An optimization-driven dynamic vehicle routing algorithm for on-demand meal delivery using drones

Yanchao Liu

Department of Industrial and Systems Engineering, Wayne State University, 4815 4th Street Rm 2609, Detroit, MI 48201, USA

A R T I C L E   I N F O

Article history:
Received 30 July 2018
Revised 25 May 2019
Accepted 26 May 2019
Available online 28 May 2019

Keywords:
Drones
Dynamic vehicle routing
Mixed integer programming

A B S T R A C T

As technology continues to improve people’s quality of life, there is a large, unfulfilled market worldwide for on-demand meal delivery services. The competitive edge of the business is foremost sharpened by the agility of the transport system. While lightweight drones are being developed as the next-generation vehicular platform for meal delivery, an efficient fleet operation becomes especially critical. This paper presents a mixed integer programming (MIP) model that comprehensively characterizes all relevant aspects of the business scenario, and proposes an optimization-driven, progressive algorithm for online fleet dispatch operations. Different from typical graph-based formulations of vehicle routing problems, the proposed temporally discrete and spatially continuous MIP formulation endogenously accounts for geometry and mobility and therefore permits dynamic input of order information with arbitrary pickup and delivery locations. The model is augmented with special constraints and an artificial objective function which effectively relay the system states between successive time horizons. The algorithm is validated through simulation case studies and is shown to meet the design objectives.

© 2019 Elsevier Ltd. All rights reserved.

1. Introduction

Morgan Stanley Research estimated that the total U.S. food delivery market could grow to as much as $210 billion over the long term, from around $11 billion today (Stanley, 2016). The growing popularity of on-demand meal delivery service is corroborated by the fast rise of online meal-ordering companies such as GrubHub and Uber Eats. Over the past three years, GrubHub’s stock price more than tripled and Uber Eats’ service quickly proliferated to more than 300 cities across 6 continents. The online-to-offline (O2O) meal ordering and delivery business model has seen a comparable, or perhaps even greater, success in China. Meituan, a leading O2O platform operator in China, estimated that the Chinese market size for meal delivery was 10 billion RMB per month in 2016. Over the past few years, there has been intense competition for market share among a handful of platforms in China, including Meituan, Baidu and Ele.me (which was acquired by Alibaba group in April 2018). Mobile app features such as instant messaging, location sharing and mobile payment have made transactions extremely easy to perform. As a result, more and more people are becoming accustomed to ordering meals online for instant or scheduled delivery. The O2O service model that connects restaurants and customers through information and logistics optimization provides great value for both sides of the market. It helps local restaurants to reach more customers even if they are not located in prime dining locations. As more dining activities take place off-site from where the food is prepared, restaurants can focus more on what they are specialized for, i.e., making healthy and tasty food, rather than being burdened by ancillary operations such as marketing and maintaining a spacious dining area. On the other side of the market, delivery service enables customers to enjoy a great variety of food at home, in the workplace or anywhere of their choice. This convenience also frees up lot of time spent on, e.g., grocery shopping, cooking and running errands through busy traffic, which is not what most working professionals are specialized for or enjoy doing.

At present, delivery trips are performed by humans on road-based vehicles, including automobiles, motorcycles and electric bikes, etc. It is a labor intensive process and labor cost, i.e., wages paid to couriers, constitutes the largest portion of a service provider’s operating expense. New modes and vehicles for meal delivery are being developed, including wheeled robots and unmanned aerial vehicles (UAV or drones) (Barr and Bensinger, 2014; Malcolm and Weise, 2015; Parmar, 2016), see Fig. 1 for illustrations. Using drones for light payload delivery provides a promising solution to many transportation problems. Battery-powered delivery drones are quiet, fast and clean, and unlike ground-based vehicles, they are not slowed by and do not contribute

E-mail address: yanchaoliu@wayne.edu

https://doi.org/10.1016/j.cor.2019.05.024
0305-0548/© 2019 Elsevier Ltd. All rights reserved.
to traffic congestion on roads, allowing for more efficient operations with higher certainty. As communication, control and navigation technologies are becoming mature, aerial logistics powered by drones has attracted great interest from the industry, government and academics in recent years. Topics have covered a wide spectrum including policy debate, market analysis, routing optimization and flight control (Charibi et al., 2016; Mahjiri et al., 2015; de Miguel Molina and Santamarina-Campos, 2018; Rao et al., 2016). In 2018, the U.S. Congress directed the Federal Aviation Administration (FAA) to develop, within a few years, standards for the safe integration of unmanned aircraft into the national airspace system, including solutions for beyond visual-line-of-sight operations, the operation of multiple small unmanned aircraft systems, and authorization for the carriage of property by small unmanned aircraft systems for compensation or hire (a.k.a. drone delivery) (Congress, 2018). This promises significant regulatory advancement in the near future, and also presents an urgent call for large-scale drone fleet management technologies.

This paper investigates the real-time routing of drones for meal pickup and delivery in a dynamic operational environment. In such an environment, meal orders are initiated randomly over time and their pickup and delivery location coordinates are also random and arbitrary across the service region. The fleet operator needs to dispatch delivery drones in near-real-time, in order to make the fastest and most efficient delivery of all orders. Factors such as food types, drones’ speed, carrying capacity and battery consumption also need to be considered. We provide a mathematical model to describe the entire operational scenario, and propose a mixed integer programming (MIP) based heuristic algorithm to facilitate the infinite-horizon decision process. The model formulation captures detailed system states over time and is amenable to drive simulation studies for understanding the steady-state system performance under various design settings. Computational experience and some simulation results are discussed as well.

Drones can be used in many transportation scenarios, from delivering medical supplies to dropping off packages to remote rural customers. We target the on-demand meal delivery scenario not only because it represents a large unfulfilled market, but also because the business reality is complicated enough to encompass many elements that are also likely to be critical in a variety of other applications. Moreover, given the limited carrying capacity of battery-powered drones, we believe that the benefit of aerial delivery systems will only materialize when a large number of such drones are deployed in the city sky. The meal delivery scenario provides a large demand basis to achieve the projected benefit. Most of the business and technical factors discussed in this paper are reality-rooted, as they come from today’s ground-based meal delivery operations.

In the methodological front, this paper contributes to the dynamic vehicle routing problem (DVRP) literature from a unique perspective. The operational scenario considered herein is a strong dynamic system (i.e., no demand is known in advance, see related taxonomy in Larsen et al., 2002) in contrast to ones with lesser degrees of dynamism typically framed as an extension to conventional (static) VRP methods; moreover, compared to existing research on strong dynamic systems such as taxi and dial-a-ride services, this paper uniquely solves a dynamic pickup and delivery problem on an Euclidean plane rather than on a road network. In contrast to a graph-based VRP formulation in which temporal and sequential relations are modeled by continuous quantities, we choose to discretize the temporal dimension in exchange for preserving continuity in the spatial dimension. Such a modeling choice not only permits arbitrary locations of vehicles and orders, but also eases the mathematical modeling process, making complicated constraints intuitive and easy to express. In addition, the solution of the proposed model is easy to visualize and even animate, therefore accessible for business users. Admittedly, the flexibility comes at the cost of some spatial inaccuracy due to approximating Euclidean geometry with linear constraints, and some temporal inaccuracy due to discretization. Nonetheless, such concerns may be minor in practice and can be ameliorated by increasing the approximation and discretization granularity. Overall, the time-discretized, network-free MIP model coupled with the rolling-horizon, optimization-driven solution framework constitutes a first attempt to address the challenge of strong dynamism. More comparative literature review will be made in Section 2. We also propose a progressive heuristic method that produces fast, consistent and good-quality solutions suitable for both real-time operation and offline simulation studies. The modeling approach is flexible enough to encompass more business needs and constraints than those included in the paper, and the solution framework is designed to be robust and adaptable to a wide range of problem size and complexity.

The remainder of the paper is organized as follows. Section 2 briefly reviews the related literature and emphasizes the unique perspective of this work. Section 3 discusses the operational problem in detail and presents a comprehensive formulation of the problem. Both a static model and a dynamic model are presented. An online dispatch algorithm driven by the optimization model is presented in Section 4, followed by simulation cases studies in Section 5. Section 6 concludes the paper and discusses future work directions.

2. Literature review

This paper presents an application of well-known MIP techniques and solution strategies to a transportation problem.
Involving drones. The characteristics of the decision-making problem, i.e., that it prescribes vehicles’ actions and movements over time and that the input information is gradually revealed while the decisions are made, suggest that the work falls in the class of dynamic vehicle routing problems (DVRP) (Pillac et al., 2013) or dynamic one-to-one pickup and delivery problems (PDP) (Berbeglia et al., 2010). Dynamic VRPs have been a lively branch in transportation research for at least thirty years following the visionary work of Psarafitis (1988). However, the meanings, scope and extent of “dynamism” or “dynamicity” have been so diversely interpreted in different methodological frameworks and application contexts that no canonical-form characterization of the dynamic VRP has yet been established (if ever possible) - in the words of Braekers et al. (2016) in their recent review paper on VRP: “Strikingly, no standard problem definitions or formulations are available for dynamic VRPs” and “no benchmark instances are available to test and compare the proposed solution methods objectively.” Notably, the two most influential taxonomic review papers on dynamic VRP, Pillac et al. (2013) and Berbeglia et al. (2010), both adopted the classic graph-based VRP formulation/terminology (see Baldacci et al., 2007; Cordeau et al., 2007; Dantzig and Ramser, 1959; Laporte, 2009; Savelsberg and Sol, 1995 for reference of this formulation) as the very context in which existing (up to their respective publication time) dynamic VRP research was reviewed. In this regard, the time-discretized network-free formulation proposed here offers a fresh perspective for modeling dynamic VRPs, thus to some extent this formulation serves to expand the landscape of the dynamic VRP literature.

There is a rich literature on dynamic VRP. Ghiani et al. (2003) reviewed the main applications that motivated the real-time VRP and pointed out a series of items for future research, including the need for heuristics with look-ahead capabilities, the need for new metaheuristics amenable for scenarios with high dynamism, and the need for algorithmic capabilities to handle varying travel times thus better integrate with intelligent transportation systems (ITS). Yang et al. (2004) modeled a real-time truckload PDP as a minimum-cost assignment problem with timing constraints, in which the truck-job assignments are binary variables and time and delay are continuous variables. Using simulated scenarios consisting of 10 to 20 jobs, the authors tested several rolling-horizon policies based on various heuristics and found that the best policy was the one that took future job distribution into consideration. Ichoua et al. (2000) incorporated the vehicle diversion capability in a well-performed parallel Tabu search strategy (Gendreau et al., 1999), and showed that the ability to dynamically divert en-route vehicles could result in a performance boost in several aspects. Ascheuer et al. (2000) adapted a dynamic machine-job scheduling algorithm in Shmoys et al. (1995) to the online dial-a-ride problem and proposed a new variant that improved the competitive ratio (defined in Section 1.1.2 of Borodin and El-Yaniv, 1998, originated by Karlin et al., 1988; Sletor and Tarjan, 1985). The improvement was achieved via balancing the two simple strategies of completely ignoring new requests until current assignments are finished and completely re-planning whenever a new request arises. Bertsimas and van Ryzin (1991) investigated a dynamic traveling repairman problem (DTRP) that involved stochastic and dynamic demands and queuing phenomena. The authors developed lower bounds on a job’s optimal expected time in the system and evaluated several policies using these bounds. Larsen et al. (2002) proposed the notion of effective degree of dynamism to characterize and classify dynamic VRPs and suggested that the best policy, i.e., the Nearest Neighbor policy, proposed in Bertsimas and van Ryzin (1991) was only suitable for moderate or strongly dynamic systems. Although optimization formulation for dynamic VRP are predominantly graph-based (Berbeglia et al., 2010; Pillac et al., 2013), research on Euclidean-space VRP also exists, e.g., the Euclidean stacker crane problem (Treleaven et al., 2012) and the Euclidean traveling salesman problem (TSP) (Percus and Martin, 1996; Remy et al., 2010), in which the primary focus has been establishing approximation bounds for the optimal tour under certain assumptions about the distribution of origins and destinations in the space.

The aspect of dynamic VRP in which a consensus is least reachable is the objective to be optimized. Depending on what is important for the underlying business scenario, diverse forms of objectives (or objective functions) have been employed in the literature. For a few examples, there are works to balance service costs and travel time (Jaw et al., 1986), to balance service costs and user satisfaction (Madsen et al., 1995), to minimize the weighted sum of total distance and total lateness (Ichoua et al., 2000), and to minimize the average time a demand spends in the system (Bertsimas and van Ryzin, 1991). Larsen et al. (2002) suggested that a sensible objective should strike a balance between minimizing routing cost and minimizing response time, depending on the system’s dynamism. They specifically suggested that the objective should bias toward cost minimization when the system is less dynamic and bias toward response time minimization when the system is more dynamic. We argue that for a dynamic, continuous and real-time business operation, without making any structural assumption it is impractical to aim for “the global” optimal solution against any single objective. Moreover, business goals are multifold. The best system design is usually the outcome of simulating and comparing various design alternatives against an array of metrics of interest and making the best compromise. Therefore, in this paper we pack a hierarchy of business objectives in a single function using weighting parameters of disparate magnitudes. The optimization process only serves to drive the system evolution in the intended direction whereas the objective value is irrelevant for the end goal.

In term of algorithm evaluation criteria, two approaches are prevalent in the literature. Performance can be evaluated analytically if specific assumptions are stipulated, e.g., in Bertsimas and van Ryzin (1991) where demands are modeled as a Poisson process; if such assumptions do not hold, evaluation is typically performed empirically via simulation studies, e.g., in Gendreau et al. (1999), Ichoua et al. (2000) and Yang et al. (2004). This makes sense, as concluded by Jaw et al. (1986): “As in the case of most heuristic algorithms that solve large-scale and complex routing and scheduling problems, it is difficult to state in quantitative terms how good the solutions are. There are no ‘optimal’ solutions to compare with”. It is also uncommon to see lateral comparisons of disparate methodologies on the same problem, perhaps attributed to the lack of a standard problem definition and data sets, as well as the operation-oriented nature of this research field. In this paper, we will use simulation and statistical analysis to evaluate solution performances of various parameters settings within the proposed framework.

