Energy-Constrained Delivery of Goods with Drones Under Varying Wind Conditions

Francesco Betti Sorbelli, Federico Corò, Sajal K. Das, and Cristina M. Pinotti

Abstract—In this paper, we study the feasibility of sending drones to deliver goods from a depot to a customer by solving what we call the Mission-Feasibility Problem (MFP). Due to payload constraints, the drone can serve only one customer at a time. To this end, we propose a novel framework based on time-dependent cost graphs to properly model the MFP and tackle the delivery dynamics. When the drone moves in the delivery area map, the global wind may change thereby affecting the drone's energy consumption, which in turn can increase or decrease. This issue is addressed by designing three algorithms for drone delivery, namely (i) compute the route of minimum energy once, at the beginning of the mission, (ii) dynamically reconsider on the fly the most convenient trip towards the destination, and (iii) dynamically select on the fly only the best local choice. We simulate our algorithms on Erdős–Rényi graphs. The changes on the drone's energy consumption are reflected by changes on the cost of the edges of the graphs. The algorithms receive the new costs every time the drone flies over a new vertex, and they have no full knowledge in advance of the weights. We compare the algorithms in terms of the percentage of missions that are completed with success (the drone delivers the goods and comes back to the depot), with delivered (the drone delivers the goods but cannot come back to the depot), and with failure (the drone neither delivers the goods nor comes back to the depot).

Index Terms—Drone delivery algorithms, energy model, wind model, cost-update time-dependent graphs, mission feasibility problem

1 Introduction

In recent years, drones or Unmanned Aerial Vehicles (UAVs) have been widely deployed in civil applications [1], such as agriculture, environmental protection, localization [2], [3], and delivery of goods. For example, Amazon, Google, UPS, and DHL are testing drone delivery systems, particularly for "last-mile" logistics of small items. There are numerous advantages of using drones for deliveries, including economic benefits and the ability to deliver in time-critical situations or hard-to-reach places. Transportation companies can further extend their business relying on drones that deliver goods directly to the customer locations in medium to large size cities or areas [4].

In our scenario, a transportation company is assumed to have a depot (warehouse) and a fleet of drones for serving the customers that are spread out in a delivery area map and can request for product deliveries. Since the payload constraint forces the drone to go back to the depot after each delivery, the Traveling Salesman Problem (TSP) is not suitable to properly model the delivery systems using UAVs; thus it is only possible to serve each customer separately [5]. In open areas, drones can fly directly (line of sight) from the source to the destination location, thereby shortening the minimum travel distance and hence extending the lifetime of the drone. In high dense urban areas, drones can fly following the routes on the ground to avoid building. Since by regulation, drones can fly up to a maximum allowable altitude depending on the landscape, we assume they fly at lower altitudes. Flying, the energy consumption is severely affected by the airflow [6], which in turn is affected by buildings, and obstacles in general. The drone will prefer to fly tailwind (or downwind) than headwind because the former wind requires less energy that the latter one [7]. The weight of the payload also impacts the autonomy of the drone's battery (or energy) during its flight.

In this paper, we investigate the delivery of goods to the customers using a drone with energy budget given by its battery. Given a customer's order of goods and an available drone for delivery, the problem is to find a cycle for the drone which is feasible, that is, which can be completed with the energy autonomy of the drone. The cycle consists of two itineraries – one from the warehouse to the customer destination with payload and the other from the customer to the warehouse without payload – that can be accomplished with the energy available in the battery. The drone's energy consumption depends on the static as well as dynamic parameters. While some parameters (e.g., the length of a traversed route, or the weight of the carried payload) are fixed and known before the mission, other parameters (e.g., the wind) may not be exactly predicted because they are exactly known only when observed, and may change several times during the mission. The drone may decide, during the mission, to take routes different from those originally planned to take advantage of the wind changes.

Our approach is to model the delivery area map with a weighted graph whose costs are time-dependent. Since varying wind conditions affect the drone's energy consumption, we propose to model the varying wind conditions with edge costs that change over the time. Instead, the topology of the graph (set of vertices and set of edges) is static because the routes that the drone can traverse on the delivery area

- *Francesco Betti Sorbelli and Sajal Das are with Dept. of Computer Science, Missouri Univ. of Science and Technology, Rolla, MO, USA, 65409.*
- *Federico Corò is with the Department of Computer Science, Sapienza University of Rome, Italy.*
- *Cristina M. Pinotti is with the Department of Computer Science and Mathematics, University of Perugia, Italy.*

cannot change. The edge costs depend not only on the relative wind condition at the edge, but also on the speed of the drone and the weight of the payload. Under this framework, we address the problem of finding which is the percentage of missions that can be accomplished with success (i.e., the drone delivers the goods and returns to the depot) with a given battery energy constraint.

Contributions: Our results are summarized as follows.

- We define the Mission-Feasibility Problem (MFP), which deals with the problem of delivering goods under a given energy budget and under varying wind conditions, not fully known a priori.
- We design three algorithms for MFP, namely Static-Offline Shortest Path (OSP), Dynamic Shortest Path (DSP), and Greedy Shortest Path (GDP) that consider different strategies to deal with the change of wind conditions during the drone's trip.
- Through simulation experiments on Erdős–Rényi random graphs with time-dependent cost edges, we evaluate the performance of our algorithms and compare them in terms of the percentage of missions completed (with delivery only, success, or failure) depending on the drone delivering the goods or not, and coming back to the depot or not.

Organization: The rest of the paper is organized as follows. Section 2 reviews related works on drone delivery problem and shortest path for dynamic graphs. Section 3 introduces the system model, such as wind and energy models, and also the time-dependent cost graph representing the delivery map area. Section 4 defines the Mission-Feasibility Problem (MFP) and observes how the energy optimal solution may vary with the full knowledge of edge costs. Section 5 proposes three algorithms for MFP, while Section 6 evaluates the effectiveness of our algorithms through simulation study. Section 7 offers conclusions with future directions.

2 Related Works

This section provides an overview of the existing literature on the drone delivery problem. We also briefly summarize the single-source shortest path algorithms for dynamic (cost and topology) graphs.

2.1 Drone Delivery Problem

In a recent work, the problem of delivering goods with drones and optimizing energy consumption, has been investigated in [7]. New exploration strategies are also proposed to generate paths from the depot to the customers locations. The estimation of actual energy consumption is hard since modeling the interaction between the drone and the environment can be quite complex. The energy consumption depends on both static and dynamic parameters, like payload, global wind speed and direction. The underlying idea in [7] is to learn a black-box model of energy consumption by exploring the environment while sending drones to deliver goods under varying wind conditions. Initially, each segment traveled by the drone is subject to an unknown cost in terms of energy. After a certain number of consecutive missions, the goal is to learn better paths from the previous routes so as to optimize its range in future missions.

In [8] is investigated what is called soaring, the process of exploiting favorable wind conditions to extend the flight duration. Here the approach involves simultaneously mapping and using a wind field for soaring with an unpowered aircraft. However, an adequate estimate of the wind field is required to generate the energy-gain paths.

Another work [9] considers that the energy required by the drone for completing a given delivery task does not exactly correspond to the energy requested to the battery, since the battery is a non-ideal power supply that delivers power depending on its state of charge. Hence, the energy required by the drone to complete the delivery task does not correspond to the energy requested to the battery because predictions of the total overall flight time may be overestimated. The authors in [9] proposed a battery-aware delivery scheduling algorithm to accomplish more deliveries with the same battery capacity.

