between successive operations $\sigma^T_i \sum_{i\in N} \gamma_i$, as described by case 1(iii). Constraints (6b) are flow conservation constraints for the truck and link the x^T_{ij} variables with the variables y^T_i and y^C_i. Constraints (6c) ensure that the truck leaves and returns to the depot exactly once. Constraints (6d)–(6e) are equivalent to constraints (6b)–(6c) but apply to the drone. Constraints (6f) ensure that each customer is a drone, truck, or combined customer and is visited exactly once. Constraints (6g) define how each combined customer is served. Constraints (6h) and (6i) ensure the compatibility between service types and operations. Constraints (6j) act as subtour elimination constraints for the truck and set the truck arrival time at each node. Constraints (6k) and (6l) set the minimum and maximum drone travel times for each drone leg $(i,j,k) \in Z$. Constraints (6m) link the variables z_{ijk} with the variables y^D_j. Constraints (6n) impose that a drone leg can contain arc $(i,j) \in \mathcal{A}$ only if arc (i,j) is traversed by the drone. Constraints (6o) set a lower bound on the arrival time for the customers served by the drone. Constraints (6p) ensure that an arc traversed only by the drone must be part of a drone leg. Constraints (6q)–(6u) model the range of the decision variables.

The computational time required to solve formulation (6) can be significantly reduced by adding the following valid inequality:

$$a_{0'} \geq \sigma^T_0 + \sum_{(i,j)\in\mathcal{A}} (t^T_{ij} + \sigma^T_j) x^T_{ij} \tag{7}$$

Constraint (7) specifies that $a_{0'}$ cannot be lower than the sum of the truck travel times of the arcs traversed by the truck plus the truck delivery times of all customers visited by the truck.

6. Genetic algorithm

The genetic algorithm we propose for the FSTSP-VDS is inspired by the Hybrid Genetic Search with Adaptive Diversity Control (HGSADC) metaheuristic of Vidal et al. (2012), one of the most effective heuristics for the Capacitated Vehicle Routing Problem. Moreover, genetic algorithms have demonstrated good performance on problems similar to the FSTSP-VDS, see for example Ha et al. (2020) and Chen et al. (2025).

The high-level scheme of our genetic algorithm is shown in Algorithm 1.

Algorithm 1: Genetic Algorithm(A^η, A^γ, A^{epochs})

Input:
A^η: population size
A^γ: number of offspring at each iteration
A^{epochs}: max number of iterations

```
Generate a random population of A^η individuals;
Evaluate the biased fitness of all individuals in the population;
while iterations < A^epochs do
    for i = 1 to A^γ do
        Select two parent individuals P1 and P2 according to their biased fitness;
        Generate offspring C from P1 and P2 (crossover);
        Apply local search procedures to C (education);
    Evaluate the biased fitness of all individuals in the population;
    Replace A^γ individuals in the population with the new offspring;
return the best individual;
```

The method initially defines a random population of A^η individuals, each of which encodes one or more FSTSP-VDS solutions. At each iteration, A^γ new individuals (offspring) are created by combining two parent individuals together (crossover), selected according to their *biased fitness* (individual's cost adjusted by its diversity contribution, see Section 6.3). Then, a local search phase is applied (education). At the end of each iteration, the fitness of all individuals is updated, and A^γ individuals that belong to the current population are replaced with the new individuals. Finally, after A^{epochs} iterations, the procedure returns the best individual found.

Section 6.1 describes how individuals encode solutions. Section 6.2 describes how the best solution associated with an individual is retrieved. Section 6.3 defines how individuals are evaluated with respect to the entire population. Section 6.4 describes how parents are selected and the crossover procedure to generate a new individual. Section 6.5 describes the local search procedures applied during the education phase. Finally, Section 6.6 describes how the population is managed during the evolution process.

6.1. Individual encoding

Each FSTSP-VDS solution can be characterized (possibly, not uniquely) by a permutation of the customers set N representing the precedence relationship among customers. Similarly to what is done by Ha et al. (2018) and Agatz et al. (2018), we define the following encoding rule. Let p be a permutation of N. A solution is encoded by p if, for each $i,j \in N$ such that i is visited before j with the same vehicle, i occurs before j in p. From now on, we refer to a chromosome sequence as a sequence with prefix 0 followed by a permutation p and $0'$. An example of this encoding is illustrated in Fig. 4, which shows the five solutions encoded by the chromosome sequence $(0,1,2,0')$.

Note that, since the precedence relationship among customers visited by the same vehicle is determined by the permutation sequence, there is a bijection between the set of operations and the set of drone legs, that is, an operation is uniquely determined by a drone leg.

The advantage of such encoding relies on the fact that, as we will show in Section 6.2, we can retrieve the best solution among all the solutions encoded by a permutation sequence in time $\Theta(|N|^3)$. Furthermore, we also propose an effective linear approximated method

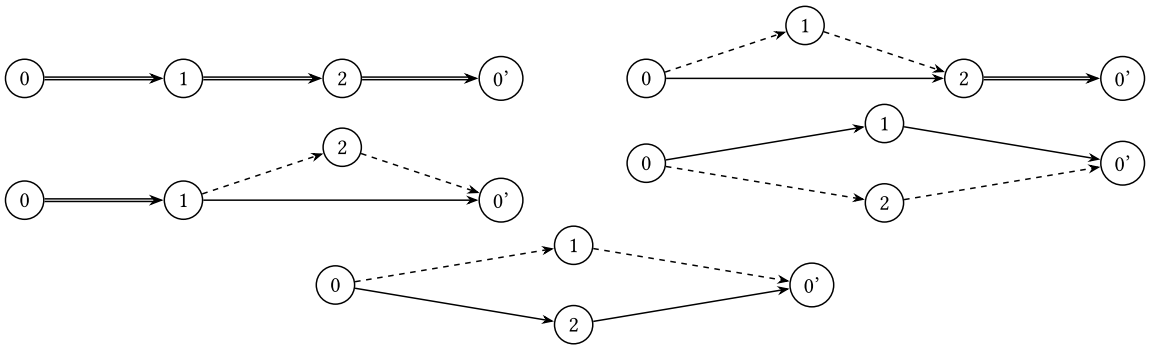

Fig. 4. The five solutions encoded by the chromosome sequence $(0, 1, 2, 0')$.

Table 2
Notation summary for π_ρ.

Notation	Start node ρ_i	End node ρ_k	Description
$\pi_\rho^{u,u}(i,k)$	Unserved	Unserved	Minimum time to go from ρ_i (not yet served) to ρ_k without serving ρ_k
$\pi_\rho^{u,s}(i,k)$	Unserved	Served	Minimum time to go from ρ_i (not yet served) to ρ_k and serve ρ_k
$\pi_\rho^{s,u}(i,k)$	Served	Unserved	Minimum time to go from ρ_i (already served) to ρ_k without serving ρ_k
$\pi_\rho^{s,s}(i,k)$	Served	Served	Minimum time to go from ρ_i (already served) to ρ_k and serve ρ_k

that runs in time $\Theta((A^b)^2|N|)$, where A^b is a parameter that sets the maximum truck leg length allowed, to limit the computational time of this procedure when dealing with large instances.

6.2. Chromosome evaluation

In this section, we describe a dynamic programming algorithm, named *split*, that, given a chromosome sequence, computes a minimum makespan solution among all the solutions encoded by the chromosome.

Let ρ be a chromosome such that $\rho = (\rho_0 = 0, \rho_1, \dots, \rho_n, \rho_{n+1} = 0')$. Let $f_k(\rho)$ and $g_k(\rho)$ denote respectively the minimum time to reach ρ_k in ρ without serving it and the minimum time to reach and serve ρ_k, under the precedence relationship defined by the chromosome sequence ρ, when ρ_k is visited by both the truck and drone.