Rolling horizon is a widely employed heuristic for solving long-term or infinite-horizon operational problems, not only in vehicle routing (Mitrovi-Mini et al., 2004; Psarafitis, 1988), but also in various other applications of MIP such as energy optimization (Bisci et al., 2017), production planning and scheduling (Li and Ierapetritou, 2010) and process systems engineering (Zamarripa et al., 2016). Though conceptually straightforward, the rolling horizon approach usually requires a meticulous implementation to work correctly, particularly when the system consists of many decision and state variables. We will explain in detail how and why our algorithm and its implementation work, emphasizing the look-far-move-small strategy and the MIP starting point strategy, which can find applications in other complex systems.
Vehicle routing involving UAVs has been studied at an increasing pace in recent years. Dorling et al. (2017) proposed two multitrip VRP models for drone delivery in which the multirotor aerial vehicle’s energy consumption model and its linear approximation were given an emphasized treatment. Simulated annealing method was used for solving the resulting MIP. Murray and Chu (2015) studied the scenario in which a drone works in tandem with a truck to distribute parcels and proposed two new problems, namely the flying sidekick TSP and the parallel drone scheduling TSP. The authors provided mathematical formulations and heuristic solution approaches for both problems. Agatz et al. (2018) modeled the TSP with drone (TSP-D) as an integer program and developed several heuristics based on local search and dynamic programming. Ha et al. (2018) introduced a variant of TSP-D in which the objective is minimizing the operational cost, and proposed a greedy randomized adaptive search procedure to solve the MIP model. Yurek and Ozmutlu (2018) proposed an iterative, two-stage approach to solve the TSP-D and demonstrated effective improvements. Wang et al. (2017) described a particular VRP scenario in which trucks and drones can be co-dispatched for delivery tasks. By comparing the truck-only problem and truck-plus-drone problem, the authors derived theoretical bounds on the benefit from using drones. Sundar and Rathinam (2014) considered fueling constraints in the route optimization of a drone and developed a fast approximation algorithm to solve the resulting MIP. In that problem, the drone has to visit one of several available depots periodically for refueling while performing the task of visiting a set of fixed target locations. In this paper, drones’ battery constraints are considered in the same spirit, though the battery levels are modeled more explicitly and the target locations come up dynamically and arbitrarily.

3. Problem formulation

To maintain generality in the problem description, we will use the terms courier, vehicle and drone depending on contexts, while their meanings should be understood interchangeably in the model formulation. The typical fulfillment process for a meal order is as follows. The customer browses the offering catalog and places a meal order. The order consists of information about the food item(s), the designated restaurant that prepares the food, and the receiving location of the order. It might also include the preferred receiving time, if it is a pre-order ahead of the meal time. Oftentimes, customers do not demand a specific time or time window for delivery. Instead, a placed order is expected to be delivered as soon as possible. Upon receiving the order information, the dispatcher queries a prediction module to estimate the time it takes for the restaurant to prepare the ordered food. Based on the system state, the dispatcher either immediately assigns a courier to pick up the order, or defers the assignment until the order is almost ready for pickup. If there are many available couriers near the order’s origin at the estimated pickup time, a decision is made regarding which courier the order is assigned to. On the other hand, if there is no available courier nearby, one from afar needs to be dispatched which is not only costly but also prone to causing delays. Some orders, such as lunch for a party of 20 people, may exceed the capacity of a single courier, in which case multiple couriers will be dispatched to pick up portions of the order. Once a courier picks up the order, it may continue picking up other orders for consolidated delivery. Couriers’ loading states and locations are constantly monitored, and the couriers can be assigned to take new orders even when they are en-route for a pre-scheduled delivery, as long as the digression makes sense economically and operationally. The travel time from a trip’s origin to its destination depends on the distance, vehicle type, traffic condition and weather condition. From the dispatcher’s point of view, the estimated travel time for any origin-destination (OD) pair is an exogenous parameter which can be instantly queried from a separate navigation module.

3.1. Assumptions

Each drone carries a single insulated food carton which can hold several orders of a standard size. Different types of food (specifically hot meals and cold drinks) cannot be carried in the same carton. The unit of time is one minute. Loading order(s) onto a drone and unloading order(s) off of a drone each takes one or several consecutive whole minute(s), a parameter independent of the drone type, order type and order size. The rate of battery consumption depends on the size of the payload and the drone’s body weight. Batteries can be swapped at charging depots and swapped-on batteries are (driven by the optimization model to be) fully charged. Battery swapping takes one or several whole minute(s) for any drone. The charging depots are not capacitated - each depot has abundant landing spots and a drone can be serviced immediately upon arrival at a depot. We assume that drones have navigational abilities and are capable of following the dispatch commands and autonomously avoiding collisions in the flight path. All meal orders by default have the same priority and in case explicit prioritization is needed to resolve staleming situations, the order that is initiated earlier enjoys a higher priority. To preserve operational reality, the drone’s mobility range in the 2D space is represented by an n-sided polygon where n is a modeling choice. However, $L_1$ norm (the Manhattan distance) is adopted for measuring proximity, i.e., for determining whether a drone has reached a pickup/dropoff/charging location, and for penalizing unnecessary wandering of drones in the objective function. The term order means the on-demand (or takeaway) meal order throughout the paper.

3.2. System components and attributes

Let us start the modeling exercise by dissecting the system components and their relevant behaviors. The performance measures for post-dispatch analysis are given in Section 5.1.3.

3.2.1. Orders

Let $\mathcal{O}$ denote the set of orders. For a realistic dispatch system there is not a definite termination point, so conceptually $\mathcal{O}$ can have infinitely many members. An order $o \in \mathcal{O}$ has the following attributes: the 2D coordinates of its origin and destination ($Ox, Oy, Dx, Dy$), the time stamp of when the order is placed ($InitT$), the size of the order ($Size$), the type of the order ($Type$), and the time (in minutes) it takes to cook the ordered meal ($PrepT$), which varies by restaurant, food and time of the day. The attributes are listed in Table 1.

Based on the basic attributes, we derive several other parameters for use in optimization modeling. Late delivery will be

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Ox_o$</td>
<td>x-coordinate of the origin</td>
</tr>
<tr>
<td>$Oy_o$</td>
<td>y-coordinate of the origin</td>
</tr>
<tr>
<td>$Dx_o$</td>
<td>x-coordinate of the destination</td>
</tr>
<tr>
<td>$Dy_o$</td>
<td>y-coordinate of the destination</td>
</tr>
<tr>
<td>$Size_o$</td>
<td>Size of the order</td>
</tr>
<tr>
<td>$Type_o$</td>
<td>Type of the order (hot meal or cold drinks)</td>
</tr>
<tr>
<td>$InitT_o$</td>
<td>Time at which the order is placed</td>
</tr>
<tr>
<td>$PrepT_o$</td>
<td>Time it takes to cook</td>
</tr>
</tbody>
</table>
penalized in proportion to the extent of the lateness. We adopt a linear lateness function here. Specifically, if an order is delivered at time $t$, it is considered late by the amount

$$\text{Lateness}_{o,t} = t - (\text{Init}T_o + \text{Prep}T_o)$$

(1)

As a component of the objective function, lateness will be minimized, which essentially drives an order to be delivered as soon as possible.

Pickup can only occur after an order becomes ready. A binary parameter is used for indicating whether order $o$ is ready for pickup at time $t$, defined as follows.

$$\text{Ready}_{o,t} = \begin{cases} 
1 & \text{if } t \geq \text{Init}T_o + \text{Prep}T_o \\
0 & \text{otherwise}
\end{cases}$$

(2)

In case of a stalemate (to be discussed later), a weighting parameter is used to arbitrate the priority among different orders. By default, $\text{Priority}_{o} = 1$ for all $o \in \Omega$.

Once an order is delivered, it should be excluded from the dispatch system. The binary parameter $\text{Delivered}_o$ records order $o$’s delivery status ($1$: delivered, $0$: not yet delivered). It is refreshed after each iteration of the dynamic dispatch algorithm.

An order will go through six stages in its life cycle in the delivery dispatch system. Fig. 2 illustrates the states and time between state transitions. The same color code will be used in the animated simulation study.

1. Initiated: the order is placed, being prepared and not ready for pickup. If the ready time is not too far in the future, an order in the initiated state is visible to the dispatcher, so that couriers can be dispatched in advance.
2. Ready: the order is prepared and is waiting for pickup, but not picked up yet.
3. Loading: the order is being loaded by a courier.
4. In-Transit: the order is riding with a courier, being transited to its destination.
5. Unloading: the order has reached its destination and is being unloaded by its courier.
6. Delivered: the order is unloaded and thus is removed from the dispatch system.

3.2.2. Drones

Let $\mathcal{R}$ denote the set of drones for order delivery. We consider three relevant attributes of drones: carrying capacity, battery capacity, and the maximum speed. $\text{Capacity}_r$ represents the number of standard-sized orders drone $r$ can carry concurrently. It primarily depends on the size of the carton used by the drone, which is fixed throughout the operation. We assume that a drone’s energy consumption is the sum over time of the instantaneous payload size. $\text{BatteryCap}_r$ is the drone’s battery capacity, measured by payload minute. For example, a battery capacity of 400 indicates that the drone can sustain 100 min of moving (not including loading and unloading) while carrying 4 standard-sized meal orders, or 80 min of moving while carrying 5 standard-sized orders, ignoring the drone’s body weight and assuming $\text{Capacity}_r = 5$ for this drone. The parameter $\text{BatThresh}_r$ indicates the safety charge level of the drone’s battery. If the battery charge drops below this value, the drone must immediately fly to a charging depot for a battery swap. $\text{MaxSpeed}_r$ represents the maximum distance the drone can fly per minute. It is a catch-all parameter that potentially aggregates many factors such as the drone’s nameplate speed, air traffic and weather condition at the time of operation. $\text{Weight}_r$ represents the body weight of a drone, to be used for calculating the battery consumption rate during flight.

Drones actions constitute the primary decision variables in the dispatch model. Let $\mathcal{A} = \{\text{Load}, \text{Unload}, \text{Move}, \text{Swap}\}$ denote the set of actions a drone can be performing at any time. The elements represent, in sequence, loading order(s), unloading order(s), flying and doing battery swap. A drone is considered to be in the “Move” state as long as it is not loading, unloading or swapping the battery, regardless of its speed. The minimum time it takes the drone to complete action $a \in \mathcal{A}$ is represented by the parameter $\text{MinTime}_a$. In other words, once a drone initiates an action, it must remain in this action for at least $\text{MinTime}_a$ minutes.

3.2.3. Charging depots

Charging depots are places where drones can stop by and swap batteries. During a battery swap, the drained battery is unloaded for a recharge at the depot, and a fully-charged battery is loaded back onto the drone. We do not treat batteries as a scarce resource here and instead assume fully-charged batteries are always available at any depot. Let $\mathcal{E}$ denote the set of charging depots available across the service area. Each depot $e \in \mathcal{E}$ is represented by $\text{Loc}_e$, the x-coordinate, $\text{Loc}_e$, the y-coordinate, and $\text{CRad}_e$, the radius of the depot-centered neighborhood within which a drone is considered to have reached the depot hence can land and do battery swaps.

3.3. Framework overview

The dispatch problem is in the context of an infinite-horizon operation. In a real-world situation, the input data is periodically refreshed as time advances, and new orders keep arriving while existing orders are being delivered. The goal is not to pursue any sort of global optimality, which is difficult to quantify given the multitude of business objectives and the infinite operating horizon. Instead, an optimization model here primarily serves as a formal description of the quantitative relations among data, decisions and the system states. A solution algorithm built upon the optimization model aims to systematically and consistently produce good-quality dispatch decisions to support the continuous operation, while respecting all relevant constraints.

Our solution framework consists of running a limited-horizon dispatch model repeatedly as time advances. Each run optimizes the dispatch decision given the current system state and available demand information. The length of each planning horizon, or the plan-ahead step, is denoted by the parameter $T$, and the length of time advancement in each iteration, or the move-ahead step, is denoted by the parameter $M$. To make the setup meaningful, we must have $T \geq 2$ and $1 \leq M \leq T$. Let time steps be indexed by natural numbers, i.e., $T = \{1, 2, \ldots\}$, then the planning horizon of the first iteration is $\{1, 2, \ldots, T\}$ and that of the second iteration is $\{1 + M, 2 + M, \ldots, T + M\}$, and so forth. In this fashion the planning horizon of, or the set of time points covered by, iteration
\( k \) is given by
\[
\mathcal{T}_k = \{1 + (k - 1)M, 2 + (k - 1)M, \ldots, T + (k - 1)M\}. \tag{3}
\]
An order is visible to the dispatch model only when it has been initiated by the beginning of the planning horizon. In addition, if an order takes too much time to prepare or if it is a pre-order with a specified ready time far in the future, it is unnecessary to expose it to the dispatch model. Let us define the set of active orders to be considered in iteration \( k \) by
\[
\mathcal{O}_k = \{ o \in \mathcal{O} : \text{Init}T_o \leq 1 + (k - 1)M \text{ and Init}T_o + \text{Prep}T_o \leq (k - 1)M + T + L \text{ and Delivery}_o = 0 \}. \tag{4}
\]
where \( L \) is the look-ahead step length, a parameter that controls the pace at which in-process orders are revealed to the dispatch model. A large \( L \) value allows more orders to be visible to the dispatcher at any time, potentially permitting a more efficient dispatch of drones. In the meantime, a large \( L \) will enlarge the model size at each iteration, making it more time-consuming to solve.

Fig. 3 provides a graphical demonstration for \( T = 10, M = 3 \) and \( L = 4 \). The process starts at time 1, at which point the model spanning time points 1–10 is executed. The optimal solution for time points 1–4 is adopted for the first four minutes of operation. The second iteration starts from time 4, and inherits the system states at the time from the previous iteration by fixing variable values to those obtained in iteration 1. Despite the solution for time 5–10 is re-optimized in iteration 2, their values computed from iteration 1 are not entirely discarded, as they will serve to produce an integer feasible incumbent for iteration 2 (to be discussed in Section 4). Three orders are illustrated in the figure. Order 1 is initiated at time 1 and is ready for pickup at time 12. Since \( \text{Init}T_1 = 1 \leq 1 \) and \( \text{Init}T_1 + \text{Prep}T_1 = 12 \leq 14 \), Order 1 is visible in iteration 1 according to formula (4). By the same principle, Order 2 is visible starting from iteration 2 and Order 3 is not visible until time reaches iteration 3.