Humanitarian assistance with the help of drones has been tackled in [10]. Here the authors considered drone applications in last-mile distribution of multiple packages of lightweight relief or first-aid items, also providing a mathematical optimization model to optimize the delivery process under realistic constraints. The objective of the model is to minimize the total traveling distance of the drone under payload and energy constraints while the recharging stations are installed to allow the extension of the drone's operating distance. The model allows to set the constraints and then minimize the objective function with a mixed integer linear program (MILP) approach.

Finally, in [11] is introduced a drone delivery system for the last-mile logistics of small packages on mixed delivery areas modeled as Euclidean and Manhattan (or EM) grids. The shortest path between two destinations of an EM-grid concatenates the Euclidean and Manhattan distance metrics. Here the drone's mission consists of one delivery for each destination of the grid, and the drone returns to the depot after each delivery. The exact solution of the problem requires logarithmic time in the length of the Euclidean side of the EM-grid. A similar approach taking into account only a subset of the whole set of vertices, is proposed in [12].

2.2 Shortest Path for Dynamic Graphs

There exist a rich body of literature on algorithms developed for single-source shortest path problem, especially for topology dynamic graphs. In [13], the authors proposed an adaptation of Dijkstra's algorithm in the case of edge deletion or insertion. This work is extended in [14] by additionally considering the change of edge costs. In [15], [16] different algorithms are proposed that are an adaption of classic static algorithms for the single-source shortest path due to Dijkstra, Bellman-Ford, and D'Esopo-Pape, and follow an interpretation of the Ball-and-String model. In [17], a new dynamic routing algorithm is proposed that uses multi-path information (i.e., evaluating all possible shortest paths) to quickly compute the new shortest paths in case of edge additions or deletions. We refer to [18] for a survey on the single-source shortest path problem on graphs whose topology and cost change over time.

To the best of our knowledge, the on-line approach for shortest path problem has been pursued mainly for stochastic shortest path problem. Such a problem has been long

studied in the machine learning community, for example, in the framework of *adversarial bandit problem* [19], [20], [21] using a Markov decision problem where an agent moves in an acyclic graph with random transitions. The context is completely different, where the reward seems local, the graph is acyclic, and the weights are stochastic. We believe these methods can be applied to our context in this paper.

3 System Model

This section introduces the relative wind and energy estimation models that will be used to measure the energy spent by the drone while traversing the delivery route. We also define the time-dependent graph that represents the possible drone movements, along with their energy consumption, on the delivery map.

3.1 The Relative Wind Model

First of all, let the *global wind* ω be the wind in the delivering area. Let ω_s and ω_d be, respectively, the speed and the direction of the global wind ω. While at any instant the entire delivery area is subject to the same wind, the conditions of the global wind can change over time.

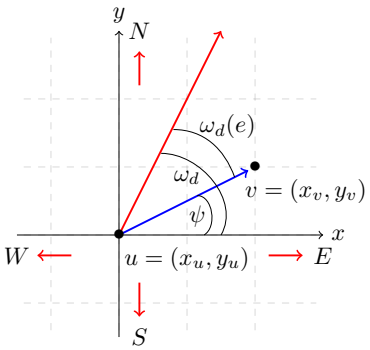


Fig. 1. Relative wind direction $\omega_d(e)$. The edge e is denoted as blue and the global wind direction as red.

To define the *relative wind direction* $\omega_d(e)$ experienced by the drone while traversing edge e, we build a Cartesian coordinate system with origin in u (see Fig. 1). On the straight line $e = (u, v)$ traversed by the drone from $u = (x_u, y_u)$ towards $v = (x_v, y_v)$, the relative wind direction is $\omega_d(e) = \omega_d - \psi$, where $\psi = \arctan(\frac{y_v - y_u}{x_v - x_u})$ is the angle direction of edge e. Flying along routes with different angle directions, the drone experiences different relative wind directions. Note that, the angle directions of the two edges $e = (u, v)$ and $e' = (v, u)$ differ by $180°$. While traversing e, the global wind is assumed stable, and hence the relative wind $\omega_d(e)$ is also stable.

Following the weather reports as in the newspapers, in this paper we simplify the occurrences of the relative wind direction by grouping the values of $\omega_d(e)$ in 8 sectors, $[iL, (i+1)L]$ for $i = 0, \dots 7$, where $L = \frac{360°}{8}$ (see Fig. 2). For reasons that will be clarified in the next section, we group the sectors two by two in 4 classes C_0, C_1, C_2, and C_3 as follows: $C_i = C_i^U \cup C_i^L$, where $C_i^U = [iL, (i+1)L]$ and $C_i^L = [(7-i)L, (8-i)L]$, for $i = 0, \dots 3$. For example, the class C_0 contains the angles in $C_0^U = [0°, 45°]$ and in $C_0^L = [315°, 0°[$. For each class, we select as the representative relative wind the angle with maximum cosine in absolute value. So, C_0 will be represented by $0°$, C_1 by $45°$, C_2 by $135°$, and C_3 by $180°$.

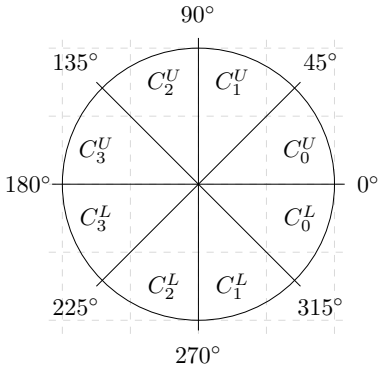


Fig. 2. Simplification of relative wind direction $\omega_d(e)$ in classes.

3.2 The Energy Model

The motion of drones is regulated by physical properties, such as the gravity and forces due to forward motion and wind [22]. The total required thrust T is defined as:

$$T = Wg + F_{\mathrm{D}},$$

where W is the total weight of the drone which includes the *payload weight* m_p, $g = 9.81\,\mathrm{m/s^2}$ is the gravitational constant, and F_{D} is the total drag force. F_{D} depends on the properties of the fluid and on the size, shape, and speed of the drone, and can be estimated by the following:

$$F_{\mathrm{D}} = \frac{1}{2}\rho\, v_a^2\, C_D A,$$

where ρ is the air density, v_a is the drone's relative air speed, $A = \pi R^2$ is the cross sectional area (R is the radius of the rotor), and C_D is the drag coefficient. The air speed v_a also depends on the global wind speed ω_s and the relative wind direction $\omega_d(e)$. The actual air speed v_a can be calculated as:

$$v_a = \sqrt{v_N^2 + v_E^2}, \tag{1}$$

where

$$v_N = v_d - \omega_s \cos(\omega_d(e))$$
$$v_E = -\omega_s \sin(\omega_d(e)),$$

where v_d is the *drone speed*. So, when $\omega_d(e) = 0°$, i.e., when the global wind direction and the drone direction have the same orientation, the East component v_E of the friction is null, and the North component v_N has the minimum value, implying v_a is minimum. Also, when $\omega_d(e) = 180°$, i.e., when wind drone direction are opposite, v_E is null, and v_N is maximum, and thus v_a is maximum. Note that the two sectors grouped in a single class in Section 3.1 span the same air speed values. Indeed, since the angles of the two grouped sectors share the same cosine and have opposite sine, they return the same value in Eq. (1).

