Journal of Artificial Intelligence Research 70 (2021) 757-788 Submitted 10/2020; published 02/2021

Efficient Large-Scale Multi-Drone Delivery using Transit Networks

Shushman Choudhury SHUSHMAN@STANFORD.EDU
Department of Computer Science
Stanford University, CA, 94035 USA

Kiril Solovey KIRILSOL@STANFORD.EDU
Mykel J. Kochenderfer MYKEL@STANFORD.EDU
Marco Pavone PAVONE@STANFORD.EDU
Department of Aeronautics and Astronautics
Stanford University, CA, 94035 USA

Abstract

We consider the problem of routing a large fleet of drones to deliver packages simultaneously across broad urban areas. Besides flying directly, drones can use public transit vehicles such as buses and trams as temporary modes of transportation to conserve energy. Adding this capability to our formulation augments effective drone travel range and the space of possible deliveries but also increases problem input size due to the large transit networks. We present a comprehensive algorithmic framework that strives to minimize the maximum time to complete any delivery and addresses the multifaceted computational challenges of our problem through a two-layer approach. First, the upper layer assigns drones to package delivery sequences with an approximately optimal polynomial time allocation algorithm. Then, the lower layer executes the allocation by periodically routing the fleet over the transit network, using efficient, bounded suboptimal multi-agent pathfinding techniques tailored to our setting. We demonstrate the efficiency of our approach on simulations with up to 200 drones, 5000 packages, and transit networks with up to 8000 stops in San Francisco and the Washington DC Metropolitan Area. Our framework computes solutions for most settings within a few seconds on commodity hardware and enables drones to extend their effective range by a factor of nearly four using transit.

1. Introduction

Rapidly growing e-commerce demands have strained dense urban communities by increasing delivery vehicle traffic and impacting travel times for public and private transit (Holguín-Veras et al., 2018). Operators need to redesign their current method of package distribution in cities (Kafle, Zou, & Lin, 2017). Drones are promising options for logistic networks due to their agility, aerial reach, and recent technological advances, and the flexibility and ease of establishing drone networks. However, they have limited travel range and carrying capacity (Sudbury & Hutchinson, 2016). In contrast, ground-based transit networks have less flexibility but greater coverage and throughput. By combining the strengths of both, we can achieve both commercial benefits and social impact, e.g., reducing ground congestion and delivering essential products.

Analysts have studied how drone-augmented commodity transport in urban areas could improve the traditional delivery ecosystem (Lohn, 2017; Gulden, 2017; D'Andrea, 2014;

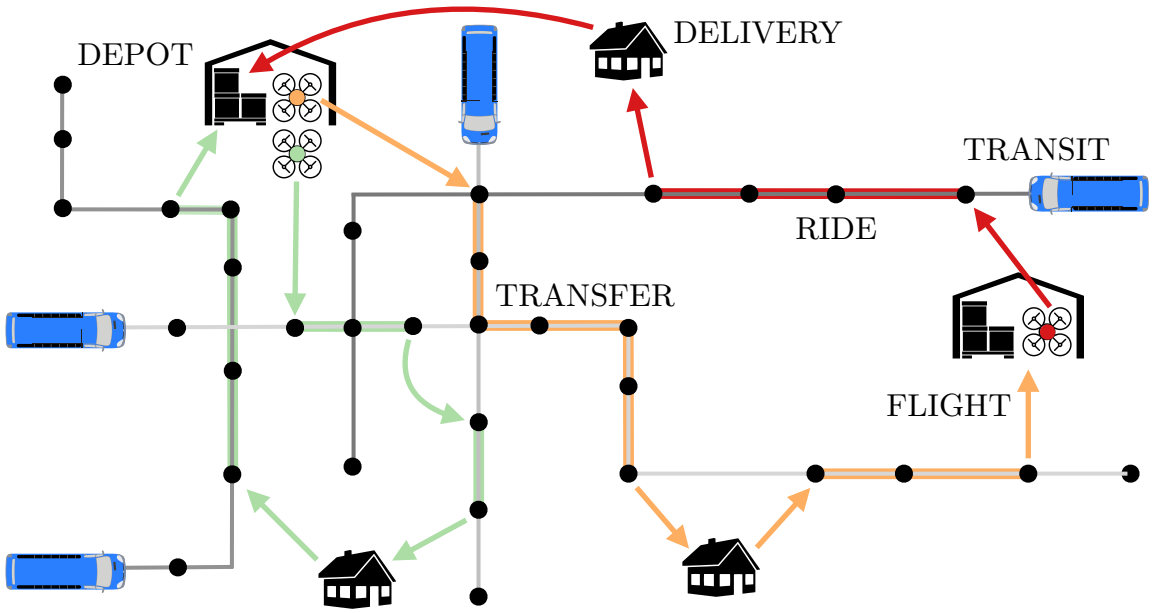


Figure 1: Our multi-drone delivery framework allocates delivery sequences to drones and plans routes for them to piggyback on public transit vehicles, while delivering packages from depots to the requested locations. Our framework is scalable and strives to minimize the *delivery time makespan*, i.e., the maximum delivery time for any individual package.

Wang et al., 2017), but researchers have made limited initial attempts to harness this synergy. Drones can be paired with a traditional delivery truck and autonomously launched to carry packages to individual customers before returning to the truck, which has been delivering to other customers in the meantime (Murray & Chu, 2015). However, this approach does not reduce the total delivery time unless drones can move at least twice as fast as trucks (Ferrandez et al., 2016), and neither exploits public transit nor addresses the multi-agent issues of task allocation and inter-agent conflicts.