### 3.4. Mathematical formulation

In what follows we will describe the main constraints and their mathematical formulation. The complete list of variables are given below. For parameters, some have been defined in preceding sections and others will be defined in the context where they are needed.

**Binary Variables**

- \( x_{a,t} \) = 1 if drone \( r \) is performing action \( a \) at time \( t \)
- \( z_{\text{Load}} \) = 1 if drone \( r \) is loading order \( o \) at time \( t \)
- \( z_{\text{Unload}} \) = 1 if drone \( r \) is unloading order \( o \) at time \( t \)
- \( z_{\text{Transit}} \) = 1 if drone \( r \) is carrying order \( o \) at time \( t \)
- \( z_{\text{Depot}} \) = 1 if drone \( r \) is at depot \( e \) at time \( t \)
- \( z_{\text{Food}} \) = 1 if drone \( r \) is carrying food type \( f \) at time \( t \)
- \( z_{\text{Stage}} \) = 1 if order \( o \) is not delivered in the planning horizon
- \( z_{\text{Load}} \) = 1 if order \( o \) is not loaded during the planning horizon

**Free Continuous Variables**

- \( x_{r,t} \) = x-coordinate of drone \( r \) at time \( t \)
- \( y_{r,t} \) = y-coordinate of drone \( r \) at time \( t \)
- \( \mathbf{v}_{r,t} \) = speed of drone \( r \) in the \( x \)-direction at time \( t \)
- \( \mathbf{v}_{r,t} \) = speed of drone \( r \) in the \( y \)-direction at time \( t \)

**Positive Continuous Variables**

- \( \mathbf{s}_{r,t} \) = x-distance between drone \( r \) and the origin of order \( o \) at time \( t \)
- \( \mathbf{y}_{r,t} \) = y-distance between drone \( r \) and the origin of order \( o \) at time \( t \)
- \( \mathbf{s}_{r,t} \) = x-distance btw. drone \( r \) and the dest. of order \( o \) at time \( t \)
- \( \mathbf{y}_{r,t} \) = y-distance btw. drone \( r \) and the dest. of order \( o \) at time \( t \)
- \( \mathbf{s}_{r,t} \) = x-distance btw. drone \( r \) and charging depot \( e \) at time \( t \)
- \( \mathbf{y}_{r,t} \) = y-distance btw. drone \( r \) and charging depot \( e \) at time \( t \)
- \( \mathbf{s}_{r,t} \) = x-distance traveled by drone \( r \) during time \( t \)
- \( \mathbf{y}_{r,t} \) = y-distance traveled by drone \( r \) during time \( t \)
- \( \mathbf{s}_{r,t} \) = x-distance btw. the stage location and destination of order \( o \)
- \( \mathbf{y}_{r,t} \) = y-distance btw. the stage location and destination of order \( o \)
- \( \mathbf{s}_{r,t} \) = distance btw. drone \( r \) and order \( o \) at step \( M \)
- \( \mathbf{s}_{r,t} \) = distance btw. drone \( r \) and depot \( e \) at step \( M \)
- \( \mathbf{s}_{r,t} \) = battery charge level of drone \( r \) at time \( t \)
- \( \mathbf{s}_{r,t} \) = battery shortage of drone \( r \) at time \( t \)

**Objective Function**

\[
\sum_{o \in \mathcal{O}} \sum_{a \in \mathcal{A}} x_{a,t} = 1, \forall r \in \mathcal{R}, t \in \mathcal{T}_k \tag{5}
\]
• Loading (unloading) can happen when the drone is performing
the corresponding action. Note that multiple orders can be
loaded and unloaded at the same time.
\[ z_{\text{load}, t}^{\text{load}} \leq z_{\text{load}, t}, \forall r \in R, o \in O_k, t \in T_k \] (6)
\[ z_{\text{unload}, t}^{\text{unload}} \leq z_{\text{unload}, t}, \forall r \in R, o \in O_k, t \in T_k \] (7)
• The action \( a \), when initiated, must last for a minimum of
\( \text{MinTime}_a \) minutes. For each \( a \in A \), \( t \in T_k \), we define a set
\( \mathcal{T} \) as \( \{ t' : t' > t \text{ and } t' \leq t + \text{MinTime}_a - 1 \} \), and enforce the
following constraint,
\[ z_{\text{a}, t} - z_{\text{a}, t-1} \leq z_{\text{a}, t}, \forall r \in R, a \in A, t \in T_k \setminus \{ 1 + (k - 1)M \}, \] (8)
If an action is initiated at time \( t \), that is, the left-hand side of
(9) is equal to 1, then the drone must be in this action for the
next \( \text{MinTime}_a \) minutes, that is, \( z_{\text{a}, t} \) must also be equal
to 1 for all \( t' \in \mathcal{T} \). The first time point in \( T_k \) is exempted from
this constraint, and will be dealt with separately in the solution
process of the dynamic model (Line 24–27 in Algorithm 4).
• A drone can have a non-zero velocity only when it is in the
“Move” state, and the speed is bounded by its maximum speed.
Technically, a light drone such as a quadcopter is able to change
direction swiftly, so we assume the unit-time flight range to be
a circular area, that is, for each \( r \in R, t \in T_k \),
\[ (x_{\text{a}, t}^{\prime})^2 + (y_{\text{a}, t}^{\prime})^2 \leq (z_{\text{move}, t})^2 \cdot (\text{MaxSpeed})^2. \] (9)
The continuous relaxation of this inequality (i.e., when \( z_{\text{move}, t} \)
is relaxed to be in \([0,1]\)) is a second-order cone constraint.
Therefore, including (9) in the optimization model would lead
to a mixed integer quadratically constrained program (MIQCP).
Several off-the-shelf solvers, such as CPLEX and Gurobi,
can handle convex MIQCPs endogenously. Nonetheless, we still
prefer an MIP formulation for its relative tractability. In
the rolling horizon framework, the dual simplex method can solve
an MIP problem quickly by exploiting the integer feasible starting
point and the advanced basis from the preceding solve. In
contrast, the algorithms used for solving MIQCPs do not have
these advantages. We will illustrate this point in Section 5.2.
To retain linearity, we will use a set of linear inequalities to
approximate the above constraint. Figuratively, we will approxi-
mately carve out the circular area using the intersection of \( n \)
halospheres, each expressed by a linear inequality of the form
\[ a_i \cdot x_{t+1}^{\prime} + b_i \cdot y_{t+1}^{\prime} \leq c_i \cdot z_{\text{move}, t} \cdot \text{MaxSpeed}, \] (9)
where \( a_i, b_i \), and \( c_i \) for each \( i \in [1, \ldots, n] \) are scalar coefficients
whose values depend on the linearization procedure as well as \( n \).
In the Appendix we develop a procedure for calculating the
coefficient values for any given \( n \). We will adopt \( n = 8 \) as the
base setting in numerical experiments.
• A drone’s location at time \( t \) is a result of movement from
its location at \( t - 1 \).
\[ x_{t, t} = x_{t, t-1} + x_{t, t}^{\prime}, \forall r \in R, t \in T_k \setminus \{ 1 + (k - 1)M \} \] (10)
\[ y_{t, t} = y_{t, t-1} + y_{t, t}^{\prime}, \forall r \in R, t \in T_k \setminus \{ 1 + (k - 1)M \} \] (11)
These equations are not defined at the first time point of each
iteration, since the location variables at such times are fixed to
the value computed in the previous iteration. For the first itera-
tion, the drone locations are initialized by the input data, i.e.,
\( x_{t, 1} = \text{Cur}X_r \) and \( y_{t, 1} = \text{Cur}Y_r \).
• In order to suppress unnecessary wandering of idle drones, we
can penalize the drones’ movement (i.e., travel distance) in
the objective function. The actual travel distance of a drone \( r \)
in time \( t \) is equal to the positive square root of the left-hand side of
(9). To avoid introducing quadratic terms in the objective
function, in the MIP formulation we instead penalize the sum of
the absolute movement along the \( x\)- and \( y\)-coordinate. The
coordinate-wise travel distances, \( x_{r, t}^{\text{Dist}} \) and \( y_{r, t}^{\text{Dist}} \), are captured in
(12)–(15).
\[ x_{r, t}^{\text{Dist}} \geq x_{t, t}, \forall r \in R, t \in T_k \] (12)
\[ x_{r, t}^{\text{Dist}} \geq -x_{t, t}, \forall r \in R, t \in T_k \] (13)
\[ y_{r, t}^{\text{Dist}} \geq y_{t, t}, \forall r \in R, t \in T_k \] (14)
\[ y_{r, t}^{\text{Dist}} \geq -y_{t, t}, \forall r \in R, t \in T_k \] (15)
Note that the movement penalty is minor compared to other
objective terms (see the objective function (57a)–(57j) and the
parameter setting in Section 5.1.2), thus the above substitution
does not impose or strongly induce Manhattan type of move-
ments - drones still travel in an approximately Euclidean fashion
as prescribed in (9). As demonstrated in Section 5.2, having
quadratic terms (in spite of its convexity) in the objective func-
tion would significantly prolong the solution time of an MIQCP
(compared to the MIQCP formulation with only the quadratic
constraints (9)). Therefore, the above substitution is practically
necessary.
• An order can be carried by at most one drone at any time.
\[ \sum_{r \in R} z_{r, a, t}^{\text{Transit}} \leq 1, \forall o \in O_k, t \in T_k \] (16)
• An order can be picked up only after it is prepared and ready.
\[ z_{r, a, t}^{\text{load}} \leq \text{Ready}_{r, a, t}, \forall r \in R, o \in O_k, t \in T_k \] (17)
• An order can be picked up by only one drone when the drone is
at the origin of the order. For all \( r \in R, o \in O_k \) and \( t \in T_k \), con-
straints (18) to (21) define the distance and constraint (22) ex-
press the above condition.
\[ x_{r, o, t}^{\text{Transit}} \geq x_{r, o, t} - OX_o \] (18)
\[ x_{r, o, t}^{\text{Transit}} \geq -x_{r, o, t} + OX_o \] (19)
\[ y_{r, o, t}^{\text{Transit}} \geq y_{r, o, t} - OY_o \] (20)
\[ y_{r, o, t}^{\text{Transit}} \geq -y_{r, o, t} + OY_o \] (21)
\[ x_{r, o, t}^{\text{Transit}} + OX_o + (M_{r, o}^{\text{DX}} + M_{r, o}^{\text{DY}}) \cdot z_{r, a, t}^{\text{load}} \leq (M_{r, o}^{\text{DX}} + M_{r, o}^{\text{DY}}) \] (22)
where \( M_{r, o}^{\text{DX}} \) and \( M_{r, o}^{\text{DY}} \) are big-M constants representing the up-
per bound on the distance between a drone’s location and the
origin of the order it could possibly be assigned to carry, in
the \( x\)- and \( y\)-dimension, respectively. Their values can be uni-
versally set to the width and length of the service area under
study, or for tightness of the LP relaxation, they can be updated
at the beginning of each iteration, based on values of \( t \) and
the drone’s maximum speed.
• Likewise, an order can be unloaded by the drone (that carries
it) only when the drone is at the destination of the order. For
all \( r \in R, o \in O_k \) and \( t \in T_k \), we have
\[ x_{r, o, t}^{\text{Dist}} \geq x_{r, o, t} - DX_o \] (23)
\[ x_{r, o, t}^{\text{Dist}} \geq -x_{r, o, t} + DX_o \] (24)
\[ y_{r, o, t}^{\text{Dist}} \geq y_{r, o, t} - DY_o \] (25)
\[ y_{r, o, t}^{\text{Dist}} \geq -y_{r, o, t} + DY_o \] (26)
\[ x_{r, o, t}^{\text{Dist}} + OX_o + (M_{r, o}^{\text{DX}} + M_{r, o}^{\text{DY}}) \cdot z_{r, a, t}^{\text{load}} \leq (M_{r, o}^{\text{DX}} + M_{r, o}^{\text{DY}}) \] (27)

where $M^\text{EX}_k$ and $M^\text{EV}_k$ are big-M constants bearing similar meanings as $M^\text{EX}_r$ and $M^\text{EV}_r$ discussed above.

- The number of orders a drone carries cannot exceed the drone's carrying capacity.
  \[
  \sum_{o \in \mathcal{C}} \text{Size}_o \cdot z_{r,o,t}^{\text{Transit}} \leq \text{Capacity}_r, \quad \forall r \in \mathcal{R}, t \in T_k
  \]  
  \[
  (28)
  \]

- Different types of food (e.g., hot and cold) cannot be carried in the same carton. Constraint (29) indicates that an order is in transit by drone r only when the drone is carrying the type of food that matches that of the order. Constraint (30) restricts the type of food a drone can carry concurrently.
  \[
  z_{r,o,t}^{\text{Transit}} \leq z_{r,o,t}^{\text{Food}}, \quad \forall r \in \mathcal{R}, o \in \mathcal{O}_k \cap \{\text{Type}_o = f\}, t \in T_k
  \]
  \[
  (29)
  \]

\[
\sum_{o \in \mathcal{C}} z_{r,o,t}^{\text{Food}} \leq 1, \quad \forall r \in \mathcal{R}, t \in T_k
\]
\[
(30)
\]

- An order's in-transit state retains until it is loaded or unloaded by a drone.
  \[
  z_{r,o,t}^{\text{Transit}} = z_{r,o,t-1}^{\text{Transit}} + z_{r,o,t}^{\text{Load}} - z_{r,o,t}^{\text{Unload}}, \quad \forall r \in \mathcal{R}, o \in \mathcal{O}_k, t \in T_k \setminus \{1 + (k - 1)M\}
  \]
  \[
  (31)
  \]

- A drone $r$ is considered to be at a charging depot $e$ at time $t$, i.e., $z_{r,e,t}^{\text{Depot}} = 1$, only when their distance is smaller than $\text{CRad}_e$.
  \[
  x_{r,e,t}^E \geq x_{r,t} - \text{LocX}_e,
  \]
  \[
  (32)
  \]

\[
 y_{r,e,t}^E \geq y_{r,t} + \text{LocY}_e,
\]
\[
 (33)
\]

\[
 y_{r,e,t}^E \geq y_{r,t} - \text{LocY}_e,
\]
\[
 (34)
\]

\[
 y_{r,e,t}^E \geq -y_{r,t} + \text{LocY}_e
\]
\[
 (35)
\]

\[
 x_{r,e,t}^E + y_{r,e,t}^E + (M^\text{EX}_r + M^\text{EV}_r) - z_{r,e,t}^{\text{Depot}} \leq (M^\text{EX}_r + M^\text{EV}_r) + \text{CRad}_e
\]
\[
 (36)
\]

where $M^\text{EX}_r$ and $M^\text{EV}_r$ are big-M constants. $\text{CRad}_e$ is a constant representing the radius of the charging depot $e$.