Having computed the value of the total thrust T, it is possible to estimate the required power P for a steady flight. With forward motion, P can be calculated from conservation of momentum and thrust T as follows [23]:

$$P = T(v_d \sin(\alpha) + v_i),$$

where $\alpha = \arctan\left(\frac{F_D}{Wg}\right)$ is the pitch angle [23], and v_i is the induced velocity required for a given thrust T. Then, v_i can be obtained by solving the implicit equation [23]:

$$v_i = \frac{v_h^2}{\sqrt{(v_d\cos(\alpha))^2 + (v_d\sin(\alpha) + v_i)^2}},$$

where v_h is the induced velocity at hover [23]:

$$v_h = \sqrt{\frac{T}{2\rho A}}.$$

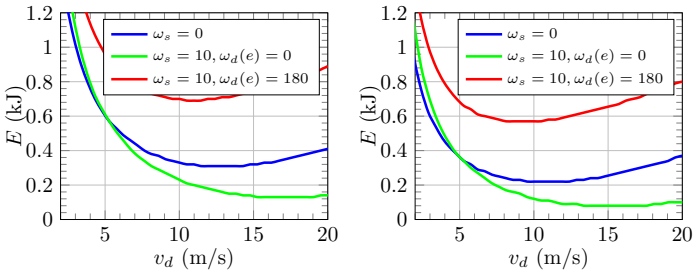
Finally, the energy efficiency, $\mu(e)$, of travel along a segment route e is calculated as the ratio between power consumption P and average ground speed v_d of the drone, i.e.,

$$\mu(e) = P/v_d.$$

Therefore, the energy consumed for traversing one edge e of length $\lambda(e)$ can be expressed as:

$$d(e) = \mu(e)\lambda(e). \tag{2}$$

For completeness, Fig. 3 plots the energy spent to traverse an edge of unitary length by an octocopter (a suitable drone for delivering using proper parameters [24]) under varying wind conditions, payloads, and the drone speed. More weight implies more energy consumption, and tail (head) wind represents less (more) energy drainage. In the rest of this work, however, we fix the payload and the drone speed; and vary only the global wind and consequently the relative wind direction at the drone.



(a) $m_p = 7\,\text{kg}$. (b) $m_p = 0\,\text{kg}$.

Fig. 3. The unitary energy consumption of a drone with varying average speed v_d of the drone.

3.3 Time-Dependent Delivery Network Graph

On a delivery mission, the drone follows the delivery area map that can be modeled by a directed weighted graph, $G = (V, E; d)$, where $d : E \to \mathbb{R}^+$ is the edge cost. The vertex set $V = \{v_0, v_1, \ldots, v_n\}$ represents points in the plane with changing relative wind. For example, the vertices are crossing-points where the drone changes its direction. For simplicity, we assume that the customers are located at the crossing-points. Vertex v_0 is the base of the depot where from the drone starts a delivery mission. The edge $e = (v_i, v_j) \in E$, for $i \neq j$, represents the route traversed by the drone to go from vertex v_i to v_j. Every edge $e \in E$ is associated with the cost function $d(e) = \mu(e)\lambda(e)$ representing the energy spent to traverse the directed edge e of length $\lambda(e)$. The unitary energy $\mu(e)$ is the energy required to traverse one unit length and depends on the payload, drone speed, and relative wind along e, as explained in Section 3.2.

Given two opposite edges $e = (v_i, v_j)$ and $e' = (v_j, v_i)$ and fixing the global wind direction ω_d, $\mu(e) \neq \mu(e')$ because it holds for angle directions $\psi(e') = \psi(e) + 180°$. Thus, $\omega_d(e') \neq \omega_d(e)$. Fig. 4 illustrated an example network graph built on the delivery area map, along with the relative wind on the edges when $\omega_d = 0$.

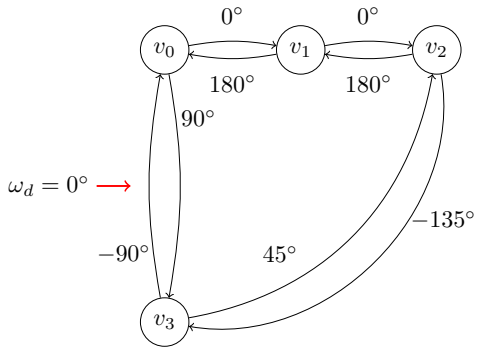


Fig. 4. Relative wind and delivery network when $\omega_d = 0°$.

For any pair of vertices $u, v \in V$, let $\pi(u, v) = (u, \ldots, v)$ denote a generic path (i.e., a sequence of vertices) from u to v. The energy cost of traversing the path $\pi(u, v)$, denoted by $d(\pi(u, v))$, is simply the sum of the costs of all the traversed edges on the given path. Moreover, let $\delta(u, v)$ be the cost $d(\pi^*(u, v))$, where $\pi^*(u, v)$ is a path of minimum cost among all the paths between the vertices u to v.

Since the global wind may vary during the mission, the relative wind and the cost of the edges may also change accordingly. To model this, we observe the cost of the delivery network at regular intervals of time, called snapshots. (Nowadays observations of the wind are available even every few minutes [25].) We associate a new graph G_t to each new wind observation and refer to it as a *snapshot*. In each graph $G_t = (V, E; d_t)$ the set of vertices and edges remain unchanged as t changes, while the costs $d_t(e)$ vary because the global wind ω and the relative wind vary, and so does the energy consumption. Graphs represented by a sequence of static graphs $G_0, \ldots, G_t$ (i.e., snapshots) are known in literature as time-dependant graphs [26]. We will denote the sequence of observed graphs as the *time-dependent cost delivery network graph*, $\mathcal{G}$.

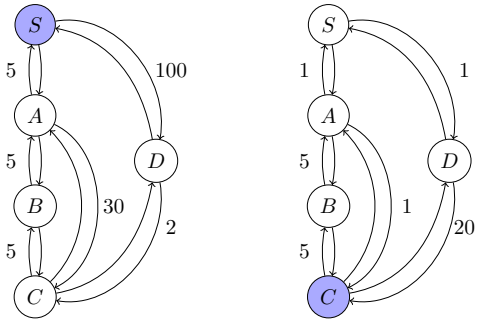
4 Mission-Feasibility Problem

Given as input a time-dependent delivery network graph $\mathcal{G} = \{G_0, \ldots, G_t\}$, a budget B that represents the full charge of the battery of the drone, a payload m_p, a drone speed v_d, and a vertex v as a destination, the *Mission-Feasibility Problem* (MFP) of delivering goods at v with a single drone aims to find if a *mission*, i.e., a complete back and forth from the drone's depot to v, is feasible. That is, MFP aims to find if there is a suitable cycle C in $\mathcal{G}$ that starts at the source node v_0 (the warehouse or depot), reaches the destination node v (the customer), and comes back to v_0 subject to the energy constraint given by B, recalling that the edge costs in $G_0, \ldots, G_t$ of $\mathcal{G}$ may vary during the mission.

Thus, the MFP aims to find two consecutive itineraries, one from the depot (warehouse) to the customer, denoted by $\pi(v_0, v)$, and a returning path $\pi(v, v_0)$ such that the total spent energy $d(C) = d(\pi(v_0, v)) + d(\pi(v, v_0)) \leq B$. Note

that, the departing and returning itineraries have different payloads because, after visiting v, the packages have been delivered, and the drone returns empty (i.e., $m_p = 0$). Moreover, we assume that, when the drone accepts a mission, it is forbidden to land at an intermediate vertex and wait there for sometime. Therefore, as soon as the drone reaches a new vertex, it receives a new snapshot, i.e., new information about the current edge costs of the graph. Then, the drone updates its itinerary towards the destination, but cannot insert any delay. It is worth repeating that the $(i+1)$-th edge in C will be weighted according to G_i, that is, $d_i(e) = \mu_i(e)\lambda(e)$ if the mission started with snapshot G_0. In other words, the consecutive edges of the mission itinerary come from different snapshots of $\mathcal{G}$.

