Transportation Research Part B 191 (2025) 103117

Contents lists available at ScienceDirect

Transportation Research Part B

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

The drone scheduling problem in shore-to-ship delivery: A time discretization-based model with an exact solving approach

Ying Yang, Xiaodeng Hao, Shuaian Wang [*]

Department of Logistics and Maritime Studies, The Hong Kong Polytechnic University, Hung Hom, Kowloon, 999077, Hong Kong, China

ARTICLE INFO

Keywords:
Shore-to-ship delivery
Drone scheduling problem
Time discretization
Branch-and-price-and-cut

ABSTRACT

Amid growing interest in the integration of drones into maritime logistics, this paper addresses the drone scheduling problem in shore-to-ship delivery (DSP-SSD), which is both significant and challenging. We introduce a mixed-integer programming model with time discretization that incorporates drone-related constraints, moving targets, and the need for multiple drone trips. While commercial solvers can handle this model in small-scale scenarios, we propose a tailored branch-and-price-and-cut (BPC) algorithm for larger and more complex cases. This algorithm integrates a drone-specific backward labeling algorithm, cutting planes, and acceleration methods to boost its effectiveness. Experiments show that the BPC algorithm substantially outperforms the commercial solvers in terms of solution quality and computational efficiency and that the inclusion of acceleration strategies in the algorithm enhances its performance. We also provide detailed sensitivity analyses of critical parameters of the model, such as the time discretization parameter and the number of ships, to gain insights into how our approach could be applied in real-world DSP-SSD operations.

1. Introduction

Shore-to-ship delivery by supply or tender boat is a common practice in maritime logistics to transport essential items such as letters, food, and medicine to ships (Squires, 1986). Ships may need to call at non-cargo destinations for replenishment during the voyage. Nevertheless, due to factors such as adverse weather conditions, limited time, crowded traffic, and the ship's excessive size, it may be neither feasible nor economical to dock a ship at a port (Sulligoi et al., 2015; Service, 2023). In such circumstances, tender boats provide the essential connection between the ship and the shore. These deliveries, which provide both logistical support and emergency aid, are crucial for facilitating continuous operations at sea and reducing the need for ships to make port calls for resupply. However, the tender boat is fraught with inefficiencies that raise both safety and environmental concerns (Nguyen et al., 2021). In busy ports, the presence of numerous tender boats darting back and forth can exacerbate congestion, cause operational inefficiencies, and increase the risk of collisions or other accidents. The need to make multiple trips to deliver supplies results in high fuel consumption, which increases the environmental footprint of maritime operations. Besides, there are mainly two modes of shore-to-ship delivery by tender boat: receiving supplies while the ship is fully anchored or receiving supplies while moving. However, halting a ship, anchoring or drifting, is sometimes impractical, especially in busy coastal areas. Anchoring incurs time and economic costs and is subject to many governmental restrictions (Davis et al., 2016). Drifting, on the other hand, is only feasible in very open waters as it increases the risk of collision (Fowler and Sørgård, 2000). Additionally, restarting and accelerating a ship consumes more energy, reduces stability, and complicates management for the crew compared to when the vessel is moving. Thus,

* Corresponding author.
E-mail addresses: shadow.yang@connect.polyu.hk (Y. Yang), shelton.hao@connect.polyu.hk (X. Hao), hans.wang@polyu.edu.hk (S. Wang).

https://doi.org/10.1016/j.trb.2024.103117
Received 30 April 2024; Received in revised form 29 October 2024; Accepted 1 November 2024
Available online 24 November 2024

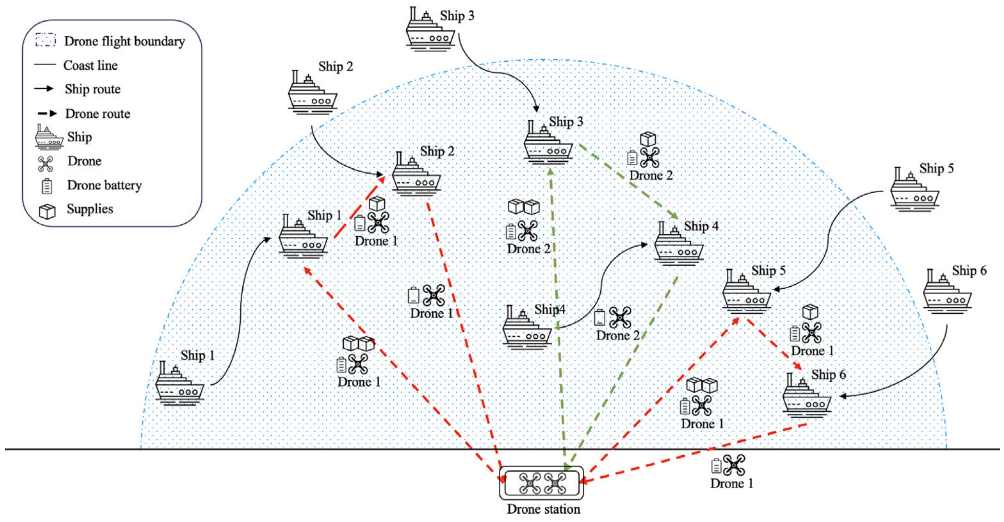

Fig. 1. Drone scheduling in shore-to-ship delivery.

maintaining a steady speed is generally more economical and practical for vessels when drones are conducting deliveries. These issues indicate the need for more safe and sustainable solutions for shore-to-ship delivery.

Recently, unmanned aerial vehicle (UAV) technology, or drones, has become a candidate solution to the longstanding logistical conundrum of shore-to-ship supply transfer (Charles Alcock, 2024; Kumar and Devi, 2023; Muhammad and Gregersen, 2022). The integration of drone technology into shore-to-ship operations promises a significant reduction in operational expenditure due to the lower battery consumption of drones compared with tender boats, and their lower labor costs due to automation. Drones can also travel on direct aerial routes, thus avoiding congested port traffic and complex docking procedures and their associated costs. From an environmental perspective, the deployment of drones for shore-to-ship delivery results in a markedly smaller carbon footprint than conventional modes of transport that are heavily reliant on fossil fuels.

Shore-to-ship delivery by drones offers great economic and environmental benefits. However, although much attention has been devoted to navigation rule design in this context (Asiimwe and Anvar, 2012; Thompson and Davies, 2021), the drone scheduling problem (DSP) has attracted much less consideration. We address the pivotal role of this problem in shore-to-ship delivery (DSP-SSD) in ensuring delivery punctuality and reducing operational costs. An illustrative example of the DSP-SSD is given as follows.

Example 1.

Fig. 1 shows the scenario of six ships sailing through a coastal area in need of supplies. Two drones are scheduled for the delivery task. Drone 1 performs two trips serving four ships, trip 1–1: {ship 1–ship 2} and trip 1–2: {ship 5–ship 6}. Drone 2 delivers supplies to ship 3 and then ship 4 in one trip. The drones depart from a drone station, deliver supplies in sequence, and then return to the drone station for battery replacement, all while the ships are moving along their designated courses. The deliveries are completed within the drone flight boundary, as constrained by the drones' carrying capacity and battery life.

Unlike the typical DSP, which is akin to the conventional vehicle routing problem (VRP), the DSP-SSD involves unique challenges. The complexity arises from two aspects. Firstly, customers in the DSP-SSD, i.e., the vessels being supplied, are not stationary but moving. Therefore, it requires novel modeling and solving schemes that can adapt to moving targets. Furthermore, the limited number of drones available necessitates the efficient planning of multiple round trips to ensure that drone resources are used optimally to meet demand.

In response to these challenges, the goal of this paper is to provide a comprehensive solution to the DSP-SSD that is practical for real-world applications. Our work makes four main contributions:

- From a modeling perspective, we formulate a mixed-integer programming (MIP) model of the DSP-SSD that considers moving targets and multiple trips (Section 3). We tackle the challenge of moving targets by constructing a time-expanded network using time discretization. We then use this network to develop the MIP model, which enables drones to perform delivery tasks sequentially using multiple trips. This model can be directly fed into an off-the-shelf solver to provide an optimal solution for drone scheduling in small-scale scenarios.
- From an algorithmic perspective, we develop a branch-and-price-and-cut (BPC) algorithm for the DSP-SSD for use at more practical scales than the basic MIP model (Section 4). In the BPC algorithm, the original model is reformulated into a route-based model and then solved by a branch-and-bound (B&B) tree in which each node represents a subproblem,

and column generation (CG) is used to add enhanced pricing and cutting plane constraints to tighten the linear relaxation. We propose several specialized acceleration strategies to the BPC algorithm based on the time-expanded network, including three types of tailored cutting planes, an efficient labeling algorithm, and an effective dominance rule, which is specialized under the time-expanded network.

- From a computational perspective, we demonstrate the ability of our MIP model and the BPC algorithm to generate high-quality solutions (Section 5.2). Various experiments reveal the superiority of our tailored BPC algorithm compared with an off-the-shelf solver in handling a complex version of the DSP-SSD. The same comparative experiments also reveal the benefits of our node selection rule, branching strategy, and acceleration strategies.
- From a practical perspective, we offer decision-makers insightful guidance on how to navigate the critical considerations of shore-to-ship delivery scheduling (Section 5.3). Sensitivity analysis of the discretization parameter reveals that adjusting the time granularity enables a customizable compromise between solution speed and decision quality. Moreover, our analysis of the relationship between drone fleet size and various operational metrics—such as delivery time, workload, and the number of trips per drone—informs drone deployment strategies.

2. Literature review

The paradigm of the DSP-SSD falls within the broader category of the DSP, which bears similarities to the VRP and has gained increasing research attention in recent years. However, the DSP-SSD diverges from the VRP in that ships, unlike traditional customers in the VRP, are moving, a feature that is characterized by a variant known as the vehicle routing problem with floating targets (VRPFT). Besides, since the number of drones is limited, the drones need to take multiple trips during the planning horizon, which is related to the multi-trip vehicle routing problem (MTVRP). Our literature review therefore mainly covers three domains: the DSP, VRPFT, and MTVRP.

2.1. Drone scheduling problem

The DSP is a contemporary optimization problem that has gained considerable research attention due to the increasing commercial and logistical applications of drones. Despite its close relationship with the VRP, the complexity of the DSP is amplified by constraints that are unique to drones, such as limited flight durations and payload capacity, and the need to navigate through changing airspace. A number of heuristic and exact methods have been developed to handle the DSP in this context, such as the vehicle routing problem with drones (VRPD), which coordinates the efforts of several trucks with a fleet of drones (Dorling et al., 2017; Poikonen et al., 2017; Wang and Sheu, 2019); the flying sidekick traveling salesman problem (FSTSP), in which a drone is launched, operated, and retrieved by its associated truck (Murray and Chu, 2015; Murray and Raj, 2020); and the traveling salesman problem with drones (TSPD), which aims to determine the most efficient path for a drone covering a group of customers by considering the complex constraints to which drones are subject (Ha et al., 2018; Roberti and Ruthmair, 2021). A state-of-the-art review of these methods is provided by Macrina et al. (2020). While these variants can be applied to various operational scenarios, our DSP-SSD features the unique difficulty that the customers are moving during delivery, which therefore requires specialized modeling and algorithmic approaches.

In practical terms, drone scheduling has applications that span a multitude of industries and services, including the logistics sector, where drones transform traditional delivery models and enable last-mile delivery solutions that significantly reduce time and costs (Dorling et al., 2017; Poikonen et al., 2017; Garg et al., 2023); the field of agriculture, where drones perform regular surveillance to improve yields and resource management (Puri et al., 2017; Rejeb et al., 2022); and disaster operations, where drones are used for search and rescue support (Pereira, 2021; Betti Sorbelli et al., 2022). A few studies have explored the DSP in the maritime field for scheduling drones to monitor pollution emissions in emission control areas (Xia et al., 2019; Shen et al., 2020; Liu et al., 2023). Specifically, Xia et al. (2019) apply the DSP in inspecting ship emissions, and use a Lagrangian relaxation-based method to provide a near-optimal solution. Based on their work, Liu et al. (2023) further develop an exact approach for vessel emission monitoring with a heterogeneous fleet of drones. However, there is a notable scarcity of research addressing the DSP in the shore-to-ship context.

2.2. Vehicle routing problem with floating targets

The VRPFT is tailored to address scenarios in which targets or destinations are not fixed but can change location dynamically, and is an advanced variation of the classic VRP. Solving the VRPFT requires sophisticated algorithms that can adapt routes to the real-time location of customers while ensuring the optimal deployment of resources and accommodating the movement of the delivery target. Some studies focus on a simpler version of this problem where the targets' locations are restricted to a set of discrete points, for example Zhen et al. (2023), Ozbaygin et al. (2017). When the targets are constantly moving, the solving complexity of the problem increases dramatically. There are two main types of model construction for the VRPFT: the coordination-based model, where time and location are continuous, and the arc flow model, which relies on time discretization. Studies applying the coordination-based model include Gambella et al. (2018), who propose a mixed-integer second-order cone optimization that can be solved by a branch-and-price (BP) algorithm, and Zhang et al. (2023), who design a new decomposition algorithm that solves the continuous model as a one-stop subproblem. However, these continuous models are unable to handle more complex multi-trip scenarios, and also rest on the stringent assumption that drones can cooperate with their customers' vehicles. This assumption is impractical in the DSP-SSD scenario, as ships lack the freedom to maneuver, particularly in coastal areas. The other type of model, the arc flow model based on time discretization, is also an effective approach to solving the VRPFT. For example, Stieber et al. (2015) study the multiple traveling salesmen problem with moving targets, in which the movement direction and speed are fixed. Xia et al. (2019) and Liu et al. (2023) focus on the multi-trip drone scheduling problem in vessel monitoring scenarios. These discrete models have greater flexibility when dealing with multi-trip scenarios.

2.3. *Multi-trip vehicle routing problem*

The MTVRP has gained increasing popularity in recent years. Azi et al. (2007) first present a two-phase exact algorithm addressing a problem where a single vehicle completes multiple routes to serve a set of customers with specific time windows. Azi et al. (2010) extend their previous work by considering the problem with multiple vehicles and propose a BP approach to tackle it. Macedo et al. (2011) then propose a new exact algorithm based on a pseudo-polynomial network flow model for this problem, which can solve more instances to optimality compared to that proposed by Azi et al. (2010). Mingozzi et al. (2013) formally propose the MTVRP and present an exact method to solve the MTVRP based on two set-partitioning-like formulations of the problem. Hernandez et al. (2016) also consider the MTVRP with time windows, where there are no restrictions on the duration but all customers should be visited. They develop a BP algorithm with tailored route-generating approaches. Paradiso et al. (2020) summarize prior studies on the MTVRP and identify four distinct variants. They develop an exact solution framework that incorporates column generation, route enumeration, and cutting plane, capable of solving all four MTVRP variants and surpassing previous state-of-the-art methods. Yang (2023b) follow the novel superstructure-based formulation proposed by Paradiso et al. (2020) and develop an exact price-cut-and-enumerate method. Yang (2023a) further put forward a variable fixing strategy that largely improves the efficiency of the CG and can be used in the MTVRP, capacitated vehicle routing problem (CVRP), and vehicle routing problem with time windows.

Some studies consider the practical applications of MTVRP, including multi-trip multi-repairman problem (Liu et al., 2018), drone delivery (Cheng et al., 2020), urban waste collection (Huang et al., 2021, 2024b), worker teams routing for airport ground (Dall'Olio and Kolisch, 2023), non-emergency patient transportation services (Huang et al., 2024a).

Our DSP-SSD presents a novel variant of the MTVRP with time windows, incorporating the unique challenge of floating customers, which adds the solving difficulty to the problem. Consequently, the existing state-of-the-art methods from previous studies cannot be directly applied to our problem.