In this paper, we consider a setting where an operator coordinates a large fleet of drones to deliver multiple packages simultaneously. The drones can use one or more vehicles in a public-transit network as modes of transportation to save their limited onboard energy and increase their effective travel range. We need to decide *which deliveries each drone should make and in what order, which modes of transit to use, and for how long* (Figure 1). Our approach must plan over large time-dependent transit networks, while accounting for flight range constraints. It must avoid inter-drone conflicts, such as multiple drones attempting to board the same vehicle at the same time, or exceeding the maximum carrying capacity of a vehicle. We seek multi-agent plans that are not only feasible but also strive to minimize a cumulative objective over all drones, namely, the makespan, i.e., the maximum individual delivery time for any drone. Finally, we must also allocate drones to package deliveries and distribution centers.

We design a comprehensive algorithmic framework for large-scale multi-drone delivery in synergy with a ground transit network. We decompose our challenging problem and solve it stage-wise with a two-layer approach. First, the upper layer assigns drones to package-

delivery sequences. Then, the lower layer executes the allocation by periodically routing the fleet over the transit network. For the upper layer, we develop an *approximately optimal polynomial time task allocation* method. For the lower layer, we extend techniques for *efficient bounded-suboptimal multi-agent pathfinding* to account for time-dependent transit networks and individual agent energy constraints. We present results supporting the scalability of our approach on simulations with up to 200 drones, 5000 packages, and transit networks of up to 8000 stops in San Francisco and the Washington DC Metropolitan area. Our framework computes solutions for most settings in a few seconds on commodity hardware and enables drones to travel up to 360% of their flight range using transit.

We wish to emphasize that due to its decoupled nature, our approach does not provide global optimality guarantees. But we do show that each individual layer has guarantees relating to the subproblems they solve, under some assumptions; we point out those assumptions throughout the text.

Our paper is structured as follows. We first highlight some relevant existing research. In Section 2 we present an overview of our two-layer approach. We then elaborate on the upper task allocation layer in Section 3 and the lower multi-agent pathfinding layer in Section 4. In Section 5 we describe the surrogate cost estimate used to couple the two layers. We then present extensive experimental results on several aspects of our framework in Section 6 and conclude with Section 7.

We presented a preliminary version of this paper in the *International Conference on Robotics and Automation* (Choudhury et al., 2020). The current work has additional discussions, proofs and experimental results, which we now summarize. Section 3 describes the task-allocation algorithm in greater detail; we include a comprehensive proof on the bounded suboptimality of solution cost, for which we had previously given a short sketch. We also analyze the algorithm's computational complexity in detail and obtain a concrete bound (we had previously only shown the polynomial runtime). Section 4 presents new speedup techniques for Focal-MCSP, which is used to find paths for individual agents within the MAPF layer, and a new diagram and pseudocode for the overall layer. Section 5 is brand new and discusses the surrogate cost estimate. For the experimental results of Section 6, we revise and update our numbers from the previous version and provide two new sets of results on replanning strategies and the relative performance of two surrogate estimates.

1.1 Overview of Related Work

We now summarize three areas of previous work related to our overall problem: drone-assisted delivery, autonomous mobility-on-demand, and multiagent pathfinding. They use a range of ideas from operations research, transportation systems, and artificial intelligence.

1.1.1 Drone-Assisted Delivery Planning

Recent work considers socially-aware motion planning for drones to operate in urban environments around humans (Yoon et al., 2019). They compute trajectories that minimize human discomfort by accounting for safety perception in close proximity to the drone. An intent-aware probabilistic approach predicts collisions in an uncertain environment to plan drone trajectories around stochastic humans. Control strategies for rendezvous between multiple agents (e.g., a drone and a ground vehicle) have also been developed (Rucco et al.,

2018; Haberfeld et al., 2020). Finally, Choudhury et al. (2019) proposed a method to plan and execute routes for a single autonomous agent using multiple modes of transit.

1.1.2 Autonomous Mobility-on-Demand Services

Our problem is similar to routing a fleet of autonomous vehicles for on-demand mobility services (Solovey et al., 2019; Iglesias et al., 2019; Wallar et al., 2018). These approaches compute routes for vehicles (customer-carrying or empty) to fulfill travel demand and minimize total operational cost. Some recent works combine such services with public transit, allowing passengers to use several modes of transportation in the same trip (Salazar et al., 2018; Zgraggen et al., 2019). However, they abstract away inter-agent constraints and dynamics and are unsuitable for autonomous pathfinding. Our task allocation setting is an instance of the vehicle routing problem (Caceres-Cruz et al., 2014; Otto et al., 2018; Toth & Vigo, 2014). The vehicle routing problem and its variants are typically solved by mixed integer linear programming formulations that are less scalable to large real-world settings of interest, or by heuristics that can have suboptimal performance.

1.1.3 Multi-Agent Path Finding

In the second layer of our approach, we must solve a multi-agent pathfinding (MAPF) problem (Erdmann & Lozano-Perez, 1987). Since all drones are on the same team, we have a centralized or cooperative pathfinding setting (Silver, 2005). The MAPF problem is NP-hard to solve optimally (Yu & LaValle, 2013), but many existing solvers do work well in practice (Felner et al., 2017).