If the wind does not change during the mission, the snapshots remain the same (i.e., $G_0 = \ldots = G_t$), then the MFP has a polynomial time solution computed at the starting of the mission. Indeed, this variant of MFP is strictly related to the Shortest Path Problem (SPP). The minimum cost cycle C^* is obtained by concatenating the two lightest paths $\pi^*(v_0, v)$ and $\pi^*(v, v_0)$ in G_0, under different payloads (the first path with payload while the second path without payload), thus obtaining a cycle of cost $\delta(C^*) = \delta(v_0, v) + \delta(v, v_0)$. Then, the mission towards destination v is feasible if and only if $\delta(C^*) \leq B$.

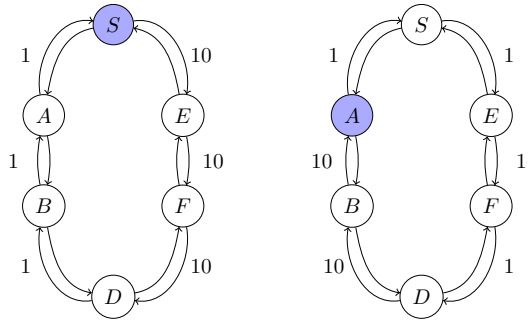


(a) G_t for $t = 0, 1, 2$. (b) G_t for $t \geq 3$.

Fig. 5. A time-dependent graph that whose optimal solution is not a simple cycle.

In general, the edge costs of $\mathcal{G}$ can change at any snapshot G_i, for $0 \leq i \leq t$, and the itinerary computed at time 0 (i.e., snapshot G_0) may become infeasible during the mission. So, the MFP can be solved by solving online the shortest-path problem from v_0 to v and vice versa. However, the shortest path from v_0 to v with full knowledge of $\mathcal{G}$ is not always a simple path, as illustrated in Fig. 5. In this example, the optimal solution for the mission having depot S and destination D is $SABCASD$. Namely, at time 0, the shortest path from S to D is $SABCD$. Since the costs remain unchanged up to time 2 (see Fig. 5a), the drone reaches vertex C. At time 3, the costs change as shown in Fig. 5b. Observe that the shortest path from C to D becomes $CASD$ whose cost, including the cost of $SABC$, is 18. For completeness, note that the optimal solution from S to D includes also the return to D and has overall cost 19.

Let us now consider another example, for which there might not be a feasible solution for MFP when the drone knows only the global wind conditions up to the current time. Consider $\mathcal{G}$ obtained by alternating between the graphs in Fig. 6. Let vertex D be the destination, and S be the drone depot. In other words, the drone receives respectively the costs in Fig. 6a and Fig. 6b at time $2i$ and $2i+1$, where $i \geq 0$. In this case, with full knowledge of $\mathcal{G}$, the best solution of MFP has cost 24 by traversing $SABDEFS$. However, with no full knowledge, the drone can decide to recompute the shortest-path at every new snapshot. At time 0, since the shortest path in G_0 is $SABD$, the drone moves to A. At time 1, the shortest path in G_1 from A to D is $ASEFD$, so the drone moves back to S. By toggling back and forth between G_0 and G_1, the drone will dissipate its energy budget (no matter how big B is) without ever reaching the destination.



(a) G_0. (b) G_1.

Fig. 6. A time-dependent graph that does not admit a feasible solution from S to D.

4.1 MFP Pre-Processing

Given the unweighted graph $G = (V, E)$ with the same V, E for each snapshot in $\mathcal{G}$ and the drone budget B, the goal of the Pre-Processing (PP) Algorithm is to classify the vertices according to their feasibility for a mission with respect to three different policies: (i) vertices in BLACK bring a mission to a failure, i.e., the drone will never be able to deliver the package nor return to the warehouse, and hence the mission is aborted in the beginning; (ii) vertices in GREEN color always allow the drone to safely accomplish its mission; and (iii) vertices in GRAY color imply the mission completion is not guaranteed a priori.

Given the lower and upper bounds on the edge cost as $\epsilon_m = \min\{\mu(e)\}$ and $\epsilon_M = \max\{\mu(e)\}$, we define the lower bound graph $G^L = (V, E^L)$ where $\forall e \in E^L, d(e) = \epsilon_m \lambda(e)$. Symmetrically, the upper bound graph is $G^U = (V, E^U)$ where $\forall e \in E^U, d(e) = \epsilon_M \lambda(e)$. Then, computing the shortest paths $\pi^*(v_0, v)$ and $\pi^*(v, v_0)$ on both the static graphs G^L and G^U, we determine the vertices BLACK for which the mission is never feasible, and also the vertices GREEN for which MFP solution is always feasible, whatever the mission parameters are.

Algorithm 1: PP (G, B)

```
1 G^L ← LOWERBOUNDGRAPH(G, ε_m)
2 G^U ← UPPERBOUNDGRAPH(G, ε_M)
3 foreach v ∈ V \ {v_0} do
4     v.col ← GRAY
5     C ← SHORTESTCYCLE(G^U, v_0, v)
6     if d(C) ≤ B then v.col ← GREEN
7     C ← SHORTESTCYCLE(G^L, v_0, v)
8     if d(C) > B then v.col ← BLACK
```

Algorithm PP, reported in Algorithm 1, works as follows. First build G^L and G^U, and then determine the minimum cost cycle C from/to the source v_0 after visiting the destination v and taking into account the vertex set V and the new edge set E^U (Line 5). The goal is to check if such a cycle C, considered in the worst-case scenario, has a cost $d(C) \leq B$. If so, the drone can safely accomplish the mission and the vertex v is marked as GREEN (Line 6). The same policy can be repeated in a symmetric way considering the best case scenario and assuming the strongest tailwind computing the edge set E^L of the new graph (Line 2). In this case, if the estimated energy consumption is larger than the budget, i.e., $d(C) > B$, the vertex is marked as BLACK (Line 8). All the vertices that do not fall in the above two categories are marked as GRAY.

5 Drone Delivery Algorithms

In this section, we solve the MFP when the destination is a GRAY vertex (the solution is trivial when the destination is GREEN or BLACK). In particular, we describe three new heuristics, called OSP (Static-Offline Shortest Path), DSP (Dynamic Shortest Path), and GDP (Greedy Shortest Path).

Observing that the usual solution for shortest path is a simple path, we aim to find a feasible cycle consisting of two simple paths, one going towards the destination v and another returning to the source v_0. In other words, our solutions forbid to visit the same vertex twice on either the onward or return path, while these two paths may intersect at a common vertex. The rationale behind this requirement is to avoid the situations illustrated in Fig. 5 and Fig. 6, and limiting the total number of edges in these cycles to $2(|V|-1)$.

Applying the MFP to a GRAY vertex, the algorithms can output the following statuses:

- **ABORT (A):** corresponds to a mission in which the drone abandons to fly from the beginning. Only the OSP algorithm can return such status.
- **PARTIAL ABORT (PA):** corresponds to an aborted mission in which the drone could have reached the delivery destination, but not the depot. We distinguish PARTIAL ABORT from ABORT because we want to count how many times the drone could have at least delivered the package. Only the OSP algorithm returns such status.
- **DELIVERED (D):** corresponds to a mission in which the drone reaches the delivery destination, but not the depot. We consider this status as a "half success". All the proposed online algorithms return such status.
- **FAIL (F):** corresponds to a mission in which the drone fails to reach both the delivery destination and the depot. This happens due to the unexpected battery drainage. All the online algorithms return such status.
- **SUCCESS (S):** corresponds to a mission in which the drone serves the customer and gets back to the depot. All the algorithms return such status.