2.4. *Summary*

This paper fills the research gap by exploring a novel DSP for shore-to-ship delivery to moving ships and providing a comprehensive solving scheme based on multiple drone trips. An arc-flow model is constructed inspired by the time discretization strategy in previous DSP in the maritime sector (Xia et al., 2019; Liu et al., 2023). However, our approach differs from their work in three main aspects. Firstly, the shore-to-ship delivery problem differs from the emissions inspection problem because it features payload capacity and nonlinear battery consumption constraints, adding new resources to the route generation process. Secondly, the time discretization granularity is modeled as a parameter to create the versatility needed to optimize solving efficiency and solution quality. Last but not least, Xia et al. (2019) propose a Lagrangian relaxation-based method that gives a feasible approximation to the optimal solution, and Liu et al. (2023) propose a structure enumeration-based exact approach whose bottleneck is reflected in a particular gap between upper and lower bounds. By contrast, we propose a tailored BPC algorithm that provides optimal solutions to large-scale instances with considerably higher quality in a short computation time.

3. Time discretization and model construction

This section begins with a clear definition of the DSP-SSD. A time-expanded network is then established based on time discretization as the foundation of the basic MIP model for the DSP-SSD.

3.1. *Problem description*

Within a planning period T, a set of ships (denoted by $\mathcal{V}$) requiring supplies are to be served by a set of identical drones (denoted by $\mathcal{D}$). The drones have a constant travel speed $S^{\mathcal{D}}$, deadweight $W^{\mathcal{D}}$, load capacity $C^{\mathcal{D}}$, and battery capacity $Q^{\mathcal{D}}$ that restrict the range that they can fly when carrying a specified payload. The drones are located in the same station with coordinates $(0, 0)$ and take off with supplies for multiple ships and return to the drone station after finishing their scheduled deliveries. When they return to the base station, their batteries are replaced, which takes τ_0 minutes. According to Dorling et al. (2017), the battery consumption $\bar{Q}$ of a drone during a certain period can be presented linearly to its flying time $\bar{T}$ and total weight $\bar{W}$, i.e., the sum of the drone's deadweight and the weight of its payload. Therefore, by introducing the parameter θ representing the battery consumption per weight over one unit of time, we can define the relationship by $\bar{Q} = \theta \bar{W} \bar{T}$ (Meng et al., 2023; Jeong et al., 2019; Cheng et al., 2020). The delivery must be finished in a coastal area within the drone flight boundary, which is defined by the maximum flying distance of a drone without payload, i.e., in a circle centered on the drone base station of radius R determined by $R = Q^{\mathcal{D}} S^{\mathcal{D}} / (2\theta W^{\mathcal{D}})$.

Each ship $v \in \mathcal{V}$ requiring supplies weighing a total of $w_v \leq C^{\mathcal{D}}$ enters the drone flight boundary and sails along its route at a constant speed of $S^{\mathcal{V}}$, where $S^{\mathcal{V}} < S^{\mathcal{D}}$. Each ship needs to be served exactly once. We define $T_v = [t_{ve}, t_{vl}]$ as the time interval during which ship v is sailing within the drone flight boundary, where the subscripts e and l represent the earliest and latest time, respectively. This time interval can be regarded as the time window for serving ship v. We assume that the navigational routes of the ships around the coastal area are relatively fixed due to the need to adhere to established shipping lanes, which are designed to ensure safe passage through potentially congested and hazardous waters. Therefore, the location coordinates of each ship $v \in \mathcal{V}$ at any time $t \in [t_{ve}, t_{vl}]$, denoted by $(\alpha_v(t), \beta_v(t))$, are known a priori. Based on the coordinates of each ship and Euclidean distance, the flying time of each drone from ship v at time t to another ship v', denoted by $\tau_{v,v'}(t)$, can be calculated by (A.1) in Appendix A. Similarly, the expression of traveling time between the drone station and ships is also provided in Appendix A. For calculation purposes, we assume that the location of ship v at time $t \leq t_{ve}$ is $(\alpha_v(t_{ve}), \beta_v(t_{ve}))$ and that at time $t \geq t_{vl}$ is $(\alpha_v(t_{vl}), \beta_v(t_{vl}))$. Based on this assumption, we always have a nonnegative delivery time for a drone to reach a ship as presented and proved in Appendix B.

In our shore-to-ship problem, each drone takes off from the base station, sequentially delivers supplies to the ships following a predetermined route, and returns to the base station for battery replacement. This constitutes a single cycle, referred to as a trip. Throughout the planning period, a drone's itinerary may encompass several such trips. The aim is to finish all of the delivery tasks within the designated planning period and optimize the scheduling of the limited drone resources at the base station by minimizing the total activity time (battery replacement time included) of the drones.

3.2. Time-expanded network

Given the coordinates of each ship at different time points, it is appropriate to design a time-expanded network such that each node represents a ship or drone station at a certain time point. The time can be discretized into $K + 1$ points with intervals of Δ minutes, where the positive discretization parameter $\Delta \in \mathbb{N}_{>0}$ and $K = \lceil T/\Delta \rceil$. Then, the set of time points is denoted by $\mathcal{T} = \{0, \Delta, 2\Delta, \cdots, K\Delta\}$. The node set $\mathcal{N}$ of this time-expanded network includes station-time nodes $\mathcal{N}^0 = \{(0,t) | t \in \mathcal{T}\}$ and ship-time nodes $\mathcal{N}^{\mathcal{V}} = \{(v,t) | v \in \mathcal{V}, t \in \mathcal{T}_v\}$, where $\mathcal{T}_v$ is the set of time points obtained by discretizing T_v into $\lfloor t_{vl}/\Delta \rfloor - \lceil t_{ve}/\Delta \rceil + 1$ elements with intervals of Δ minutes, i.e., $\mathcal{T}_v = \{\lceil t_{ve}/\Delta \rceil \Delta, \cdots, \lfloor t_{vl}/\Delta \rfloor \Delta\}$. We denote u_i and t_i as the corresponding ship (or station) and time of node i ($i \in \mathcal{N}$), respectively. The weight of supplies required by node i is w_{u_i}. As the weight of the required supplies of the station-time nodes is 0, we have $w_{u_i} = 0$ if $i \in \mathcal{N}^0$.

Using these nodes, the ship-ship arcs, denoted by the set $\mathcal{A}^{\mathcal{V}}$, are constructed by connecting ship node (v,t) and ship node (v',t') if $\tau_{v,v'}(t)$ lies in $(t' - t - \Delta, t' - t]$. Similarly, the ship-station arcs, denoted by the set $\mathcal{A}^0$, are constructed by connecting station node $(0,t)$ and ship node (v,t') if $\tau_{0,v}(t)$ lies in $(t' - t - \Delta, t' - t]$, and by connecting ship node (v,t) and station node $(0,t')$ if $\tau_{v,0}(t) + \tau_0$ lies in $(t' - t - \Delta, t' - t]$. The battery replacement time τ_0 is included in the ship-station arc to reduce the total number of arcs in the network, which implies that there is no station-station arc. We denote each arc $a_{i,j}$ in $\mathcal{A} = \mathcal{A}^0 \cup \mathcal{A}^{\mathcal{V}}$ as the arc from node i to node j.

The complete time-expanded graph can be presented as $\mathcal{G}_\Delta = (\mathcal{N}, \mathcal{A})$, where the parameter Δ determines the degree of segmentation of the network. We denote z_t as the number of available drones in the station at time $t \in \mathcal{T}$ (if there are drones landing or taking off at time t, z_t reflects the number of drones available immediately after this time point). All of the drones located in the station have a full battery prior to flight, and the battery is replaced with a full battery on returning to the station at the end of the planning horizon. Therefore, we have $z_{-\Delta} = |\mathcal{D}|$ and $z_{K\Delta} = |\mathcal{D}|$, where $z_{-\Delta}$ is a constant that represents the initial number of available drones in the station at virtual time point $-\Delta$. The original problem can thus be conceptualized as selecting the appropriate arcs to finish all of the delivery tasks while meeting the payload capacity and battery consumption constraints of the drones.

3.3. Original model

The important notation for model construction can be summarized as follows.

• **Ship-related parameters:**
$\mathcal{V}$: the set of ships, indexed by v.
$S^{\mathcal{V}}$: the sailing speed of the ships.
$\mathcal{T}_v$: the set of discretized time points within the time window for serving ship v.
w_v: the weight of supplies required by ship v.

• **Drone-related parameters:**
$\mathcal{D}$: the set of drones, indexed by d.
$S^{\mathcal{D}}$: the traveling speed of the drones.
$W^{\mathcal{D}}$: the deadweight of the drones.
$C^{\mathcal{D}}$: the load capacity of the drones.
$Q^{\mathcal{D}}$: the battery capacity of the drones.

• **Network-related parameters:**
$\mathcal{T}$: the set of discretized time points during the time planning period T.
Δ: the discretization parameter.
$\mathcal{N}$: the set of nodes in the time-expanded network, which includes the station-time node set $\mathcal{N}^0$ and the ship-time node set $\mathcal{N}^{\mathcal{V}}$.
u_i: the ship (or station) of node $i \in \mathcal{N}$.

t_i: the time of node $i \in \mathcal{N}$.
$\mathcal{A}$: the set of arcs in the time-expanded network, which includes the ship-ship arc set $\mathcal{A}^{\mathcal{V}}$ and the ship-station arc set $\mathcal{A}^0$.
$a_{i,j}$: the arc from node $i \in \mathcal{N}$ to node $j \in \mathcal{N}$.

• **Decision variables:**

$x_{i,j}$: a binary variable. $x_{i,j} = 1$ if arc $a_{i,j}$ is included in the drones' itineraries; and $x_{i,j} = 0$ otherwise.
y_i: a continuous variable that indicates the loaded weight of the drone when arriving at node $i \in \mathcal{N}^{\mathcal{V}}$.
q_i: a continuous variable that indicates the remaining battery level of the drone when arriving at node $i \in \mathcal{N}^{\mathcal{V}}$.
z_t: an integer variable that captures the number of available drones with a full battery in the station at time $t \in \mathcal{T}$.
Based on the discretized time-expanded network, we can formulate the following MIP model:

[Original model]

$$\min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{q},\boldsymbol{z}} \sum_{a_{i,j}\in\mathcal{A}} (t_j - t_i)x_{i,j} \tag{1a}$$

$$\text{s.t.} \sum_{j\in\mathcal{N}^{\mathcal{V}}} x_{i,j} - \sum_{j\in\mathcal{N}^{\mathcal{V}}} x_{j,i} = z_{t_i-\Delta} - z_{t_i} \quad \forall i \in \mathcal{N}^0, \tag{1b}$$

$$\sum_{i\in\mathcal{N}} x_{i,j} - \sum_{i\in\mathcal{N}} x_{j,i} = 0 \quad \forall j \in \mathcal{N}^{\mathcal{V}}, \tag{1c}$$

$$\sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}^{\mathcal{V}}:u_j=v} x_{i,j} = 1 \quad \forall v \in \mathcal{V}, \tag{1d}$$

$$y_i - w_{u_i}x_{i,j} + C^{\mathcal{D}}\left(1 - x_{i,j}\right) \geq y_j \quad \forall i \in \mathcal{N}^{\mathcal{V}}, \forall j \in \mathcal{N}^{\mathcal{V}}, \tag{1e}$$

$$y_i - w_{u_i}x_{i,j} \geq 0 \quad \forall i \in \mathcal{N}^{\mathcal{V}}, \forall j \in \mathcal{N}^0, \tag{1f}$$

$$Q^{\mathcal{D}} - \theta(t_j - t_i)(y_j + W^{\mathcal{D}})x_{i,j} \geq q_j \quad \forall i \in \mathcal{N}^0, \forall j \in \mathcal{N}^{\mathcal{V}}, \tag{1g}$$

$$q_i - \theta(t_j - t_i)(y_j + W^{\mathcal{D}})x_{i,j} + Q^{\mathcal{D}}\left(1 - x_{i,j}\right) \geq q_j \quad \forall i \in \mathcal{N}^{\mathcal{V}}, \forall j \in \mathcal{N}^{\mathcal{V}}, \tag{1h}$$

$$q_i - \theta(t_j - t_i - \tau_0)W^{\mathcal{D}}x_{i,j} + Q^{\mathcal{D}}\left(1 - x_{i,j}\right) \geq 0 \quad \forall i \in \mathcal{N}^{\mathcal{V}}, \forall j \in \mathcal{N}^0, \tag{1i}$$

$$x_{i,j} \in \{0, 1\} \quad \forall a_{i,j} \in \mathcal{A}, \tag{1j}$$

$$0 \leq y_i \leq C^{\mathcal{D}} \quad \forall i \in \mathcal{N}^{\mathcal{V}}, \tag{1k}$$

$$0 \leq q_i \leq Q^{\mathcal{D}} \quad \forall i \in \mathcal{N}^{\mathcal{V}}, \tag{1l}$$

$$z_{-\Delta}, z_{K\Delta} = |\mathcal{D}|, \tag{1m}$$

$$z_t \in \{0, 1, \cdots, |\mathcal{D}|\} \quad \forall t \in \mathcal{T}. \tag{1n}$$

The objective of the model is to minimize the total time taken to deliver the supplies over the planning horizon. The battery replacement time is included in this horizon. Constraints (1b) are the flow balance restriction of the station-time nodes, which determine the number of available drones at each station-time node by relating the decision variable $x_{i,j}$ to z_t. Constraints (1c) are the flow balance restriction on each ship-time node. Constraints (1d) mandate that each ship must be served. Constraints (1e)–(1f) enforce the load capacity of the drones when arriving at each node. Constraints (1g)–(1i) specify the battery depletion during drone service and the remaining battery level of the drone when arriving at each node. Constraints (1j)–(1n) define the domains of the variables.

As the battery consumption constraints (1g) and (1h) are nonlinear due to the term $y_j x_{i,j}$, we introduce the new continuous variables $\gamma_{i,j} = y_j x_{i,j}$ ($i \in \mathcal{N}, j \in \mathcal{N}^{\mathcal{V}}$) to linearize it. Specifically, constraints (1g) and (1h) are transformed as follows:

$$Q^{\mathcal{D}} - \theta\left(t_j - t_i\right)(\gamma_{i,j} + W^{\mathcal{D}}x_{i,j}) \geq q_j \quad \forall i \in \mathcal{N}^0, \forall j \in \mathcal{N}^{\mathcal{V}}, \tag{2a}$$

$$q_i - \theta\left(t_j - t_i\right)(\gamma_{i,j} + W^{\mathcal{D}}x_{i,j}) + Q^{\mathcal{D}}\left(1 - x_{i,j}\right) \geq q_j \quad \forall i \in \mathcal{N}^{\mathcal{V}}, \forall j \in \mathcal{N}^{\mathcal{V}}, \tag{2b}$$

$$\gamma_{i,j} \geq y_j - C^{\mathcal{D}}(1 - x_{i,j}) \quad \forall i \in \mathcal{N}, j \in \mathcal{N}^{\mathcal{V}}, \tag{2c}$$

$$\gamma_{i,j} \geq 0 \quad \forall i \in \mathcal{N}, \forall j \in \mathcal{N}^{\mathcal{V}}. \tag{2d}$$

This model is a variant of the CVRP, which is a classic NP-hard problem. There are no known algorithms that can solve all instances of the problem in polynomial time, which reflects the computational difficulty of the problem.

4. Branch-and-price-and-cut algorithm

In this section, a BPC algorithm is presented to solve the DSP-SSD problem. We first reformulate the original model into a set covering-like model by Dantzig–Wolfe decomposition. Then, we demonstrate the specific procedures of the CG and B&B schemes, where a set of cuts are proposed to enhance the solving efficiency. Finally, the overall framework of the BPC algorithm is given.

4.1. Dantzig–Wolfe decomposition

The original model (1) resembles the classic CVRP and can be formulated as a set covering-like model by Dantzig–Wolfe decomposition. The following notation is defined:

• **Parameters:**

$\mathcal{P}$: the set of feasible trips for the drones, indexed by p. Trip p is defined as a single cycle $(i_0, i_1, \cdots, i_n)$ in which the drone takes off from the station-time node i_0, sequentially delivers supplies to ship-time nodes $i_1, \cdots, i_{n-1}$, and returns to the station-time node i_n for battery replacement.