From an algorithmic perspective, our approach for decomposing the overall drone delivery problem into a two-layered solution of task allocation and MAPF, shares similarities with approaches for pickup and delivery in a warehouse setting that use a similar decomposition (Ma et al., 2017; Hönig et al., 2018; Liu et al., 2019). However, our method differs in several crucial aspects. First, we develop a specialized task allocation algorithm that exploits the problem structure, whereas the previous works use a black-box solver for the travelling salesman problem without any polynomial-time guarantees. Second, these previous approaches have not considered pathfinding over large time-dependent transit networks, let alone with constraints on the drone flight range and constraints on inter-drone conflicts. These new constraints and features make our MAPF problem even more difficult to solve. Also note that our MAPF solver uses models, algorithms and practical techniques from the transportation planning community (Delling et al., 2009; Bast et al., 2016).

2. Methodology

We describe our formulation and approach at a high-level and illustrate the various interacting components.

2.1 Problem Formulation

We operate a centralized homogeneous fleet of m drones in a city-scale domain. There are ℓ product *depots* with known geographic locations, denoted by $V_D := \{d_1, \ldots, d_\ell\} \subset \mathbb{R}^2$. The depots are both product dispatch centers and drone-charging stations. At the start of

a suitable time interval (e.g., a day), we receive a batch of delivery request locations for k different *packages*, denoted $V_P := \{p_1, \ldots, p_k\} \subset \mathbb{R}^2$, where $k \gg m$. We assume that any package can be dispatched from any depot; our approach exploits this property to optimize the *makespan*, i.e., the maximum execution time for any drone delivery and subsequent return to a depot. In Section 3, we mention how we can accommodate dispatch constraints.

The drones carry packages from depots to delivery locations. They can extend their effective travel range by using public transit vehicles, which are unaffected by drone actions. We must route drones to deliver all packages while minimizing makespan. A drone path consists of its current location and the sequence of depot and package locations to visit with a combination of flying and riding on transit. The limited drone energy imposes a maximum flight distance constraint. A feasible solution must also satisfy inter-drone constraints such as collision avoidance and transit vehicle capacity limits. We assume that a drone carries one package at a time, which is reasonable given state-of-the-art drone payloads (Sudbury & Hutchinson, 2016); drones are recharged in negligible time upon visiting a depot (e.g., a battery replacement); depots have unlimited drone capacity; the transit network is deterministic with respect to locations and vehicle travel times. Some of these assumptions will be justified subsequently in the proper context.

2.2 Approach Overview

In principle, we could solve our entire problem as a mixed integer linear program (MILP). However, for real-world problems (hundreds of drones, thousands of packages, and large transit networks), even state-of-the-art MILP approaches will not scale well. Even a simpler problem that ignores interaction constraints is an instance of the notoriously challenging multi-depot vehicle routing problem (Otto et al., 2018). Thus, we decouple our problem into two distinct subproblems that we solve stage-wise in layers.

The upper layer (Section 3) performs *task allocation* to decide which packages are delivered by which drone and in what order. It takes as input the known depot and package delivery locations and an estimate of the drone travel time between every pair of locations. It then solves an optimization problem that minimizes delivery makespan and computes three sets of decisions: assigning to each package (i) the dispatch depot and (ii) the delivery drone, and to each drone (iii) the order of package deliveries. We develop an efficient polynomial-time task-allocation algorithm with an approximately optimal makespan.

The lower layer (Section 4) performs *route planning* for the drone fleet to execute the allocated delivery tasks. It generates detailed routes of drone locations in space and time and the transit vehicles used, while accounting for the time-varying transit network. It also ensures that (i) multiple drones do not board transit simultaneously, (ii) no transit vehicle exceeds its drone-carrying capacity, and (iii) drone energy constraints are respected. To efficiently handle individual and inter-drone constraints, we frame the routing problem as an extension of multi-agent path finding (MAPF) to transit networks. We adapt a scalable, bounded suboptimal variant of a highly effective MAPF solver called Conflict-Based Search (Sharon et al., 2012) to solve the problem of planning a set of routes, one for each drone (to deliver its current package). Finally, we can execute the full sequence of delivery tasks in a receding horizon fashion by replanning routes for the next delivery task of a drone after it has completed its current one.