5.1 The OSP Algorithm

The Static-Offline Shortest Path (OSP) algorithm, reported in Algorithm 2, computes a cycle C considering the cost function as input at snapshot zero, i.e., G_0.

Algorithm 2: OSP $(G, B, \text{destination } v)$

```
1  S ← ∅, flag ← false, t ← 0
2  C ← SHORTESTCYCLE(G_0, v_0, v)
3  if d_0(C) ≤ B then
4      foreach e = (v_i, v_j) ∈ C do
5          B ← B − d_t(e)
6          if v_j = v then flag ← true
7          if v_j = v_0 then S ← SUCCESS
8          if B < 0 then
9              if flag = true then S ← DELIVERED
10             else S ← FAIL
11         t ← t + 1
12 else
13     π(v_0, v) ← SHORTESTPATH(G_0, v_0, v)
14     if d_0(π(v_0, v)) ≤ B then S ← PARTIAL ABORT
15     else S ← ABORT
```

Initially, the current status S is unset and the delivery $flag$ is set to false (Line 1). Given the graph G_0 at time zero, the budget B and a destination node v, the OSP algorithm calculates the shortest cycle C (Line 2) taking into account only the current wind. In other words, the drone estimates the route only once at the beginning. If $d(C) \leq B$, the drone starts and follows C. However, during the mission the wind varies as indicated in $\mathcal{G}$; so for each edge the amount of energy actually consumed can be different from the amount accounted at G_0. After each edge, the residual energy budget is computed according to the actual edge cost. In this case, we distinguish the status among SUCCESS, FAIL, and DELIVERED. This algorithm also sets a policy such that if the estimated budget at the beginning is larger than B, the mission is aborted with a final status ABORT (Line 15).

Given a destination, the shortest cycle can be computed (Line 2) by implementing Dijkstra's algorithm. Therefore, the OSP algorithm has time complexity of $\mathcal{O}(|E| + |V| \log |V|)$, considering an implementation with an efficient priority queue data-structure, such as Fibonacci Heap.

Example: Let us illustrate algorithm OSP using Fig. 5. It computes the path $SABCDCBAS$ at snapshot G_0 when the cost is 34. However, after performing the cycle C, the cost becomes 35 to reach D and it is 66 overall to return to the depot. So, for $B < 34$, the mission returns ABORT; for $B = 34$, the mission returns FAIL; and for $35 \leq B < 66$, the mission returns DELIVERED; otherwise it returns SUCCESS. For the example in Fig. 6, recalling that the optimal solution has cost 24 traversing $SABDFES$, invoking the OSP algorithm at snapshot G_0, we get the shortest-path $SABDBAS$ with cost 6. However, while the drone flies, it spends energy 24. So, if the budget is $B < 6$, the mission returns ABORT. If $6 \leq B < 24$, the mission returns DELIVERED. Whereas, if $B \geq 24$, the mission returns SUCCESS.

5.2 The DSP Algorithm

The Dynamic Shortest Path (DSP) is an on-line algorithm that dynamically recomputes the shortest path from each intermediate vertex to the destination. Here no check is done on the cost of the entire path with respect to the residual budget. The recomputed path could be infeasible for the current budget, but the drone moves in any case because it

is the path of minimum cost and any other path will cost more than the one just recomputed.

Algorithm 3: DSP $(G, B, \text{destination } v)$

```
1  S ← ∅, flag ← false, C ← ∅, t ← 0, u ← v_0
2  while S = ∅ do
3      π(u, v) ← SHORTESTPATH(G_t, u, v)
4      e = (u, v_j) ← first edge of π(u, v)
5      B ← B − d_t(e), C ← C ∪ e
6      V ← V \ {u}, E ← E \ Adj(u)
7      t ← t + 1, u ← v_j
8      if B < 0 then
9          if flag = true then S ← DELIVERED
10         else S ← FAIL
11     if v_j = v then
12         if flag = false then
13             flag ← true
14             (V, E) ← (G.V, G.E), u ← v, v ← v_0
15         else S ← SUCCESS
```

Algorithm DSP (reported in Algorithm 3) works as follows. Initially, the current status S is unset, the first snapshot $t = 0$ is set, the cycle C does not contain edges, and the current source for the static path is $u = v_0$ (Line 1). Then, the algorithm starts to repeat the main loop until a final status is declared (Line 2). The first step is to calculate the shortest path $\pi(u, v)$ in G_t from the current vertex in which the drone resides (i.e., u) to the current destination, i.e., v (Line 3). The drone will visit the first edge of such a path (Line 4), and then the remaining budget B and current cycle C are updated (Line 5). The residual graph is also updated removing the vertex u and its edges (Line 6), followed by the update on the currently visited new vertex (Line 7). This step is fundamental to avoid the drone to loop endlessly and never reach the destination. Nevertheless, this step could also lead to failing the mission. The aforementioned loop is done until either the drone reached the customer or the warehouse. However, if for any reason the current budget is exhausted during the mission, a failure is declared (Line 10).

Similar to OSP, the time complexity of the DSP algorithm is $\mathcal{O}(|C| \cdot (|E| + |V| \log |V|))$, since for each step of the cycle we have to recompute a new shortest path.

Example: Referring to the example in Fig. 5, the DSP algorithm computes the solution at snapshot G_0 which is $SABCDCBAS$, and proceeds until it reaches vertex C paying a cost of 15. Then the shortest path towards D is CD because the vertices $\{S, A, B\}$ have been removed. The destination is reached with an overall cost of 35. From D, the entire graph is reactivated, and the shortest path towards S costs 1. Hence, the trip computed by DSP costs 36. Therefore, for $B < 35$, the mission returns FAIL; for $B = 35$, the mission returns DELIVERED; and for $B \geq 36$ SUCCESS is returned. As regards to the example in Fig. 6, in the DSP algorithm the drone moves in A since the shortest path at snapshot G_0 is $SABD$. At time 1, the shortest-path is recomputed, but since the vertex S cannot be selected, the shortest-path is ABD and so the drone moves to B and then to D. Then all the vertices and edges of G are reactivated. At G_3, the shortest path towards S is $DFES$ which remains as the unique path until the end.

5.3 The GDP Algorithm

The Greedy Shortest Path (GDP) algorithm, reported in Algorithm 4, is a very short-sighted algorithm that computes the cycle C considering only the most favorable edge at the current time step, instead of the entire shortest path from the current vertex to the destination as computed by algorithm DSP. In particular, at each time step t, the strategy is simply to reach the neighbor vertex with the lightest edge cost considering graph G_t. There is no guarantee that GDP moves in the direction of the shortest path to reach the destination, but it is locally very thrifty. Similar to the DSP, in this case also we forbid the drone to consider already visited vertices as a next step. The mission is declared FAIL if the residual budget is not enough to traverse any outgoing edge, or because the drone cannot continue its path which may occur if the current vertex has no outgoing edges.