$\mathcal{A}_p$: the set of arcs contained in trip p.

$\mathcal{V}_p$: the set of vessels visited in trip p.

$\mathcal{T}_p$: the set of time points contained in trip p. For the trip $p = (i_0, i_1, \cdots, i_n)$, we assume that $\mathcal{T}_p = \left\{ t \mid t = k\Delta, t_{i_0} \le t < t_{i_n}, k \in \mathbb{N}_0 \right\}$.

$c_{i,j}$: the time cost of arc $a_{i,j}$, i.e., $t_j - t_i$.

c_p: the total time cost of trip p, i.e., $t_{i_n} - t_{i_0}$, which includes the total flying time and the time required to replace the battery once.

$\mathbb{1}_X(x)$: an indicator function that takes the value of 1 if $x \in X$, and 0 otherwise.

• **Decision variable:**

χ_p: a binary variable. $\chi_p = 1$ if the trip is included in the drones' itineraries; and $\chi_p = 0$ otherwise.

The original model can be reformulated to select the appropriate trips to serve all ships with a minimized total drone activity time, written as follows:

[Set covering-like model]

$$\min_{\chi} \sum_{p \in \mathcal{P}} c_p \chi_p \tag{3a}$$

$$\text{s.t.} \sum_{p \in \mathcal{P}} \mathbb{1}_{\mathcal{V}_p}(v) \chi_p \ge 1 \qquad \forall v \in \mathcal{V}, \tag{3b}$$

$$\sum_{p \in \mathcal{P}} \mathbb{1}_{\mathcal{T}_p}(t) \chi_p \le |\mathcal{D}| \qquad \forall t \in \mathcal{T}, \tag{3c}$$

$$\chi_p \in \mathbb{N}_0 \qquad \forall p \in \mathcal{P}. \tag{3d}$$

The objective function (3a) attempts to minimize the total activity time of the drones including battery replacement. Constraints (3b) ensure that every vessel is served. We note that the constraints are relaxed to greater than or equal to 1, whereas under the optimal solution, the left-hand side will naturally be 1. Constraints (3c) restrict the available number of drones. Constraints (3d) regulate the variables are nonnegative integers.

Proposition 1. *Constraints (3c) are equivalent to constraints (1b) and (1n).*

The proof of Proposition 1 is given in Appendix C.

In the model, each column, corresponding to the variable χ_p, represents a feasible trip that a drone can undertake. The number of columns increases dramatically as the problem size grows. In this case, CG is crucial, as it introduces only the most promising columns iteratively. This technique significantly reduces the computational burden, allowing the practical solution of large-scale problems that would otherwise be intractable (Desrochers et al., 1992). The linear relaxation of the master model is presented in Section 4.2, where the candidate columns are generated by resolving a subproblem through dynamic programming. Typically, the resulting solution offers superior relaxation, which is then integrated into the B&B framework introduced in Section 4.3 to produce integer solutions.

4.2. Column generation

In this section, the two-phase CG scheme is introduced. In the first phase, the optimal solution of the linear master model is obtained, and in the second phase promising columns with negative reduced costs are generated.

4.2.1. Restricted master problem

The CG starts with a restricted master problem (RMP), which relaxes model (3) by linearizing the binary variables and only introducing a subset of all possible columns. The RMP can be written as follows:

[Restricted master problem]

$$\min_{\chi} \sum_{p \in \mathcal{P}'} c_p \chi_p \tag{4a}$$

$$\text{s.t.} \sum_{p \in \mathcal{P}'} \mathbb{1}_{\mathcal{V}_p}(v) \chi_p \ge 1 \qquad \forall v \in \mathcal{V}, \tag{4b}$$

$$\sum_{p \in \mathcal{P}'} \mathbb{1}_{\mathcal{T}_p}(t) \chi_p \le |\mathcal{D}| \qquad \forall t \in \mathcal{T}, \tag{4c}$$

$$\chi_p \geq 0 \qquad \forall p \in \mathcal{P}', \tag{4d}$$

where $\mathcal{P}' \subseteq \mathcal{P}$ and constraints (4d) are obtained by relaxing variable constraints (3d) into nonnegative continuous variables.

As an initial step, a dummy column corresponding to χ_0 is added into the RMP, where $\mathbb{1}_{\mathcal{V}_0}(v)$ for each ship v is set to 1, the occupied number of drones for each time point t is set to $|\mathcal{D}|$, and c_0, which represents a huge cost in this column (also known as big-M), is set to an empirical value of $10^4 \cdot |\mathcal{V}| \cdot 2R/S^{\mathcal{D}}$. Theoretically, we will have $\chi_0 = 0$ after a certain number of iterations by adding more promising columns with smaller costs; otherwise, the model is infeasible.

4.2.2. Cutting planes

To accelerate the convergence, we augment model (4) by incorporating the rounded capacity inequalities (RCIs), 2-path inequalities (TPIs), and subset-row inequalities (SRIs). The first two inequalities are robust cuts that do not change the structure of the pricing problem, while SRIs belong to nonrobust cuts which are based on a subset of the constraints in the RMP (Costa et al., 2019). We generate these inequalities at the early stage of the BPC algorithm, for example, only the root node of the B&B tree.

Rounded capacity inequalities. We employ the RCIs first proposed by Augerat (1995) to strengthen the RMP, which is to enforce drone capacity constraints more effectively by considering the total demand of a subset of vessels and ensuring that it cannot be served by a given number of drones unless their combined capacity is sufficient. For notation convenience, we denote $\mathcal{RC}$, indexed by rc, as the set of RCIs. Specifically, for a subset of vessels $\mathcal{V}^{rc}$, an RCI states that the total number of drones required to serve $\mathcal{V}^{rc}$ must be at least $\lceil \sum_{v\in\mathcal{V}^{rc}} w_v / C^{\mathcal{D}} \rceil$.

In this paper, we implement an exact separation procedure initially proposed by Fukasawa et al. (2006) to effectively generate the RCIs. Specifically, given a solution χ^* to the RMP, we can obtain $x^*_{i,j}$ for each arc $a_{i,j} \in \mathcal{A}$. Then, we define a support graph $\bar{\mathcal{G}}_\Delta = \{\mathcal{N}, \bar{\mathcal{A}}\}$, where $\bar{\mathcal{A}} = \{a_{i,j} \in \mathcal{A} : x^*_{i,j} > 0\}$. We denote binary variable r_v, $v \in (\mathcal{V} \cup \{0\})$, that represents whether a vessel v (0 is the station) is included in $\mathcal{V}^{rc}$ and variable $s_{i,j}$, $a_{i,j} \in \bar{\mathcal{A}}$, that equals 1 if arc $a_{i,j}$ satisfies $u_i \in \mathcal{V}^{rc}$ and $u_j \notin \mathcal{V}^{rc}$. A given parameter M represents the number of drones that cannot serve the subset of vessels $\mathcal{V}^{rc}$. Then, for each value of M in $\{1, \cdots, \lceil \sum_{v\in\mathcal{V}} w_v / C^{\mathcal{D}} \rceil - 1\}$, we can solve the following model:

$$\min_{\boldsymbol{s},\boldsymbol{r}} \sum_{a_{i,j}\in\bar{\mathcal{A}}} x^*_{i,j} s_{i,j} \tag{5a}$$

$$\text{s.t.}\, s_{i,j} \geq r_{u_i} - r_{u_j} \qquad \forall a_{i,j} \in \bar{\mathcal{A}}, \tag{5b}$$

$$s_{i,j} \geq r_{u_j} - r_{u_i} \qquad \forall a_{i,j} \in \bar{\mathcal{A}}, \tag{5c}$$

$$\sum_{v\in\mathcal{V}} w_v r_v \geq M \cdot C^{\mathcal{D}} + 0.001, \tag{5d}$$

$$r_0 = 0, \tag{5e}$$

$$r_v \in \{0, 1\} \qquad \forall v \in \mathcal{V}, \tag{5f}$$

$$s_{i,j} \geq 0 \qquad \forall a_{i,j} \in \bar{\mathcal{A}}. \tag{5g}$$

If the optimal objective $\sum_{a_{i,j}\in\bar{\mathcal{A}}} x^*_{i,j} s^*_{i,j} < 2(M+1)$, the capacity restriction over set $\mathcal{V}^{rc} = \{v \in \mathcal{V} : r^*_v = 1\}$ is violated and we add the following cut into the RMP:

$$\sum_{p\in\mathcal{P}'} \sum_{a\in\mathcal{A}^{rc}} \mathbb{1}_{\mathcal{A}_p}(a)\chi_p \geq M + 1, \tag{6}$$

where $\mathcal{A}^{rc} = \{a_{i,j} \in \mathcal{A} : u_i \in \mathcal{V}^{rc}, u_j \notin \mathcal{V}^{rc}\}$ is the set of arcs flowing out of the subset of vessels $\mathcal{V}^{rc}$.

2-path inequalities. Since considering capacity alone is not strong enough, we also apply the TPIs to our problem. These cuts mitigate solutions involving subsets of vessels that cannot be served by a single drone due to capacity, battery consumption, and time-window constraints. Similarly, we denote $\mathcal{TP}$, indexed by tp, as the set of TPIs. Firstly, we use the heuristic proposed by Kohl et al. (1999) to generate candidate subsets $\mathcal{V}^{tp}$. For each candidate subset, we check whether this subset of vessels can be served by one drone (in one trip) satisfying all resource constraints. As the RCI has separated all subsets of vessels that cannot be served by one drone due to capacity constraints, only energy consumption and time window have to be checked here. In traditional TPIs, this is equivalent to checking whether there is a feasible solution to the TSPTW defined on $\mathcal{V}^{tp} \cup \{0, |\mathcal{V}^{tp}| + 1\}$ (with origin 0 and destination $|\mathcal{V}^{tp}| + 1$). While this problem can be solved by a classic dynamic programming (DP) method (Dumas et al., 1995), our problem differs as the vessels are floating and nodes are defined on a time-expanded network. Therefore, we propose a tailored DP algorithm specified in Appendix D to check the feasibility of serving the subset of vessels with one drone. It should be noted that this DP algorithm is similar to but much simpler than the labeling algorithm that will be proposed in Section 4.2.3 as we only need to check the feasibility property here. Then, the TPI can be written as below.

$$\sum_{p\in\mathcal{P}'} \sum_{a\in\mathcal{A}^{tp}} \mathbb{1}_{\mathcal{A}_p}(a)\chi_p \geq 2, \tag{7}$$

where $\mathcal{A}^{tp} = \{a_{i,j} \in \mathcal{A} : u_i \in \mathcal{V}^{tp}, u_j \notin \mathcal{V}^{tp}\}$ is the set of arcs flowing out of the subset of vessels $\mathcal{V}^{tp}$.

Partial reduced cost. The RMP is solved to optimality, yielding dual prices $\boldsymbol{\pi} \geq 0$ and $\boldsymbol{\lambda} \leq 0$ associated with constraints (4b) and (4c), respectively. We also denote $\boldsymbol{\delta} \geq 0$ and $\boldsymbol{\xi} \geq 0$ as the dual variables corresponding to RCIs and TPIs, respectively. Therefore, we let b_p be the partial reduced cost of trip path p (we call b_p a "partial" reduced cost because it does not consider the nonrobust cuts), which can be written as

$$b_p = c_p - \sum_{v\in\mathcal{V}} \mathbb{1}_{\mathcal{V}_p}(v)\pi_v - \sum_{t\in\mathcal{T}} \mathbb{1}_{\mathcal{T}_p}(t)\lambda_t - \sum_{rc\in\mathcal{RC}}\sum_{a\in\mathcal{A}^{rc}} \mathbb{1}_{\mathcal{A}_p}(a)\delta_{rc} - \sum_{tp\in\mathcal{TP}}\sum_{a\in\mathcal{A}^{tp}} \mathbb{1}_{\mathcal{A}_p}(a)\xi_{tp}. \tag{8}$$

Then, the partial reduced cost for each arc $a_{i,j} \in \mathcal{A}_p$ can be written as

$$b_{i,j} = c_{i,j} - \sum_{v\in\mathcal{V}} \mathbb{1}_{u_i}(v)\pi_v - \sum_{t\in\mathcal{T}} \mathbb{1}_{\mathcal{T}_{(i,j)}}(t)\lambda_t - \sum_{rc\in\mathcal{RC}} \mathbb{1}_{\mathcal{A}^{rc}}(a_{i,j})\delta_{rc} - \sum_{tp\in\mathcal{TP}} \mathbb{1}_{\mathcal{A}^{tp}}(a_{i,j})\xi_{tp}, \tag{9}$$

where $\mathcal{T}_{(i,j)} = \left\{t \mid t = k\Delta, t_i \le t < t_j, k \in \mathbb{N}_0\right\}$.

Subset-row inequalities. We concentrate on the inequalities specified for three vessels because they can be separated by straightforward enumeration (Jepsen et al., 2008; Tilk et al., 2018). We denote the index set of vessel subsets as $\mathcal{SR}$, where $|\mathcal{SR}| = \binom{|V|}{3} = \frac{|V|!}{3!(|V|-3)!}$, and the subsets of all three vessels as $\mathcal{V}^{sr}$, $sr \in \mathcal{SR}$. For each subset $\mathcal{V}^{sr}$, an SRI is added to the RMP, which can be written as follows:

$$\sum_{p\in\mathcal{P}'} \left\lfloor \frac{\sum_{v\in\mathcal{V}} \mathbb{1}_{\mathcal{V}^{sr}\cap\mathcal{V}_p}(v)}{2} \right\rfloor \chi_p \le 1. \tag{10}$$

By incorporating inequality (10) for each $sr \in \mathcal{SR}$ into the RMP, we can cut out many fractional solutions so as to lift the LB of the B&B tree. These SRIs require us to make specific modifications to our pricing problems. We denote $\mu_{sr} \le 0$ as the dual price associated with $sr \in \mathcal{SR}$. The value μ_{sr} is subtracted from the reduced cost of path p if the path includes two or three vessels in subset $\mathcal{V}^{sr}$. Therefore, the full reduced cost of p, denoted by $\bar{c}_p$, can be computed as follows:

$$\bar{c}_p = b_p - \sum_{sr\in\mathcal{SR}} \mathbb{1}_{\mathcal{SR}(p)}(sr)\mu_{sr}, \tag{11}$$

where $\mathcal{SR}(p) = \{sr \in \mathcal{SR} : |\mathcal{V}_p \cap \mathcal{V}^{sr}| \ge 2\}$.

4.2.3. Pricing problem

The pricing problem in the CG is to identify new columns with a negative reduced cost, i.e., promising trip path $p \in \mathcal{P}\backslash\mathcal{P}'$ for the drones. The binary decision variable $x_{i,j}$ takes the value of 1 if arc $a_{i,j}$ is included in the new path, and 0 otherwise. We further define $\hat{\mathcal{V}} := \{u_i \in \mathcal{V} : x_{i,j} = 1, i \in \mathcal{N}^{\mathcal{V}}, j \in \mathcal{N}\}$ and $\hat{\mathcal{SR}} := \{sr \in \mathcal{SR} : |\hat{\mathcal{V}} \cap \mathcal{V}^{sr}| \ge 2\}$. Then, given the dual prices obtained from the RMP, the pricing problem in our case can be presented as follows.

[Pricing problem]

$$\min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{q}} \sum_{a_{i,j}\in\mathcal{A}} b_{i,j}x_{i,j} - \sum_{sr\in\mathcal{SR}} \mathbb{1}_{\hat{\mathcal{SR}}}(sr)\mu_{sr} \tag{12a}$$

$$\text{s.t.} \sum_{a_{i,j}\in\mathcal{A}^0 : u_i=0} x_{i,j} = 1, \tag{12b}$$

$$\sum_{a_{j,i}\in\mathcal{A}^0 : u_i=0} x_{j,i} = 1, \tag{12c}$$

(1c),(1e)–(1l).