We denote with $\pi_\rho^{u,u}(i,k)$ the minimum time to go from ρ_i not yet served, to ρ_k without serving it, either with a combined arc if $k = i+1$ or with an operation that starts from ρ_i and ends in ρ_k if $k > i+1$. The definitions of $\pi_\rho^{u,s}(i,k)$, $\pi_\rho^{s,u}(i,k)$ and $\pi_\rho^{s,s}(i,k)$ follow by analogy, where the first superscript is equal to s if the starting node has been already served, u otherwise, and the second superscript is equal to s if the ending node must be served, u otherwise. Table 2 summarizes the notation for the above definitions.

It is easy to see that $f_k(\rho)$ and $g_k(\rho)$ satisfy the Bellman Eqs. (8a) and (8b). The minimum time to reach ρ_k without serving it (8a), can be computed as the minimum time to go from ρ_i plus the minimum time to go from ρ_i to ρ_k, for each $i < k$, both when ρ_i has been already served $(g_i(\rho) + \pi_\rho^{s,u}(i,k))$ and when ρ_i has not yet been served $(f_i(\rho) + \pi_\rho^{u,u}(i,k))$. The same reasoning applies to compute the minimum time to reach and serve ρ_k(8b), whereas, instead of $\pi_\rho^{u,u}(i,k)$ and $\pi_\rho^{s,u}(i,k)$, we have respectively $\pi_\rho^{u,s}(i,k)$ and $\pi_\rho^{s,s}(i,k)$.

$$f_k(\rho) = \begin{cases} 0 & \text{if } k = 0 \\ \min_{0 \le i < k} \left\{ \min \left\{ f_i(\rho) + \pi_\rho^{u,u}(i,k);\ g_i(\rho) + \pi_\rho^{s,u}(i,k) \right\} \right\} & \text{if } k \ge 1 \end{cases} \tag{8a}$$

$$g_k(\rho) = \begin{cases} \sigma_{i_0}^T & \text{if } k = 0 \\ \min_{0 \le i < k} \left\{ \min \left\{ f_i(\rho) + \pi_\rho^{u,s}(i,k),\ g_i(\rho) + \pi_\rho^{s,s}(i,k) \right\} \right\} & \text{if } k \ge 1 \end{cases} \tag{8b}$$

Algorithm 2 shows the pseudocode of the dynamic programming algorithm that can be derived from the recurrence Eqs. (8a) and (8b) to compute a minimum makespan solution among all the solutions encoded by a given chromosome sequence. The algorithm begins by initializing $f_i(\rho)$ and $g_i(\rho), 0 \le i < |\rho| - 1$, as the truck arrival times associated with the truck-only solution. Then, starting from $i = 0$ to $i = |\rho| - 2$, all the minimum makespan operations starting from ρ_i and ending in $\rho_k, k > i$ are evaluated, and the quantities $f_k(\rho)$ and $g_k(\rho)$ are updated accordingly. The correctness of the algorithm relies on the fact that at the beginning of each iteration i, the sub-problems referenced by $f_i(\rho)$ and $g_i(\rho)$ have already been solved.

Algorithm 2: *split*(ρ)

Input:
$\rho = (\rho_0, \rho_1, \dots, \rho_{|\rho|-1})$: chromosome sequence
`//Initialization (truck-only solution)`
$f_0(\rho) \leftarrow 0.0$
$g_0(\rho) \leftarrow \sigma_0^T$
for $i = 1$ **to** $|\rho| - 1$ **do**
  $f_i(\rho) \leftarrow g_{i-1}(\rho) + t^T_{i_{i-1}, i_i}$
  $g_i(\rho) \leftarrow f_i(\rho) + \sigma^T_{\rho_i}$
`//Recursive Phase`
for $i = 0$ **to** $|\rho| - 2$ **do**
  for $k = i + 1$ **to** $|\rho| - 1$ **do**
    compute $\pi_\rho^{u,u}(i,k), \pi_\rho^{u,s}(i,k), \pi_\rho^{s,u}(i,k), \pi_\rho^{s,s}(i,k)$
    $f_k(\rho) \leftarrow \min\{f_k(\rho), f_i(\rho) + \pi_\rho^{u,u}(i,k), g_i(\rho) + \pi_\rho^{s,u}(i,k)\}$
    $g_k(\rho) \leftarrow \min\{g_k(\rho), f_i(\rho) + \pi_\rho^{u,s}(i,k), g_i(\rho) + \pi_\rho^{s,s}(i,k)\}$
best_sol $\leftarrow$ backtrack optimal solution
return $g_{|\rho|-1}(\rho)$, *best_sol*

The initialization phase (Lines 1 to 5) requires $\Theta(|\rho|)$ operations. Line 8 requires $\Theta(|\rho|)$ operations, since there are exactly $k - i - 1$ drone legs that start from ρ_i, end in ρ_k, and that satisfy the precedence relationship defined by ρ. Lines 9 and 10 require $\Theta(1)$ operations. Hence, Algorithm 2 runs in time $\Theta(|N|^3)$, since $|\rho| = |N| + 2$.

If we fix the maximum truck leg length to A^b, the time complexity of Algorithm 2 is reduced to $\Theta((A^b)^2|N|)$ because now Line 8 requires $\Theta(A^b)$ operations, and Line 8 is now executed $\Theta(A^b|N|)$ times rather than $\Theta(|N|^2)$ times. The idea of limiting the truck leg length stems from observing that solutions with low parallelism between the truck and the drone due to long truck legs are typically associated with a high makespan since they tend to be structurally more similar to TSP solutions.

6.3. Biased fitness

A typical problem of genetic algorithms is premature convergence, which happens when the best individual in the population is far from optimality and the algorithm does not improve anymore because all individuals in the population resemble each other in terms of genetic material. Therefore, a key element of genetic algorithms is maintaining a diversified population in the evolution process to avoid premature convergence (see Fogel (1994)).

In population-based metaheuristics, a value representing the "quality" of each individual with respect to the entire population is defined, usually expressed as a function of the individual's cost, also called *fitness*, and one or more penalty terms. This value is therefore called *biased fitness* and is used to guide the "natural selection" during the evolution process. Inspired by the work of Vidal et al. (2012), we define the *diversity contribution* $\Delta(P)$ for an individual P, as the average distance to its A^{close} neighbors in the population, represented by the set $N_{close}(P)$.

Let $\Gamma^T(P)$ and $\Gamma^D(P)$ be the arrays of successors of, respectively, the truck and the drone routes of the best solution among all the ones encoded by the individual P, such that, $\Gamma_i^T(P)$ is the next node visited after i in the truck route, and $\Gamma_i^D(P)$ is the next node visited after i in the drone route. Then, the distance $\delta(P_1, P_2)$ between two individuals P_1 and P_2 is defined as the Hamming distance between the array of successors $\Gamma^T(P_1)$ and $\Gamma^T(P_2)$, plus the Hamming distance between the array of successors $\Gamma^D(P_1)$ and $\Gamma^D(P_2)$. We assume that if node i is not visited by the truck or the drone, $\Gamma_i^T(P) = -1$ and $\Gamma_i^D(P) = -1$ respectively. Therefore, the diversity contribution $\Delta(P)$ of an individual P is computed according to Eq. (9) where $\mathbf{1}(x)$ is the indicator function equal to 1 if x is true, 0 otherwise:

$$\begin{aligned}\Delta(P) &= \frac{1}{A^{close}} \sum_{P_n \in N_{close}(P)} \delta(P, P_n) \\ &= \frac{1}{A^{close}} \sum_{P_n \in N_{close}(P)} \sum_{i=0}^{|N|} \left(\mathbf{1}(\Gamma_i^T(P) \neq \Gamma_i^T(P_n)) + \mathbf{1}(\Gamma_i^D(P) \neq \Gamma_i^D(P_n))\right)\end{aligned} \tag{9}$$

Let $fit(P)$ and $dc(P)$ be the positions of the individual P in the fitness ranking and the diversity contribution ranking in ascending and descending order, respectively, with respect to the entire population. Then, we define the *biased fitness* $BF(P)$ of an individual P as

$$BF(P) = fit(P) + \lambda\, dc(P) \tag{10}$$

where $\lambda \in [0, 1]$ is a trade-off parameter used to control the relative importance of the diversity contribution over the fitness contribution of an individual. The *biased fitness* is thus a measure that balances the evolutionary pressure based on the fitness of the individuals (elitism) with the genetic diversity of the population, by penalizing individuals very "similar" to their neighbors and individuals with low fitness.