- A drone can swap the battery only when it is at one of the charging depots. For all $r \in \mathcal{R}$ and $t \in T_k$.
  \[
  z_{r,\text{Swap},t} \leq \sum_{e \in \mathcal{C}} z_{r,e,t}^{\text{Depot}}
  \]
  \[
  (37)
  \]

- The battery level of a drone is equal to its battery level at the preceding time point minus the energy consumed in the current period, unless the battery is swapped. The instantaneous battery consumption is proportional to total weight (body weight plus payload size) at the time. Battery shortage is defined as the amount of energy shortage of the service level. For all $r \in \mathcal{R}$ and $t \in T_k \setminus \{1 + (k - 1)M\}$.
  \[
  w_{r,t}^\text{Bat} \leq w_{r,t-1}^\text{Bat} - \sum_{o \in \mathcal{C}} \text{Size}_o \cdot z_{r,o,t}^{\text{Transit}} - \text{Weight}_{r,t} \cdot z_{r,\text{Move},t} + \text{BatteryCap}_{r,t} \cdot z_{r,\text{Swap},t}
  \]
  \[
  (38)
  \]

\[
 w_{r,t}^\text{Bat} \leq \text{BatteryCap}_{r,t}
\]
\[
 (39)
\]

Constraints (5)–(39) define the essence of the business logic. However, several critical constraints as well as an objective function are still missing to complete a working model. How these elements are handled draws the critical line between a static model and a dynamic model. We will discuss them separately.

A static model assumes that all input data are available upfront and the task is to optimally complete the pickup and delivery of all orders in a single time horizon of a fixed length $T'$. In solving a static model there is only one iteration, $k = 1$ with $\mathcal{O}_k = \mathcal{O}$ and $T_k = T' = \{1, 2, \ldots, T'\}$. The parameter $M$ is irrelevant here and can be set to 0. A static model that minimizes the total latency is defined as follows.

\[
\text{Minimize } \sum_{o \in \mathcal{O}} \text{Lateness}_{o,t} \cdot \sum_{r \in \mathcal{R}, r \in T'} \text{Load}_{r,o,t}^{\text{Unload}}
\]
\[
 (40)
\]

\[
\text{s.t. } \sum_{r \in \mathcal{R}, r \in T'} \text{Load}_{r,o,t}^{\text{Unload}} = 1, \quad o \in \mathcal{O}
\]
\[
 (41)
\]

\[
\sum_{r \in \mathcal{R}, r \in T'} \text{Unload}_{r,o,t}^{\text{Unload}} \leq \sum_{r \in \mathcal{R}, r \in T'} \text{Load}_{r,o,t}^{\text{Unload}}, \quad \forall r \in \mathcal{R}, o \in \mathcal{O}, t \in T'
\]
\[
 (42)
\]

\[
\text{and (5) – (31)}.
\]
\[
 (44)
\]

Constraints (41) and (42) require that each order must be picked up and dropped off, respectively, exactly once in the entire planning horizon. A drone can drop-off an order only if it has picked up the order earlier, which is enforced by constraint (43). In addition to the assumption for complete information, this model assumes that all orders can be delivered within $T'$ time periods, and the model will be infeasible if the assumption is not true. It is nontrivial to set an appropriate value for $T'$, since using an unnecessarily large $T'$ as an attempt to ensure feasibility will increase the size of the model thus hurt its solvability. As demonstrated by numerical experiments, the model is extremely difficult to solve even for a small problem instance.

In a dynamic model which the machinery in Section 3.3 is set up for, global-scope constraints such as those in (41)–(43) are more complicated to express. The key challenge is that the whole delivery process of an order may span multiple iterations. The solution framework should work (i.e., produce a feasible solution) regardless of how small the plan-ahead step $T$ is. The values of parameters $T$, $M$ and $I$ should be determined solely based on computational allowance and the real-timeliness requirement. Therefore, we need to implement certain heuristic mechanisms in the model to ensure: (1) proper relay of system states across iterations, and (2) tactical consistency among successive decision epochs. The following constraints will be included in a dynamic model defined on the sets $\mathcal{R}$, $\mathcal{O}$ and $\mathcal{T}_k$.

- Let $\text{Loaded}_{r,o}$ be a binary parameter indicating whether order $o$ has been loaded by drone $r$. Its value is updated at the beginning of each iteration $k$ by the formula $\text{Loaded}_{r,o} = \sum_{t \in \mathcal{T}_k} \text{Load}_{r,o,t}^{\text{Unload}}$, where the hat symbol denotes the optimal value of the corresponding decision variable. With this, the requirement that “an order can be picked up at most once” is given by

\[
\sum_{r \in \mathcal{R}, r \in T'} \text{Load}_{r,o,t}^{\text{Unload}} + \sum_{r \in \mathcal{R}} \text{Loaded}_{r,o} \leq 1, \quad o \in \mathcal{O}_k
\]
\[
(45)
\]

- If an order is in-transit at the end of the planning horizon, it is considered staged. The staging state indicator $\text{Stage}_{o}$ is defined below.

\[
\sum_{r \in \mathcal{R}, r \in T'} \text{Transit}_{r,o,t} \geq \text{Stage}_{o}, \quad o \in \mathcal{O}_k
\]
\[
(46)
\]

Staging will be penalized in the objective function. In the beginning of the next scheduling cycle, staged orders will be fixed to be in-transit by the same drone at the location where it is staged, hence enabling a smooth relay from one iteration to the next. Staging may be unavoidable even though it is penalized. For instance, if an order’s OD distance is greater than the distance traversible by the fastest drone in 10 minutes, the delivery process of the order will inevitably span more than one planning horizon. Penalizing the act of staging alone will not guarantee the order’s final delivery. The drone needs incentive to gravitate
toward the order’s destination even though the order cannot be delivered within the current planning horizon. This is enabled by penalizing the distance gap between the staging location and the order’s destination. The time distances are defined as follows. For all \( r \in R \) and \( o \in C_{o} \),

\[
\begin{align*}
    x_{o}^{\text{Stage}} & \geq [x_{r,a,t} - DX_{o} - (1 - \alpha_{r,a,t}) \cdot M_{\text{DX}_{o}}^{\text{Stage}}] / \text{MaxSpeed}_{r}, \\
    x_{o}^{\text{Stage}} & \geq [-x_{r,a,t} + DX_{o} - (1 - \alpha_{r,a,t}) \cdot M_{\text{DX}_{o}}^{\text{Stage}}] / \text{MaxSpeed}_{r}, \\
    y_{o}^{\text{Stage}} & \geq [y_{r,a,t} - DY_{o} - (1 - \alpha_{r,a,t}) \cdot M_{\text{DY}_{o}}^{\text{Stage}}] / \text{MaxSpeed}_{r}, \\
    y_{o}^{\text{Stage}} & \geq [-y_{r,a,t} + DY_{o} - (1 - \alpha_{r,a,t}) \cdot M_{\text{DY}_{o}}^{\text{Stage}}] / \text{MaxSpeed}_{r},
\end{align*}
\]  

(47) (48) (49) (50)

In the above constraints, the variable \( \alpha_{r,a,t} \) indicates whether the order will be staged at the end of the current planning horizon. If yes, the time distances \( x_{o}^{\text{Stage}} \) and \( y_{o}^{\text{Stage}} \) from the order’s staging location to its destination are defined which will be minimized in the objective; otherwise, those distances are irrelevant hence are not defined due to the big-Ms.

- It may happen that a new order cannot be assigned to any drone in the given planning horizon due to capacity limitation. This situation should be tolerated with penalty. An active order is deemed “not started” if it is neither loaded by, nor in-transit with, any drone. The “not started” state \( z_{o}^{\text{Start}} \) is characterized as follows.

\[
\begin{align*}
    x_{o}^{\text{Start}} + \text{Load} \leq 1, & \forall r \in R, o \in C_{o}, t \in T_{k} \tag{51} \\
    x_{o}^{\text{Start}} + \alpha_{r,a,t} \leq 1, & \forall r \in R, o \in C_{o}, t \in T_{k} \tag{52}
\end{align*}
\]

- During any planning horizon, an active order must have reached one of the three states: (1) unloaded, (2) staged at the end of the horizon and (3) not picked up at all.

\[
\sum_{r \in R, t \in T_{k}} \text{Load}_{r,o,t} + x_{o}^{\text{Stage}} + z_{o}^{\text{Start}} = 1, \forall o \in C_{o} \tag{53}
\]

This is a substitute for the constraint (42) in the static model. The fact regarding whether an order is unloaded (hence delivered) is recorded in the parameter \( \text{Delivered}_{r,o,t} \) during the iteration state updates. By definition, delivered orders are excluded from the set \( C_{o} \).

- If an order’s origin is not reachable by any drone due to the limited planning horizon, the order may never be picked up. To remedy this situation, a “not started” order will be matched to its nearest idle drone (if one is available) and by penalizing their distance, the drone will be driven to the order. An idle drone is one that has neither loaded nor carried any order during the current iteration. Let \( R_{\text{Idle}}^{k} = \{ r \in R : \sum_{o \in C_{o} \cap \text{Load}_{r,o,t}} (\text{Load}_{r,o,t} + \alpha_{r,a,t}) = 0 \} \) be the set of idle drones, updated after each iteration \( k \). From the set \( R_{\text{Idle}}^{k} \) we select the drone that is closest to the origin of the order, and put the drone-order pair in the set \( R_{O} \). The updating procedure for \( R_{O} \) is given in Algorithm 2. In iteration \( k \), for each \((r, o) \in R_{O}\) which is updated at the end of the previous iteration, the distance \( v_{r,o,t}^{E} \) between \( o \) and \( r \) at time \( t' = 1 + (k - 1) M + M \) is characterized by

\[
\begin{align*}
    v_{r,o,t}^{E} = (x_{o}^{0} - x_{r,o,t}^{0}) + (y_{o}^{0} - y_{r,o,t}^{0}) - (M_{\text{DX}_{o}}^{\text{Stage}} + M_{\text{DY}_{o}}^{\text{Stage}}) \cdot (1 - z_{o}^{\text{Start}}). \tag{54}
\end{align*}
\]

This distance will be penalized in the objective function. Note that we single out the time point \( t' \) because that is when the system state is passed to the next iteration, which is what matters to the progression of the algorithm.

- Unlike in the static model where the drones’ energy sustainability is simply ensured by constraints (38) and (39), the battery levels in the dynamic model must be explicitly managed. Specifically, the battery life can practically span a long period and due to the limited-horizon of each iteration, by the time the battery level is low, there might not be enough time for the drone to complete the trip to a depot in any single iteration. Two measures are taken to manage this situation. First, we set a safety battery threshold \( \text{BatThresh}_{r} \) for each drone \( r \), and penalize the battery shortage relative to this threshold. The parameter \( \text{BatThresh}_{r} \) is set (at least) equal to the amount of energy needed to fly to the nearest charging depot from anywhere in the service region, assuming the drone is carrying a full payload. The shortage amount \( v_{r,t}^{\text{Short}} \), which is a positive variable for each \( r \in R \) and \( t \in T_{k} \), is defined as

\[
\begin{align*}
    v_{r,t}^{\text{Short}} \geq \text{BatThresh}_{r} - v_{t}^{\text{Bat}}. \tag{55}
\end{align*}
\]

The second measure is to drive any low-battery drone to a charging depot directly, via heavily penalizing the distance to the depot. To do this, the optimal value \( v_{r,t}^{\text{Short}} \) is checked at the end of each iteration \( k \). For any drone \( r \in R \) if \( \sum_{t \in T_{k}} v_{t}^{\text{Short}} > 0 \), the charging depot \( e \) nearest to the drone’s location at time \( t' = 1 + (k - 1) M + M \) will be assigned as the next destination of the drone. The set of such drone-depot pairs \( R_{E} \) is updated at the end of each iteration \( k \) by Algorithm 3 and will be used in the subsequent iteration. In any iteration \( k \), the drone-depot distance \( v_{r,e}^{E} \) for each \((r, e) \in R_{E} \) at time \( t' = 1 + (k - 1) M + M \) is characterized by

\[
\begin{align*}
    v_{r,e}^{E} = t_{r,e,t}^{E} + t_{r,e,t}^{E} - (M_{t_{r,e,t}^{E}} + M_{t_{r,e,t}^{E}}) \cdot \sum_{t \in T_{k}} z_{r,\text{Swap},t}. \tag{56}
\end{align*}
\]

and will be penalized in the objective function. The last term in the right side of (56) serves to relax the penalization if the battery is swapped in the current iteration.