The classic pricing problem, which is always reduced to an elementary shortest path problem with resource constraints (ESPPRC), is NP-hard (Dror, 1994; Garey and Johnson, 1983). The labeling algorithm, a DP approach, has been demonstrated to be an effective method for solving the ESPPRC (Feillet et al., 2004; Righini and Salani, 2006). Our model (12) is a variant of the ESPPRC where the depot is differentiated by distinct time points and battery consumption constraints are imposed. As a drone's battery consumption is associated with its total loaded weight, i.e., the total load required by the subsequent ships that the drone visits, we propose a backward labeling algorithm to generate the trip path.

Let $L_p = (\mathcal{V}_p, y^p, q^p, \mathcal{T}_p, i^p, b_p, \bar{c}_p)$ be the label vector of the elementary backward path $p = (i_0, i_1, \cdots, i^p)$, which means that the drone flies to vessel-time nodes $i^p, \cdots, i_1$ sequentially, and finally arrives at the station-time node i_0. $\mathcal{V}_p$ is the visited vessel on path p; y^p is the total weight of supplies on path p; q^p is the battery consumption of a drone traveling from i^p to i_0; $\mathcal{T}_p$ is the set of time points covered on path p; i^p is the last node of backward path p; b_p is the partial reduced cost of path p, including costs defined on every single arc and excluding the cost contributed by the dual variables corresponding to the SRIs; $\bar{c}_p$ is the full reduced cost corresponding to path p. Due to the load capacity and battery consumption constraints, a label L_p is feasible only if:

$$y^p \le C^{\mathcal{D}}, \quad q^p \le Q^{\mathcal{D}}. \tag{13}$$

The feasible backward path is generated following Appendix F.1. Denote $\bar{\mathcal{V}}_p$ as the set of vessels that path p can further access (satisfying capacity, battery consumption, and time-window constraints). To speed up the solution process and limit the number of labels generated, the following dominance rule can be employed, the proof of which is presented in Appendix E.

Proposition 2. *Given two labels $L_{p_1} = (\mathcal{V}_{p_1}, y^{p_1}, q^{p_1}, \mathcal{T}_{p_1}, i^{p_1}, b_{p_1}, \bar{c}_{p_1})$ and $L_{p_2} = (\mathcal{V}_{p_2}, y^{p_2}, q^{p_2}, \mathcal{T}_{p_2}, i^{p_2}, b_{p_2}, \bar{c}_{p_2})$ in $List_L$ or $List_{L^*}$, L_{p_2} is dominated by L_{p_1} if the following conditions are satisfied: (i) $\bar{\mathcal{V}}_{p_2} \subseteq \bar{\mathcal{V}}_{p_1}$, (ii) $y^{p_1} \leq y^{p_2}$, (iii) $q^{p_1} \leq q^{p_2}$, (iv) $\mathcal{T}_{p_1} \subseteq \mathcal{T}_{p_2}$, (v) $i^{p_1} = i^{p_2}$, (vi) $b_{p_1} \leq b_{p_2}$, and (vii) $(\mathcal{V}^{sr} \cup \mathcal{V}_{p_1}) \subseteq (\mathcal{V}^{sr} \cup \mathcal{V}_{p_2}), \forall sr \in \mathcal{SR}$.*

Heuristic labeling method. To avoid generating too many labels with poorly reduced costs, we selectively retain only the first Num_L partial labels with the most negative reduced costs during the label generation process. Notably, when no complete label with a negative reduced cost can be generated through the selected partial labels, we extend all of the partial labels to check again whether there are still columns with negative reduced costs. An indicator parameter "flag" is used to control whether to extend all of the partial labels. Additionally, when returning columns after each round of label generation, we return the top Num_{L^*} columns with the most negative reduced costs (if there are fewer than Num_{L^*} columns with negative reduced costs, we return as many as there are). The new columns corresponding to the backward paths are then added to the RMP.

4.3. Branch-and-bound

The RMP with SRIs is solved by the CG algorithm until no more columns with negative reduced costs can be generated. If the solution to the RMP is not integral, the B&B scheme is introduced by selecting a branching variable and creating two new subproblems, each with an additional constraint that forces the variable to take an integer value on one branch. In this section, we first discuss how to obtain the upper bound (UB) and lower bound (LB) of the B&B tree. We then demonstrate the specific branching strategy.

4.3.1. Upper and lower bounds

In a B&B tree, the UB is the value of the best feasible solution found so far, which serves as a benchmark for evaluating the potential of other solutions. To obtain a UB at the early stage of the B&B tree, we apply two methods in the root node. Firstly, we adopt the feasibility pump (FP) algorithm proposed by Fischetti et al. (2005), which is a heuristic algorithm designed to quickly find a feasible solution by repeatedly adjusting a non-integer solution to the RMP. The main idea of the algorithm is to round the fractional variables to the nearest integer and then solve a modified version of the RMP to minimize the distance from this rounded solution. This is called a pumping action between continuous and discrete spaces, which seeks to strike a balance between satisfying the problem's constraints and adhering to the integrality requirements of the variables. The pseudocode of the FP algorithm is given in Appendix F.2. If no feasible solution is found by FP, we solve the IP form of the RMP in the root node, which includes a small scale of columns and can be solved quickly by the state-of-the-art solver (Yang, 2023b). If a feasible integer solution is found, the global UB is set to the objective value of this solution; otherwise, the global UB is set to $+\infty$. The global UB is updated when a new feasible integer solution is found to have an objective value smaller than the global UB.

The local LB is obtained by solving the RMP (4) with inequalities after CG, which significantly improves the objective value compared with solving the simple linearized version of the set covering-like model (Jepsen et al., 2008). The global LB is updated to the smallest local LB of all leaf nodes. Notably, we will stop the CG prematurely if the algorithm has reached the time limit and will compute an LB based on the LP value of the current RMP and the current reduced cost (Vanderbeck and Wolsey, 1996).

4.3.2. Branching scheme

In our B&B tree, we adopt a branching strategy by branching on arcs, which has been proven to be powerful for the VRP in expediting convergence and simplifying the problem structure through straightforward implementation (Desaulniers et al., 2006). Specifically, we adhere to the procedures outlined below.

Node selection: When there are multiple nodes with fractional solutions to be extended, we follow the best-bound-first strategy proposed by Toth and Vigo (2014), Land and Doig (1960). This strategy aims to speed up the convergence process by concentrating on the region that is most likely to contain the optimal solution. In our minimization problem, this strategy selects the currently unexplored node with the smallest estimated bound for branching, where the estimated bound for each node is set to the objective value of its parent node. When multiple nodes have the same estimated bound, e.g., they are child nodes branching from the same parent node, we randomly choose one of them.

Arc selection: Given a solution χ^* to the RMP of the chosen node, we can calculate the value of $x^*_{i,j}$ for each arc $a_{i,j} \in \mathcal{A}$. When the solution contains multiple fractional arcs, we follow the arc selection rule proposed by Desrochers et al. (1992) and look for arc $a_{i,j}$ such that $x^*_{i,j}$ is closest to 0.5 for branching. Ties are broken by selecting the arc with the smallest cost $c_{i,j}$.

Branching on arc: The chosen arc is fixed at 0 and 1 in each child node. In a node where arc $a_{i,j}$ is fixed at 0, the arc is directly removed from the pricing subproblem of the node. The columns corresponding to the path that contains arc $a_{i,j}$ are removed from the RMP. In the other nodes where arc $a_{i,j}$ is fixed at 1, we remove the arcs in

$$\bar{\mathcal{A}}_1(a_{i,j}) = \{(k,l) \in \mathcal{A} | (u_k = u_i, k \neq i) \vee (k = i, l \neq j) \vee (u_l = u_i, l \neq i)\} \text{ if } i \in \mathcal{N}^{\mathcal{V}}, \tag{14}$$

and

$$\bar{\mathcal{A}}_2(a_{i,j}) = \{(k,l) \in \mathcal{A} | (u_k = u_j, k \neq j) \vee (l = j, k \neq i) \vee (u_l = u_j, l \neq j)\} \text{ if } j \in \mathcal{N}^{\mathcal{V}}. \tag{15}$$

Specifically, the three conditions in Eq. (14) forbid three kinds of arcs, i.e., the arcs starting from the other nodes of the same vessel of node i, the arcs starting from node i but not connecting to node j, and the arcs ending at other nodes of the same vessel of node i. The three conditions in Eq. (15) forbid three kinds of arcs, i.e., the arcs starting from the other nodes of the same vessel of node j, the arcs ending at node j but not starting to node i, and the arcs ending at other nodes of the same vessel of node j. Therefore, the arc $a_{i,j}$ must be chosen as long as the vessel u_i is served. Similarly, columns with paths that use specific arcs are removed from the RMP according to (14)–(15).

The overall pseudocode of the BPC algorithm is presented in Appendix F.3.

5. Numerical experiments

In this section, we generate several testing instances based on vessel data from real ports and drone-related data obtained from UAV companies to implement our proposed BPC algorithm and analyze the critical parameters. All of the experiments are carried out on three computers running the Windows operating system with a 3.70-GHz Intel Core i7 processor with 64 GB of RAM, using 12 threads to take advantage of the multiple cores available. The models and algorithms are implemented by Java and solved by CPLEX 12.7.

5.1. Experiment setting

This section introduces the data used to generate the instances. The drone-related data are mostly sourced from (DJI, 2024) with the parameter θ adjusted to meet the shore-to-ship delivery duration and the ship-related parameters are generated following the procedures in Xia et al. (2019).

• **Drones.** Each drone travels at a constant speed (S^D) of 0.72 km/min. The maximum payload of a drone (C^D) is 30 kg, where the dead weight (W^D) is 65 kg. The battery's capacity (Q^D) is 38, 000 mAh and the battery consumption ratio (θ) is 4.5 mAh/(kg min). Battery replacement takes 12 min (τ_0).

• **Ships.** The speed of the ships is randomly generated from a uniform distribution ranging from 5 to 7 knots (0.154 to 0.216 km/min). The demand of each ship (w_v) is generated from a uniform distribution over the interval $[5, 8]$ kg. The earliest and latest service points of each ship are determined by generating random coordinates in a semi-circle of radius 32 km, which is the boundary of the service range that a fully loaded drone can cover, centered on the drone station. The ships are assumed to sail along a straight line within the service boundary. The earliest serving time, t_{ve}, for each ship is obtained from a uniform distribution over the interval $[0, 300]$ min, and the latest serving time, t_{vl}, is then computed based on the earliest service time, the ship's speed, and the earliest and latest service points.

• **Instances.** We denote each instance as $V - D$, where V is the number of ships and D is the number of drones. The number of drones is selected from $\{5, 10, 15\}$, and the number of ships is chosen from $\{10, 20, 30, 40, 50\}$. For each combination of V and D, we generate 10 instances. For a particular instance at each scale, we add the suffix i to represent the ith instance in this scale, denoted as $V - D - i$. To ensure the feasibility of the instance as far as possible, we set the value of T to $\max\{t_{vl}, \forall v \in \mathcal{V}\} + R/S^D + \tau_0$, which is the latest delivery time of the ship plus the maximum travel time of a drone from the ship to the station plus the battery replacement time.

• **Algorithm parameters.** The discretization granularity, Δ, in all of the experiments except that in Section 5.3.1 is set to 5 min. The time limit is designated as 3600 s for small- and medium-scale instances where $V < 30$, and 7200 s for large-scale instances where $V = 30$. The optimality gap is set to 0.001%. The label reduction parameters Num_L and Num_{L*} are set to 2000 and 200, respectively. The maximum iteration number is set to 100 and the number of variables to be flipped is set to $|\chi^*|/6$ in the FP algorithm.

5.2. Computational results

In this section, we solve the instances using CPLEX, the BPC algorithm, and variants of the BPC algorithm to demonstrate the superiority of our proposed BPC algorithm in terms of computational efficiency and effectiveness.

5.2.1. Performance comparison of CPLEX and the BPC algorithm

Comparison of the average performance of CPLEX and the BPC algorithm for different scales of instances are provided in Table 1. The following details are provided.

• **#Opt**: the number of instances that are solved to optimality at each scale within the time limit.

• **#Fea**: the number of instances where feasible integer solutions are obtained at each scale within the time limit but the algorithm has not converged to the optimality gap.

• **#Unk**: the number of instances where no feasible integer solution is obtained at each scale within the time limit, i.e., the solution status is unknown.

• $\overline{\textbf{Gap}}$ (%): the average gap of the instances where feasible solutions are obtained at each scale. The gap(%) of each instance is calculated by $(\mathrm{UB} - \mathrm{LB})/\mathrm{UB} \times 100$.

• **Gap**$_{\max}$ (%): the maximum (worst) gap of the instances where feasible solutions are obtained at each scale.

• **CPU (s)**: the total computational CPU time in seconds.

• **Avg.**: the average results of each metric for all instances at different scales.

For small-scale instances such as 10–5, both algorithms achieve optimal solutions with a zero gap. However, the BPC algorithm demonstrates a clear advantage in computational efficiency, solving the problem in an average of 1 s compared with CPLEX's 841.10 s. As the problem size increases to instances like 20–5 and 20–10, CPLEX struggles to find optimal solutions, hitting the maximum computational time cap of 3600 s with high average gaps exceeding 38%. It fails to find an optimal solution in any of the instances and the worst gaps of 20–5 and 20–10 soar to 47.6% and 50.1%, respectively. In contrast, BPC manages to adapt more effectively to these moderate problem sizes, resolving many instances with substantially lower gaps and in considerably less time, achieving a worst-case gap of just 3.1% and 3.5% in the 20–5 and 20–10 instances, respectively.

Table 1
The results of CPLEX and BPC on different instances.

Instance	CPLEX						BPC				
	#Opt	#Fea	#Unk	$\overline{\text{Gap}}$ (%)	$\text{Gap}_{\max}$ (%)	CPU (s)	#Opt	#Fea	$\overline{\text{Gap}}$ (%)	$\text{Gap}_{\max}$ (%)	CPU (s)
10–5	10	0	0	0.00	0.00	841.10	10	0	0.00	0.00	1.00
20–5	0	10	0	38.67	47.60	3600.00	6	4	0.62	3.10	1823.13
20–10	0	10	0	41.90	50.10	3600.00	8	2	0.56	3.50	1103.83
30–5	0	1	9	60.40	60.40	7200.00	2	8	1.62	3.80	6103.45
30–10	0	9	1	50.58	61.10	7200.00	3	7	1.01	2.70	5259.07
40–10	0	4	6	63.80	72.90	7200.00	1	9	3.02	12.80	6767.17
40–15	0	6	4	69.62	75.30	7200.00	0	10	2.14	9.03	7200.00
50–15	0	2	8	71.65	75.70	7200.00	0	10	5.21	20.20	7200.00
Avg.	1.25	5.25	3.50	49.58	55.39	5505.14	3.75	6.25	1.77	6.89	4432.20

Table 2
The results of different node selection rules.

Instance	Best-bound-first			Depth-first			Width-first			Random-selection		
	Gap (%)	CPU (s)	#Node	Gap (%)	CPU (s)	#Node	Gap (%)	CPU (s)	#Node	Gap (%)	CPU (s)	#Node
20–5–1	0.20	3600.00	410	2.10	3600.00	136	1.70	3600.00	174	1.70	3600.00	168
20–5–2	0.00	627.69	18	2.00	3600.00	162	1.90	3600.00	224	1.90	3600.00	228
20–5–3	0.00	23.77	0	0.00	24.74	0	0.00	24.24	0	0.00	24.20	0
20–5–4	3.10	3600.00	500	3.70	3600.00	440	3.70	3600.00	578	3.70	3600.00	558
20–5–5	0.00	278.03	10	0.00	530.66	18	0.00	540.44	18	0.00	673.36	24
20–5–6	0.40	3600.00	130	0.70	3600.00	150	0.10	3600.00	74	1.10	3600.00	70
20–5–7	0.00	2710.94	230	1.40	3600.00	194	1.40	3600.00	296	1.40	3600.00	258
20–5–8	0.00	153.93	0	0.00	156.87	0	0.00	155.36	0	0.00	160.61	0
20–5–9	0.00	36.74	0	0.00	36.83	0	0.00	36.60	0	0.00	37.49	0
20–5–10	2.50	3600.00	156	2.60	3600.00	78	2.60	3600.00	110	2.80	3600.00	106
Avg.	0.62	1823.11	145.4	1.25	2234.91	117.8	1.14	2235.66	147.4	1.26	2249.57	141.2

As the problem complexity further increases (for instances like 30−5 and 30−10), CPLEX's performance deteriorates with no optimal solutions and worst-case gaps reaching up to 61.1% in the 30−10 instance. The computational time reaches 7200 s for these instances, highlighting significant limitations in scalability. The BPC algorithm, on the other hand, demonstrates superior performance across all metrics, successfully finding feasible solutions in all instances. It maintains a remarkably low average gap of 1.62% and 1.01% for instance 30−5 and 30−10, respectively, indicating its effectiveness at closely approximating or achieving an optimal solution. The BPC algorithm's average CPU time is also less than 7200 s, which is significantly lower than that of CPLEX.