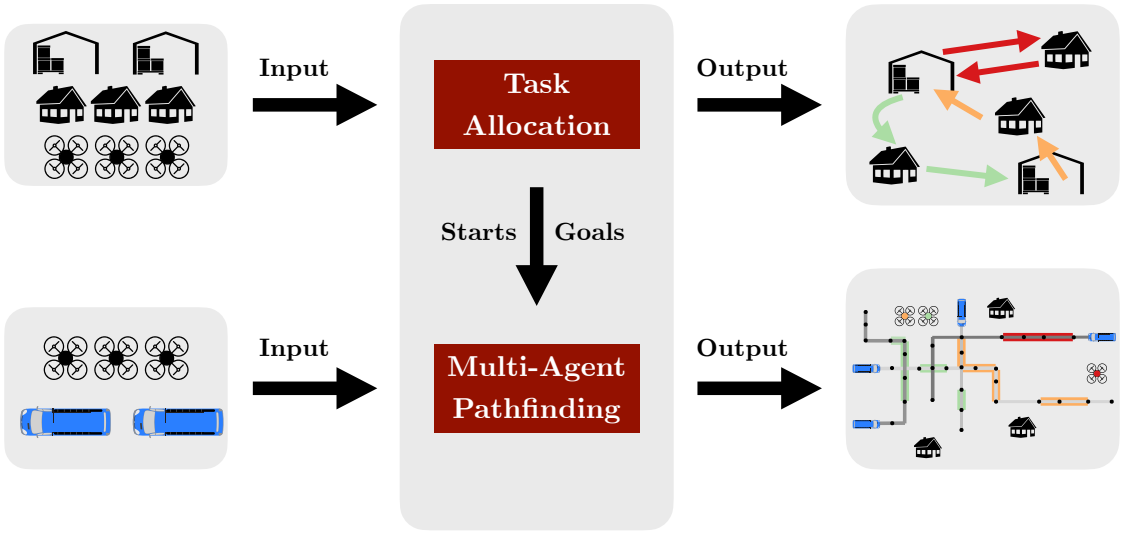


Figure 2: We design a two-layer framework for solving the problem of multi-drone delivery using transit networks. The upper task allocation layer takes as input the depot and package delivery locations and an estimate of drone travel time between locations, and assigns each package to a dispatch depot and delivery drone, and each drone to a sequence of deliveries. The lower multi-agent pathfinding layer takes as input the transit network and the drone start and goal locations from the allocation layer, and generates detailed routes in space and time that respects all the system constraints and the transit timetable.

Our overall framework is two-staged; its solution quality depends on the surrogate cost estimate that we use for the drone travel time in the task allocation layer. The allocation computed by the first stage determines the start and goal locations for the MAPF layer and thus constrains the set of overall solutions. The better the surrogate estimate we use for allocation, the more *coupled* the layers are, i.e., the better is the solution of the first stage for the second one. We use a surrogate that accounts for the transit network, at the expense of modest preprocessing. We precompute the pairwise shortest travel times between locations spread around the city, over a representative snapshot of the transit network, and look up these estimates at runtime (Section 5 has more details).

3. Delivery Sequence Allocation

We leverage our problem structure to design an algorithm called MergeSplitTours, which guarantees an approximately optimal solution for the task allocation layer in polynomial time; in this context, the tasks are delivery sequences for each drone. This layer (i) distributes the set of packages V_P among m drones, (ii) assigns each package destination $p \in V_P$ to a depot $d \in V_D$, and (iii) assigns drones to a sequence of depot pickups and package de-

liveries. Our objective is to **minimize the maximum travel time among all drones** while computing the above three sets of decisions.

We begin this section by formally defining the allocation problem and discussing its relation to the m travelling salesman problem. We then describe our MergeSplitTours algorithm and present proofs for approximate optimality and polynomial-time complexity.

For the allocation layer, we seek a set of m paths of minimum makespan (the maximum travel time for a path in the set) and we call this the *m minimal visiting paths problem* (m-MVP). We only need *paths* that start and end at depots, not tours. A key element of m-MVP is the *allocation graph* $G_A = (V_A, E_A)$, with vertex set $V_A = V_D \cup V_P$ (Figure 3a). Each directed edge $(u, v) \in E_A$ is weighted according to an estimated travel time c_{uv} from the location of u to that of v in the city. We now define the m-MVP problem in full:

Definition 1. (The m-MVP problem) Given allocation graph G_A, the m minimal visiting paths problem (m-MVP) requires finding m paths $P_1^*, P_2^*, \ldots, P_m^*$ (denoted compactly as $P_{1:m}^*$) on G_A, such that (1) each path P_i^* starts at some depot $d \in V_D$ and terminates at the same or different $d' \in V_D$, (2) exactly one path visits each package $p \in V_P$, and (3) the maximum travel time of any of the paths is minimized.

Let OPT be the optimal makespan, i.e., $\text{OPT} := \max_{i \in [m]} \text{LENGTH}(P_i^*)$, where $\text{LENGTH}(\cdot)$ denotes the total travel time along a given path or tour. We make four observations. First, if a path contains the sub-path $(d, p), (p, d')$, for some $d, d' \in V_D, p \in V_P$, then p should be dispatched from depot d and the drone delivering p will return to d' after delivery. Second, a package p being found in P_i^* indicates that drone $i \in [m]$ should deliver it. Third, P_i^* fully characterizes the order of packages delivered by drone i. Fourth, the solution determines the initial depot locations of the drones (represented by the first vertex along P_i^*), to further optimize the solution cost.

3.1 Assumptions

The m-MVP problem is a special case of the *asymmetric* m-travelling salesman problem (Bektas, 2006), for a *directed* underlying graph, which is NP-hard even for $m = 1$ on general graphs (Asadpour et al., 2017); it is not known whether the specific instance of our problem is NP-hard as well. Moreover, the current best polynomial-time approximation yields a fairly large approximation factor $O(\log n / \log \log n)$, where n is the number of vertices. The problem is made even harder by the fact that the triangle inequality does not apply to our metric of travel time.

To mitigate these difficulties, we make two simplifying assumptions concerning the structure of the allocation graph G_A. The first assumption excludes edges that require more than half of the allowed flight range between package and depot locations. Later on we will exploit this to develop a polynomial-time approximation algorithm.

Assumption 1. If $(d, p) \in E_A$, where $d \in V_D, p \in V_P$ (and similarly for edges from p to d), then it is possible to travel from d from p within *half* the allowed travel range, while accounting for energy savings due to transit travel. Additionally, under those conditions, if there exist $(d, p) \in E_A$, where $d \in V_D, p \in V_P$, then there must also exist $d' \in V_D$ such that $(p, d') \in E_A$, to allow the drone to successfully return from a delivery mission. If

$(d, d') \in E_A$, where $d, d' \in V_D$, then it is possible to travel from d from d' within the *full* allowed flight range, while accounting for energy savings due to transit travel.