Minimize

\[
\begin{align*}
    \sum_{r \in R, t \in T_{k}} \frac{\alpha_{r,t}^{\text{Short}} \cdot v_{r,t}^{\text{Short}}}{\text{Capacity}_{r}} \tag{57a} \\
    + \sum_{r \in R, t \in T_{k}} \frac{\alpha_{r,t}^{\text{Swap}} \cdot v_{r,t}^{\text{Swap},t}}{\text{MaxSpeed}_{r}} \tag{57b} \\
    + \sum_{t \in T_{k}} (\max(\text{Lateness}_{o,t}^{\text{Start}}) + \beta_{\text{Pickup}}) \cdot z_{o}^{\text{Start}} \tag{57c} \\
    + \sum_{t \in T_{k}} (\max(\text{Lateness}_{o,t}^{\text{Stage}}) + \beta_{\text{Stage}}) \cdot x_{o}^{\text{Stage}} \tag{57d} \\
    + \sum_{t \in T_{k}} (\text{Lateness}_{o,t}^{\text{Start}} \cdot \sum_{r \in R} (\text{Load}_{r,o,t} + \alpha_{r,a,t})) \tag{57e} \\
    + \sum_{t \in T_{k}} \frac{v_{r,t}^{\text{Dist}}}{\text{MaxSpeed}_{r}} \tag{57f} \\
    + \sum_{t \in T_{k}} \text{Priority}_{o,t} \cdot (x_{o}^{\text{Stage}} + y_{o}^{\text{Stage}}) \tag{57g} \\
    + \sum_{r \in R, t \in T_{k}} \frac{\alpha_{r,t}^{\text{Dist}} \cdot v_{r,t}^{\text{Dist}}}{\text{MaxSpeed}_{r}} \tag{57h} \\
    + \sum_{t \in T_{k}} z_{r,\text{Swap},t} \tag{57i} \\
    - \sum_{r \in R, t \in T_{k}} \frac{\alpha_{r,t}^{\text{Bat}} \cdot v_{r,t}^{\text{Bat}}}{\text{Capacity}_{r}} \tag{57j}
\end{align*}
\]

Finally, we need an objective function that drives the iterative solution toward achieving an ultimate goal, that is, delivering all orders as soon as possible and as efficiently as possible. From the business perspective, prioritization of different objectives is inherently hierarchical - important goals are considered first, then come the secondary goals and so forth. In designing the objective function, we follow the hierarchical structure to mitigate different objectives, as shown in Fig. 4. In particular, we consider three tiers of objectives. The first and foremost tier is ensuring safety - no
drone should run out of battery any time. This is enabled by two terms in the objective function: minimize the distance between drone-depot pairs that are in $\mathcal{R}_E$ and penalize the battery shortage at all time to hasten the battery swap in case of a shortage. The second tier is ensuring fast delivery of orders, which contributes four terms to the objective function: minimize the number of “not started” orders, minimize the number of staged orders, expedite the unload action and arrange a pickup for out-of-range orders. The tertiary objective is maximizing efficiency, which consists of avoiding unnecessary movement and maintaining tactical consistency among iterations.

The objective function is given in (57a)–(57j). The terms are in the same order as they appear in the hierarchy of Fig. 4. The top-tier terms, i.e., (57a) and (57b), account for the battery shortage and the distance gap between low-battery drones to the charging depot, respectively. The factors $\alpha^{\text{Short}}$ and $\alpha^{\text{Swap}}$ are for elevating the relevant importance of the corresponding objectives. Let us look at the second-tier objective of minimizing lateness. For any order $o \in \mathcal{O}_k$, it is either delivered during $\mathcal{T}_k$ or not. The latter case further leads to two possible scenarios: the order is either staged at the end of $\mathcal{T}_k$ or not picked up at all. Furthermore, for a “not started” order, it is either visible to some drone or not visible to any drone. Therefore, the order’s lateness is captured in one of four terms of (57c)–(57e). Specifically, the summand in (57e) records the lateness, in terms of the number of time periods, of the order’s pickup and delivery, if such an event occurs at all during $\mathcal{T}_k$. If the order is staged at the end of $\mathcal{T}_k$, i.e., $Z^{\text{Stage}} = 1$, then the lateness is at least $\max_{t \in \mathcal{T}_k}(\text{Lateness}_{o,t})$. In addition, an extra penalty $\beta^{\text{Stage}}$ is included to account for the lateness to be incurred in future time periods. Similarly, if the order is not started, i.e., $Z^{\text{Short}} = 1$, the lateness cost is $\max_{t \in \mathcal{T}_k}(\text{Lateness}_{o,t})$ plus an even greater penalty value of $\beta^{\text{Pickup}}$. Finally, if the order is too far away from all drones to be visible, an idle drone will be assigned to gravitate toward its origin for pickup, as captured in the term (57f). The third tier of the objective consists of terms (57g)–(57j). Scaling factors are used for approximately converting the magnitude of the corresponding term to the order of traveling time. The factor $\alpha^{\text{Dist}}$ is used for suppressing the relative importance of traveling, i.e., making distance minimization the secondary goal. As long as $\alpha^{\text{Dist}}$ is positive, drones will not wander around. The term (57i) associates a small cost to the battery swap action to avoid senseless battery swaps, i.e., an idle drone sits at the depot and repeatedly swaps its battery for nothing. The term (57j), with a small positive factor $\alpha^{\text{Flat}}$, expels the leeway brought by the inequality (38) hence ensures that the battery level is topped up to BatteryCap$_i$ after a swap, rather than to some less-than-maximum levels.

The dynamic dispatch model for iteration $k$ is an MIP composed of the objective function (57a)–(57j) and constraints (5)–(39) and (45) and (56). Let us call this model DISPATCH($k$).

Recall that the objective term (57h) together with the constraints (12)–(15) constitutes a linear approximation for movements in the Euclidean space. If the Euclidean metric were to be preserved in an MIQCP formulation, the term (57h) could be replaced by

$$\sum_{r \in \mathcal{R}, j \in \mathcal{T}_k} \alpha^{\text{Dist}} \cdot \frac{\left( (x'_{r,t})^2 + (y'_{r,t})^2 \right)}{\text{MaxSpeed}_{o,r,t}^2}. \quad (57h)$$

We will experiment this variant (namely MIQCP2) in Section 5.2.

4. The online dispatch algorithm

In Section 3, we have formulated both a static model and a dynamic model for drone dispatch. We now present a progressive algorithm for online dispatch operations using the dynamic model DISPATCH($k$) as the essential building block.

When a drone carries multiple orders that cannot be delivered within the current iteration, the drone will be simultaneously attracted to the destination of all these orders according to the objective term (57g). To avoid the potential staleming situation, artificial priorities (i.e., Priority$_o$) should be assigned so as to articulate a clear priority rank among orders loaded on the same drone. We adopt a first-come first-serve principle to resolve the conflict, as implemented in Algorithm 1. Orders with a smaller InitT$_o$ has a higher priority for delivery.

**Algorithm 1** Update Priorities for Loaded Orders.

1: **procedure** UPDATEPriority($k$)
2: Priority$_o \leftarrow 1, \forall o \in \mathcal{O}_k$
3: for $r \in \mathcal{R}$ do
4: \quad Let $\mathcal{O}_r = \{o : \text{Loaded}_r,o = 1\}$, sorted by InitT$_o$ in descending order
5: \quad $i \leftarrow 0$
6: \quad for $o \in \mathcal{O}_r$ do
7: \quad \quad Priority$_o \leftarrow (\text{Capacity}_o)^i$
8: \quad \quad $i \leftarrow i + 1$
9: \quad end for
10: end for
11: **end procedure**
Algorithms 2 and 3 are used for updating the sets \( R O \) and \( RE \) for the objective terms (57f) and (57b), and the idea is outlined where constraints (54) and (55) are discussed. Note that in Line 4 of Algorithm 2, the condition \( \sum_{r \in R, t_0 \leq t \leq t_f} g_{\text{load}}^{\text{a, t}} = 1 \) is not equivalent to the expression \( g_{\text{NSStart}}^o = 0 \). The former means not only that order \( o \) is scheduled to be picked up by some drone but also that the scheduled pickup will actually be executed. In contrast, the counterfactual latter expression only indicates that order \( o \)'s pickup is scheduled within the current planning horizon, but if the pickup time was beyond the move-ahead step, it is not guaranteed to be executed. Using the latter condition instead of the former would result in staleness in some cases.

Algorithm 2: Update Drone-Order Bundle \( RO \).

1: \textbf{procedure} UpdateRO(k)
2: \hspace{1em} t_0 \leftarrow 1 + (k-1)M, t' \leftarrow 1 + (k-1)M + M
3: \hspace{1em} for \( (r, o) \in RO \) do
4: \hspace{2em} if \( \sum_{r' \in R, t_0 \leq t' \leq t_f} g_{\text{load}}^{\text{a, t'}} = 1 \) or \( r \notin R_k^\text{idle} \) then
5: \hspace{3em} RO \leftarrow RO \setminus \{(r, o)\}
6: \hspace{1em} end if
7: \hspace{1em} end for
8: \hspace{1em} end for
9: \hspace{1em} for \( o \in O_k \cap \{g_{\text{NSStart}}^o = 1\} \) do
10: \hspace{2em} \( d_{\text{Min}} \leftarrow \infty \), \( r^* \leftarrow \emptyset \)
11: \hspace{2em} for \( r \in R_k^\text{idle} \) do
12: \hspace{3em} if \( d_{r, o} \leq d_{\text{Min}} \) then
13: \hspace{4em} \( d_{\text{Min}} \leftarrow d_{r, o} \), \( r^* \leftarrow r \)
14: \hspace{1em} end if
15: \hspace{1em} end for
16: \hspace{1em} if \( r^* \neq \emptyset \) then
17: \hspace{2em} RO \leftarrow RO \cup \{r^*, o\}
18: \hspace{1em} end if
19: \hspace{1em} end for
20: \textbf{end procedure}

Algorithm 3: Update Drone-Depot Bundle \( RE \).

1: \textbf{procedure} UpdateRE(k)
2: \hspace{1em} t_0 \leftarrow 1 + (k-1)M, t' \leftarrow 1 + (k-1)M + M
3: \hspace{1em} for \( (r, e) \in RE \) do
4: \hspace{2em} if \( \sum_{t_0 \leq t \leq t_f} g_{\text{SwapInter}}^t \geq 1 \) then
5: \hspace{3em} RE \leftarrow RE \setminus \{(r, e)\}
6: \hspace{2em} end if
7: \hspace{2em} end for
8: \hspace{2em} for \( r \in [R : (r, e') \notin RE, \forall e' \in E \text{ and } \sum_{t_0 \leq t \leq t_f} g_{\text{Short}}^t (r, e') > 0] \) do
9: \hspace{3em} \( d_{\text{Min}} \leftarrow \infty \), \( e^* \leftarrow \emptyset \)
10: \hspace{3em} for \( e \in E \) do
11: \hspace{4em} \( d_{r, e} \leftarrow |x_{r, e} - \text{LocO}_k| + |y_{r, e} - \text{LocY}_e| \)
12: \hspace{4em} if \( d_{r, e} \leq d_{\text{Min}} \) then
13: \hspace{5em} \( d_{\text{Min}} \leftarrow d_{r, e} \), \( e^* \leftarrow e \)
14: \hspace{4em} end if
15: \hspace{4em} end if
16: \hspace{4em} if \( e^* \neq \emptyset \) then
17: \hspace{5em} RE \leftarrow RE \cup \{r, e^*\}
18: \hspace{5em} end if
19: \hspace{3em} end for
20: \hspace{2em} end for
21: \hspace{1em} end for
22: \textbf{end procedure}

Algorithm 4: Dynamic Dispatch Algorithm for Order Pickup and Delivery.

1: \textbf{procedure} Dispatch(\( R, E \))
2: \hspace{1em} k \leftarrow 1
3: \hspace{1em} while (true) do
4: \hspace{2em} Stream in new orders initiated in the past \( M \) minutes, add them to \( O \)
5: \hspace{2em} Construct \( T_k \), \( C_k \), and \( T_{a,t} \), \( \forall a \in A, t \in T_k \), \( t_0 \leftarrow (k-1)M + 1 \)
6: \hspace{2em} Update Latenness_{\text{Min}} and \text{Ready}_t \, \forall 0 \leq o \in C_k, t \in T_k
7: \hspace{2em} Loaded_o \leftarrow 0, \text{Delivered}_o \leftarrow 0 \, \forall 0 \leq o \in C_k
8: \hspace{2em} if \( k = 1 \) then
9: \hspace{3em} \text{Delivered}_o \leftarrow 0, \forall o \in C_k, \text{Loaded}_o \leftarrow 0, \forall r \in R, \, o \in C_k
10: \hspace{3em} \text{Fix } \text{Load}_t \leftarrow 0, \forall r, o
11: \hspace{3em} else
12: \hspace{4em} \text{Loaded}_o \leftarrow 1, \forall (r, o) \in R \times C_k \cap \left\{(r, o) : \text{MinTime}_o \geq 1 \right\} or \( \text{Load}_t = 1 \) \right\}
13: \hspace{4em} \text{Delivered}_o \leftarrow 1, \forall o \in C_k \cap \{\sum_{t_0 \leq t \leq t_f} g_{\text{load}}^{\text{a, t}} = 1 \}
14: \hspace{4em} \text{end if}
15: \hspace{4em} \text{end for}
16: \hspace{4em} \text{end else}
17: \hspace{4em} if \( r, o \) such that \( \text{MinTime}_o > 1 \) and \( \text{R}_{a,t} = 1 \) do
18: \hspace{5em} \( t' \leftarrow \max[t : (t_0 - \text{MinTime}_o) \leq t \leq t_0 \text{ and } \text{R}_{a,t} = 1 \}
19: \hspace{5em} \text{end if}
20: \hspace{4em} \text{end for}
21: \hspace{4em} \text{end if}
22: \hspace{4em} \text{end for}
23: \hspace{4em} \text{end for}
24: \hspace{4em} \text{end for}
25: \hspace{4em} \text{end for}
26: \hspace{4em} \text{end if}
27: \hspace{4em} \text{end for}
28: \hspace{4em} \text{end if}
29: \hspace{4em} \text{end if}
30: \hspace{4em} \text{end if}
31: \hspace{4em} \text{end if}
32: \hspace{4em} \text{end if}
33: \hspace{4em} \text{end if}
34: \hspace{4em} \text{end if}
35: \hspace{4em} \text{end if}
36: \hspace{4em} \text{end if}
37: \hspace{4em} \text{end if}
38: \hspace{4em} \text{end if}
39: \hspace{4em} \text{end if}
40: \hspace{4em} \text{return} All variable values
41: \textbf{end procedure}

Algorithm 4 is the dynamic dispatch algorithm suitable for online use or for offline simulation studies. The algorithm takes as static input a set of drones \( R \) and a set of charging depots \( E \), while new orders will be gradually revealed as time advances and hence the set of orders is periodically augmented. The algorithm starts at iteration \( k = 1 \) and keeps looping, until a user-defined termination condition is met (line 32–34). In a simulation run, the algorithm execution can be terminated when the set of active orders \( O_k \) becomes empty, i.e., when all active orders are delivered. The solution consistency across iterations is enabled by fixing the variable values at the first time point \( t_0 \) of each iteration to values obtained from the previous iteration (lines 15–16) or to the initial
values read from input data (lines 10–11). Lines 17–22 are used for setting integer feasible starting points for the newly revealed M periods, which, together with the current solution values for the overlapping time periods, will form an incumbent solution for the model DISPATCH(k) in line 30 (with solver option mipstart=1). The incumbent is critical in some cases for expediting the MIP solution process. Lines 24–27 are used for relaying the minimum time constraint (8) between successive iterations. Specifically, at the beginning of an iteration (i.e., time $t_0$) if a drone is found to be performing an action which has a minimum time requirement, then the action must continue until its duration reaches the required value. After solving the model, decision variables for the first $M$ periods are output (either recorded for future analysis or output for real-time execution) and relevant system states are updated in preparation for the next iteration (lines 27–30).