In the largest and most complex instances, such as 40−10, 40−15, and 50−15, the performance differential between CPLEX and BPC is markedly pronounced. CPLEX fails to find feasible solutions for more than half of the instances and encounters the worst-case gap peaking at 75.70% for the 50−15 instances, while consistently exhausting the computational limit of 7200 s. BPC, while also reaching maximum computational time, maintains significantly lower average gaps, with a comparatively small worst-case gap of 20.20% in the 50−15 instance. Moreover, it can always find feasible solutions for all instances.

In summary, our proposed BPC algorithm displays better performance not only in consistently finding optimal solutions to more instances but also in maintaining lower gap percentages. Moreover, this superiority is evident despite the BPC algorithm's higher CPU time in complex cases, as it still offers much lower average and worst gaps than CPLEX, suggesting that the BPC algorithm is more effective at navigating and solving large-scale problems.

5.2.2. *Effectiveness of the branching and node selection strategies*

In this section, we compare our branching and node selection strategies with other methods commonly used in the literature.

Firstly, we test our best-bound-first strategy with depth-first, width-first, and random-selection strategies on each instance on a scale of 20−5, the results of which are presented in Table 2. It can be observed that the best-bound-first strategy demonstrates significant efficacy by consistently achieving the lowest optimality gaps across multiple instances, surpassing other strategies. The average gap for best-bound-first is 0.62%, considerably lower than depth-first (1.25%), width-first (1.14%), and random (1.26%). In terms of computational efficiency, best-bound-first also outperforms the other methods with an average CPU time of 1823.11 s, which is lower than the other strategies, reflecting not only its ability to find closer solutions to the optimal but also its efficiency in computational expenditure. Furthermore, the number of nodes processed by best-bound-first (average of 145.4) illustrates a balanced approach, effectively navigating between exploring many paths and focusing on promising ones.

We also validate our branching strategy by comparing it to the strategy of branching on the number of drones and then on the arc. The specific procedure of branching on the number of drones is given in Appendix G and the performances of both methods

Table 3
The results of different branching strategies.

Instance	Branch on arc			Branch on drone number then on arc		
	Gap (%)	CPU (s)	#Node	Gap (%)	CPU (s)	#Node
20–5–1	0.20	3600.00	410	1.40	3600.00	172
20–5–2	0.00	627.69	18	1.60	3600.00	282
20–5–3	0.00	23.77	0	0.00	24.94	0
20–5–4	3.10	3600.00	500	3.40	3600.00	640
20–5–5	0.00	278.03	10	0.00	557.65	18
20–5–6	0.40	3600.00	130	0.10	3600.00	80
20–5–7	0.00	2710.94	230	1.10	3600.00	300
20–5–8	0.00	153.93	0	0.00	162.75	0
20–5–9	0.00	36.74	0	0.00	37.80	0
20–5–10	2.50	3600.00	156	2.60	3600.00	116
Avg.	0.62	1823.11	145.4	1.02	2238.31	160.8

Table 4
The acceleration effects of BPC on solution quality and solving efficiency.

Instance	BPC				BPC-V1				BPC-V2				BPC-V3			
	UB	Gap (%)	CPU (s)	#Node	UB	Gap (%)	CPU (s)	#Node	UB	Gap (%)	CPU (s)	#Node	UB	Gap (%)	CPU (s)	#Node
20–5–1	13.50	0.20	3600.00	410	13.50	1.30	3600.00	176	13.50	2.30	3600.00	5132	13.50	1.70	3600.00	90
20–5–2	13.00	0.00	627.69	18	/[a]	100.00	3600.00	290	13.00	1.60	3600.00	4902	13.00	1.70	3600.00	162
20–5–3	13.50	0.00	23.77	0	13.50	0.00	24.62	0	13.50	0.00	2.71	0	13.50	0.00	155.22	0
20–5–4	14.67	3.10	3600.00	500	/	100.00	3600.00	634	14.75	5.80	3600.00	7250	14.75	4.00	3600.00	644
20–5–5	13.50	0.00	278.03	10	13.50	0.00	561.48	18	13.50	1.40	3600.00	4868	13.50	0.00	1196.24	14
20–5–6	13.83	0.40	3600.00	130	14.83	6.90	3600.00	126	13.83	0.60	3600.00	2802	13.83	0.20	3600.00	24
20–5–7	15.25	0.00	2710.94	230	/	100.00	3600.00	326	15.25	0.70	3600.00	4172	15.42	2.20	3600.00	190
20–5–8	12.08	0.00	153.93	0	12.08	0.00	160.80	0	12.08	0.00	121.72	70	12.08	0.00	813.49	0
20–5–9	14.75	0.00	36.74	0	14.75	0.00	37.43	0	14.75	0.00	282.54	328	14.75	0.00	324.10	0
20–5–10	12.58	2.50	3600.00	156	/	100.00	3600.00	116	12.58	2.40	3600.00	1810	13.25	7.50	3600.00	30
Avg.	13.67	0.62	1823.11	145.4	/	40.82	2238.43	168.6	13.67	1.48	2560.70	3133.4	13.76	1.73	2408.91	115.4

[a] It means no upper bound is found in the time limit and the gap is 100%.

are shown in Table 3. These results highlight the efficiency and effectiveness of our strategy. Firstly, our branching rule achieves consistently lower optimality gaps, with a remarkable zero gap in 6 instances and an average gap of 0.62% versus 1.02% for the alternative approach. Moreover, this strategy often requires less computational time, as evidenced by quicker resolutions in instances like 20–5–2 and 20–5–5, and processes fewer nodes on average (145.4 compared to 160.8), suggesting a more efficient and focused search path.

5.2.3. *Benefits of acceleration strategies for the BPC algorithm*

In this section, we verify the effect of acceleration strategies on the solving efficiency of the BPC algorithm. We denote BPC-V1 as the BPC algorithm without performing the FP algorithm or solving the IP form of RMP in the root node, BPC-V2 as the BP algorithm without SRIs, RCIs, and TPIs, and BPC-V3 as the BPC algorithm without the heuristic labeling method. The original BPC algorithm and the variants with the acceleration strategies are tested on medium-scale instances (20–5), the detailed results of which are provided in Table 4. Note that the UB and LB are in hours.

As shown in Table 4, from the perspective of solution quality, the original BPC algorithm demonstrates robust performance, achieving optimal solutions in most instances with an average gap of just 0.62%. This result indicates that using a combination of acceleration strategies with the BPC is highly effective. BPC-V1, which does not use IP form or the FP algorithm to derive the UBs in the root node, fails to find a UB in four of the ten instances and has a much larger gap of 40.82%, which suggests that the importance of recognizing a feasible integer solution at the early stages of the BPC algorithm. The second variant, which removes the inequalities used to cut out fractional solutions, results in a significant drop in performance, with the average gap increasing to 1.48%. The increased gap indicates that the inequalities are vital for the BPC algorithm's ability to efficiently solve these problems, and their absence leads to looser bounds and reduced overall efficiency. Similar to BPC-V2, excluding the heuristic labeling method causes the average gap and average UB to increase to 1.73% and 13.76, respectively. The degradation in the optimality gaps and recognized UBs demonstrates the importance of heuristic labeling in achieving efficient solutions. Heuristic labeling appears to be crucial for navigating the solution space effectively and ensuring that solutions remain optimal.

Table 4 also compares the original BPC algorithm with its three variants on the CPU time in seconds (CPU (s)) and the number of nodes explored (#Node). It can be observed that BPC-V1 explores a greater number of nodes in five of the ten instances, indicating that the lack of effective UBs results in inefficient pruning in the B&B tree, as it cannot discard suboptimal branches early in the process. Therefore, it also has a larger average computational time of 2238.43 s than the BPC. BPC-V2 often explores significantly more nodes, indicating less effective pruning. Despite exploring more nodes, BPC-V2 also shows a larger average CPU time than both the original BPC algorithm and the other two variants. This result suggests that BPC-V2, by not including the inequalities, is likely

Table 5
The results of applying different cuts.

Instance	All cuts			SRI only			RCI only			TPI only		
	Gap (%)	CPU (s)	#Node	Gap (%)	CPU (s)	#Node	Gap (%)	CPU (s)	#Node	Gap (%)	CPU (s)	#Node
20–5–1	0.20	3600.00	410	1.60	3600.00	2390	0.90	3600.00	3126	1.60	3600.00	8290
20–5–2	0.00	627.69	18	1.50	3600.00	3032	1.70	3600.00	7284	1.70	3600.00	7308
20–5–3	0.00	23.77	0	0.00	24.27	0	0.00	1.68	0	0.00	1.66	0
20–5–4	3.10	3600.00	500	4.80	3600.00	4114	4.30	3600.00	7704	4.30	3600.00	8040
20–5–5	0.00	278.03	10	1.70	3600.00	3928	1.70	3600.00	7414	1.50	3600.00	5836
20–5–6	0.40	3600.00	130	1.10	3600.00	2632	1.10	3600.00	1176	1.10	3600.00	32
20–5–7	0.00	2710.94	230	0.40	3600.00	4066	1.00	3600.00	2622	0.30	3600.00	1940
20–5–8	0.00	153.93	0	0.00	155.65	0	0.10	3600.00	2168	0.00	3600.00	0
20–5–9	0.00	36.74	0	0.00	37.31	0	0.00	8.34	2	0.00	35.81	2
20–5–10	2.50	3600.00	156	2.60	3600.00	2038	2.60	3600.00	2698	2.60	3600.00	248
Avg.	0.62	1823.11	145.4	1.37	2541.72	2220.0	1.34	2881.00	3419.4	1.31	2883.75	3169.6

to achieve faster processing times per node because it deals with a less constrained and thus simpler pricing problem at each node. However, it leads to poorer relaxations in each node, which makes it harder for the algorithm to converge. BPC-V3 explores fewer nodes but has a larger computation time on average than the original BPC algorithm, which suggests that each node's processing is computationally more intensive due to the presence of too many labels in each iteration of the CG procedure.

5.2.4. *Detailed analysis on different cuts*

To assess the impact of each type of cut on our algorithm, we solve the medium-scale instances (20−5) using algorithms that exclusively utilize either SRI, RCI, or TPI. The detailed results are shown in Table 5. Our findings illustrate that employing a combination of all cuts significantly enhances performance over using any single type of cut. For instance, the average optimality gap with all cuts is considerably lower at 0.62%, as opposed to 1.37% for SRI-only, 1.34% for RCI-only, and 1.31% for TPI-only. Furthermore, both the computation time and the number of nodes processed are substantially reduced when all types of cuts are employed. The average computation time for the algorithm incorporating all cuts stands at 1823.11 s, significantly lower than the times observed when only one type of cut is used, which range from 2541.72 to 2881.00 s, indicating a more efficient search process and potentially superior pruning within the B&B tree.

When comparing the performances of employing each type of cut individually, notable differences in efficiency and effectiveness emerge. The SRI-only algorithm exhibits a slightly higher average optimality gap at 1.37%, closely followed by RCI-only and TPI-only at 1.34% and 1.31%, respectively. Interestingly, both the SRI-only and TPI-only algorithms achieve the optimal solution in three out of ten instances, whereas the RCI-only algorithm obtains the optimal solution in only two. The relatively smaller optimality gap of applying only TPI may stem from its effectiveness in cutting out the infeasible solutions by including the capacity, battery consumption, and time-window constraints. In terms of computational efficiency, SRI yields the longest CPU time per node. Specifically, the average CPU times for the algorithms with each type of cut are all above 2500 s. However, algorithms employing only RCI and TPI explore a significantly larger average number of nodes (3419.4 and 3169.6, respectively) compared to those using SRI-only (2220). This suggests that the SRI may be the least effective at reducing the gap when used alone, but it is crucial in closing the gap for certain instances.

In addition, we also compare the algorithm efficiency by applying the SRI in different layers of the B&B tree with results detailed in Table 6. The outcomes reveal marked improvements in solving efficiency with the increased number of layers applying SRI. Specifically, when SRI is applied only at the root node, the average optimality gap is 1.45%, which progressively decreases to 0.62% as SRI is extended to all nodes, highlighting a significant tightening of the solution space. Concurrently, the average CPU time decreases from 2541.94 s to 1823.11 s, and the number of nodes processed dramatically reduces from 2822.6 to just 145.4 on average, indicating more effective pruning and faster convergence towards optimal solutions when SRI is more extensively applied. It is worth noting that the number of layers applying the SRI can serve as a hyperparameter, which should be chosen reasonably for different datasets.

5.2.5. *Summary of the algorithm comparison*

Overall, the experiments in Section 5.2 reveal the superiority of the BPC in terms of effective exploration of the solution space and handling of the complexities inherent in large-scale instances. This superiority may be due to the advantages of a trip-based model structure and algorithm design. Firstly, the BPC algorithm, with its CG technique, dynamically generates and incorporates only the most promising trips into the model, efficiently handling this complexity. In contrast, the arc-flow model used by CPLEX involves variables for every arc in the time-expanded network, leading to a large number of variables and constraints. Secondly, our design and choice of branching and node selection strategies are appropriate for this problem. Thirdly, the BPC algorithm is enhanced by integrating several acceleration strategies, which results in the algorithm handling complex problems more effectively than CPLEX in specific problem contexts. For instance, implementing problem-specific inequalities and heuristic labeling methods allows the algorithm to quickly find high-quality feasible solutions, guiding the search process more effectively than the general-purpose heuristics used in CPLEX.

Table 6
The results of applying SRI in different nodes.

Instance	Only in root node			First 3 levels of the B&B tree			First 5 levels of the B&B tree			In all nodes		
	Gap (%)	CPU (s)	#Node	Gap (%)	CPU (s)	#Node	Gap (%)	CPU (s)	#Node	Gap (%)	CPU (s)	#Node
20–5–1	2.40	3600.00	5032	1.60	3600.00	4006	1.60	3600.00	4316	0.20	3600.00	410
20–5–2	1.80	3600.00	4304	1.50	3600.00	4094	0.00	616.59	238	0.00	627.69	18
20–5–3	0.00	23.20	0	0.00	23.03	0	0.00	24.03	0	0.00	23.77	0
20–5–4	5.20	3600.00	5442	4.80	3600.00	4972	4.80	3600.00	8428	3.10	3600.00	500
20–5–5	1.40	3600.00	4634	1.40	3600.00	4308	0.00	280.87	10	0.00	278.03	10
20–5–6	0.60	3600.00	2660	1.10	3600.00	2844	1.10	3600.00	2524	0.40	3600.00	130
20–5–7	0.70	3600.00	4064	0.60	3600.00	5414	0.60	3600.00	9458	0.00	2710.94	230
20–5–8	0.00	158.41	0	0.00	151.82	0	0.00	151.26	0	0.00	153.93	0
20–5–9	0.00	37.84	0	0.00	35.56	0	0.00	35.28	0	0.00	36.74	0
20–5–10	2.40	3600.00	2090	2.60	3600.47	2162	2.60	3600.00	1996	2.50	3600.00	156
Avg.	1.45	2541.94	2822.6	1.36	2541.09	2780.0	1.07	1910.80	2697.0	0.62	1823.11	145.4

Table 7
The scale of the expanded-time network under different discretization granularity.