As we flagged in Section 2.1, we can also model any other dispatch constraints by excluding the necessary edges from the corresponding depot. The next assumption ensures that a drone can move between any pair of depots, possibly by going through a sequence of depots along the way, while potentially recharging when passing through a depot.

Assumption 2. The subgraph $G_A(V_D)$ induced by the vertices V_D is strongly connected. That is, $G_A(V_D)$ contains a directed path from any $d \in V_D$ to any other vertex $d' \in V_D$.

Below, we develop a new polynomial-time algorithm for m-MVP that returns a near optimal solution, under the above assumptions.

3.2 MergeSplitTours Algorithm

Before describing our MergeSplitTours algorithm for solving m-MVP (Algorithm 1), we present a key subroutine that solves the minimal connecting tours (MCT) problem. For simplicity, we formally define MCT as an integer program in Table 1. But we can efficiently implement MergeSplitTours in polynomial time *without invoking an integer program solver*; we will discuss this point further in the next subsection.

The solution to MCT provides MergeSplitTours with an initial set of tours $\mathbb{T}$ that connects packages to depots within tours to minimize the total edge weight in Equation (1). The constraint in Equation (4) ensures that each package is connected to precisely one incoming and one outgoing edge from and to depots, respectively. The final constraint in Equation (5) enforces inflow and outflow equality for every depot. Edges connecting packages can be used at most once, whereas edges connecting depots can be used multiple times. Solving MCT yields the assignment $\{x_{uv}\}_{(u,v)\in E_A}$, i.e., which edges of G_A are used and how many times. This assignment implicitly represents the desired collection of the tours $T_1, \ldots, T_t$.

We can now describe MergeSplitTours in detail. The algorithm consists of the following three main steps, which are illustrated in Figure 3:

Step 1 (Line 1): We generate a collection of t tours ($1 \leqslant t \leqslant k$) of minimum total distance, denoted as $T_1, \ldots, T_t$, such that exactly one tour covers every package $p \in V_P$. This step is achieved by solving the minimal connecting tours (MCT) problem in Table 1. The solution to MCT is given by an assignment $\{x_{uv}\}_{(u,v)\in\mathbb{E}}$, which indicates which edges of G are used and for how many times (we denote this edge assignment compactly as $\boldsymbol{x}$ hereafter). Lemma 1 below discusses why this assignment implicitly encodes $T_1, \ldots, T_t$, the desired collection of tours.

Lemma 1. *(Disjoint Tours of MCT) Let $\boldsymbol{x}$ be the output of $MCT(G_A, V_P)$. There exists a collection of vertex-disjoint tours $T_1, \ldots, T_{m'}$, such that for every $(u, v) \in E_A$ for which $x_{uv} > 0$, there exists T_i in which (u, v) appears exactly x_{uv} times.*

Proof. By the definition of MCT, for every $p \in V_p$, there is precisely one incoming edge (d, p) and one outgoing edge (p, d') such that $x_{dp} = x_{pd'} = 1$. Also, by Equation (5) the in-degree and out-degree of every $d \in V_D$ are equal. We can thus form a Eulerian tour that traverses every edge (u, v) exactly x_{uv} times. □

Table 1: An integer programming formulation of minimal connecting tours (MCT).

Given the allocation graph $G_A = (V_A, E_A)$, with $V_A = V_D \cup V_P$,

$$\text{minimize} \sum_{(u,v)\in E_A} x_{uv} \cdot c_{uv} \tag{1}$$

subject to

$$x_{uv} \in \{0,1\}, \quad \forall (u,v) \in E_A, u \in V_P \vee v \in V_P, \tag{2}$$

$$x_{uv} \in \mathbb{N}_{\geqslant 0}, \quad \forall (d,d') \in E_A, d, d' \in V_D, \tag{3}$$

$$\sum_{d\in N_+(p)} x_{dp} = \sum_{d\in N_-(p)} x_{pd} = 1, \quad \forall p \in V_P, \tag{4}$$

$$\sum_{v\in N_+(d)} x_{vd} - \sum_{v\in N_-(d)} x_{dv} = 0, \quad \forall d \in V_D. \tag{5}$$

where $N_+(v)$ and $N_-(v)$ denote the incoming and outgoing neighbors of $v \in V_A$.