5. Numerical experiments

The performance of this algorithm can be examined from three aspects. First, the algorithm’s execution time must meet the real-timeliness requirement. In the context of Algorithm 4, each iteration should not take more than $M$ minutes of clock time. Second, the algorithm should not report an “infeasible” solution or be trapped in any type of stalematating situation. Three, the long-run dispatch performance should be satisfactory. We will run several simulated cases to demonstrate the algorithm’s performance from these aspects, and showcase its usage as a simulation tool for system design analysis.

5.1. Computing environment and parameters

5.1.1. Computing environment

The algorithms are coded in GAMS modeling language, version 25.1.3. Computations were performed on a Dell Precision Tower 3420 with Intel Core i7-7700 CPU at 3.60 GHz and 16GB RAM on Windows 10 operating system. The MIP solver is CPLEX, version 12.8.0.0 that came with GAMS. The following CPLEX options were used: mipstart=1, names=no, scalind=1, numericalemphasis=1, eprhs=1e-9, epopt=1e-9, epint=0; and the following GAMS options were used: threads=0, optca = 1e-3. Rationales for using these options are discussed in Section 5.4.

5.1.2. Input parameters

Throughout the experiments, values of the constants used in the model are listed below.

Big-M constants:

$M^W_{i,r}$ = max(OX, 500 – OX), $M^D_{r,o}$ = max(OY, 500 – OY), ∀r, o

$M^L_{i,r}$ = max(DX, 500 – DX), $M^R_{r,o}$ = max(DY, 500 – DY), ∀r, o

$M^F_{X,Y} = \text{max}(\text{LocX}, 500 – \text{LocX}),$ ∀r, e

$M^{\text{Util}}_{i,r} = \text{max}(\text{LocY}, 500 – \text{LocY})$, ∀r, e

Weighting factors in the objective function: $\alpha^{\text{Short}} = 100$, $\alpha^{\text{Swap}} = 1$, $\beta^{\text{Dist}} = 0.1$, $\alpha^{\text{Bat}} = 0.001$, $\beta^{\text{Pickup}} = 1000$ and $\beta^{\text{Stage}} = 100$. The vicinity radius CRad (used in constraint (36) is set to 20 for each charging depot $e$. The look-ahead step $L$ is set to 5 min. In the base case, the number of polygonal sides $n$ is set to 8 and the minimum action time $\text{MinTime}_a$ is set to 1 for all $a \in A$, while their effects are probed statistically in Section 5.3.

5.1.3. Performance measures

For orders, Time to Delivery (TTD) is defined as the time difference between when an order is delivered and when it is ready for pickup. The inherent transit time from an order’s origin to destination contributes to the irreducible part of TTD, while the time spent on waiting or taking detours accounts for the reducible part of TTD, which we call delay. Delay is defined as TTD minus the shortest travel time between the order’s origin and destination on the fastest vehicle. We will report the average Delay over all orders in the experimental results.

For vehicles, a drone’s instantaneous Capacity Utilization is the ratio between the total size of loads and the drone’s carrying capacity. It is only defined on those time instants when the drone is flying, i.e., the speed is positive. The long-term average utilization is the instantaneous utilization averaged over the qualified time periods. The Residual Battery at Swap, or bSWP, is the average residual power in the battery, expressed in the number of minutes of flight time, when the battery is swapped at a depot. Efficient dispatch should result in a low bSWP, which is a mark of good power management. Deliveries per Distance Traveled, or DPDT, is the average number of orders delivered for each unit of distance traveled. A higher DPDT signifies more efficient use of the vehicle. We will report the total distance traveled (Distance), average capacity utilization (Util) and the average bSWP over all drones in the experimental results. In addition, we report the total time intervals (minutes) it takes to complete the whole operation (DoneT) and the total number of battery swaps (nSWP) in the whole operational duration.

5.2. Case 1: a small-scale demonstration

We generated experimental data on a $[0, 500]^2$ square area (similar to the setup in Gendreau et al., 1999). The first data set is a small case consisting of 10 drones and 20 orders. Table 2 lists the vehicle information, including the starting coordinate, maximum speed and carrying capacity of each drone. For example, it takes the slower drone 100 min, and the faster drone 50 min, to traverse the area from coordinate (0,0) to (500,500). Each drone’s battery capacity is set to 90 min of flight time with full payload, $\text{BatteryCap}_r = 90 \times (\text{Weight}_r + \text{Capacity}_r)$, $\forall r \in R$

Table 2 Vehicle (drone) configuration in Case 1.

<table>
<thead>
<tr>
<th>Vehicle</th>
<th>CurX</th>
<th>CurY</th>
<th>Speed</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>30</td>
<td>440</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>V2</td>
<td>120</td>
<td>380</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>V3</td>
<td>500</td>
<td>420</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>V4</td>
<td>440</td>
<td>270</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>V5</td>
<td>390</td>
<td>400</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>V6</td>
<td>440</td>
<td>370</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>V7</td>
<td>280</td>
<td>310</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>V8</td>
<td>310</td>
<td>410</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>V9</td>
<td>390</td>
<td>370</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>V10</td>
<td>280</td>
<td>200</td>
<td>20</td>
<td>6</td>
</tr>
</tbody>
</table>

where Weight, is universally set to 1. We placed four charging depots in the area, with coordinates (125,125), (125,375), (375, 125) and (375,375), respectively. The maximum distance to the nearest charging depot from anywhere in the area needs to be computed in general, e.g., by some geometric or optimization-based algorithms. For this simple setting, the distance is 250. Accordingly, we set the safety battery level BatThrsh to

$\text{BatThrsh} = (250/\text{MaxSpeed}) \times (\text{Weight}_r + \text{Capacity}_r)$, $\forall r \in R$

so that each drone can have enough time to fly to a battery depot from anywhere in the area.

Table 3 lists the order data, including the coordinates of the origin and destination, order size, food type (C for cold and H for hot), time when the order is placed, and the time it takes to prepare the order. Let us look at Order 10 for example. It is a 2-pack hot-food order that requests shipment from location (210,400) to location (230,480). The order is initiated at time $t = 8$ and it takes.
Table 3
Orders in Case 1.

<table>
<thead>
<tr>
<th>Order</th>
<th>OX</th>
<th>OY</th>
<th>DX</th>
<th>DY</th>
<th>Size</th>
<th>Type</th>
<th>InitT</th>
<th>PrepT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>400</td>
<td>410</td>
<td>160</td>
<td>480</td>
<td>1</td>
<td>C</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>250</td>
<td>260</td>
<td>450</td>
<td>80</td>
<td>1</td>
<td>H</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td>50</td>
<td>390</td>
<td>300</td>
<td>3</td>
<td>H</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>350</td>
<td>320</td>
<td>170</td>
<td>60</td>
<td>2</td>
<td>C</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>260</td>
<td>30</td>
<td>480</td>
<td>2</td>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>80</td>
<td>500</td>
<td>320</td>
<td>370</td>
<td>2</td>
<td>C</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>450</td>
<td>100</td>
<td>50</td>
<td>400</td>
<td>1</td>
<td>H</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>190</td>
<td>30</td>
<td>60</td>
<td>80</td>
<td>3</td>
<td>C</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>170</td>
<td>420</td>
<td>340</td>
<td>200</td>
<td>1</td>
<td>H</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>210</td>
<td>400</td>
<td>230</td>
<td>480</td>
<td>2</td>
<td>H</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>360</td>
<td>260</td>
<td>10</td>
<td>50</td>
<td>3</td>
<td>H</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>30</td>
<td>120</td>
<td>240</td>
<td>40</td>
<td>1</td>
<td>H</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>180</td>
<td>330</td>
<td>90</td>
<td>190</td>
<td>3</td>
<td>C</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>290</td>
<td>480</td>
<td>330</td>
<td>40</td>
<td>1</td>
<td>C</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>370</td>
<td>200</td>
<td>460</td>
<td>470</td>
<td>1</td>
<td>C</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>420</td>
<td>350</td>
<td>350</td>
<td>240</td>
<td>1</td>
<td>H</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>17</td>
<td>250</td>
<td>420</td>
<td>490</td>
<td>280</td>
<td>3</td>
<td>H</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>18</td>
<td>210</td>
<td>410</td>
<td>460</td>
<td>330</td>
<td>2</td>
<td>H</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>470</td>
<td>40</td>
<td>500</td>
<td>290</td>
<td>3</td>
<td>H</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>70</td>
<td>10</td>
<td>450</td>
<td>1</td>
<td>C</td>
<td>12</td>
<td>2</td>
</tr>
</tbody>
</table>

6 min for the origin (restaurant) to prepare the order, so it is ready for pickup anytime after \( T = 14 \), according to formula (1). Suppose for example an algorithmic setting of \( T = 5 \), \( M = 2 \) and \( L = 5 \), then according to formula (4) this order will be available for dispatch starting from iteration \( k = 4 \), which plans for the time range \( t \in [7, 11] \). Fig. 5 shows the initial placement of all the 10 drones (blue circles, the size indicates the maximum speed) and 20 orders (marker size indicates the order size). Note that in a simulation run, only active orders are visible on the plot.

We first compare the computing times of solving MIP problems and solving MIQCP problems as the building block in the rolling horizon framework, to justify the practical importance of having the MIP model. Three formulations are experimented, MIP, MIQCP1 and MIQCP2. MIP is the original formulation of the model DISPATCH\((k)\) with the number of polygonal sides \( n \) equal to 8, MIQCP1 substituted the SOCP constraint \((9^*)\) for \((9)\), and MIQCP2 additionally substituted \((57h^*)\) for the objective term \((57h)\). We consider four settings with increasing plan-ahead horizon \( T \) values at a given move-ahead step \( M = 2 \). CPLEX offers two main search strategies to solve an MIQCP model (IBM, 2019): (1) solve a QCP relaxation of the model at each node of the branch-and-bound search tree (option miqcpstrat=1), and (2) solve an LP relaxation (formed by using first order approximations of the quadratic constraints) of the model at each node (option miqcpstrat=2). For the problems at hand, Strategy 1 has been much slower than Strategy 2, so we set miqcpstrat=2 in these experiments. We also limit the computing time per iteration to 300 s, via the GAMS option reslim=300. The results are given in Table 4. The column...
Table 5
Dispatch Results for Case 1 at different T and M values. Notes: Distance is the total flight distance (in Euclidean metric) of all drones, Delay is the average delay (in minutes) for all orders, Util is the average capacity utilization over all drones, nsWP is the number of battery swaps, bsWP is the average battery level before a swap, DoneT is the time point at which all orders are delivered, aCPU is the average computing time (in seconds) per iteration and mCPU is the maximum computing time over all iterations.