6.4. Parent selection and crossover

An important concept in genetics is genetic recombination, in which the genetic material of different organisms is combined to generate offspring with combinations of traits that differ from those found in both parents (see Fogel (1994)). This process, also known as *crossing over* (or crossover), works by breaking and rejoining chromosome segments. In the classic genetic algorithm paradigm, the biological crossover operation is simulated by combining the genetic information of two parent individuals. Various types of crossover have been proposed in the literature. In this work, we focus on the single-point crossover. The idea of the single-point crossover is to define a random crossover point c such that the resulting offspring is the concatenation of the first c elements of the first parent, with the last $n - c$ elements of the second parent, where n is the length of the two chromosomes.

However, when a single-point crossover is applied, duplicates can occur. Therefore, after copying the first c elements of the first parent, we copy the last $n - c$ elements of the second parent without inserting duplicates. The leftover elements are then shuffled and added to the offspring chromosome one by one in a position that minimizes the cost associated with the current partial chromosome. Let r be the leftover element we want to add. The minimum cost position can be found by calling the *split* procedure $O(|N|)$ times, every time on a different chromosome sequence that differs for the position of the element r. Since there can be at most $|N|/2$ leftover elements, and each element must be evaluated in at most $|N|$ different positions – each of which requires $O(|N|^3)$ operations – the time complexity of this crossover procedure is $O(|N|^5)$. If the approximated *split* procedure is chosen, the time complexity can be reduced to $O(A^b|N|^3)$.

For instances with hundreds of customers, the crossover procedure can become a bottleneck, so we propose a relaxed version with a reduced time complexity. Rather than evaluating the *split* procedure on the whole chromosome sequence, we only evaluate a partial chromosome sequence centered at r including at most A^τ elements, e.g., $A^\tau = 30$. In this way, the *split* procedure requires $O((A^b)^2 A^\tau)$ operations, and the whole crossover procedure has a complexity of $O((A^b)^2 A^\tau |N|^2)$.

With probability 0.5, instead of concatenating the last $n-c$ elements of the second parent after the first c elements of the first parent, we concatenate the first c elements of the second parent before the last $n - c$ elements of the second parent.

Finally, parents are selected using a binary tournament procedure that randomly (with a uniform probability) selects two individuals from the population and returns the one with the best biased fitness.

6.5. Education

When an offspring is generated, it undergoes an *education* phase, which consists of different local search procedures applied to the best solution associated with the individual to improve the solution cost. The local search procedures devised for the education phase apply the following operations in this exact order:

- **Reverse operations**: for each operation, check if the solution cost decreases by reversing the operation, and if so, apply the inversion;
- **Swap customers**: for each operation, check if the solution cost can be improved by swapping the takeoff node or the rendezvous node with one of its neighbor nodes. If so, apply the most improving substitution;
- **Optimize truck and combined legs**: for each truck/combined leg with at most three truck customers, optimize the order of the truck customers. For truck/combined legs with more than three customers, apply the most improving two-opt move until there are no more improvements.

Table 3
GA parameters.

Parameter	Meaning	Value
A^{epochs}	Number of Iterations	2000
A^{η}	Population size	400
A^{γ}	Number of offspring in a generation	20
A^{close}	Number of close individuals considered for distance evaluation	15
A^{λ}	Biased fitness trade-off parameter	0.92
A^{max}	Max Iterations without improvement	400
A^{b}	Max truck leg length	12
A^{τ}	Max length of partial chromosomes to consider during crossover	30
A^{r}	Fraction of individuals to reset during diversification	0.9
A^{c}	Max percentage of chromosome sharing	0.03

6.6. Population management

In the last sections, we have presented a crossover operator that describes how two individuals combined generate an offspring, an education operator that describes the local search procedures an offspring undergoes when it is created, and how parents are selected. The population management operator complements these operators to propagate good and promising solutions while enhancing the diversity in the population. By following the work of Vidal et al. (2012), our population management mechanism consists of three components: *initialization*, *diversification*, and *survivor selection*.

In the *initialization* phase, A^{η} individuals are randomly generated by assigning to each of them a random chromosome sequence. The *diversification* phase is activated when the genetic algorithm cannot improve the best solution for A^{max} consecutive iterations. This stage aims to add new genetic material to the population to escape from the local minimum where the algorithm is currently stuck. In this phase, $A^{r} \cdot A^{\eta}$ individuals with the worst biased fitness are replaced with new random individuals.

As anticipated in Section 6.3, one of the main challenges of population-based metaheuristics is to avoid premature convergence. This problem is particularly exacerbated when, as in our case, the repair phase in the crossover operator and the education operator are greedy procedures. The result is a general tendency to favor individuals with high fitness at the expense of the diversity of the population. For this purpose, the *biased fitness* measure has been defined to prevent premature convergence by discarding individuals with a low diversity contribution. Then, our *survivor selection* policy replaces after each iteration, the A^{γ} individuals with the worst biased fitness in the population with the offspring just generated.

Other than that, after every epoch we check if there are *clones* in the population, that is, if two individuals P_1, P_2 are such that $\delta(P_1, P_2) \leq 1 + A^{c} \cdot 2|V|$. If two clones are found, we replace the individual with the worst fitness with a new offspring.

7. Computational results

In this section, we discuss the computational results achieved by the exact solution method (hereafter CF) described in Section 5 and by the metaheuristic solution method (hereafter GA) described in Section 6. This section is organized as follows. Section 7.1 describes the test instances used in the computational experiments. Section 7.2 presents the results of CF. Section 7.3 analyzes the drone speeds associated with an optimal FSTSP-VDS solution. Section 7.4 provides insights on fixed vs variable drone speeds. In Section 7.5, we compare the results achieved by GA with those achieved by Raj and Murray (2020) on single-drone FSTSP-VDS instances with up to 100 customers, available at https://github.com/optimatorlab/mFSTSP-VDS. In Section 7.6, we assess GA on two FSTSP variants with up to 99 customers in which the drone speed is assumed to be constant. Finally, in Section 7.7, we evaluate the impact of the main key components proposed in GA. The computational tests of Raj and Murray (2020) have been performed on an Intel i7-6700 processor with 16 GB RAM running Ubuntu 14.04. Both CF and GA have been implemented in C and compiled with GCC 11.4 and with the optimization flag -O3. All experiments have been conducted on a 64-bit desktop computer with an AMD Ryzen 5 PRO 4650GE processor running at 3.3 GHz and with 16 GB of RAM on a GNU/Linux Ubuntu 22.04.2 LTS operating system. The MILP formulation of CF was solved with CPLEX 22.1.1 with a time limit of one hour. We use the default parameters of CPLEX, except for CPX_PARAM_THREADS, set equal to 1 to use a single thread, CPX_PARAM_EPINT (set to 0 to decrease integrality tolerance), CPX_PARAM_EPRHS (set to 1e-6 to decrease feasibility tolerance) and CPX_PARAM_EPGAP (set to 1e-5 to decrease relative MIP gap tolerance).