Algorithm 4: GDP $(G, B, \text{destination } v)$

```
1  S ← ∅, flag ← false, C ← ∅, t ← 0, u ← v_0
2  while S = ∅ do
3      e = (u, v_j) ← arg min_{j ∈ Adj(u)} d(u, v_j)
4      if e = ∅ then S ← FAIL
5      else
6          B ← B − d(e), C ← C ∪ e
7          V ← V \ {u}, E ← E \ Adj(u)
8          u ← v_j, t ← t + 1
9          if B < 0 then
10             if flag = true then S ← DELIVERED
11             else S ← FAIL
12         if v_j = v then
13             if flag = false then
14                 flag ← true
15                 (V, E) ← (G.V, G.E), u ← v, v ← v_0
16             else S ← SUCCESS
```

We remark that the difference between algorithms DSP and GDP is that the latter selects only the first edge $e = (u, v_j)$ of minimum cost with respect to the current time step (Line 3), while the former selects the shortest path from u to v. The former, however, utilizes only the first edge of the computed shortest path. If there is no available edge to visit, the mission is declared FAIL (Line 4).

The time complexity of Algorithm GDP is $\mathcal{O}(\max_{v \in V} |Adj(v)| \cdot |C|)$, since at each step we have to recompute the minimum outgoing edge from the current vertex. Without having full knowledge of $\mathcal{G}$, it is not possible to pre-compute this minimum, since the cost of each edge may change as the wind changes.

Example: For the example in Fig. 5, the GDP algorithm returns the same solution as DSP. For this example, none of the algorithms finds the optimal solutions whose cost is 19, as discussed earlier. Now, for the example in Fig. 6, and recalling that the optimal solution has cost 24 traversing $SABDEFS$, the GDP algorithm finds the solution $SABDFES$ whose cost is 24, equaling the optimal solution.

6 Simulation Experiments

We generate at random undirected graph G with $n = 25 + 1$ vertices in the interval $[0, 2]\,\text{km}^2$, modeled according to the

Erdős–Rényi model. Such a graph is characterized by the probability $p(n)$ of the existence of edges. In general, given $p(n) = c\frac{\log n}{n}$ for a specific constant $c > 0$, the probability of G being connected tends to 0 if $c < 1$ and to 1 if $c > 1$ [27] (We discard disconnected graphs). Then, we split each edge in G into two edges e and e' in opposite directions, thus making G a directed graph. The rationale is that we believe at the moment there are no one-way routes in the sky. In other words, if a drone can go from v_i to v_{i+1}, it can also go from v_{i+1} to v_i. The cost are computed as explained in Section 3.1, and they can be different.

Note that we commensurate the width of the area to the drone battery, as we will point out later looking at the number of BLACK vertices. Moreover, we have imagined 25 crossing points, which we believe is reasonable in $4\,\mathrm{km}^2$ of moderate dense urban area, like a middle city. However, note that out algorithms study point-to-point shortest paths, so the network density is not so important, but it was mainly a parameter to generate the Erdős–Rényi graphs.

We considered four different graph layouts (scenarios) by varying the value of $c = 0.5, 1, 1.5, 2$ and generating 50 different random graphs for each c. Notice that for smaller values of c result in longer graph diameter, and also smaller average vertex-degree. When c increases, the graphs become denser with shorter diameter and larger average degree.

As regards to the drone parameters, we set a single speed $v_d = 20\,\mathrm{m/s}$, for the deliveries, we use the maximum allowable payload of $7\,\mathrm{kg}$ for a decent octocopter [24]. The drone has also a maximum energy budget of $B = 5000\,\mathrm{kJ}$. For the wind parameters, we uniformly generate at random the global wind speed ω_s assuming only four distinct values, i.e., $0\,\mathrm{m/s}$ (calm), $5\,\mathrm{m/s}$ (light), $10\,\mathrm{m/s}$ (moderate), and $15\,\mathrm{m/s}$ (fast) which corresponds to the usual speed wind reported in weather forecast [28]. Global wind direction ω_d is generated according to the uniform distribution from $0°$ to $359°$. Recall that for the relative wind direction $\omega_d(e)$, we simplified the occurrences in four classes defined by $\omega_d(e) = 0°$ (tail), $\omega_d(e) = 45°$ (semi-tail), $\omega_d(e) = 135°$ (semi-head), and $\omega_d(e) = 180°$ (head).

Table 1 recaps the parameters used in our experiments.

TABLE 1
Used parameters.

Description	Variable	Unit	Value
number of vertices	n	–	$25 + 1$
constant probability of edge	c	–	$0.5, 1, 1.5, 2$
drone speed	v_d	m/s	20
weight of payload	m_p	kg	$0, 7$
global wind speed	ω_s	m/s	$0, 5, 10, 15$
global wind direction	ω_d	°	$0, \ldots, 359$
relative wind direction	$\omega_d(e)$	°	$0, 45, 135, 180$
max battery budget	B	kJ	5000

In the following, we evaluate the performance of our algorithms by measuring how many delivery destinations fall in each possible output status described in Section 5.

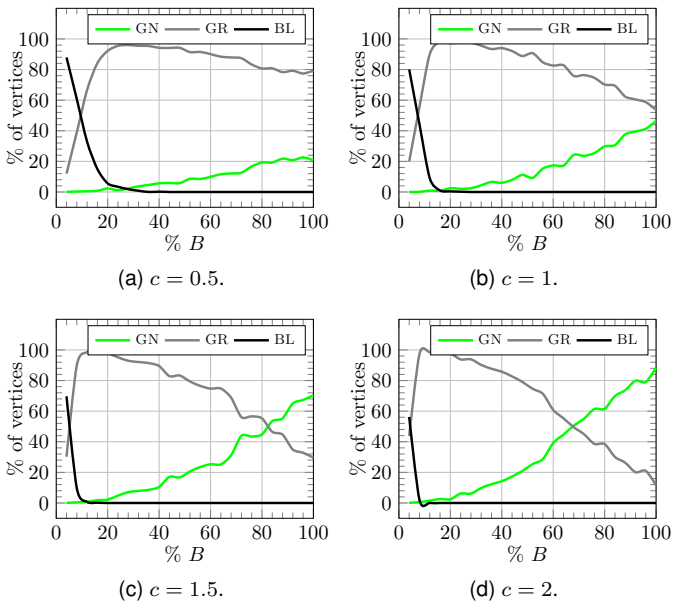
An interesting way to interpret the importance of c in our results is as follows. A mission in a delivery network with a small diameter ($c > 1$) lasts for a few snapshots, and it is considered a *short* mission. A mission in a delivery network with a large diameter ($c < 1$) lasts for several snapshots, and it is considered a *long* mission. Obviously, the wind conditions have more chances to change during long missions than during short missions. So we can expect that algorithm OSP works better when the mission is short than when it is long. Similarly, algorithm GDP, that is the most thrifty one, can face better long missions.

6.1 Pre-Processing Results

Fig. 7 plots the results of our Pre-Processing (PP) algorithm by varying the constant c and the budget B. The number of BLACK vertices is large only for very small budgets, which means that in general the entire area can be served by the drone under good wind conditions. This means the size of the area is adequate to the size of battery.

As expected, increasing the budget also increases the number of GREEN vertices, which can be served whatever the wind is and for both long and short missions.

The value of the budget for which the number of GREEN vertices outdoes the number of GRAY vertices decreases when c increases. That is, the short missions requires less energy than the long ones. We explain this with a generalization of the triangular inequality, because in G^L and G^U the cost are Euclidean (i.e, the cost are distances scaled all by the same constant). Our missions are point-to-point: they connect the depot with the customer. It is well known that the shortest way to connect the depot to the customer distance is to trace a straight line. When instead of going directly we fix an intermediate point (outside the segment), we know that the trip that passes for the intermediate point is longer then the straight line. This reasoning can be repeated inductively increasing the number of intermediate points, and obtaining always path longer than the previous time. When the mission is long, the same two points are connected by a path longer than when the mission is short.