<table>
<thead>
<tr>
<th>Reveals</th>
<th>T</th>
<th>M</th>
<th>Distance</th>
<th>Delay</th>
<th>Util</th>
<th>nsWP</th>
<th>bsWP</th>
<th>DoneT</th>
<th>aCPU</th>
<th>mCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grad</td>
<td>5</td>
<td>5</td>
<td>8826</td>
<td>46.1</td>
<td>0.39</td>
<td>1</td>
<td>22.7</td>
<td>151</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>Grad</td>
<td>4</td>
<td>4</td>
<td>9200</td>
<td>43.5</td>
<td>0.38</td>
<td>2</td>
<td>279</td>
<td>145</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>Grad</td>
<td>3</td>
<td>3</td>
<td>9298</td>
<td>41.3</td>
<td>0.39</td>
<td>3</td>
<td>24.4</td>
<td>151</td>
<td>0.1</td>
<td>0.3</td>
</tr>
<tr>
<td>Grad</td>
<td>2</td>
<td>2</td>
<td>8750</td>
<td>40.7</td>
<td>0.38</td>
<td>1</td>
<td>27.3</td>
<td>137</td>
<td>0.1</td>
<td>0.3</td>
</tr>
<tr>
<td>Grad</td>
<td>1</td>
<td>1</td>
<td>8300</td>
<td>35.0</td>
<td>0.39</td>
<td>0</td>
<td>0.0</td>
<td>127</td>
<td>0.2</td>
<td>0.3</td>
</tr>
<tr>
<td>Grad</td>
<td>7</td>
<td>7</td>
<td>8207</td>
<td>34.8</td>
<td>0.42</td>
<td>0</td>
<td>0.0</td>
<td>111</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>Grad</td>
<td>7</td>
<td>6</td>
<td>7621</td>
<td>29.7</td>
<td>0.48</td>
<td>0</td>
<td>0.0</td>
<td>100</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>Grad</td>
<td>7</td>
<td>5</td>
<td>8523</td>
<td>38.1</td>
<td>0.44</td>
<td>1</td>
<td>16</td>
<td>163</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>Grad</td>
<td>9</td>
<td>9</td>
<td>8726</td>
<td>37.8</td>
<td>0.42</td>
<td>0</td>
<td>0.0</td>
<td>127</td>
<td>0.3</td>
<td>0.6</td>
</tr>
<tr>
<td>Grad</td>
<td>9</td>
<td>7</td>
<td>8944</td>
<td>39.4</td>
<td>0.44</td>
<td>0</td>
<td>0.0</td>
<td>120</td>
<td>0.3</td>
<td>0.6</td>
</tr>
<tr>
<td>Grad</td>
<td>9</td>
<td>5</td>
<td>7623</td>
<td>31.5</td>
<td>0.44</td>
<td>0</td>
<td>0.0</td>
<td>111</td>
<td>0.3</td>
<td>0.6</td>
</tr>
<tr>
<td>Grad</td>
<td>9</td>
<td>3</td>
<td>8415</td>
<td>33.9</td>
<td>0.49</td>
<td>0</td>
<td>0.0</td>
<td>112</td>
<td>0.3</td>
<td>0.7</td>
</tr>
<tr>
<td>Grad</td>
<td>11</td>
<td>11</td>
<td>7727</td>
<td>33.7</td>
<td>0.46</td>
<td>0</td>
<td>0.0</td>
<td>122</td>
<td>0.5</td>
<td>3.4</td>
</tr>
<tr>
<td>Grad</td>
<td>11</td>
<td>9</td>
<td>9411</td>
<td>45.7</td>
<td>0.52</td>
<td>1</td>
<td>12.3</td>
<td>154</td>
<td>0.4</td>
<td>1.3</td>
</tr>
<tr>
<td>Grad</td>
<td>11</td>
<td>7</td>
<td>9045</td>
<td>52.1</td>
<td>0.40</td>
<td>0</td>
<td>0.0</td>
<td>134</td>
<td>0.4</td>
<td>0.8</td>
</tr>
<tr>
<td>Grad</td>
<td>11</td>
<td>5</td>
<td>8873</td>
<td>42.0</td>
<td>0.43</td>
<td>0</td>
<td>0.0</td>
<td>121</td>
<td>0.4</td>
<td>1.0</td>
</tr>
<tr>
<td>Full</td>
<td>5</td>
<td>5</td>
<td>9034</td>
<td>39.1</td>
<td>0.4</td>
<td>0</td>
<td>0.0</td>
<td>131</td>
<td>0.2</td>
<td>0.3</td>
</tr>
<tr>
<td>Full</td>
<td>5</td>
<td>4</td>
<td>8807</td>
<td>43.9</td>
<td>0.41</td>
<td>1</td>
<td>22.5</td>
<td>165</td>
<td>0.1</td>
<td>0.3</td>
</tr>
<tr>
<td>Full</td>
<td>5</td>
<td>3</td>
<td>9223</td>
<td>40.8</td>
<td>0.41</td>
<td>1</td>
<td>23.3</td>
<td>136</td>
<td>0.2</td>
<td>0.3</td>
</tr>
<tr>
<td>Full</td>
<td>5</td>
<td>2</td>
<td>9339</td>
<td>41.5</td>
<td>0.36</td>
<td>4</td>
<td>29.2</td>
<td>251</td>
<td>0.1</td>
<td>0.3</td>
</tr>
<tr>
<td>Full</td>
<td>7</td>
<td>7</td>
<td>8100</td>
<td>33.4</td>
<td>0.41</td>
<td>0</td>
<td>0.0</td>
<td>120</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>Full</td>
<td>7</td>
<td>5</td>
<td>7990</td>
<td>32.8</td>
<td>0.44</td>
<td>0</td>
<td>0.0</td>
<td>126</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>Full</td>
<td>7</td>
<td>3</td>
<td>8045</td>
<td>30.5</td>
<td>0.45</td>
<td>0</td>
<td>0.0</td>
<td>100</td>
<td>0.2</td>
<td>0.5</td>
</tr>
<tr>
<td>Full</td>
<td>7</td>
<td>2</td>
<td>8085</td>
<td>27.2</td>
<td>0.41</td>
<td>0</td>
<td>0.0</td>
<td>113</td>
<td>0.2</td>
<td>0.5</td>
</tr>
<tr>
<td>Full</td>
<td>9</td>
<td>9</td>
<td>8788</td>
<td>31.4</td>
<td>0.41</td>
<td>0</td>
<td>0.0</td>
<td>118</td>
<td>0.3</td>
<td>0.8</td>
</tr>
<tr>
<td>Full</td>
<td>9</td>
<td>7</td>
<td>7497</td>
<td>27.7</td>
<td>0.42</td>
<td>0</td>
<td>0.0</td>
<td>92</td>
<td>0.2</td>
<td>0.7</td>
</tr>
<tr>
<td>Full</td>
<td>9</td>
<td>5</td>
<td>8283</td>
<td>30.4</td>
<td>0.41</td>
<td>0</td>
<td>0.0</td>
<td>111</td>
<td>0.3</td>
<td>0.7</td>
</tr>
<tr>
<td>Full</td>
<td>9</td>
<td>3</td>
<td>8071</td>
<td>29.4</td>
<td>0.51</td>
<td>0</td>
<td>0.0</td>
<td>112</td>
<td>0.3</td>
<td>0.7</td>
</tr>
<tr>
<td>Full</td>
<td>11</td>
<td>11</td>
<td>7454</td>
<td>26.7</td>
<td>0.42</td>
<td>1</td>
<td>25.3</td>
<td>144</td>
<td>0.4</td>
<td>1.2</td>
</tr>
<tr>
<td>Full</td>
<td>11</td>
<td>9</td>
<td>8649</td>
<td>35.3</td>
<td>0.43</td>
<td>0</td>
<td>0.0</td>
<td>127</td>
<td>0.4</td>
<td>1.0</td>
</tr>
<tr>
<td>Full</td>
<td>11</td>
<td>7</td>
<td>8217</td>
<td>36.6</td>
<td>0.39</td>
<td>1</td>
<td>25.7</td>
<td>141</td>
<td>0.6</td>
<td>4.2</td>
</tr>
<tr>
<td>Full</td>
<td>11</td>
<td>5</td>
<td>7273</td>
<td>26.3</td>
<td>0.43</td>
<td>0</td>
<td>0.0</td>
<td>101</td>
<td>0.6</td>
<td>3.8</td>
</tr>
</tbody>
</table>

*aCPU* is the average computing time (in seconds) per iteration and *mCPU* is the maximum computing time over all iterations. We can see that as the problem size increases, both MIQCP formulations scaled much worse than the MIP formulation, and the presence of quadratic objective terms aggravated the intractability in computing time. Furthermore, both MIQCP formulations failed to leverage (i.e., recognize as an incumbent solution) the starting point from the preceding iteration. The order assignments and vehicle routes are different under different formulations. In terms of the total Distance traveled, the average difference among the three formulations is within 3%, while no consistent superiority of one over another is observed. Therefore, we will adopt the MIP formulation in all subsequent experiments.

Next, we run the algorithm with different combinations of plan-ahead T and move-head M values, and captured the routing performances. The results are listed in the upper half of Table 5, where the order information unfolds gradually (Grad), following the arrival process and the rule in Eq. (4). The column headers are annotated in the caption of the table and their definitions are given in Section 5.1.3.

All parameter combinations have successfully completed the dispatch. We can see that, as intuitively sensible, the combination of a large T and a small M tends to produce better dispatching results. We have also run the same set of experiments for an artificial scenario where all orders’ information is assumed known upfront (i.e., Full revelation). This is equivalent to setting InitT to 1 and setting PrepT to the value (InitT + PrepT - 1). The results are listed in the lower half of Table 5 where Reveal is set to Full. With the clairvoyant information, the algorithm performed better (smaller Distance, smaller Delay) for most of the parameter combinations. This indicates that the information evolution, which is a natural stochastic process, does affect and complicate the problem.

One might wonder why it is not always the case that (1) a larger T and a smaller M improve the performances and (2) the full revelation gives a better performance than gradual revelation of information. There are several reasons. First, the actual objective function being optimized is a complex composite of multiple performance metrics, including distance, delay and overall feasibility, etc., so monotone change in any single performance metric with any input parameter (e.g., T or M) should not be expected. The same reasoning applies for the comparison between Gradual and Full revelation. Second, the whole algorithm involves solving MIPs in a rolling horizon fashion, and there is no theory of optimality for such frameworks. While the algorithm is designed to work with any reasonable choice of (T, M) combination, the specific choice is subject to practical considerations.

The best (T, M) combination for use in practice will depend on three factors: (1) the actual computational load, which is affected by, e.g., the number of vehicles under dispatch, the arrival rate of orders and the typical OD distance of orders, (2) the computing hardware, such as the CPU speed and the number of parallel cores available and (3) the granularity of the temporal discretization, in other words, the clock time equivalence of a unit time interval in the algorithm. A smaller T limits the look-ahead horizon and diminishes the benefit of optimization, while a larger T results in a larger MIP model which takes more time to solve. A smaller M requires a faster turnaround in terms of computing for each iteration, while a larger M reduces the system’s responsiveness to arriving orders.
For comparison, we have also run the static model (40)–(44) on the Full information case by setting the time scope T to 92, the lowest DoneT in Table 5. However, even by supplying an integer feasible starting point (from the solution of Algorithm 4), the static model was still ineffective - it not only failed to significantly close the optimality gap within 20 h of computation, but also failed to improve the incumbent during the entire time. This demonstrates the extreme difficulty of the problem. We comment that part of the difficulty is due to the expansive formulation associated with temporal discretization and explicit representation of the relation between geometric locations and binary actions.

5.3. Case 2: a simulation study

We simulate a 6-hour operation scenario. Orders arrive at the rate of one arrival per minute, following a Poisson process. The OD coordinates are randomly generated in the [0, 500]² area. The order size is randomly assigned to 1, 2 or 3 with equal probability and the order type is also randomly assigned to Hot or Cold. Each order's preparation time is independent of other attributes and is uniformly distributed between 2 and 7 min. A total of 353 orders were generated for a 360-min period. We randomly scatter 50 drones across the service area, of which 24 are big drones with a carrying capacity of 6 and a maximum speed of 20, and the other 26 are small drones whose maximum speed is 10. Among the small drones, 13 have a carrying capacity of 2 and the other 13 have a carrying capacity of 3. Note the subtle complexity that capacity-2 drones are unable to pick up size-3 orders. Four battery depots are located in the same way as Case 1. The data sets (same format as Tables 2 and 3) is provided as supplemental material.

We have experimented different values of T and M to process the orders, simulating an online operation. The overall performance metrics are reported in Table 6. The first thing to note is that computing time per iteration increased significantly due to the increased number of drones and orders under dispatch. The average computing time (aCPU) is also directly related to the value of T. For the same T, the value of M affects the maximum computing time (mCPU) in a reverse way - a smaller M means more iterations to process in the whole simulation horizon, hence more chance to experience extreme cases in term of computing time. The computing times are all within the tolerable range for online deployment. The Delay generally decreases as more computing effort is committed. The average power remaining at battery swaps is in the range of 15–25 min. Comparing this to the safety level (BatThresh) of 25 min for slow drones and 12.5 min for fast drones, we conclude that power management is on the conservative side (more safety and less efficiency). The risk altitude can be adjusted by tweaking some of the penalty coefficients in the model objective.

Fig. 6 presents more perspectives from which the simulation performance can be examined. When multiple system configurations are compared, these plots could make insights more accessible. For this specific case, the following points could be observed. The Delay distribution is highly skewed - most orders experience a short delay while only a few orders experience extremely long delays. This indicates that some order statistics, such as the sample median or a percentile, could be more appropriate to summarize and compare the performances. The relatively concentrated distribution of the per-iteration solution time indicates that extremely long computing time is practically a rare event, even though MILPs are in general intractable. The Capacity-6 drones are on average less well-utilized than drones with smaller capacities, suggesting that the overall objective biases toward fast delivery rather than high consolidation. The configuration that the faster drone has twice the speed of the slower drone is reflected in the total work they perform - overall, faster drones doubles the amount of traveling and delivery of small drones, as shown in the lower right subplot.

Fig. 7 plots some system state indicators over time. After a short ramp-up period, the system reaches a steady state in which at any moment a steady (though fluctuating) number of orders are in-transit and are waiting for pickup, respectively. Ramp-down starts at around t = 360 when the order arrival process was shut down and the process terminates when all orders are delivered. The MIP solution time is somewhat positively correlated with the number of waiting orders in the system. Intuitively, the main work for optimization is arbitrating what each drone should do to handle the waiting orders. Once an order is in-transit with a drone, the dispatcher's task becomes relatively straightforward, i.e., maintaining a reasonable route to the order's destination. This suggests opportunities to decompose some of the tasks by drones, i.e., delegating routing decisions to drones, which will be explored in the future work.

The algorithm development process benefited significantly from visualization and animation. Fig. 8 demonstrates a temporal snapshot of the animated traffic scene in the steady state. Only orders visible at the time are plotted. For clarity the vehicle labels are not shown in the plot. Order status is color-coded using the same scheme as in Fig. 2. For instance, Order 232 (gray) is initiated but not ready for pickup yet, and a vehicle to the southwest of it (which has just unloaded order 188) is already traveling toward it for a timely pickup as soon as the order becomes ready. Orders 202 and 212 are carried by the same vehicle traveling east-

<table>
<thead>
<tr>
<th>T</th>
<th>M</th>
<th>Distance</th>
<th>Delay</th>
<th>Util</th>
<th>nSWP</th>
<th>bSWP</th>
<th>DoneT</th>
<th>aCPU</th>
<th>mCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>128453</td>
<td>14.3</td>
<td>0.48</td>
<td>47</td>
<td>17.7</td>
<td>441</td>
<td>2.1</td>
<td>3.9</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>127607</td>
<td>12.6</td>
<td>0.48</td>
<td>48</td>
<td>20.1</td>
<td>433</td>
<td>2.3</td>
<td>3.6</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>132498</td>
<td>13.6</td>
<td>0.48</td>
<td>49</td>
<td>20.3</td>
<td>439</td>
<td>2.4</td>
<td>4.3</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>130002</td>
<td>11.5</td>
<td>0.47</td>
<td>49</td>
<td>22.1</td>
<td>417</td>
<td>2.5</td>
<td>4.0</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>129056</td>
<td>14.1</td>
<td>0.49</td>
<td>39</td>
<td>16.0</td>
<td>449</td>
<td>3.1</td>
<td>6.1</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>130903</td>
<td>12.5</td>
<td>0.48</td>
<td>43</td>
<td>18.2</td>
<td>441</td>
<td>3.6</td>
<td>9.8</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>140203</td>
<td>15.1</td>
<td>0.46</td>
<td>48</td>
<td>21.2</td>
<td>457</td>
<td>3.8</td>
<td>7.6</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>142975</td>
<td>17.5</td>
<td>0.48</td>
<td>53</td>
<td>22.6</td>
<td>425</td>
<td>4.3</td>
<td>8.1</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>128166</td>
<td>12.6</td>
<td>0.45</td>
<td>39</td>
<td>15.9</td>
<td>442</td>
<td>5.0</td>
<td>11.2</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>129168</td>
<td>11.8</td>
<td>0.46</td>
<td>34</td>
<td>16.2</td>
<td>428</td>
<td>5.6</td>
<td>11.8</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>135575</td>
<td>12.6</td>
<td>0.45</td>
<td>42</td>
<td>19.6</td>
<td>426</td>
<td>6.3</td>
<td>16.3</td>
</tr>
<tr>
<td>3</td>
<td>141164</td>
<td>14.7</td>
<td>0.46</td>
<td>39</td>
<td>20.8</td>
<td>412</td>
<td>6.8</td>
<td>15.9</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>130218</td>
<td>12.7</td>
<td>0.49</td>
<td>34</td>
<td>16.8</td>
<td>463</td>
<td>12.4</td>
<td>174.0</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>131303</td>
<td>13.0</td>
<td>0.45</td>
<td>32</td>
<td>18.2</td>
<td>487</td>
<td>9.1</td>
<td>22.3</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>137701</td>
<td>12.8</td>
<td>0.43</td>
<td>35</td>
<td>17.2</td>
<td>442</td>
<td>12.9</td>
<td>127.1</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>139586</td>
<td>13.1</td>
<td>0.44</td>
<td>39</td>
<td>22.7</td>
<td>441</td>
<td>11.0</td>
<td>42.1</td>
</tr>
</tbody>
</table>
Fig. 6. Simulation result analysis for the case $T = 7$ and $M = 7$.