The minimum and maximum travel times of all feasible drone legs are computed in preprocessing as described in Section 4 using the derivative-free optimization algorithm COBYLA of the open-source nonlinear-optimization library NLopt 2.7.1 of Johnson (2007). We set a feasibility tolerance equal to 1e-6, a relative tolerance on the optimization variables equal to 1e-6, and a relative optimality tolerance of 1e-8. We impose a maximum number of NLopt iterations equal to 500 and 800, respectively, for the minimum and maximum drone leg travel time optimization problems. If an optimization problem terminates before reaching the maximum number of iterations, we set the optimal (minimum or maximum) drone leg travel time equal to the returned optimal solution's cost. Otherwise, if we are solving the minimization (maximization) problem, we define an upper (lower) bound as the best feasible solution found during the optimization procedure and a lower (upper) bound as the travel time required to traverse the drone leg at the maximum (minimum) speed. We then apply the bisection method until the difference between the lower and upper bounds is not greater than 1e-6 by solving at each step the convex optimization problem (5) to determine the minimum energy required to traverse a drone leg in a target time. In this case, we solve (5) without imposing any limit on the number of iterations. To reduce the computational burden of this phase, we apply the bisection procedure just described only for the exact method CF or if NLopt returns an error, otherwise we define the minimum/maximum drone leg travel time as the best feasible solution's cost found during the optimization procedure.

All computing times reported in this section are in seconds. Due to the problem complexity, CF was tested only on FSTSP-VDS instances with 10 customers. For each instance tested, both CF and GA are executed 10 times with a different random seed from 0 to 9. The parameters of GA used in the experiments have been defined after some preliminary testing on a subset of instances and can be found in Table 3. The source code and the solutions obtained in these experiments are available at http://github.com/federicomichelotto/FSTSP-VDS.

7.1. Test instances

The FSTSP-VDS instances used in these experiments are the ones introduced by Raj and Murray (2020), available at https://github.com/optimatorlab/mFSTSP-VDS. For this set of instances, the maximum

Table 4
Computational results of CF for different maximum drone speeds.

Instance	$v^D_{max} = 10$			$v^D_{max} = 20$			$v^D_{max} = 30$			$v^D_{max} = 40$		
	Root	Δ	Time	Root	Δ	Time	Root	Δ	Time	Root	Δ	Time
45624016194	53.56	28.57	302.82	45.87	7.98	34.16	41.55	0.00	14.50	41.55	0.00	12.68
45645377021	42.27	15.11	71.19	46.37	4.19	149.41	44.12	0.00	87.38	44.12	0.00	145.74
45706863992	41.70	22.12	80.02	56.45	1.94	69.14	55.61	0.00	14.31	55.61	0.00	11.07
45728368390	48.78	20.50	189.96	43.22	4.18	152.17	40.84	0.00	72.14	40.84	0.00	84.50
45749863540	40.36	11.75	159.23	36.71	0.95	78.91	36.11	0.00	30.80	36.11	0.00	31.34
45854314056	47.77	16.25	251.93	53.41	1.29	173.42	52.81	0.00	150.00	52.81	0.00	134.67
45916460302	53.90	30.71	299.20	56.61	7.13	170.68	54.33	1.79	71.18	53.51	0.00	85.13
45938067895	57.92	15.05	109.05	52.06	0.15	118.49	51.99	0.00	91.32	51.99	0.00	78.53
45959409904	47.06	28.85	42.61	48.22	2.43	67.30	46.97	0.02	29.07	46.96	0.00	30.20
50020711011	44.30	7.06	96.71	52.29	0.52	66.31	52.04	0.00	29.73	52.04	0.00	26.49
All	47.76	19.60	160.27	49.12	3.08	108.00	47.64	0.18	59.04	47.55	0.00	64.04

payload capacity is 5 lbs, and parcel weights are randomly chosen to be integer between 1 and 5 lbs for 85% of the customers, and more than 5 lbs for the remaining customers. Thus, the set of possible drone payloads is equal to $W = \{0, 1, 2, 3, 4, 5\}$. For each $w \in W$, the second order derivative of $E^{PM}(v, w)$ with respect to v, is a function whose numerator and denominator are strictly positive functions for $v > 0$, and consequently, $E^{PM}(v, w)$ is convex for $v > 0$ for each $w \in W$. Therefore, for each drone leg, the set of feasible drone leg travel times is a convex set.

The test set consists of two sets with customer locations in the Seattle and the Washington area. The first set consists of 10 problems for each of four levels of the number of customers (10, 25, 50, and 100) generated on a fixed service area of 918.0 km^2, for a total of 40 problem instances. The second set consists of 10 problems, each one with 50 customers, for each of four levels of service area (57.4, 229.5, 516.4, and 918.0 km^2), for a total of 40 problems. In the following, we will refer to these two datasets as dataset A and B. In all instances, customer locations are generated from a uniform distribution on the road network within the service area. For each level of customers and each level of service area, 5 instances featured a centrally-located depot, and 5 instances contained a depot at the periphery. The drone battery capacity is equal to $B = 500$ kJ. The takeoff and landing drone speeds are constants and set respectively to 10 m/s and 5 m/s. The cruise altitude h is set to 50 m. Truck and drone delivery times σ^T and σ^D are set to 30 and 60 s, respectively. Drone launch and retrieval times, s_L and s_R, are set to 60 and 30 s, respectively. The distance d between two locations is computed using the Haversine distance formula as follows

$$d = 2R \arcsin\left(\sqrt{\sin^2\left(\frac{\theta_2 - \theta_1}{2}\right) + \cos(\theta_1)\cos(\theta_2)\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right)}\right)$$

Where θ_1, φ_1 represent the latitude and longitude of the first location, θ_2, φ_2 the latitude and longitude of the second location, and $R = 6378100.0$ denotes the radius of the earth in meters. The Liu et al. (2017) model is used to characterize the drone power consumption with respect to speed and payload weight, whose parameters are defined in Section 3.1. We test all instances with different maximum drone speeds $v^D_{max} = 10, 20, 30, 40$ m/s, for a total of 320 instances. The minimum drone speed v^D_{min} is set equal to 0.1 m/s.

The FSTSP instances used in these experiments are the ones used by Blufstein et al. (2024) (available at https://github.com/fsoulignac/tspd-release), an adaptation of the instances originally introduced by Poikonen et al. (2019) for the TSP-D. In particular, the test set consists of 250 instances with $n = 9, 19, \ldots, 99$ customers, 25 for each problem size. The instances were created by randomly locating the depot and the customers on a 50-by-50 grid using a uniform distribution on both axes. Given two locations i and j and the corresponding x, y coordinates (x_i, y_i) and (x_j, y_j), truck travel times are computed according to the Manhattan (taxicab) metric, i.e., $t^T_{ij} = \lfloor |x_i - x_j| + |y_i - j_j| \rfloor$, drone travel times are instead computed according to the Euclidean distance metric at a speed that is μ times faster than the truck, i.e., $t^D_{ij} = \lfloor \sqrt{(x_i - x_j)^2 + (y_i - j_j)^2}/\mu \rfloor$. Each instance is solved with $\mu = 1, 2, 3$, for a total of 750 instances.

7.2. Compact formulation

In this section, we evaluate the proposed exact method CF on the ten instances with 10 customers of dataset A. Table 4 evaluates CF for different values of v^D_{max}: 10, 20, 30 and 40 m/s. Column instance reports the name of the instance evaluated. All instances begin with the prefix 20191230T1, which, for clarity, has been omitted from the table. Columns root and Δ report respectively the integrality gap, and the gap between the optimal solution obtained and the optimal solution for the FSTSP-VDS when $v^D_{max} = 40$ m/s. Specifically, the integrality gap is measured as $(z^* - LB0)/z^*$, where z^* is the optimal solution cost and $LB0$ is the linear relaxation cost. Column time reports the overall computing time, including preprocessing.

First of all, all instances tested have been solved to optimality by CF. The average computing time goes from 160.27 s when $v^D_{max} = 10$ m/s to 64.04 s when $v^D_{max} = 40$ m/s. The average integrality gap is about 50% independently of the value of v^D_{max}. Clearly, Δ is higher for lower values of v^D_{max}, and it is respectively 19.60%, 3.08% and 0.18% for v^D_{max} values equal to 10, 20 and 30 m/s.