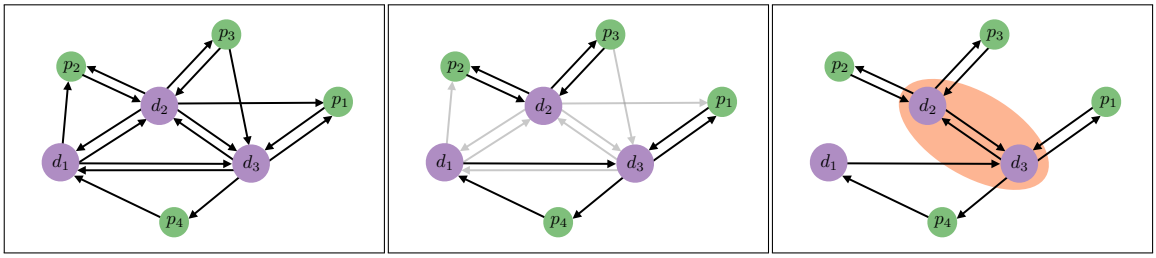
Step 2 (Lines 2-10): The $T_1, \ldots, T_t$ tours are iteratively merged until a single tour T is generated. We first identify $t \geqslant 1$ connected depot sets $\mathcal{D} = \{D_1, \ldots, D_t\}$, induced by the MCT solution (line 2). Every depot set D_i consists of all depots that belong to one specific tour T_i, as encoded in the edge assignment $\boldsymbol{x}$. We then merge the tours and thus the connected depot sets; we iterate over all combinations of $D, D' \in \mathcal{D}, d \in D, d' \in D'$ (lines 5-8) and choose $(d,d'), (d',d)$ to minimize $c_{dd'} + c_{d'd}$, and then update the edge assignment $\boldsymbol{x}$ and depot set collection $\mathcal{D}$ appropriately (lines 9, 10). For a given $\mathcal{D}$ and $d \in V_D$, the notation $\mathcal{D}(d)$ represents the depot component $D \in \mathcal{D}$ that contains d. The resulting merged tour is of the form $(d_1, p_1, d'_1, d_2, p_2, d'_2, \ldots, d_l, p_{l-1}, d'_l)$, where for a given $1 \leqslant j \leqslant l$, p_i denotes the package delivery location, d_i is the depot from which the package was taken, and d'_i is the depot to which the drone returns after the delivery. It is possible that $d'_i = d_{i+1}$, i.e., the return depot is also the location from which the next package will be taken. In the latter case, the cost $c_{d'_i d_{i+1}} = 0$.

Step 3 (Lines 11-18): We split the merged tour T into m paths $P_{1:m}$ where the length of each path is proportional to the length of T divided by m. Also, every path P_i starts and ends at a depot (not necessarily the same one). We derive this step from the algorithm of Frederickson et al. (1976) for m-TSP in undirected graphs.

3.3 Completeness and Optimality

We now analyze the theoretical guarantees of MergeSplitTours. The following theorem shows that our algorithm returns a feasible solution to m-MVP that is also close to optimal.

Theorem 1. *(Approximate Optimality of* MergeSplitTours*) Suppose that Assumptions 1 and 2 hold. Let $P_{1:m}$ be the output of* MergeSplitTours*. Then, every package $p \in V_P$ is contained in exactly one path P_i, and every P_i starts and ends at a depot.*



(a) (b) (c)

Figure 3: The MergeSplitTours algorithm for delivery sequence allocation has three key steps: (a) The allocation graph is defined for the depots $d_{1:3}$ and package destinations $p_{1:4}$. (b) The MCT step yields a solution that connects each package delivery with the depot from which the drone is dispatched and the depot to which it returns. (c) A tour merging steps merges the depots d_2 and d_3 into a single cluster, corresponding to line 10 of the algorithm.

Moreover, $\max_{i\in[m]} \textsc{length}(P_i) \leqslant \textsc{opt} + \alpha + \beta$ *holds,*

$$\textit{where } \alpha := \max_{d,d'\in V_D} c_{dd'} + c_{d'd} \ , \ \beta := \max_{d,d'\in V_D, p\in V_P} c_{dp} + c_{pd'}.$$

We assume hereafter that G_A is strongly connected, which follows from the two previous assumptions. Additionally, for simplicity of notation, we can assume that $G_A(V_D)$ is a directed clique. We do so by connecting every pair of depots with an edge, where an edge can potentially encode a sequence of visited depot nodes with charging in between.

We now present the main proof. The key idea is that the total cost of the tours from MCT cannot exceed the total length of $P^*_{1:m}$ (the set of paths in OPT, the optimal makespan solution). We then adapt the MCT solution to m paths with an additional overhead of $\alpha + \beta$ per path. When $m \ll |V_P|$ (typically the case), α and β are small compared to OPT, making the bound tight. For instance, in our randomly-generated scenarios in Section 6.1, for $m = 5$ and $k = 200$, the approximation ratio $\max_{i\in[m]} \textsc{length}(P_i)/\textsc{opt} = 1.09$, and for $m = 10, k = 500$, the factor is 1.06.

Proof of Theorem 1. After every iteration of the "while" loop, the updated assignment $\boldsymbol{x}$ still represents a collection of tours. This loop is repeated at most $m - 1$ times; t, which represents the initial number of connected depots (line 2), is at most m, since every tour induced by MCT must contain at least one depot.

Let OPT be the optimal solution to m-MVP, i.e., let m paths $\{P^*_1, \ldots, P^*_m\}$ represent the solution to m-MVP, and for every $i \in [m]$, $\textsc{length}(P^*_i) \leqslant \textsc{opt}$. It follows that

$$\sum_{i=1}^{m} \textsc{length}(P_i) \leqslant m \cdot \max_{i\in[m]} \textsc{length}(P^*_i) = m \cdot \textsc{opt},$$

where $P_{1:m}$ is the set of m paths computed by MergeSplitTours. Next, by definition of α, we have that $\textsc{length}(T) \leqslant m \cdot \textsc{opt} + m\alpha$. Lastly, by definition of β, we have that

$$\textsc{length}(T_i) \leqslant \textsc{length}(T)/m + \beta \leqslant \textsc{opt} + \alpha + \beta. \qquad \square$$

Algorithm 1: MergeSplitTours

Input: Allocation graph $G_A = (V_A, E_A)$, with $V_A = V_D \uplus V_P$; $m \geqslant 1$ drones.
Output: Paths $\{P_1, \ldots, P_m\}$, such that every package is visited exactly once.