(a) $c = 0.5$. (b) $c = 1$. (c) $c = 1.5$. (d) $c = 2$.

Fig. 7. The colors of vertices for different values of c and budget B after the Pre-Processing (PP) algorithm.

6.2 Performance of Proposed Algorithms

Fixing the value of c, we test our algorithms by considering one mission for each GRAY vertex of the 50 random graphs generated for that c.

For a given algorithm, we vary the budget B (battery 100% fully charged) and plot the percentage of ABORT, FAIL, DELIVERED, and SUCCESS recorded in all the 50 graphs generated for any c value.

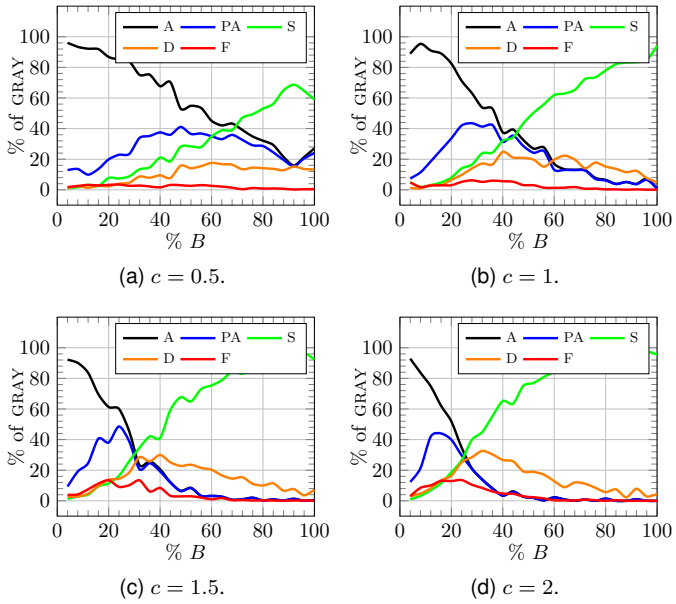


Fig. 8. Each plot shows the results of the Static-Offline Shortest Path (OSP) algorithm for different values of c and budget B.

Fig. 8 shows the result of Algorithm Static-Offline Shortest Path (OSP). Recall OSP selects the shortest cycle under the wind condition G_0, and try to follow that cycle up to completing the mission. OSP is conservative: if the budget is not sufficient to complete the mission under the condition G_0, the mission is aborted. This approach is reasonable if, based on the weather forecast, G_0 is the frequent snapshot in the area. Otherwise, OSP pays a large number of ABORT that is not fully justified. This negative effect of OSP is emphasized when the mission is long (small values of c).

Another interesting aspect is the convergence of the ABORT and PARTIAL ABORT lines for different values of c. A PARTIAL ABORT is one mission such that, if the drone did not give up, it could have reached the customer. In other words, the PARTIAL ABORT curve can be summed to the DELIVERED curve in a less conservative approach. Since the ABORT and PARTIAL ABORT curves converge when c is large, we conclude that the conservative approach stops fewer short missions than long ones.

As expected, in OSP, among the undertaken missions, the percentage of success is very high when the mission is short. The percentage of failure increases when the budget is moderate and the mission is short. We can explain this looking at the length of the edges. When the missions are short, any point of the delivery map is reached with few hops, but long. If the unitary energy efficiency $\mu(e)$ increases, the cost variation is more relevant because $\lambda(e)$ is large. Therefore, we suspect that missions that, at time G_0 was supposed to consume almost all B, have more difficult to tolerate a cost variation of a long edge than a short one.

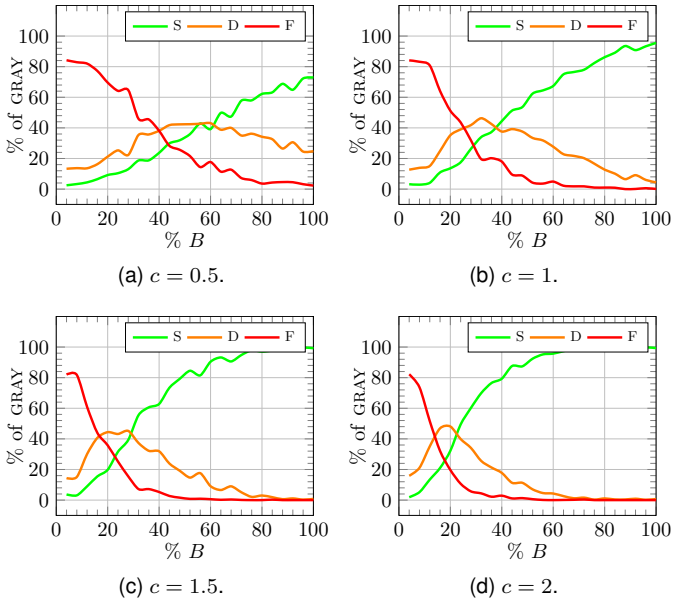
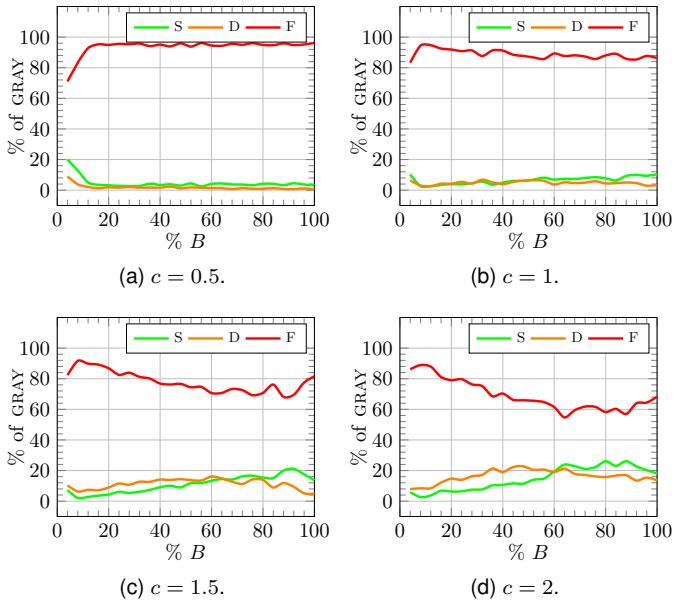


Fig. 9. Each plot shows the results of the Dynamic Shortest Path (DSP) algorithm for different values of c and budget B.

Fig. 9 demonstrates the result of Algorithm Dynamic Shortest Path (DSP). The DSP algorithm, which is computationally most expensive, yields larger percentage of success than OSP. In DSP, when the drone is charged at about 30% and the missions are short, the percentage of successes is higher than 70%; whereas with the same amount of energy, OSP is able to accomplish only 40% of the missions. So DSP requires less energy to accomplish a mission. Nonetheless, the percentage of failing is also higher in DSP than OSP. With a small budget, most of the missions either return FAIL or DELIVERED. It seems that those missions aborted in OSP cannot be completed in DSP either. Summing the DELIVERED and SUCCESS, even with a minimal budget between 20% and 30%, DSP serve about 40% of the customers. In conclusion, the performance of DSP is quite good, which will be even more clear when compared with that of GDP.