7.3. Optimal drone speeds

Observe that, in general, a drone leg $(i, j, k) \in Z$ can be completed in $t \in [t^D_{ijk}, T^D_{ijk}]$ seconds (considering the potential hovering above the rendezvous node) by more than one drone speed configuration (v^D_{ij}, v^D_{jk}), and that the set of all possible drone speed configurations is convex if the drone energy-per-meter function is convex (see Section 4 and in particular, formulation (5)). Therefore, reporting the drone speeds associated with a FSTSP-VDS solution is not always possible. This section addresses this point and aims to provide meaningful insights.

Fig. 5 depicts the optimal solution found for the ten-customer instance 45624016194 of dataset A. Table 5 shows the main features of the operations in this solution. Columns i, j and k denote, respectively, the takeoff node, the drone customer, and the rendezvous node. Column $t^T_{i\to k}$ denotes the truck leg travel time to go from i to k, including the delivery times to serve all customers in between, and, for the sake of simplicity, also customer k since all rendezvous customers in this solution are served before the truck and drone synchronization ($\alpha_k = 1$). Columns t^D_{ijk} and T^D_{ijk} denote, respectively, the minimum and maximum drone leg travel times. Therefore, the optimal drone leg travel time for each operation must be equal to $\max\{t^D_{ijk}, t^T_{i\to k}\}$ seconds, and the maximum of these two values is highlighted in bold. Column w^T denotes the waiting time of the truck for the drone arrival at the rendezvous node k, computed as $\max\{t^T_{i\to k} - t^D_{ijk}, 0.0\}$. Columns $\overline{v}^D_{ij}$ and $\overline{v}^D_{jk}$ denote the feasible drone speed configurations that allows to complete the drone leg $(i, j, k) \in Z$ (considering the potential hovering

Table 5
Main features of the operations corresponding to the optimal solution depicted in Fig. 5.

i	j	k	$t^T_{i\to k}$	t^D_{ijk}	T^D_{ijk}	w^T	$\overline{v}^D_{ij}$	$\overline{v}^D_{jk}$	V^D_{ij}	V^D_{jk}
0	3	7	1596.30	**1671.07**	2162.50	74.77	20.58	16.14	[20.58, 20.58]	[16.14, 16.14]
7	5	1	308.44	**368.53**	2485.57	60.09	34.11	32.96	[34.10, 34.11]	[32.96, 32.96]
1	4	10	1003.58	**1047.17**	2473.26	43.59	22.60	21.60	[22.60, 22.60]	[21.60, 21.60]
10	8	9	**820.09**	779.51	2434.47	0.00	22.10	24.87	[20.64, 29.73]	[19.92, 26.71]
9	6	0'	**397.52**	389.72	2390.55	0.00	30.03	34.54	[29.25, 36.74]	[27.57, 35.87]

above the rendezvous node) in exactly $\max\{t^D_{ijk}, t^T_{i\to k}\}$ seconds, in a way that the energy consumption is minimized. To compute these quantities, we solved the convex optimization problem (5). Finally, column V^D_{ij} (V^D_{jk}, resp.) denotes the convex set of drone speeds to go from i to j (from j to k, resp.) such that there exists at least a feasible drone speed to go from j to k (from i to j, resp.) that allows to complete the drone leg $(i, j, k) \in Z$ (considering the potential hovering above the rendezvous node) in exactly $\max\{t^D_{ijk}, t^T_{i\to k}\}$ seconds without exceeding the battery capacity. To compute these convex sets, we implemented a bisection procedure that recursively solves a sub-problem of (5) in which either t^D_{ij} or t^D_{jk} is fixed. Note that the cartesian product $V^D_{ij} \times V^D_{jk}$ is a super-set of the convex set of all feasible drone speed configurations. All times and speeds reported in Table 5 are expressed, respectively, in seconds and m/s.

The solution's makespan is equal to the sum of all times reported in Fig. 5 (truck travel and waiting times), plus the time required to launch the drone ($s_L = 60$ seconds) and retrieve it ($s_R = 30$ seconds) multiplied by the number of drone legs in the solution (5), yielding a total of 4754.38 s.

Notice that the duration of the first three operations is given by t^D_{ijk} since the truck must wait for the drone, and the possible speed ranges are quite small. On the other hand, the last two operations have a duration equal to $t^T_{i\to k}$, since the drone must wait for the truck, and the possible drone speed ranges are much larger.

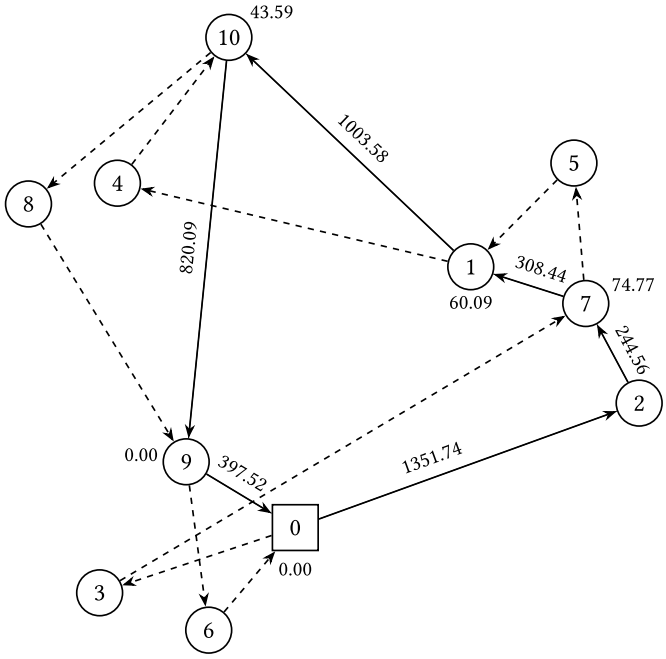

Fig. 5. Optimal solution for the instance 45624016194 with a makespan of 4754.38 s. Values above the edges and close to the rendezvous nodes denote, respectively, the truck travel times and truck waiting times for the drone arrival, in seconds. Since all rendezvous customers are served before the truck and drone synchronization in this solution, truck delivery times have been included in the truck travel times. Times related to the drone launching and retrieval operations are not shown and are equal to $s_L + s_R = 90$ seconds for each drone leg.

7.4. Fixed vs variable drone speeds

Fig. 6 illustrates the advantage of not fixing a priori the drone speed over the ten instances with ten customers solved with CF. The figure shows how the gap with respect to the optimal FSTSP-VDS solution with $v^D_{max} = 40$ m/s is affected by varying the value of v^D_{max}, both when the drone speed is fixed and when it is not. All instances have been solved to optimality. The red line represents the average gap when the drone speed is not fixed, i.e., the Δ values of Table 4. The blue line denotes the average gap when the drone speed is fixed to the maximum allowed speed v^D_{max}. When v^D_{max} is 10 or 20 m/s, fixing or not the drone speed produces a gap difference of 0.0% and 1.99%, respectively. Instead, when v^D_{max} is 30 or 40 m/s, fixing the drone speed to v^D_{max} results in a gap increase of 22.82% and 32.06%, respectively. The best results when the drone speed is fixed have been obtained with a speed of 20 m/s; however, even in this case, the gap with respect to the optimal FSTSP-VDS solution with $v^D_{max} = 40$ m/s is 5.07%. Finally, we also evaluated the case in which the drone speed for each drone flight is set equal to the speed that maximizes the drone range (it depends only on the payload weight) and obtained a gap of 7.79%, even larger than always flying at 20 m/s.