Fig. 7. Evolvement of system states in the simulation timeline.
bound. The vehicle carrying the Order 218 is low in battery (hence shown in red), and it is gravitating toward the nearest charging depot (B3). Overall, visualization is a great tool for reading the solution, revealing potential problems and identifying opportunities for parameter tuning and algorithmic improvement.

The above computational exercise has verified that the dispatch algorithm is able to work correctly as designed. We now probe the effects of two design parameters ($n$, the number of sides of the approximating polygon, and LT, the minimum time required for loading) on three performance metrics (Distance, Delay, Computing Time) via statistical analysis on random permutations of the Case 2 data. Specifically, we randomly permute the arrival time of the 353 orders in Case 2 data while keeping all other attributes unchanged, thereby creating 10 random samples of the data set. While each sample represents a unique 6-hour operational scenario, the total payloads and OD distances are the same across all samples, which controls for unnecessary variability and simplifies the data creation process.

Adopting the algorithmic configuration of $T = 7$ and $M = 7$, we solve the 10 permutations three times, each time using a different ($n$, LT) setting, i.e., the base setting has ($n$, LT) = (8, 1), and the other two has (16, 1) and (8, 3), respectively. We then conducted paired t-tests on Distance, Delay and the average MIP solution time (aCPU) to compare the base setting to each of the other settings. The performance metrics as well as test results are listed in Table 7. The small p-values under the setting $n = 16$ indicates that doubling the granularity of vehicles’ mobility representation will make a statistically significant difference in solution quality as well as in computing time - it would result in less detour and less delay at the cost of an increased computing time. The practical significance of these differences, however, is up for the practitioner’s judgment. For the other comparative case: the increase in Loading Time creates a substantial additional\(^1\) increase in the average Delay, while not making any significant difference on Distance and computing time. This result is expected and sensible. The additional Delay can be attributed to the increased need for battery swaps brought about by the prolonged operational delay.

5.4. Notes on solving the MIP

Several mixed integer modeling techniques that are generally regarded as daunting from the computational perspective have been employed in the model DISPATCH($k$). First, big-M constants are perversely used, i.e., in constraints (22), (27), (36), (47)–(50), (54) and (56). Large big-M values can impede closing of the optimality gap. Second, penalty factors of disparate magnitudes are applied to objective terms in order to stratify levels of priority. This calls for a more aggressive scaling method (the scenaid option) as well as more emphasis on numerical precision (the numerical emphasis option), which will compromise the solution speed. Without turning on these options, CPLEX in certain cases falsely reported near-optimal solutions as optimal (at optcr=0 and optca=0). While a near-optimal solution may be acceptable in many other applications, our algorithm demands strict integer optimality of the DISPATCH($k$) solution in order to ensure a feasible and properly-advancing progression. This is because the optimization model not only performs incremental dispatch decisions within its planning horizon but is also responsible for relaying system states between

\(^1\) The null hypothesis in this test is the equality of the base setting Delay and the alternative setting (i.e., LT = 3) Delay minus two, where “minus two” is for offsetting the known and expected two-minute difference in Delay between the two settings. Therefore, the study is about the additional increase in Delay caused by the prolonged loading time.
successive iterations. A mere feasible solution to DISPATCH(k) may not guarantee sufficient progress to move the system states forward. For example, delaying the pickup action for a couple of time periods from the optimal pickup point still constitutes a feasible solution to DISPATCH(k). However, such a delay may cause the action uncaptured in iteration k + 1 as time advances M (assuming M < T) steps, and if this situation persists, a stalemating situation will arise. This type of stalemating situation has stopped arising once we set a small enough tolerance (optca=1e–3) for the optimality gap while running DISPATCH(k).

We have primarily focused on expressing the business operation in a convenient and intuitive way, and delegated much of the model reduction effort to the CPLEX solver’s presolve functionality, which was quite effective. In the experiments, no prolonged solution time has been encountered, as corroborated by the computational results (mCPU). The incumbent solution prepared in Lines 17–22 of Algorithm 4 has always been leveraged by the solver. We have also experimented not providing an incumbent solution (setting mipstart to 0) and in most iterations the solution speed was unaffected. Nonetheless, in certain iterations finding a first incumbent took an extremely long time, in which case the supplied incumbent would be hugely beneficial. Therefore, the starting point strategy is overall effective and is included in the main algorithm. The rationale for the GAMS option “optca=1e–3” is that when the absolute optimality gap is less than 0.001, the optimal integer solution can be deemed found because any integral change in binary variables would incur a change in the objective value greater than that threshold, therefore we can terminate the MIP process at this point to save some time. The time saving achieved in this way turned out significant at times, when the solver would otherwise spend a long time proving optimality.

6. Conclusion

The market potential of on-demand meal delivery businesses will continue to materialize in the years to come. Drones are a promising platform to serve the need for agile transportation in meal delivery and other similar businesses. In this paper, we have presented an optimization-driven progressive algorithm for drone dispatch and order delivery in a dynamic, real-time operational environment. The foundation of the algorithm includes a comprehensive mathematical model of the business operations and a temporal progression framework that connects decisions across time periods and processes new information with agility. The model is designed for real-world situations where order information is revealed gradually as time advances and dispatch decisions are updated periodically with short intervals. Practical factors such as the orders’ locational uncertainty, food type restriction, order size variation, drones’ mobility range, carrying capacity, battery consumption and battery swapping operation are all considered in the model. As the driving force of the algorithmic progression, a sophisticated objective function, as well as the parameter updating procedures, has been developed to mitigate multiple layers and multiple aspects of the operational requirements and goals. Computational and simulation experiments have shown that the algorithm is capable of handling online dispatches for moderately sized systems.

Dynamic, infinite-horizon VRP with arbitrary pickup and delivery locations and en-route vehicle diversion is a real operational problem and is becoming increasingly important with the development of agile transportation. This paper has made a first attempt at presenting and addressing this challenge which could serve to expand the scope of VRP research. Though not extensively discussed here, parameter configuration in the objective function plays an important role in the actual performance of the algorithm. Improper settings that fail to elicit the desired hierarchy of objectives could stalemate the algorithmic progression. Using an optimization model to drive the dynamics of a complex system in an infinite horizon is an interesting challenge to be investigated in the future work. In spite of its novelty and demonstrated effectiveness, the temporarily discrete, spatially continuous MIP formulation is by no means the only option that fits well in the rolling horizon framework to address the challenge. At least two other formulations are worth investigating: (1) construct a static node-arc representation of the low-altitude air traffic network, e.g., assuming fixed, straight-line routes between pairs of points of interest (POIs) and including sufficient number of popular address-level or block-level POIs that capture most orders’ origin and destination coordinates, then periodically re-optimize the route plan based on this static network; (2) in each optimization epoch, dynamically construct a new graph on the Euclidean plane with nodes being the exact coordinates of relevant entities (i.e., orders and vehicles) at the time and arcs representing the straight-line distances between pairs of entities. The data cases used in this paper will be applicable for these formulations and lateral comparisons among different formulations can be conducted.

As more and more drones, wheeled robots and other autonomous vehicles are deployed in the logistics system, routing will inevitably become too complex to perform centrally. Central dispatch is especially impractical when traffic conditions, such as corridor congestion and collision avoidance, are explicitly considered. Future work will also focus on extending the proposed framework to allow for decentralized decision-making, more tolerance for uncertainty and improved computational parallelism, therefore adapting the framework to production-scale systems.

Acknowledgments

The author would like to thank three anonymous referees for comments and suggestions that improved the quality of this paper.

Table 7

<table>
<thead>
<tr>
<th>Perm</th>
<th>Base setting</th>
<th>n = 16</th>
<th>LT = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dist</td>
<td>Delay</td>
<td>aCPU</td>
</tr>
<tr>
<td>1</td>
<td>130032</td>
<td>13.3</td>
<td>3.3</td>
</tr>
<tr>
<td>2</td>
<td>130703</td>
<td>14.2</td>
<td>3.2</td>
</tr>
<tr>
<td>3</td>
<td>128131</td>
<td>13.6</td>
<td>3.2</td>
</tr>
<tr>
<td>4</td>
<td>130578</td>
<td>14.6</td>
<td>3.1</td>
</tr>
<tr>
<td>5</td>
<td>128958</td>
<td>16.8</td>
<td>3.5</td>
</tr>
<tr>
<td>6</td>
<td>130954</td>
<td>16.0</td>
<td>3.5</td>
</tr>
<tr>
<td>7</td>
<td>131238</td>
<td>15.8</td>
<td>3.4</td>
</tr>
<tr>
<td>8</td>
<td>129376</td>
<td>13.1</td>
<td>3.1</td>
</tr>
<tr>
<td>9</td>
<td>131348</td>
<td>15.6</td>
<td>3.3</td>
</tr>
<tr>
<td>10</td>
<td>129559</td>
<td>15.9</td>
<td>3.3</td>
</tr>
<tr>
<td>p-val</td>
<td>-</td>
<td>-</td>
<td>0.004</td>
</tr>
</tbody>
</table>
Appendix A. Inner polyhedral approximation of a circular area

A circular area of radius $R$ centered at the origin consists of points in the set

$$B := \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 \leq R^2\} \quad \text{(A.1)}$$

The defining inequality constitutes a nonlinear (though convex) constraint if included in an optimization problem, which will limit the selection of amenable solvers, particularly when the formulation also contains integer variables. We will provide a generic formula for constructing a set of linear inequalities that approximate the circular area while maintaining feasibility, that is, any point that satisfies the linear inequalities is guaranteed to be in $B$. The idea is to algebraically characterize the inscribing polyhedron of the circle.

Suppose we want to use $n$ linear inequalities of the form

$$a_i x + b_i y \leq c_i, \quad \text{for } i = 1, \ldots, n$$

to approximately characterize the circular area, our task is to determine the coefficients $a_i, b_i$ and the right-hand side constant $c_i$. Let us assume for practicality that at least four inequalities are used, i.e., $n \geq 4$.

For $i = 1, \ldots, n$, define the following quantities,

$$x_i = R \cdot \cos(2\pi (i - 1)/n), \quad y_i = R \cdot \sin(2\pi (i - 1)/n)$$

$$s_i = \text{sgn} \left( \frac{x_i}{x_{i+1} - x_i} - \frac{y_i}{y_{i+1} - y_i} \right)$$

where $\text{sgn}(a)$ is equal to 1 if $a \geq 0$, equal to $-1$ if $a < 0$. Note that $s_i$ is well-defined only when $x_{i+1} \neq x_i$ and $y_{i+1} \neq y_i$. Then we define the $i$th linear inequality as

$$s_i \cdot \left( \frac{x_i}{x_{i+1} - x_i} - \frac{y_i}{y_{i+1} - y_i} \right) \leq 1$$

(A.2)

for $i = 1, \ldots, n$.

The parameter $s_i$ is used for determining the direction of the corresponding inequality. To see why this is correct, first note that when $x_{i+1} \neq x_i$ and $y_{i+1} \neq y_i$, the two-point form the $i$th boundary line is given by the equation

$$\frac{x - x_i}{x_{i+1} - x_i} = \frac{y - y_i}{y_{i+1} - y_i}$$

and to make it into an inequality of the correct direction, just note that the point $(0,0)$ should always lie on the feasible side of the inequality, which is ensured by the use of $s_i$ in conjunction with the $\leq$ sign. When we have either $x_{i+1} = x_i$ or $y_{i+1} = y_i$, the boundary line becomes a vertical or a horizontal line, respectively, and the corresponding formulas given in (A.2) are apparently the intended ones. The $a_i, b_i$ and $c_i$ values can be directly read off from formula (A.2).

Fig. A.1 illustrates how the polyhedral approximation look like at three $n$ values, and Table A.1 lists the coefficient values for $n = 8$.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>$\sqrt{2}$</td>
<td>$-\sqrt{2}$</td>
<td>$\sqrt{2}$</td>
<td>$\sqrt{2}$</td>
<td>$-\sqrt{2}$</td>
<td>$-\sqrt{2}$</td>
<td>$\sqrt{2}$</td>
<td>$\sqrt{2}$</td>
</tr>
<tr>
<td>$b_i$</td>
<td>$\sqrt{2}$</td>
<td>$\sqrt{2}$</td>
<td>$\sqrt{2}$</td>
<td>$\sqrt{2}$</td>
<td>$\sqrt{2}$</td>
<td>$\sqrt{2}$</td>
<td>$\sqrt{2}$</td>
<td>$\sqrt{2}$</td>
</tr>
<tr>
<td>$c_i$</td>
<td>$1/2$</td>
<td>$1/2$</td>
<td>$1/2$</td>
<td>$1/2$</td>
<td>$1/2$</td>
<td>$1/2$</td>
<td>$1/2$</td>
<td>$1/2$</td>
</tr>
</tbody>
</table>

Table A.1

Coeficients for approximating linear inequalities with $n = 8$.

References


Parnar, T., 2016. This tech giant has kicked off drone delivery in rural China. Fortune.