1 $\boldsymbol{x} := \{x_{uv}\}_{(u,v)\in E_A} \leftarrow \text{MCT}(G_A, V_P)$
2 $\mathcal{D} := \{D_1 \ldots, D_t\} \leftarrow \text{ConnectedDepots}(G_A, \boldsymbol{x})$
3 **while** $|\mathcal{D}| > 1$
4 $\quad c_{\min} \leftarrow \infty, d_{\min} \leftarrow \emptyset, d'_{\min} \leftarrow \emptyset$
5 $\quad$ **for** $D, D' \in \mathcal{D}, D \neq D', d \in D, d' \in D'$
6 $\quad\quad$ **if** $c_{dd'} + c_{d'd} < c_{\min}$
7 $\quad\quad\quad c_{\min} \leftarrow c_{dd'} + c_{d'd}$, $d_{\min} \leftarrow d, d'_{\min} \leftarrow d'$
8 $\quad x_{d_{\min} d'_{\min}} \leftarrow 1, x_{d'_{\min} d_{\min}} \leftarrow 1$
9 $\quad \mathcal{D} \leftarrow (\mathcal{D} \setminus \{\mathcal{D}(d_{\min}), \mathcal{D}(d'_{\min})\}) \cup \{\mathcal{D}(d_{\min}) \cup \mathcal{D}(d'_{\min})\}$
10 $T := (d_1, p_1, d'_1, d_2, p_2, d'_2, \ldots, d_l, p_{l-1}, d'_l) \leftarrow \text{GetTour}(G_A, \boldsymbol{x})$
11 $i \leftarrow 1$; $j \leftarrow 1$
12 **for** $i = 1$ to m
13 $\quad P_i \leftarrow \{(d_j, p_j), (p_j, d'_j)\}, L_i \leftarrow c_{d_j p_j} + c_{p_j d'_j}, j \leftarrow j + 1$
14 $\quad$ **while** $L_i \leqslant \text{length}(T)/m$ **and** $j \leqslant l$
15 $\quad\quad L_i \leftarrow L_i + c_{d'_{j-1} d_j} + c_{d_j p_j} + c_{p_j d'_j}$
16 $\quad\quad P_i \leftarrow P_i \cup \{(d'_{j-1}, d_j), (d_j, p_j), (p_j, d'_j)\}$
17 $\quad\quad j \leftarrow j + 1$
18 **return** $\{P_1, \ldots, P_m\}$

3.4 Computational Complexity

We conclude this section by proving that we can implement MergeSplitTours in time polynomial in the input size.

Theorem 2. *(Polynomial time complexity of* MergeSplitTours*) The time complexity of Algorithm 1 is* $\mathcal{O}(m \log n(m + n \log n))$*, where* $n = |V_A|$ *and* $m = |E_A|$*.*

Proof. The minimal connecting tours (MCT) routine is the computational bottleneck of the MergeSplitTours algorithm. The MCT problem corresponds to the minimum cost circulation problem (Williamson, 2019, Definition 5.1). If all edge capacities are integral, the linear relaxation of the circulation problem has a constraint matrix that is totally unimodular (Ahuja et al., 1993). Hence, *the linear relaxation will necessarily have an integer optimal solution*, which will be a fortiori an optimal solution to the original circulation problem with integral constraints, as we have here. We can solve the minimum cost circulation problem itself in time $\mathcal{O}(m \log n(m + n \log n))$, using Orlin's algorthim, which is the fastest known strongly polynomial algorithm (Orlin, 1993; Williamson, 2019).

We now analyze the rest of Algorithm 1. We can implement line 2 in time linear in the size of G_A by identifying the strongly connected components of the graph induced by $\boldsymbol{x}$ (Cormen et al., 2009, Chapter 22.5). Next, we claim that we can implement lines 3–9 in time $\mathcal{O}(\ell^2 \log \ell)$, where $\ell = |V_D|$. We precompute the values $\{\tilde{c}_{dd'} := c_{dd'} + c_{d'd}\}_{d,d' \in V_D}$ and sort them in ascending order in $\mathcal{O}(\ell^2 \log \ell)$ time. Next, we traverse this list from the smallest

value, and for each item $\tilde{c}_{dd'}$, we check if d, d' are already in the same depot set $D \in \mathcal{D}$. In case they are not, we update $x_{dd'} = x_{d'd} = 1$ and merge D and D'. This process terminates when $\mathcal{D}$ contains a single set. We can maintain $\mathcal{D}$ with a simple vector that maintains for each $d \in V_D$ the component of $\mathcal{D}$ to which it belongs. With this representation, each merge operation for $\mathcal{D}$ requires $\mathcal{O}(\ell)$ time. We must repeat the latter operation $\mathcal{O}(\ell)$ times. Thus, the complexity of lines 3–9 is still bounded by $\mathcal{O}(\ell^2 \log \ell)$.