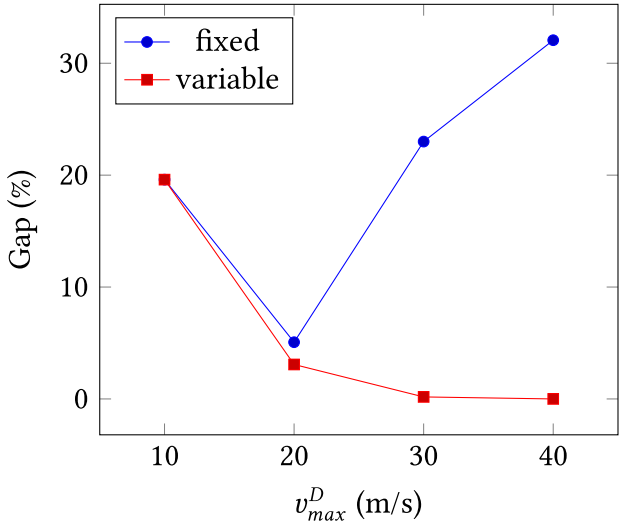

Fig. 6. Average gap with respect to the optimal FSTSP-VDS solutions with $v^D_{max} = 40$ m/s, for different maximum drone speeds. In blue, the drone speed is fixed and equal to v^D_{max}. In red, the drone speed is variable and can vary between v^D_{min} and v^D_{max}.

7.5. Computational results of GA on FSTSP-VDS instances

In this section, we assess GA on FSTSP-VDS instances. To the best of our knowledge, the only known solution method for the FSTSP-VDS in literature is the heuristic method of Raj and Murray (2020), hereafter RM. Therefore, we refer to the best-known solution as the best solution between our method GA and RM for the instances with more than ten customers, and the optimal solutions obtained by CF for the instances with ten customers.

Table 6
Comparison between GA and RM for different maximum drone speeds on dataset A.

$\|N\|$	v^D_{max}	GA				RM	
		Min	Avg	Preprocessing	Time	Avg	Time
10	10	0.00	0.00	0.29	6.94	4.55	0.32
	20	0.00	0.00	0.09	6.40	4.87	0.28
	30	0.00	0.00	0.11	6.41	6.73	0.20
	40	0.00	0.00	0.12	6.42	7.07	0.20
25	10	0.00	0.00	4.09	43.55	7.93	3.56
	20	0.00	0.01	1.30	27.30	7.15	3.24
	30	0.00	0.07	1.60	27.43	7.09	2.81
	40	0.00	0.00	1.86	28.24	7.34	2.83
50	10	0.00	0.08	35.78	157.14	9.21	28.92
	20	0.00	0.15	11.54	91.05	10.47	31.01
	30	0.00	0.22	14.20	90.56	7.82	28.43
	40	0.00	0.29	16.65	91.83	8.68	28.50
100	10	0.00	0.37	274.66	735.30	9.74	491.46
	20	0.00	0.45	92.97	411.99	8.42	580.25
	30	0.00	0.46	109.78	400.82	8.41	465.52
	40	0.00	0.48	128.89	412.08	9.06	454.70
All	10	0.00	0.11			7.86	
	20	0.00	0.15			7.72	
	30	0.00	0.19			7.51	
	40	0.00	0.19			8.04	
All		0.00	0.16			7.78	

Table 7
Comparison between GA and RM for different maximum drone speeds on dataset B.

$\|N\|$	v^D_{max}	GA				RM	
		Min	Avg	Preprocessing	Time	Avg	Time
50	10	0.00	0.17	84.74	191.21	7.73	36.03
	20	0.00	0.17	14.16	91.20	7.72	34.33
	30	0.00	0.19	24.06	96.81	5.92	28.96
	40	0.00	0.22	31.00	102.91	6.12	27.99
All		0.00	0.19			6.87	

Tables 6 and 7 compare GA with RM respectively on the dataset A and B. Columns min and avg report, respectively, the aggregate average gap and the aggregate minimum gap, in percentage, with respect to best-known solutions, columns preprocessing and time report the average preprocessing time required to compute the minimum and maximum drone leg time of all drone legs, and the average overall computing time, including preprocessing.

For all instances, the best-known solution was obtained by GA. For all instances with 10 customers, GA was able to find an optimal solution for all random seeds tested. Across the 160 instances of dataset A, the aggregate average gap is 0.16% for the proposed method GA and 7.78% for RM. The computing time of GA ranges from a few seconds on the instances with ten customers up to about 10 min on the instances with 100 customers. The computing time of RM is significantly lower than GA up to 50 customers, but it reduces as the instance size increases, and on the largest instances with 100 customers, the two methods have comparable computing times, showing that GA scales better than RM. The preprocessing time grows exponentially and requires hundreds of seconds on the largest set of instances with 100 customers. No significant differences can be observed on dataset B.

7.6. Computational results of GA on FSTSP instances

In this Section, we evaluate the computational results of GA on FSTSP instances with respect to the solutions found by the exact solution method of Blufstein et al. (2024) (hereafter PFA). Table 8 reports the computational results of GA on two FSTSP variants evaluated by Blufstein et al. (2024), named BASE and MHD. In the BASE variant, the drone endurance is infinite, a node must be visited exactly once, and the drone cannot be launched and retrieved in the same location. The MHD variant has the same constraints as the BASE variant, plus the following ones. The drone has a finite endurance equal to $e = 20$, it cannot land and wait at the rendezvous location, the time to launch and retrieve the drone requires a time equal to $s_L = s_R = 1$, and 20% of customers are drone-incompatible (in particular, a customer $j \in N$ is drone-incompatible if $j - 1$ is a multiple of 5). The launch time s_L of the operations that start from the depot is not taken into account. Following Blufstein et al. (2024), for instances with $n \geq 49$ customers, nodes' coordinates, e, s_L and s_R are multiplied by 100. The results of PFA were obtained with a time limit of one hour, and to the best of our knowledge, all best-known solutions for this set of instances belong to PFA.

Columns Ψ and opt indicate respectively the number of instances solved to optimality by PFA, and how many of these instances are solved to optimality by GA, by at least one run. Columns min and avg report respectively the aggregate minimum and average percentage gap of GA with respect to the best-known solutions. Column avg_opt reports the aggregate average percentage gap of GA with respect only to the optimal solutions found by PFA. Column time reports the average computing time of GA in seconds.

For the BASE configuration, GA obtains an aggregate minimum and average gap with respect to the best-known solutions, respectively of 0.07% and 0.39%. GA obtains an aggregate average gap of 0.33% with respect to the 659 proven optimal solutions found by PFA, and for 501 of them, GA finds an optimal solution (76%). When the drone endurance is limited (MHD) GA obtains an aggregate minimum and average gap with respect to the best-known solutions, respectively of 0.05% and 0.25%. For the MHD variant, GA obtains an aggregate average gap of 0.18% with respect to the 594 proven optimal solutions found by PFA, and for 489 of them, GA finds an optimal solution (82%). The average computing time on these instances ranges from a few seconds on the smallest instances with 9 customers up to about 4 min on the largest instances with 99 customers. If we analyze the gap with respect to the different drone flying speed levels $\mu = 1, 2, 3$, we observe an aggregate average gap respectively equal to 0.17%, 0.36%, and 0.64% for the BASE variant, and a gap equal to 0.10%, 0.29%, and 0.36% for the MHD variant. Both for the BASE and the MHD variant, the increase of μ (i.e., the increase of the drone speed) is associated with an increase in the gap. It is worth noting that for $\mu = 1$, GA was able to find an optimal solution for all but 21 (6 if we consider only the MHD variant) of the 417 instances tested. Moreover, GA was also able to find optimal solutions for some of the largest instances with 99 customers.

7.7. Impact of key components of GA

In this section, we evaluate the impact of two key components of the proposed heuristic method GA: the repair phase and the approximated split procedure. To evaluate the effectiveness of these components – described in Section 6 – we compare the results obtained by GA on Dataset A, against two variants, namely, GA_{repair} and GA_{split} (see Table 9).