Fig. 10 depicts the result of Algorithm Greedy Shortest Path (GDP), which is computationally the less expensive but works poorly. Its performance does not depend on the budget because most often the drone does not continue with the on-line strategy because it reaches a dead end. This is probably because the local choice creates holes in the graph. So, although the algorithm is thrifty, it moves vaguely and disperses its energy. When the mission is short, the failure decreases and the local saving due to short-sighted heuristic alleviates the poor performance. In this case, about 40% of the missions reach the customer. When the mission is long

In summary, both OSP and DSP algorithms perform better on short missions than long ones, but DSP achieves a very good percentage of success even for long missions when the budget is above 50%. Thus DSP copes with wind changes. The OSP runs the fastest. Finally, GDP is extremely fast, but its performance is poor. It could be improved by searching in a sightly larger neighborhood to avoid the path is trapped at dead end.



(a) $c = 0.5$. (b) $c = 1$. (c) $c = 1.5$. (d) $c = 2$.

Fig. 10. Each plot shows the results of the Greedy Shortest Path (GDP) algorithm for different values of c and budget B.

7 Conclusion

In this paper we introduce the MFP which studies whether it is feasible sending a drone to deliver goods to a given customer or not. For this goal, we proposed a framework based on time-dependent graphs to properly model the MFP, and we proposed three algorithms for solving MFP. We have evaluated the performance of our algorithms in terms of percentage of missions completed (with success, delivery only, and failure) in a synthetic set of Erdős–Rényi graphs. In future, it is reasonable, to think to a probabilistic approach based on weather forecast, to guess the most likely wind directions during the mission. With such information, if the mission is too energy-demanding, the drone can delay the starting of a mission waiting for better wind conditions , or the drone can stop and resume the mission at intermediate vertices when there are hopes that the wind will change in a favorable direction.

From theoretical point of view, it would be interesting to find the competitive ratio of the proposed online algorithms. More theoretical results can be derived confining the delivery network to special graph classes, like acyclic graphs.

References

[1] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges," *IEEE Access*, vol. 7, pp. 48 572–48 634, 2019.

[2] F. Betti Sorbelli, S. K. Das, C. M. Pinotti, and S. Silvestri, "Range based Algorithms for Precise Localization of Terrestrial Objects using a Drone," *Pervasive and Mobile Computing*, vol. 48, pp. 20–42, 2018.

[3] F. Betti Sorbelli, C. M. Pinotti, and V. Ravelomanana, "Range-Free Localization Algorithm Using a Customary Drone: Towards a Realistic Scenario," *Pervasive and Mobile Computing*, vol. 54, pp. 1–15, 2019.

[4] B. Rao, A. G. Gopi, and R. Maione, "The societal impact of commercial drones," *Technology in Society*, vol. 45, pp. 83–90, 2016.

[5] P. M. Kornatowski, A. Bhaskaran, G. M. Heitz, S. Mintchev, and D. Floreano, "Last-centimeter personal drone delivery: Field deployment and user interaction," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3813–3820, 2018.

[6] M.-h. Hwang, H.-R. Cha, and S. Y. Jung, "Practical endurance estimation for minimizing energy consumption of multirotor unmanned aerial vehicles," *Energies*, vol. 11, no. 9, p. 2221, 2018.

[7] T. Nguyen and T.-C. Au, "Extending the range of delivery drones by exploratory learning of energy models," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 1658–1660.

[8] N. R. Lawrance and S. Sukkarieh, "Autonomous exploration of a wind field with a gliding aircraft," *Journal of guidance, control, and dynamics*, vol. 34, no. 3, pp. 719–733, 2011.

[9] D. Baek, Y. Chen, A. Bocca, A. Macii, E. Macii, and M. Poncino, "Battery-aware energy model of drone delivery tasks," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2018, pp. 1–6.

[10] B. Rabta, C. Wankmüller, and G. Reiner, "A drone fleet model for last-mile distribution in disaster relief operations," *International Journal of Disaster Risk Reduction*, vol. 28, pp. 107–112, 2018.

[11] L. Bartoli, F. Betti Sorbelli, F. Corò, C. M. Pinotti, and A. Shende, "Exact and Approximate Drone Warehouse for a Mixed Landscape Delivery System," in *2019 IEEE International Conference on Smart Computing, Washington, DC, USA, 12-14 June 2019*, 2019.

[12] F. Betti Sorbelli, F. Corò, C. M. Pinotti, and A. Shende, "Automated Picking System Employing a Drone," in *15th International Conference on Distributed Computing in Sensor Systems, DCOSS 2019, Santorini, Greece, May 29-31, 2019*, 2019, pp. 633–640.

[13] G. Ramalingam and T. Reps, "On the computational complexity of dynamic graph problems," *Theoretical Computer Science*, vol. 158, no. 1-2, pp. 233–277, 1996.

[14] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni, "Fully dynamic algorithms for maintaining shortest paths trees," *Journal of Algorithms*, vol. 34, no. 2, pp. 251–281, 2000.

[15] P. Narváez, K.-Y. Siu, and H.-Y. Tzeng, "New dynamic algorithms for shortest path tree computation," *IEEE/ACM Transactions On Networking*, vol. 8, no. 6, pp. 734–746, 2000.

[16] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng, "New dynamic spt algorithm based on a ball-and-string model," *IEEE/ACM transactions on networking*, vol. 9, no. 6, pp. 706–718, 2001.

[17] V. Eramo, M. Listanti, and A. Cianfrani, "Design and evaluation of a new multi-path incremental routing algorithm on software routers," *IEEE Transactions on Network and service management*, vol. 5, no. 4, pp. 188–203, 2008.

[18] G. Nannicini and L. Liberti, "Shortest paths on dynamic graphs," *International Transactions in Operational Research*, vol. 15, no. 5, pp. 551–563, 2008.

[19] A. György, T. Linder, G. Lugosi, and G. Ottucsák, "The online shortest path problem under partial monitoring," *Journal of Machine Learning Research*, vol. 8, no. Oct, pp. 2369–2403, 2007.

[20] G. Neu, A. György, and C. Szepesvári, "The online loop-free stochastic shortest-path problem." in *COLT*, vol. 2010. Citeseer, 2010, pp. 231–243.

[21] N. Cesa-Bianchi and G. Lugosi, "Combinatorial bandits," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1404–1422, 2012.

[22] M.-D. Hua, T. Hamel, P. Morin, and C. Samson, "A control approach for thrust-propelled underactuated vehicles and its application to vtol drones," *IEEE Transactions on Automatic Control*, vol. 54, no. 8, pp. 1837–1853, 2009.

[23] W. Johnson, *Helicopter theory*. Courier Corporation, 2012.

[24] J. K. Stolaroff, C. Samaras, E. R. O'Neill, A. Lubers, A. S. Mitchell, and D. Ceperley, "Energy use and life cycle greenhouse gas emissions of drones for commercial package delivery," *Nature communications*, vol. 9, no. 1, pp. 1–13, 2018.

[25] Y.-K. Wu and J.-S. Hong, "A literature review of wind forecasting technology in the world," in *2007 IEEE Lausanne Power Tech*. IEEE, 2007, pp. 504–509.

[26] Y. Wang, Y. Yuan, Y. Ma, and G. Wang, "Time-dependent graphs: Definitions, applications, and algorithms," *Data Science and Engineering*, vol. 4, no. 4, pp. 352–366, 2019.

[27] R. Durrett, *Random graph dynamics*. Cambridge university press Cambridge, 2007, vol. 200, no. 7.

[28] LaMMA. Tuscany weather forecast. [Online]. Available: http://www.lamma.rete.toscana.it/en/meteo