Instance	Δ	#Avg. time-space nodes	#Avg. arcs
20–5	1	3765.4	29 195.6
	2	1884.6	14 614.5
	5	755.6	5844.4
	10	379.0	2920.7

Table 8
The results of CPLEX and BPC with different discretization granularity.

Instance	Δ	CPLEX						BPC					
		#Opt	#Fea	#Unk	$\overline{\text{Gap}}$ (%)	UB	CPU (s)	#Opt	#Fea	#Infea[a]	$\overline{\text{Gap}}$ (%)	UB	CPU (s)
20–5	1	0	10	0	48.57	13.06	3600.00	7	3	0	0.84	11.97	1597.38
	2	0	10	0	44.03	13.81	3600.00	7	3	0	0.60	12.56	1426.49
	5	0	10	0	38.67	14.47	3600.00	6	4	0	0.62	13.67	1823.13
	10	0	7	3	36.89	17.48	3600.00	7	1	2	0.28	16.29	484.73

[a] It presents that the model is infeasible. Notably, it differs from "#Unk" where the model may be feasible but no feasible integer solution is found by the algorithm.

5.3. Sensitivity analysis

5.3.1. Discretization granularity

In the following experiments, instances at scales of 20−5 are tested with different discretization granularity values Δ, varying from $\{1, 2, 5, 10\}$ (min). The average numbers of time-space nodes and arcs with different discretization granularity are presented in Table 7. As the granularity increases from 1 to 10, there is a notable reduction in both the average number of time-space nodes and arcs across the instance 20−5. Specifically, when Δ is set at 1, the network is most dense with 3765.4 time-space nodes and 29195.6 arcs. However, increasing Δ to 10 results in a significant decrease, reducing the number of time-space nodes and arcs to 379.0 and 2920.7, respectively. This trend suggests that while finer granularity enhances the network's detail and potentially its modeling accuracy, it also substantially increases computational complexity. Based on the constructed time-expanded network, the average performances of CPLEX and BPC are reported in Table 8.

From the perspective of algorithm efficiency, we can observe that the BPC algorithm consistently finds a larger number of optimal solutions across different granularities, while CPLEX shows a general increase in the number of optimal solutions as Δ increases. BPC experiences a notable reduction in CPU time as Δ increases (from 1597.38 s at $\Delta = 1$ min to 484.73 s at $\Delta = 10$ min), suggesting that a coarser time granularity substantially reduces computational complexity, while the CPU time of CPLEX remains at 3600 s, indicating notably worse solving efficiency than BPC even with finer discretizations.

From the standpoint of decision quality, increasing the value of Δ simplifies the problem, leading to shorter solution times but at the cost of solution quality (increased drone delivery times). When the value of Δ increases to 10 min, the model may even become infeasible. Therefore, selecting an appropriate Δ value is crucial for balancing model accuracy and computational feasibility in network-based modeling. For real-world applications requiring precise drone scheduling decisions, a finer granularity is preferable despite the higher computational cost. However, when computational resources or time are constrained, a coarser granularity might be a viable compromise.

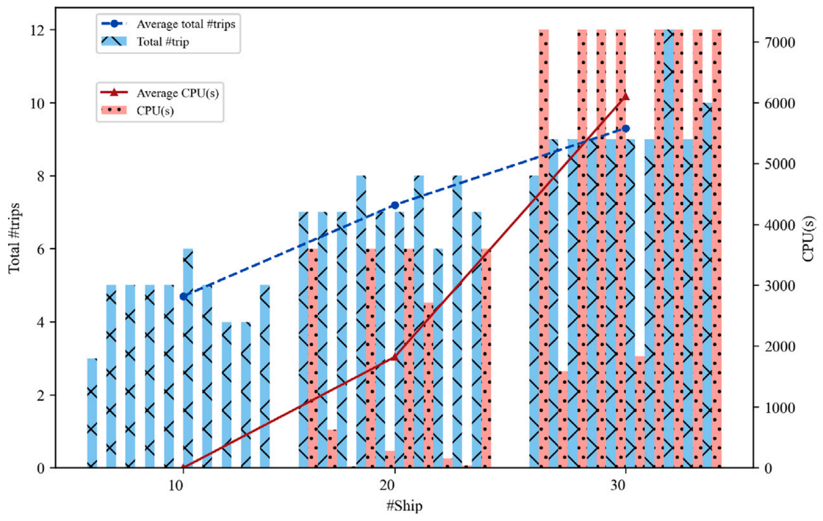

Fig. 2. Different number of ships delivered by the same number of drones.

5.3.2. Number of ships

We fix the number of drones at 5 but vary the number of ships to 10, 20, and 30 to compare the total number of trips and computation time, where the feasible integer solutions are obtained by the BPC algorithm.

In Fig. 2, the x-axis represents the total number of ships, i.e., $|\mathcal{V}|$. The left y-axis depicts the total number of trips taken by the drones over the planning horizon, while the right y-axis shows the CPU time in seconds. The histogram displays these two metrics for each instance, while the line graph presents the average values of these metrics across instances of the same scale.

It can be seen that scalability issues become apparent as the number of ships increases, as evinced by significant increases in the computation time. For instances with 30 ships, the computation time is notably high, often reaching the maximum time limit of 7200 s. For instances with 10 ships, the computation time is drastically smaller, mostly approximately 0 s. In terms of the total number of drone trips, the results reveal a more steady increasing trend with the number of ships. For instances with 20 or 30 ships, the number of trips varies from 6 to 12, indicating multiple trips made by each drone. This feature is less pronounced in instances of 10 ships with the number of trips ranging from 3 to 6.

5.3.3. Summary of the sensitivity analysis

The sensitivity analysis conducted on the discretization granularity parameter demonstrates that a finer granularity (smaller Δ) enhances the solution quality at the expense of an increased computation time. Decision-makers are thus presented with the choice of balancing these factors according to their strategic priorities. Furthermore, the experiments reveal that a coarser granularity (larger Δ) can lead to model infeasibility, mainly attributable to the inability to complete services within the required time window after rough time discretization. The experiment on the number of ships highlights the capability of drones to undertake multiple trips in scenarios of high delivery demand versus limited resources. These insights are invaluable for decision-makers determining the optimal number of drones to deploy to ensure a balance between service efficiency and drone workload for effective shore-to-ship delivery operations.

6. Conclusion

Motivated by the lack of attention paid to drone-assisted shore-to-ship delivery, we put forward the DSP-SSD and provide a comprehensive solution to this problem. Firstly, we develop the MIP model, which is tailored for scenarios involving dynamic targets and multiple trips and can be solved directly by off-the-shelf solvers in small-scale instances. This model leverages a time-expanded network created through time discretization, enabling drones to execute sequential delivery tasks effectively.

To apply the model to larger, more practical scenarios, we introduce a tailored BPC algorithm. This advanced algorithm transforms the original model into a route-based framework, which is then processed using a B&B strategy. The algorithm incorporates CG for dynamic variable integration and employs cutting planes to strengthen the linear relaxations. Our computational tests confirm that the BPC algorithm outperforms standard solvers, particularly in managing the frequent demands of the DSP-SSD, and our acceleration techniques further enhance its efficiency.

We offer decision-makers detailed analyses of how changes to the time discretization parameter can optimize the trade-off between solution speed and decision accuracy, and how varying the number of ships affects delivery times, workloads, and trip frequencies, thereby aiding optimal drone fleet management.

This research could be further extended in several aspects. Methodologically, more advanced methods could be adopted to find a better UB at the early stages of the BPC algorithm (Desaulniers et al., 2006). Anti-degradation techniques such as the dual stabilization method could be considered (Neame, 2000) to avoid the discrete time-expanded network inducing many solutions with the same objective value. Practically, to promote the application of shore-to-ship delivery, more realistic factors could be considered in the model, such as synergy between ships and tender boats, the impact of uncertain weather conditions on drone services (Cheng et al., 2024), and the power consumption of drones.

CRediT authorship contribution statement

Ying Yang: Writing – review & editing, Writing – original draft, Software, Methodology, Conceptualization. **Xiaodeng Hao:** Visualization, Validation, Software, Data curation, Methodology. **Shuaian Wang:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Methodology.

Acknowledgments

This work was supported by the National Natural Science Foundation of China [Grant Nos. 72071173, 72371221]. The authors would like to express gratitude to the PolyU Maritime Data and Sustainable Development Centre (PMDC) and Research Centre for Digital Transformation of Tourism for their invaluable support in providing data and equipment.

Appendix A. Calculating the traveling time

The flying time of each drone from ship v at time t to another ship v', denoted by $\tau_{v,v'}(t)$, can be calculated by

$$\left[S^{\mathcal{D}}\tau_{v,v'}(t)\right]^2 = \left[\alpha_v(t) - \alpha_{v'}\left(t+\tau_{v,v'}(t)\right)\right]^2 + \left[\beta_v(t) - \beta_{v'}\left(t+\tau_{v,v'}(t)\right)\right]^2. \tag{16}$$

Similarly, the flying time of a drone from ship v at time t to the drone station, denoted by $\tau_{v,0}(t)$, can be calculated by

$$\left[S^{\mathcal{D}}\tau_{v,0}(t)\right]^2 = \left[\alpha_v(t) - 0\right]^2 + \left[\beta_v(t) - 0\right]^2. \tag{17}$$

The flying time from the drone station at time t to ship v, denoted by $\tau_{0,v}(t)$, can be computed by

$$\left[S^{\mathcal{D}}\tau_{0,v}(t)\right]^2 = \left[0 - \alpha_v\left(t+\tau_{0,v}(t)\right)\right]^2 + \left[0 - \beta_v\left(t+\tau_{0,v}(t)\right)\right]^2. \tag{18}$$

The valid ranges of t in the above equations Eq. (16)–Eq. (18) are provided as follows.

Notably, we abbreviate $\tau_{v,v'}(t)$, $\tau_{v,0}(t)$ and $\tau_{0,v}(t)$ as τ in the following proof.

(i) For equation Eq. (17), the range of t is $[t_{ve}, t_{vl}]$.

(ii) For equation Eq. (18), a drone leaves the station at time t and arrives at the ship v at time $t+\tau$, which should be in the vessel's time window, i.e., $[t_{ve}, t_{vl}]$. Therefore, we denote the range of t as $[t_{\min}, t_{\max}]$, where $t_{\min}$ is calculated by

$$\begin{aligned}
&t_{\min} + \tau = t_{ve},\\
&\tau \ge 0,\\
&\left[S^{\mathcal{D}}\tau\right]^2 = \left[0 - \alpha_v\left(t_{ve}\right)\right]^2 + \left[0 - \beta_v\left(t_{ve}\right)\right]^2,
\end{aligned}$$

and $t_{\max}$ is calculated by

$$\begin{aligned}
&t_{\max} + \tau = t_{vl},\\
&\tau \ge 0,\\
&\left[S^{\mathcal{D}}\tau\right]^2 = \left[0 - \alpha_v\left(t_{vl}\right)\right]^2 + \left[0 - \beta_v\left(t_{vl}\right)\right]^2.
\end{aligned}$$

(iii) For equation Eq. (16), firstly, we have $t \in [t_{ve}, t_{vl}]$. Besides, the drone leaving vessel v at time t should arrive at vessel v' within its time window $[t_{v'e}, t_{v'l}]$. Therefore, we have

$$\hat{\mathcal{T}} = \{t \in [t_{ve}, t_{vl}] \mid t_{v'e} \le \tau + t \le t_{v'l}, \left[S^{\mathcal{D}}\tau\right]^2 = \left[\alpha_v(t) - \alpha_{v'}(t+\tau)\right]^2 + \left[\beta_v(t) - \beta_{v'}(t+\tau)\right]^2, \tau \ge 0\}.$$

In this case, the valid t is in $\hat{\mathcal{T}}$.

Appendix B. Existence of delivery time

Proposition 3. *Given $S^{\mathcal{D}} > S^{\mathcal{V}}$, there always exists a unique and positive root of $\tau_{\cdot,\cdot}(t)$ for equations Eq. (16), Eq. (17) and Eq. (18).*

The proof is given as follows. Notably, we abbreviate $\tau_{v,v'}(t)$, $\tau_{v,0}(t)$ and $\tau_{0,v}(t)$ as τ in the following proof.

(i) For equation Eq. (17), the right-hand side is a positive value. Therefore, we have

$$\tau = \pm\sqrt{\frac{\left[\alpha_v(t) - 0\right]^2 + \left[\beta_v(t) - 0\right]^2}{S^{\mathcal{D}}}}.$$

Then, there always exists one unique positive root of $\tau = \sqrt{\frac{[\alpha_v(t)]^2 + [\beta_v(t)]^2}{S^{\mathcal{D}}}}$.

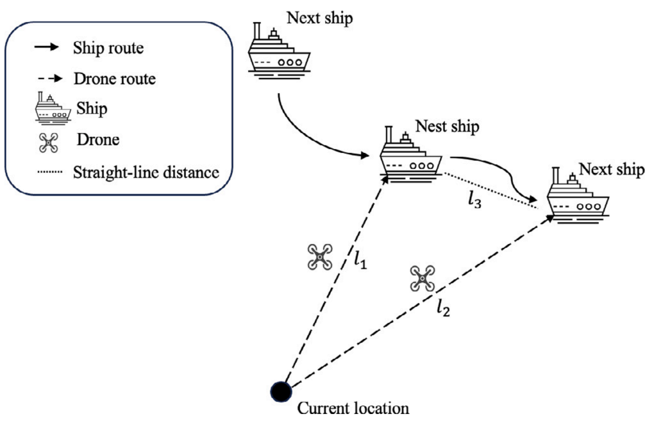

Fig. B.1. The situation that the drone can arrive at the next ship at two different time points.

(ii) For equations Eq. (16) and Eq. (18), we first prove that there is at least one positive root of τ. We assume that the drone leaves the current station (or ship) at time t_1 and arrives at the next ship v at t_2 and $t_2 = t_1 + \tau$. The original location of the drone is (α_0, β_0), and its distance to the next ship v at time t_1 is denoted as l_0. Then, we define $g(\tau) = [S^{\mathcal{D}}\tau]^2 - [\alpha_v(t_2) - \alpha_0]^2 - [\beta_v(t_2) - \beta_0]^2$, which is a function of τ.

When $\tau = 0$, we have

$$g(0) = [S^{\mathcal{D}} \cdot 0]^2 - l_0^2 < 0.$$

When $\tau > 0$, we have

$$\begin{aligned} g(\tau) &= [S^{\mathcal{D}}\tau]^2 - [\alpha_v(t_2) - \alpha_0]^2 - [\beta_v(t_2) - \beta_0]^2 \\ &= [S^{\mathcal{D}}\tau]^2 - [\alpha_v(t_1+\tau) - \alpha_0]^2 - [\beta_v(t_1+\tau) - \beta_0]^2 \\ &\geq [S^{\mathcal{D}}\tau]^2 - [l_0 + S^{\mathcal{V}} \cdot \tau]^2 && \text{(By triangle inequality)} \\ &= ([S^{\mathcal{D}}]^2 - [S^{\mathcal{V}}]^2)\tau^2 - (l_0)^2 - 2l_0 S^{\mathcal{V}}\tau. \end{aligned} \tag{19}$$

The quadratic formula Eq. (19) is equal to 0 when

$$\begin{aligned} \tau &= \frac{2l_0 S^{\mathcal{V}} \pm \sqrt{4(l_0)^2[S^{\mathcal{V}}]^2 + 4(l_0)^2([S^{\mathcal{D}}]^2 - [S^{\mathcal{V}}]^2)}}{2([S^{\mathcal{D}}]^2 - [S^{\mathcal{V}}]^2)} \\ &= \frac{2l_0 S^{\mathcal{V}} \pm 2l_0 S^{\mathcal{D}}}{2([S^{\mathcal{D}}]^2 - [S^{\mathcal{V}}]^2)} \\ &= \frac{l_0(S^{\mathcal{V}} \pm S^{\mathcal{D}})}{[S^{\mathcal{D}}]^2 - [S^{\mathcal{V}}]^2} && \text{(since } S^{\mathcal{D}} > S^{\mathcal{V}} \text{ and } \tau > 0) \\ &= \frac{l_0}{S^{\mathcal{D}} - S^{\mathcal{V}}}. \end{aligned}$$

Therefore, we have $\tau > \frac{l_0}{S^{\mathcal{D}} - S^{\mathcal{V}}}$ such that the quadratic formula Eq. (19) is greater than 0. Therefore, when τ grows large enough, we have $g(\tau) > 0$. Together with $g(0) < 0$ and $g(\tau)$ is continuous, we have at least one positive root of τ.