In the variant GA_{repair}, instead of using the *split* procedure to reinsert the missing customers into the offspring chromosome, a more classical and simpler distance-based reinsertion is adopted. In particular, the missing customers are reinserted one by one, in random order, in one of the empty positions that minimizes the sum of distances to the predecessor and successor nodes. Conversely, the variant GA_{split} removes the limit on the truck leg length within the split procedure, equal to $A^b = 12$.

Columns avg and time report the average gap and the computing time for each method, respectively. To ensure a fair comparison, GA_{repair} has not been evaluated with an iteration-limit but, rather, with a time-limit equal to the average computing time obtained by GA to

Table 8
Computational Results of GA on FSTSP instances.

$\|N\|$	μ	BASE						MHD					
		Ψ	Opt	Min	Avg	avg_opt	Time	Ψ	Opt	Min	Avg	avg_opt	Time
9	1	25	25	0.00	0.00	0.00	5.03	25	25	0.00	0.00	0.00	5.09
	2	25	25	0.00	0.00	0.00	4.99	25	25	0.00	0.00	0.00	5.08
	3	25	25	0.00	0.00	0.00	4.99	25	25	0.00	0.00	0.00	5.02
19	1	25	25	0.00	0.00	0.00	11.09	25	25	0.00	0.00	0.00	14.01
	2	25	25	0.00	0.01	0.01	10.47	25	25	0.00	0.00	0.00	12.12
	3	25	25	0.00	0.03	0.03	10.38	25	25	0.00	0.01	0.01	10.59
29	1	25	25	0.00	0.01	0.01	21.72	25	25	0.00	0.00	0.00	29.98
	2	25	25	0.00	0.03	0.03	19.20	25	25	0.00	0.02	0.02	22.93
	3	25	23	0.11	0.37	0.37	18.74	25	25	0.00	0.08	0.08	19.17
39	1	25	25	0.00	0.04	0.04	34.98	25	25	0.00	0.00	0.00	49.20
	2	25	24	0.02	0.23	0.23	28.45	25	25	0.00	0.04	0.04	35.76
	3	25	20	0.18	0.52	0.52	26.79	25	23	0.07	0.24	0.24	29.30
49	1	25	22	0.01	0.07	0.07	49.52	25	25	0.00	0.01	0.01	77.04
	2	25	24	0.01	0.23	0.23	34.64	25	21	0.02	0.08	0.08	47.43
	3	25	15	0.09	0.55	0.55	33.17	25	17	0.10	0.45	0.45	37.71
59	1	25	21	0.02	0.17	0.17	61.80	25	25	0.00	0.05	0.05	103.10
	2	25	19	0.04	0.38	0.38	44.88	25	21	0.03	0.22	0.22	61.82
	3	25	15	0.10	0.59	0.59	42.91	22	11	0.15	0.38	0.39	50.45
69	1	25	23	0.00	0.18	0.18	78.17	23	23	0.00	0.09	0.08	126.97
	2	25	13	0.07	0.47	0.47	59.90	23	14	0.08	0.50	0.52	81.68
	3	23	10	0.13	0.72	0.72	57.64	16	4	0.09	0.41	0.49	67.70
79	1	23	19	0.01	0.28	0.27	97.02	20	17	0.02	0.20	0.21	153.56
	2	22	8	0.11	0.49	0.50	74.74	18	4	0.19	0.63	0.72	102.51
	3	23	2	0.34	1.02	0.98	71.46	17	0	0.23	0.72	0.72	84.68
89	1	13	11	0.01	0.45	0.53	123.66	11	9	0.00	0.20	0.16	189.22
	2	19	3	0.18	0.83	0.82	93.74	8	0	0.17	0.59	0.87	131.94
	3	17	1	0.56	1.30	1.32	92.82	6	0	0.17	0.64	1.00	107.59
99	1	1	1	0.00	0.49	0.11	152.16	1	0	0.00	0.44	0.27	230.49
	2	10	2	0.09	0.90	0.83	118.65	3	0	0.04	0.80	1.26	163.29
	3	8	0	0.16	1.30	1.52	114.10	1	0	0.10	0.68	0.72	132.40
All	1	212	197	0.01	0.17	0.12		205	199	0.00	0.10	0.05	
	2	226	168	0.05	0.36	0.30		202	160	0.05	0.29	0.22	
	3	221	136	0.17	0.64	0.57		187	130	0.09	0.36	0.29	
All		659	501	0.07	0.39	0.33		594	489	0.05	0.25	0.18	

execute $A^{epochs} = 2000$ iterations, depending on the instance size $|N|$ and the maximum drone speed v^D_{max}.

For GA_{repair}, the column time is replaced by the column iterations, which reports the number of iterations executed by GA_{repair}. The variant GA_{repair} executes 10 to 100 times more iterations than GA. However, it achieves gaps similar to those of GA only on the ten-customer instances; thereafter, the gap grows substantially, reaching a gap of 17.10% on the 100-customer instances with $v^D_{max} = 10$ m/s. Overall, the variant GA_{split} achieves results comparable to those of GA, but at the cost of higher computing times. Some small gap improvements can be observed on the 100-customer instances with $v^D_{max} = \{10, 40\}$ m/s; however, the computing time on these instances is two to three times higher than that of GA.

Table 9
Comparison between GA and two variants for different maximum drone speeds on dataset A.

$\|N\|$	v^D_{max}	GA		GA_{repair}		GA_{split}	
		Avg	Time	Avg	Iterations	Avg	Time
10	10	0.00	6.94	0.00	21 782.10	0.00	6.96
	20	0.00	6.40	0.02	72 918.90	0.00	6.41
	30	0.00	6.41	0.02	127 013.10	0.00	6.42
	40	0.00	6.42	0.06	204 590.00	0.00	6.44
25	10	0.00	43.55	1.87	22 784.90	0.00	65.99
	20	0.01	27.30	1.78	49 843.80	0.01	37.11
	30	0.07	27.43	2.27	88 764.90	0.07	37.21
	40	0.00	28.24	2.21	154 214.20	0.00	38.42
50	10	0.08	157.14	6.66	22 657.10	0.05	294.49
	20	0.15	91.05	5.94	49 304.90	0.14	145.86
	30	0.22	90.56	6.77	85 126.00	0.22	144.62
	40	0.29	91.83	5.98	142 370.20	0.29	145.03
100	10	0.37	735.30	17.10	22 584.90	0.30	2005.46
	20	0.45	411.99	15.42	50 736.70	0.45	934.84
	30	0.46	400.82	15.52	83 285.70	0.46	903.85
	40	0.48	412.08	16.40	138 589.10	0.40	906.59
All	10	0.11		6.41		0.09	
	20	0.15		5.79		0.15	
	30	0.19		6.14		0.19	
	40	0.19		6.16		0.17	
All		0.16		6.13		0.15	

8. Conclusions

Most drone-aided routing problems assume a constant drone energy consumption model, which is, however, an assumption that does not hold in practice. The goal of this work was to study and propose effective solution methods for a more realistic variant of the Flying Sidekick Traveling Salesman Problem in which the drone power consumption is modeled as a more realistic non-linear function with respect to the speed and parcel weight.

We proposed an exact approach based on a compact MILP formulation and a genetic algorithm. Both solutions are based on the idea of computing in advance the minimum and maximum feasible drone leg travel times to avoid handling drone speeds as decision variables. To do so, we showed that the set of feasible drone leg travel times

of a generic drone leg is convex if the energy-per-meter function is convex with respect to the speed, and devised a procedure to compute the minimum and the maximum of this set. We point out that the underlying assumption of our proposed methods, i.e., that the energy-per-meter function that models the drone power consumption is convex with respect to the speed, is not an isolated case, but instead, a common characteristic of other popular drone energy models reported in literature, see for example Stolaroff et al. (2018) and Zhang et al. (2021).