Next, we prove that the root is unique by contradiction. As shown in Fig. B.1, assume that there are two positive roots of τ, i.e., the drone can leave its current location and arrive at the next ship after τ_1 or τ_2, where $\tau_1 < \tau_2$. The distance between the current location of the drone and the location where the drone arrives at the next ship is l_1 and l_2, respectively. The distance between the locations of the ship after τ_1 and τ_2 is l_3.

Then, we have

$$\begin{aligned} l_2 - l_1 &= S^{\mathcal{D}}\tau_2 - S^{\mathcal{D}}\tau_1 \\ &= S^{\mathcal{D}}(\tau_2 - \tau_1) \\ &\leq l_3 && \text{(By triangle inequality)} \\ &\leq S^{\mathcal{V}}(\tau_2 - \tau_1) && \text{(The ship's sailing route cannot be shorter than the straight line).} \end{aligned}$$

The result implies $S^{\mathcal{D}}(\tau_2 - \tau_1) \leq S^{\mathcal{V}}(\tau_2 - \tau_1)$, i.e., $S^{\mathcal{D}} \leq S^{\mathcal{V}}$, which contradicts our assumption $S^{\mathcal{V}} < S^{\mathcal{D}}$. Therefore, there can not be more than one positive root of τ.

Conclusively, given $S^{\mathcal{D}} > S^{\mathcal{V}}$, there always exists one unique and positive root of $\tau_{\cdot,\cdot}(t)$ for equations Eq. (16), Eq. (17) and Eq. (18).

Appendix C. Proof of Proposition 1

Constraints Eq. (1b) and Eq. (1n) restrict that the number of available drones in the station at each time point should be larger than 0 and smaller than $|\mathcal{D}|$, that is, the number of working drones at each time point should be smaller than $|\mathcal{D}|$ and larger than 0, which is exactly the meaning of constraints Eq. (3c). This completes the proof.

Appendix D. Dynamic programming algorithm for 2-path inequalities

A tailored backward DP formulation is introduced to check whether a subset of vessels $\mathcal{V}^{tp}$ can be served by one drone. We perform the DP for each station-time node $i_0 \in \mathcal{N}^0$, and the procedure stops once a feasible solution that visits all vessels can be found. Firstly, we define the state $(\widetilde{\mathcal{V}}, j, \widetilde{q})$ as the family of all feasible paths, which starts in vessel u_j at time t_j, visits an unordered set of vessels $\widetilde{\mathcal{V}} \subseteq \mathcal{V}^{tp}$, and arrives at the station-time node i_0. $\widetilde{q}$ represents the minimum battery consumption of a path starting at node j, passing through every vessel of $\widetilde{\mathcal{V}}$ exactly once, ending at node i_0. Let $\mathcal{S}_k$ be the set of states that visits k vessels.

Step 1: Initialization. Set $\mathcal{S}_0 = \{(\emptyset, i_0, 0)\}$ and $\mathcal{S}_k = \emptyset$, for $k = 2, \cdots, |\mathcal{V}^{tp}|$. Let $\hat{k} = 1$.

Step 2: Expansion of every state in set $\mathcal{S}_{\hat{k}-1}$. For each $(\widetilde{\mathcal{V}}, j, \widetilde{q}) \in \mathcal{S}_{\hat{k}-1}$, repeat the Step 3.

Step 3: Generate reachable states of $\mathcal{S}_{\hat{k}}$ from $(\widetilde{\mathcal{V}}, j, \widetilde{q})$. For each $a_{i,j} \in \mathcal{A}$ such that $u_i \in \mathcal{V}^{tp} \backslash \widetilde{\mathcal{V}}$, we generate $(\widetilde{\mathcal{V}}', i, \widetilde{q}')$, where $\widetilde{\mathcal{V}}' = \widetilde{\mathcal{V}} \cup \{u_i\}$ and $\widetilde{q}' = \widetilde{q} + \theta \left(\sum_{v \in \widetilde{\mathcal{V}}'} w_v + W^D\right)\left(t_i - t_j\right)$. The state is rejected if any of the following conditions folds: (i) feasibility check: $\widetilde{q}' \geq Q^D$; (ii) dominance check: $\mathcal{S}_{\hat{k}}$ contains a state $(\widetilde{\mathcal{V}}', i, \widetilde{q}'')$ such that $\widetilde{q}' > \widetilde{q}''$. If the state is not rejected, let $\mathcal{S}_{\hat{k}} = \mathcal{S}_{\hat{k}} \cup \{(\widetilde{\mathcal{V}}', i, \widetilde{q}')\}$ and remove any state in $\mathcal{S}_{\hat{k}}$ that is dominated by $(\widetilde{\mathcal{V}}', i, \widetilde{q}')$.

Step 4: Update. Set $\hat{k} \leftarrow \hat{k} + 1$; if $\hat{k} \leq |\widetilde{\mathcal{V}}|$ go to Step 2, otherwise the DP is finished.

If we have any state $(\widetilde{\mathcal{V}}, j, \widetilde{q}) \in \mathcal{S}_{|\widetilde{\mathcal{V}}|}$ such that $\widetilde{q} + \theta\left(\sum_{v \in \widetilde{\mathcal{V}}} w_v + W^D\right)\left(t_i - t_j - \tau_0\right) \leq Q^D$, this subset of vessels can be visited by one drone; otherwise, perform the DP for another unexplored station-time node until all station-time nodes have been tested.

Appendix E. Proof of Proposition 2

It suffices to show that if (i) to (vii) hold, then $\forall j \in \mathcal{N}^{\mathcal{V}} : u_j \in \bar{\mathcal{V}}_{p_2}$ such that L_{p_2} can be extended to a feasible $L_{p_2'}$, we have that L_{p_1} can also be extended to a feasible $L_{p_1'}$ by visiting j and (i) to (v) still hold for $L_{p_1'}$ and $L_{p_2'}$.

Step 1: We show that the resource vector of $L_{p_1'}$ is less than $L_{p_2'}$. Firstly, according to (20), condition (i), (ii), and (iv) hold trivially. Note that when we extend L_{p_1} to $L_{p_1'}$, $i^{p_1} = i^{p_2}$ cannot be a station-time node. Then, the battery energy consumption of p_1' and p_2' are $q^{p_1'} = q^{p_1} + \theta(t_{i^{p_1}} - t_j)(y^{p_1'} + W^D)$ and $q^{p_2'} = q^{p_2} + \theta(t_{i^{p_2}} - t_j)(y^{p_2'} + W^D)$, respectively. According to condition (v), we have $t_{i^{p_1}} = t_{i^{p_2}}$. Therefore, we can prove that

$$\begin{aligned}
q^{p_1'} &= q^{p_1} + \theta(t_{i^{p_1}} - t_j)(y^{p_1'} + W^D) & \\
&\leq q^{p_2} + \theta(t_{i^{p_1}} - t_j)(y^{p_1'} + W^D) & \text{(By initial condition (ii) } q^{p_1} \leq q^{p_2}\text{)} \\
&= q^{p_2} + \theta(t_{i^{p_2}} - t_j)(y^{p_1'} + W^D) & \text{(Since } t_{i^{p_1}} = t_{i^{p_2}}\text{)} \\
&\leq q^{p_2} + \theta(t_{i^{p_2}} - t_j)(y^{p_2'} + W^D) & \text{(Since } y^{p_1'} \leq y^{p_2'}\text{)} \\
&= q^{p_2'},
\end{aligned}$$

which completes condition (iii).

In addition, recalling the definition of the partial reduced cost, we have

$$\begin{aligned}
b_{p_1'} &= b_{p_1} + b_{j,i^{p_1}} & \text{(By definition)} \\
&= b_{p_1} + b_{j,i^{p_2}} & \text{(By initial condition (iv) } i^{p_1} = i^{p_2}\text{)} \\
&\leq b_{p_2} + b_{j,i^{p_2}} & \text{(By initial condition (v) } b_{p_1} \leq b_{p_2}\text{)} \\
&= b_{p_2'}. & \text{(By definition)}
\end{aligned}$$

Therefore, condition (vi) holds.

In view of the condition (vii), since $(\mathcal{V}^{sr} \cup \mathcal{V}_{p_1}) \subseteq (\mathcal{V}^{sr} \cup \mathcal{V}_{p_2}), \forall sr \in \mathcal{SR}$, we have $(\mathcal{V}^{sr} \cup \mathcal{V}_{p_1'}) \subseteq (\mathcal{V}^{sr} \cup \mathcal{V}_{p_2'}), \forall sr \in \mathcal{SR}$.

Step 2: We show that the full reduced cost of $L_{p_1'}$ is less than $L_{p_2'}$. Since $\mathcal{V}_{p_1'} \subseteq \mathcal{V}_{p_2'}$, we have

$$\mathcal{K}_{p_1'} = \{k \in \mathcal{K} : |\mathcal{V}_{p_1'} \cap \mathcal{V}^k| \geq 2\} \subseteq \{k \in \mathcal{K} : |\mathcal{V}_{p_2'} \cap \mathcal{V}^k| \geq 2\} = \mathcal{K}_{p_2'}.$$

Thus, $\forall k \in \mathcal{K}$, we have $0 \leq \mathbb{1}_{\mathcal{K}_{p_1'}}(k) \leq \mathbb{1}_{\mathcal{K}_{p_2'}}(k)$. Furthermore, since $\mu_k \leq 0$, we have

$$-\sum_{k \in \mathcal{K}} \mathbb{1}_{\mathcal{K}_{p_1'}}(k)\mu_k \leq -\sum_{k \in \mathcal{K}} \mathbb{1}_{\mathcal{K}_{p_2'}}(k)\mu_k.$$

Considering initial condition (vi) $b_{p_1'} \leq b_{p_2'}$, we can prove that

$$\bar{c}_{p_1'} = b_{p_1'} - \sum_{k \in \mathcal{K}} \mathbb{1}_{\mathcal{K}_{p_1'}}(k)\mu_k \leq b_{p_2'} - \sum_{k \in \mathcal{K}} \mathbb{1}_{\mathcal{K}_{p_2'}}(k)\mu_k = \bar{c}_{p_2'}.$$

Step 3: We then prove the feasibility of $L_{p_1'}$ by showing that constraints Eq. (13) hold for $L_{p_1'}$. Since conditions (ii) and (iii) hold for $L_{p_1'}$ and $L_{p_2'}$, i.e., $y^{p_1'} \leq y^{p_2'}$ and $q^{p_1'} \leq q^{p_2'}$. Therefore, considering the feasibility of $L_{p_2'}$, we have

$$y^{p_1'} \leq y^{p_2'} \leq C^{\mathrm{D}},$$

$$q^{p'_1} \leq q^{p'_2} \leq Q^{\mathrm{D}}.$$

The proof is completed.

Appendix F. Algorithms

F.1. Labeling algorithm pseudocode

Algorithm F.2: Feasibility pump algorithm that generates a feasible integer solution.

Input: The number of variables to be flipped κ, maximum iteration number $\hat{\eta}$.

Output: Feasible integer solution χ^*.

```
Compute $\chi^* \in \arg\min\{\sum_{p\in\mathcal{P}'} c_p\chi_p$ : s.t.Eq. (4b)-Eq. (4d), Eq. (10)$\}$
if $\chi^*$ is integer then
    return $\chi^*$
else
    Let $\tilde{\chi} := [\chi^*]$, $\eta = 0$ // $[x]$ means round $x$ to the nearest integer.
    while $\eta < \hat{\eta}$ do
        Let $\eta = \eta + 1$
        Compute $\chi^* \in \arg\min\{\sum_{p\in\mathcal{P}':\tilde{\chi}_p=0} \chi_p + \sum_{p\in\mathcal{P}':\tilde{\chi}_p=1}(1-\chi_p)$ : s.t. Eq. (4b)-Eq. (4d), Eq. (10)$\}$
        if $\chi^*$ is integer then
            return $\chi^*$
        end
        if $\exists p \in \mathcal{P}' : \tilde{\chi}_p \neq \left[\chi_p^*\right]$ then
            $\tilde{\chi} := [\chi^*]$
        else
            Flip the $\tilde{\kappa}$ entries $\tilde{\chi}_p$ $(p \in \mathcal{P}')$ with highest $\left|\chi_p^* - \tilde{\chi}_p\right|$, where $\tilde{\kappa}$ is a random integer between $[\kappa/2, 3\kappa/2]$
            /* Flip is to prevent stalling issues by taking the reverse rounding of some variables
               when all rounded variables are the same as the last iteration.                       */
        end
    end
end
```

F.2. Feasibility pump pseudocode

Algorithm F.1: Backward labeling algorithm to generate drone path.

Input: The time-expanded network and the reduced cost of each arc. Parameters Num_L, Num_{L^*} and $flag$.
Output: The generated columns and revised $flag$.

```
1  $List_{L^*} = \emptyset$ and $List_L = \emptyset$
2  foreach $i_0 \in \mathcal{N}^0$ do
3      Initialization: a single label $L_p = (\mathcal{V}_p, y^p, q^p, \mathcal{T}_p, i^p, b_p, \bar{c}_p)$ is generated, where $\mathcal{V}_p = \emptyset$, $q^p = 0$, $y^p = 0$, $\mathcal{T}_p = \emptyset$, $i^p = i_0$, $b_p = 0$, $\bar{c}_p = 0$, $List_L$.add($L_p$)
4      if $List_L$.size> $Num_L$& flag then
5          Sort $List_L$ by the increasing order of the reduced costs of the labels
6          Delete the last $List_L$.size$-Num_L$ labels with the largest reduced costs
7      end
8      while $List_L \neq \emptyset$ do
9          Get label $L_p = (\mathcal{V}_p, y^p, q^p, \mathcal{T}_p, i^p, b_p, \bar{c}_p)$ from $List_L$
10         foreach $a_{j,i^p} \in \mathcal{A}$ do // There are arcs whose target node is $i^p$
11             Extension: extend $L_p$ towards node $j$ such that $a_{j,i^p} \in \mathcal{A}$ and $u_j \notin \mathcal{V}_p$; generate label $L_{p'} = (\mathcal{V}_{p'}, y^{p'}, q^{p'}, \mathcal{T}_{p'}, i^{p'}, b_{p'}, \bar{c}_{p'})$ if it is feasible according to Eq. (13), where
12                 $$\begin{cases} \mathcal{V}_{p'} = \begin{cases} \mathcal{V}_p & \text{if } j \in \mathcal{N}^0 \\ \mathcal{V}_p \cup \{u_j\} & \text{if } j \in \mathcal{N}^{\mathcal{V}} \end{cases} \\ y^{p'} = y^p + w_{u_j} \\ q^{p'} = \begin{cases} q^p + \theta(y^p + W^{\mathcal{D}})(t_{i^p} - t_j - \tau_0) & \text{if } i^p \in \mathcal{N}^0 \\ q^p + \theta(y^p + W^{\mathcal{D}})(t_{i^p} - t_j) & \text{if } i^p \in \mathcal{N}^{\mathcal{V}} \end{cases} \\ \mathcal{T}_{p'} = \mathcal{T}_p \cup \left\{ t \mid t = k\Delta, t_j \le t < t_{i^p}, k \in \mathbb{N}_0 \right\} \\ i^{p'} = j \\ b_{p'} = b_p + b_{j,i^p} \\ \text{compute } \bar{c}_{p'} \text{ by Eq. (11)} \end{cases} \quad (20)$$
               if $i^{p'} \in \mathcal{N}^0$ then // A complete feasible path is generated
13                 $List_{L^*}$.add($L_{p'}$)
14             else
15                 $List_L$.add($L_{p'}$)
16                 Dominance rule: apply Proposition 2 and delete the dominated labels from $List_L$ or $List_{L^*}$
17             end
18         end
19         Delete $L_p$ from $List_L$
20     end
21 end
22 Sort $List_L^*$ by the increasing order of the reduced costs of the labels
23 if the first label's reduced cost of $List_L^*$ is negative then
24     Select the first $Num_{L^*}$ paths with the most negative reduced costs from $List_{L^*}$ and set $flag$ = true
25 else
26     Set $flag$ = false
27 end
28 Return the columns corresponding to paths and flag.
```