The resulting compact MILP formulation allows to easily solve to optimality FSTSP-VDS instances with ten customers, but struggles with larger instances. For this purpose, we devised a genetic algorithm and evaluated it both on FSTSP-VDS and FSTSP (in which the drone speed is constant) instances. For the FSTSP-VDS, the proposed heuristic obtains an average makespan improvement of about 7% with respect to the state-of-the-art. Our heuristic also finds the optimal solutions of all the instances with ten customers solved to optimality by the MILP formulation. Regarding the FSTSP, the proposed heuristic obtains an average gap of 0.26% among the 1253 optimally solved instances ranging from 9 to 99 customers, and finds an optimal solution for 990 of them, including some of the largest instances with 99 customers.

From a managerial perspective, we investigate how fixing the drone speed affects the makespan. To get reliable insights, we only solved the ten instances with 10 customers solvable by the MILP formulation. The results show that even a reasonable drone speed like 10 m/s produces a massive 19.60% increase in the makespan. At 20 m/s, the gap decreases to 5.07%, at 30 m/s and 40 m/s becomes respectively 23.00% and 32.66%. Choosing the maximum range speed for each drone flight resulted in a gap of 7.79%, greater than always flying at 20 m/s. These results suggest that choosing over-conservative speeds or high speeds but too "expensive" or too stringent selection policies, easily lead to a large increase in the makespan, and highlight the importance of the drone speeds selection. This becomes even more prominent if we observe that these experiments have been performed on small instances with ten customers, thus, we can expect even larger gaps on larger real-world instances.

Computing the minimum and maximum drone leg times for all drone legs can become an issue when dealing with instances with more than 100 customers. For these instances, however, only heuristic approaches seem practicable, and therefore one could think of heuristic approaches also for computing these quantities, and of re-optimization procedures to call at runtime for promising drone legs only.

CRediT authorship contribution statement

Federico Michelotto: Writing – original draft, Validation, Software, Methodology, Formal analysis, Conceptualization. **Roberto Roberti:** Writing – original draft, Supervision, Methodology, Funding acquisition, Formal analysis, Conceptualization.

Funding

Roberto Roberti's work is supported by Ministero dell'Università e della Ricerca (MUR), PRIN 2022 project 20229ZWC97 – Revise and Enhance: Win-win INsights for home Delivery services (REWIND).

Acknowledgments

We are grateful to Francisco Soulignac for his help with interpreting the test instances of Blufstein et al. (2024). We are also grateful to the three anonymous referees for their helpful comments.

Data availability

We have shared the link to the data in the paper.

References

Agatz, N., Bouman, P., Schmidt, M., 2018. Optimization approaches for the traveling salesman problem with drone. Transp. Sci. 52 (4), 965–981.

Banker, S., 2022. The race for last mile drones. https://www.forbes.com/sites/stevebanker/2022/08/16/the-race-for-last-mile-drones/. (last accessed on 6 June 2025).

Blufstein, M., Lera-Romero, G., Soulignac, F.J., 2024. Decremental state-space relaxations for the basic traveling salesman problem with a drone. INFORMS J. Comput. 36 (4), 1064–1083.

Boccia, M., Mancuso, A., Masone, A., Sterle, C., 2023. A new MILP formulation for the flying sidekick traveling salesman problem. Networks 82 (3), 254–276.

Boccia, M., Masone, A., Sforza, A., Sterle, C., 2021. A column-and-row generation approach for the flying sidekick travelling salesman problem. Transp. Res. Part C Emerg. Technol. 124, 102913.

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

Campuzano, G., Lalla-Ruiz, E., Mes, M., 2023. The drone-assisted variable speed asymmetric traveling salesman problem. Comput. Ind. Eng. 176, 109003.

Chen, C., Demir, E., Hu, X., Huang, H., 2025. Transforming last mile delivery with heterogeneous assistants: drones and delivery robots. J. Heuristics 31, 8.

de Freitas, J.C., Penna, P.H.V., 2020. A variable neighborhood search for flying sidekick traveling salesman problem. Int. Trans. Oper. Res. 27 (1), 267–290.

Dell'Amico, M., Montemanni, R., Novellani, S., 2021a. Algorithms based on branch and bound for the flying sidekick traveling salesman problem. Omega 104, 102493.

Dell'Amico, M., Montemanni, R., Novellani, S., 2021b. Modeling the flying sidekick traveling salesman problem with multiple drones. Networks 78 (3), 303–327.

Dell'Amico, M., Montemanni, R., Novellani, S., 2022. Exact models for the flying sidekick traveling salesman problem. Int. Trans. Oper. Res. 29 (3), 1360–1393.

El-Adle, A.M., Ghoniem, A., Haouari, M., 2023. A variable neighborhood search for parcel delivery by vehicle with drone cycles. Comput. Oper. Res. 159.

Es Yurek, E., Ozmutlu, H.C., 2018. A decomposition-based iterative optimization algorithm for traveling salesman problem with drone. Transp. Res. Part C Emerg. Technol. 91, 249–262.

Fogel, D.B., 1994. An introduction to simulated evolutionary optimization. IEEE Trans. Neural Netw. 5 (1), 3–14.

Ha, Q.M., Deville, Y., Pham, Q.D., Hà, M.H.n., 2018. On the min-cost traveling salesman problem with drone. Transp. Res. Part C Emerg. Technol. 86, 597–621.

Ha, Q.M., Deville, Y., Pham, Q.D., Hà, M.H.n., 2020. A hybrid genetic algorithm for the traveling salesman problem with drone. J. Heuristics 26 (2), 219–247.

Johnson, S.G., 2007. The NLopt nonlinear-optimization package. https://github.com/stevengj/nlopt.

Liang, Y.-J., Luo, Z.-X., 2022. A survey of truck–drone routing problem: Literature review and research prospects. J. Oper. Res. Soc. China 10 (2), 343–377.

Liu, Z., Sengupta, R., Kurzhanskiy, A., 2017. A power consumption model for multi-rotor small unmanned aircraft systems. In: 2017 International Conference on Unmanned Aircraft Systems. ICUAS, pp. 310–315.

Macrina, G., Di Puglia Pugliese, L., Guerriero, F., Laporte, G., 2020. Drone-aided routing: A literature review. Transp. Res. Part C Emerg. Technol. 120, 102762.

Murray, C., Chu, A., 2015. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. Transp. Res. Part C Emerg. Technol. 54, 86–109.

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

Raj, R., Murray, C., 2020. The multiple flying sidekicks traveling salesman problem with variable drone speeds. Transp. Res. Part C Emerg. Technol. 120, 102813.

Roberti, R., Ruthmair, M., 2021. Exact methods for the traveling salesman problem with drone. Transp. Sci. 55 (2), 315–335.

Schermer, D., Moeini, M., Wendt, O., 2020. A branch-and-cut approach and alternative formulations for the traveling salesman problem with drone. Networks 76 (2), 164–186.

Stolaroff, J.K., Samaras, C., O'Neill, E.R., Lubers, A., Mitchell, A.S., Ceperley, D., 2018. Energy use and life cycle greenhouse gas emissions of drones for commercial package delivery. Nat. Commun. 9 (409).

Tamke, F., Buscher, U., 2023. The vehicle routing problem with drones and drone speed selection. Comput. Oper. Res. 152, 106112.

Vásquez, S.A., Angulo, G., Klapp, M.A., 2021. An exact solution method for the TSP with drone based on decomposition. Comput. Oper. Res. 127, 105127.

Vidal, T., Crainic, T.G., Gendreau, M., Lahrichi, N., Rei, W., 2012. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Oper. Res. 60 (3), 611–624.

Zhang, J., Campbell, J.F., Sweeney, II, D.C., Hupman, A.C., 2021. Energy consumption models for delivery drones: A comparison and assessment. Transp. Res. D Transp. Env. 90, 102668.