F.3. Branch-and-price-and-cut pseudocode

Algorithm F.3: Branch-and-price-and-cut algorithm for drone scheduling problem.

```
Input: The constructed expanded network and a convergence gap $\epsilon$.
Output: Optimal integer solution $\chi^*$.
1  Create a B&B tree and insert the root node into the tree
2  Set LB= $-\infty$, UB= $+\infty$, BestSol← $\emptyset$
3  while $UB-LB \geq \epsilon$ do
4      Select a node from the tree by best-bound-first strategy
5      if the current node is the root node then
6          Add the SRIs
7          Solve the RMP by CG and generate the RCIs and TPIs iteratively
8      else
9          Solve the RMP by CG with existing SRIs, RCIs and TPIs
10     end
11     if the RMP is infeasible then
12         Mark the current node as explored
13     else
14         Get an optimal solution $\bar{\chi}$ with objective value $\Psi$
15         Set LB = min{the local objective of all leaf nodes}
16         if $\bar{\chi}$ is integral then
17             if $\Psi$ <UB then Set BestSol← $\bar{\chi}$ and UB = $\Psi$
18             Mark the current node as explored
19         else
20             if the current node is the root node then
21                 Perform the FP algorithm or solve the IP form of RMP to get the solution $\hat{\chi}$ with objective value $\hat{\Psi}$
22                 if $\hat{\chi}$ is integeral then Set BestSol← $\hat{\chi}$, UB= $\hat{\Psi}$
23             end
24             if $\Psi$ <UB then
25                 Branch on $\bar{\chi}$ on the arc
26                 Set the estimated bound for each child node as $\Psi$
27                 Insert two resulting child nodes into the tree
28             end
29             Mark the current node as explored
30         end
31     end
32 end
33 Return the BestSol and UB
```

Appendix G. Branching on the number of drones

For each station-time node $i \in \mathcal{N}^0$, we let d_i be the number of drones flying out of node i, which can be calculated by $\sum_{p\in\mathcal{P}'} \sum_{a_{i,j}\in\mathcal{A}:j\in\mathcal{N}^{\mathcal{V}}} \mathbb{1}_{\mathcal{A}_p}(a_{i,j})\chi_p^*$. If d_i is factional, we impose a constraint $\sum_{p\in\mathcal{P}'} \sum_{a_{i,j}\in\mathcal{A}:j\in\mathcal{N}^{\mathcal{V}}} \mathbb{1}_{\mathcal{A}_p}(a_{i,j})\chi_p \geq \lceil d_i \rceil$ to the RMP of one child node and force $\sum_{p\in\mathcal{P}'} \sum_{a_{i,j}\in\mathcal{A}:j\in\mathcal{N}^{\mathcal{V}}} \mathbb{1}_{\mathcal{A}_p}(a_{i,j})\chi_p \leq \lfloor d_i \rfloor$ to the RMP of the other child node. We choose to branch on the earlier time point when there are multiple fractional d_i. Notably, the dual prices associated with each drone-number branching constraint should also be included when calculating the reduced cost of each path.

References

Asiimwe, R., Anvar, A., 2012. Automation of the maritime UAV command, control, navigation operations, simulated in real-time using Kinect sensor: A feasibility study. Int. J. Comput. Syst. Eng 6 (12), 2606–2610.

Augerat, P., 1995. Polyhedral Approach of the Vehicle Routing Problem (Ph.D. thesis). Université Joseph Fourier, Available at HAL.

Azi, N., Gendreau, M., Potvin, J.-Y., 2007. An exact algorithm for a single-vehicle routing problem with time windows and multiple routes. European J. Oper. Res. 178 (3), 755–766.

Azi, N., Gendreau, M., Potvin, J.-Y., 2010. An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. European J. Oper. Res. 202 (3), 756–763.

Betti Sorbelli, F., Corò, F., Das, S.K., Palazzetti, L., Pinotti, C.M., 2022. Greedy algorithms for scheduling package delivery with multiple drones. In: Proceedings of the 23rd International Conference on Distributed Computing and Networking. pp. 31–39.

Charles Alcock, 2024. Shore-to-Ship drone deliveries could cut costs and carbon. https://www.ainonline.com/aviation-news/business-aviation/2022-02-16/shore-ship-drone-deliveries-could-cut-costs-and-carbon. (Accessed April 8, 2024).

Cheng, C., Adulyasak, Y., Rousseau, L.-M., 2020. Drone routing with energy function: Formulation and exact algorithm. Transp. Res. B 139, 364–387.
Cheng, C., Adulyasak, Y., Rousseau, L.-M., 2024. Robust drone delivery with weather information. Manuf. Serv. Operat. Manag (in press).
Costa, L., Contardo, C., Desaulniers, G., 2019. Exact branch-price-and-cut algorithms for vehicle routing. Transp. Sci. 53 (4), 946–985.
Dall'Olio, G., Kolisch, R., 2023. Formation and routing of worker teams for airport ground handling operations: A branch-and-price-and-check approach. Transp. Sci. 57 (5), 1231–1251.
Davis, A.R., Broad, A., Gullett, W., Reveley, J., Steele, C., Schofield, C., 2016. Anchors away? The impacts of anchor scour by ocean-going vessels and potential response options. Mar. Policy 73, 1–7.
Desaulniers, G., Desrosiers, J., Solomon, M.M., 2006. Column Generation, vol. 5. Springer Science & Business Media, New York.
Desrochers, M., Desrosiers, J., Solomon, M., 1992. A new optimization algorithm for the vehicle routing problem with time windows. Oper. Res. 40 (2), 342–354.
DJI, 2024. DJI FlyCart 30. https://www.dji.com/hk/flycart-30/specs. (Accessed April 8, 2024).
Dorling, K., Heinrichs, J., Messier, G.G., Magierowski, S., 2017. Vehicle routing problems for drone delivery. IEEE Trans. Syst, Man, Cybern: Syst 47 (1), 70–85.
Dror, M., 1994. Note on the complexity of the shortest path models for column generation in VRPTW. Oper. Res. 42 (5), 977–978.
Dumas, Y., Desrosiers, J., Gelinas, E., Solomon, M.M., 1995. An optimal algorithm for the traveling salesman problem with time windows. Oper. Res. 43 (2), 367–371.
Feillet, D., Dejax, P., Gendreau, M., Gueguen, C., 2004. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. Netw: An Int. J 44 (3), 216–229.
Fischetti, M., Glover, F., Lodi, A., 2005. The feasibility pump. Math. Program. 104, 91–104.
Fowler, T.G., Sørgård, E., 2000. Modeling ship transportation risk. Risk Anal. 20 (2), 225–244.
Fukasawa, R., Longo, H., Lysgaard, J., Aragão, M.P.d., Reis, M., Uchoa, E., Werneck, R.F., 2006. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Math. Program. 106, 491–511.
Gambella, C., Naoum-Sawaya, J., Ghaddar, B., 2018. The vehicle routing problem with floating targets: Formulation and solution approaches. INFORMS J. Comput. 30 (3), 554–569.
Garey, M.R., Johnson, D.S., 1983. Computers and intractability: A guide to the theory of NP-completeness. The J. Symbolic Logic 48 (2), 498–500.
Garg, V., Niranjan, S., Prybutok, V., Pohlen, T., Gligor, D., 2023. Drones in last-mile delivery: A systematic review on efficiency, accessibility, and sustainability. Trans. Res. D: Transp Environ 123, 103831.
Ha, Q.M., Deville, Y., Pham, Q.D., Hà, M.H., 2018. On the min-cost traveling salesman problem with drone. Transp. Res. C 86, 597–621.
Hernandez, F., Feillet, D., Giroudeau, R., Naud, O., 2016. Branch-and-price algorithms for the solution of the multi-trip vehicle routing problem with time windows. European J. Oper. Res. 249 (2), 551–559.
Huang, N., Li, J., Zhu, W., Qin, H., 2021. The multi-trip vehicle routing problem with time windows and unloading queue at depot. Transp. Res. E: Logist. Transp. Rev 152, 102370.
Huang, N., Qin, H., Du, Y., Wang, L., 2024a. An exact algorithm for the multi-trip vehicle routing problem with time windows and multi-skilled manpower. European J. Oper. Res. 319 (1), 31–49.
Huang, N., Qin, H., Xu, G., Wan, F., 2024b. An enhanced exact algorithm for the multi-trip vehicle routing problem with time windows and capacitated unloading station. Comput. Oper. Res. 168, 106688.
Jeong, H.Y., Song, B.D., Lee, S., 2019. Truck-drone hybrid delivery routing: Payload-energy dependency and no-fly zones. Int. J. Prod. Econ. 214, 220–233.
Jepsen, M., Petersen, B.r., Spoorendonk, S., Pisinger, D., 2008. Subset-row inequalities applied to the vehicle-routing problem with time windows. Oper. Res. 56 (2), 497–511.
Kohl, N., Desrosiers, J., Madsen, O.B., Solomon, M.M., Soumis, F., 1999. 2-path cuts for the vehicle routing problem with time windows. Transp. Sci. 33 (1), 101–116.
Kumar, R.P., Devi, V.A., 2023. Air drone as an alternative to supply boats in sea ports. Mar. Eng. Rev (India) 2, 19–24.
Land, A., Doig, A., 1960. An automatic method of solving discrete programming problems. Econometrica 28 (3), 497–520.
Liu, S., Qin, S., Zhang, R., 2018. A branch-and-price algorithm for the multi-trip multi-repairman problem with time windows. Transp. Res. E: Logist. Transp. Rev 116, 25–41.
Liu, B., Wang, Y., Li, Z.-C., Zheng, J., 2023. An exact method for vessel emission monitoring with a ship-deployed heterogeneous fleet of drones. Transp. Res. C 153, 104198.
Macedo, R., Alves, C., de Carvalho, J.V., Clautiaux, F., Hanafi, S., 2011. Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model. European J. Oper. Res. 214 (3), 536–545.
Macrina, G., Pugliese, L.D., Guerriero, F., Laporte, G., 2020. Drone-aided routing: A literature review. Transp. Res. C 120, 102762.
Meng, S., Guo, X., Li, D., Liu, G., 2023. The multi-visit drone routing problem for pickup and delivery services. Transp. Res. E: Logist. Transp. Rev 169, 102990.
Mingozzi, A., Roberti, R., Toth, P., 2013. An exact algorithm for the multitrip vehicle routing problem. INFORMS J. Comput. 25 (2), 193–207.
Muhammad, B., Gregersen, A., 2022. Maritime drone services ecosystem-potentials and challenges. In: 2022 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom). IEEE, pp. 6–13.
Murray, C.C., Chu, A.G., 2015. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. Transp. Res. C 54, 86–109.
Murray, C.C., Raj, R., 2020. The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones. Transp. Res. C 110, 368–398.
Neame, P.J., 2000. Nonsmooth dual methods in integer programming (Ph.D. thesis). University of Melbourne, Department of Mathematics and Statistics.
Nguyen, H.P., Hoang, A.T., Nizetic, S., Nguyen, X.P., Le, A.T., Luong, C.N., Chu, V.D., Pham, V.V., 2021. The electric propulsion system as a green solution for management strategy of CO2 emission in ocean shipping: A comprehensive review. Int. Trans. Electr. Energy Syst. 31 (11), e12580.
Ozbaygin, G., Karasan, O.E., Savelsbergh, M., Yaman, H., 2017. A branch-and-price algorithm for the vehicle routing problem with roaming delivery locations. Transp. Res. B 100, 115–137.
Paradiso, R., Roberti, R., Laganá, D., Dullaert, W., 2020. An exact solution framework for multitrip vehicle-routing problems with time windows. Oper. Res. 68 (1), 180–198.
Pereira, F.L., 2021. Optimal control problems in drone operations for disaster search and rescue. Procedia Comput. Sci. 186, 78–86.
Poikonen, S., Wang, X., Golden, B., 2017. The vehicle routing problem with drones: Extended models and connections. Networks 70 (1), 34–43.
Puri, V., Nayyar, A., Raja, L., 2017. Agriculture drones: A modern breakthrough in precision agriculture. J. Statist. Manag. Syst 20 (4), 507–518.
Rejeb, A., Abdollahi, A., Rejeb, K., Treiblmaier, H., 2022. Drones in agriculture: A review and bibliometric analysis. Comput. Electron. Agric. 198, 107017.
Righini, G., Salani, M., 2006. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. Discrete Optim. 3 (3), 255–273.
Roberti, R., Ruthmair, M., 2021. Exact methods for the traveling salesman problem with drone. Transp. Sci. 55 (2), 315–335.
Service, S.D., 2023. The power of ship-to-shore drone delivery in maritime supply chains. https://skyportsdroneservices.com/2023/01/the-power-of-ship-to-shore-drone-delivery-in-maritime-supply-chains/. (Accessed July 15, 2024).
Shen, L., Wang, Y., Liu, K., Yang, Z., Shi, X., Yang, X., Jing, K., 2020. Synergistic path planning of multi-UAVs for air pollution detection of ships in ports. Transp. Res. E: Logist. Transp. Rev 144, 102128.
Squires, A.M., 1986. The Tender Ship: Governmental Management of Technological Change. Springer Science & Business Media, New York.
Stieber, A., Fügenschuh, A., Epp, M., Knapp, M., Rothe, H., 2015. The multiple traveling salesmen problem with moving targets. Optim. Lett. 9 (8), 1569–1583.
Sulligoi, G., Bosich, D., Pelaschiar, R., Lipardi, G., Tosato, F., 2015. Shore-to-ship power. Proc. IEEE 103 (12), 2381–2400.

Thompson, M., Davies, M., 2021. Maritime uses of drones. In: Drone Law and Policy. Routledge, pp. 79–113.
Tilk, C., Bianchessi, N., Drexl, M., Irnich, S., Meisel, F., 2018. Branch-and-price-and-cut for the active-passive vehicle-routing problem. Transp. Sci. 52 (2), 300–319.
Toth, P., Vigo, D., 2014. Vehicle Routing: Problems, Methods, and Applications. Society for industrial and applied mathematics, Philadelphia.
Vanderbeck, F., Wolsey, L.A., 1996. An exact algorithm for IP column generation. Oper. Res. Lett. 19 (4), 151–159.
Wang, Z., Sheu, J.-B., 2019. Vehicle routing problem with drones. Transp. Res. B 122, 350–364.
Xia, J., Wang, K., Wang, S., 2019. Drone scheduling to monitor vessels in emission control areas. Transp. Res. B 119, 174–196.
Yang, Y., 2023a. Deluxing: Deep Lagrangian underestimate fixing for column-generation-based exact methods. Available at SSRN 4585724.
Yang, Y., 2023b. An exact price-cut-and-enumerate method for the capacitated multitrip vehicle routing problem with time windows. Transp. Sci. 57 (1), 230–251.
Zhang, W., Jacquillat, A., Wang, K., Wang, S., 2023. Routing optimization with vehicle–customer coordination. Manage. Sci. 69 (11), 6876–6897.
Zhen, L., He, X., Wang, S., Wu, J., Liu, K., 2023. Vehicle routing for customized on-demand bus services. IISE Trans 55 (12), 1277–1294.