Next, we analyze the complexity of line 10. We can extract T through an Eulerian tour on the subgraph of G_A induced by $\boldsymbol{x}$, in time linear in $|T|$. We can bound the latter as $\mathcal{O}(k\ell)$, where $k = |V_P|$, as a tour can visit potentially all depots between any two consecutive packages p_i, p_{i+1} on it. Finally, lines 11–18 are clearly linear in $|T|$. To conclude, the MCT computation dominates the overall running time of the algorithm, which is thus $\mathcal{O}(m \log n(m + n \log n))$. □

4. Multi-Agent Path Finding

For each drone $i \in [m]$, the allocation layer yields a sequence of deliveries $d_1 p_1 \dots p_l d_{l+1}$. The drone route planning layer treats each dpd' subsequence *as an individual task* that the drone needs to execute—leave with the package from depot d, carry it to package location p, and return to the (same or different) depot d'—without exceeding the total energy capacity. We seek an efficient and scalable method to obtain high-quality (with respect to travel time) feasible paths for m different drone tasks simultaneously, using transit options to extend range. We can execute the full set of delivery sequences by replanning when a drone finishes its current task and begins a new one; we discuss and compare two replanning strategies in Section 6.3. Thus, we formulate the problem of multi-drone routing to satisfy a set of delivery sequences as *receding-horizon multi-agent path finding (MAPF) over transit networks*. In this section, we describe the graph representation of our problem and present an efficient bounded suboptimal algorithm.

For sake of clarity, we restate the simplifying (though realistic) assumptions from Section 2 that apply to this layer. Underpinning these assumptions is our focus on the algorithmic challenges that arise in MAPF problems when considering time-dependent transit networks and per-agent range constraints. The myriad drone-specific engineering issues and corner cases are better suited for a separate research project. We assume a deterministic setting where buses follow the timetable precisely; dealing with delays and disruptions is a separate subfield of transportation planning research with tailored methods and heuristics. Previous work on a single-agent version of this problem considers both uncertainty and delays and could be integrated here (Choudhury et al., 2019). We ignore energy loss due to hovering while waiting for a bus by assuming the drone can fly at a lower speed to reach the bus just-in-time. Drone recharge time at a depot can be ignored by having extra batteries, replacing the used ones immediately, and recharging them offline. We could accommodate non-zero recharge times anyway by replanning for a drone after it has been recharged, since we only plan routes for one delivery task per drone at a time.

4.1 Multi-Agent Path Finding with Transit Networks (MAPF-TN)

We frame the problem of Multi-Agent Path Finding with Transit Networks (MAPF-TN) by extending standard MAPF to allow agents to use one or more modes of transit, besides

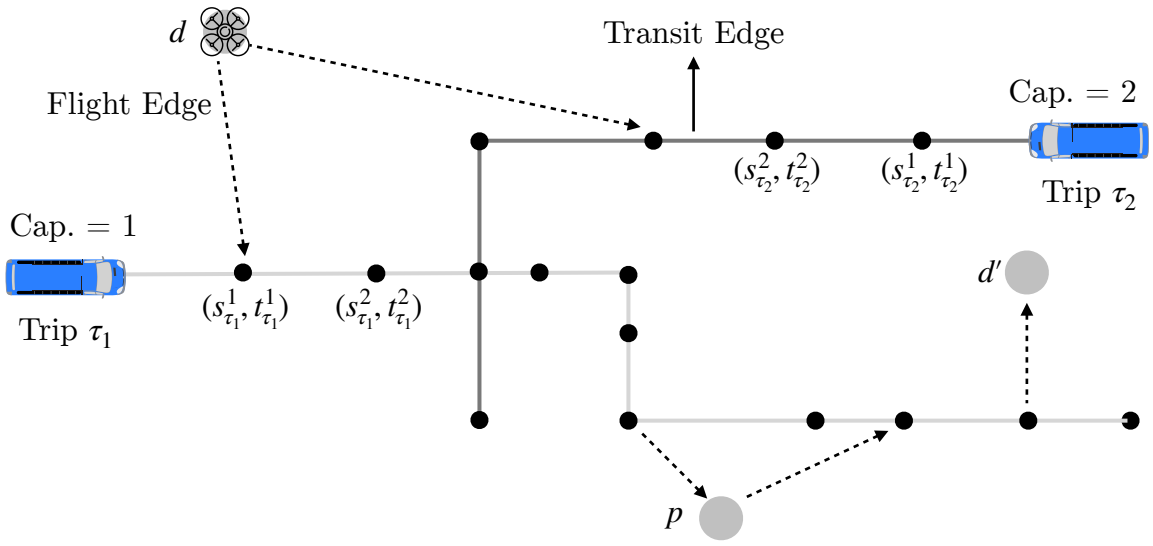


Figure 4: We illustrate the key elements of the directed operation graph in MAPF-TN. To aid visualization, we show only a single drone task (planning from depot d to package p and then returning to depot d') and two bus trips τ_1 and τ_2, with drone-carrying capacities of 1 and 2 respectively. Each trip is a directed sequence of transit vertices, which are represented as time-stamped locations or coordinates, e.g., $(s^1_{\tau_1}, t^1_{\tau_1})$ is the first stop for bus trip τ_1, with the bus arriving at location $s^1_{\tau_1}$ at time $t^1_{\tau_1}$. A transit edge is one between two consecutive transit vertices, and a flight edge is one from a drone location to a transit vertex.

moving themselves. Incorporating transit networks introduces additional challenges and underlying structure. The **input** to MAPF-TN is the set of m tasks $(d, p, d')_{1:m}$ and the directed operation graph $G_O = (V_O, E_O)$. In Section 3, the allocation graph G_A only considered depots and packages and edges between them. Here, in addition to the depot and package locations, the MAPF-TN operation graph V_O also includes the set of vertices V_{TN} from the transit network. Therefore, we have $V_O = V_D \cup V_P \cup V_{TN}$. In general, the depot and package locations (specific street addresses) are distinct from the transit network stops. Figure 4 illustrates the key components of the MAPF-TN operation graph.

We first describe how we define the transit vertices V_{TN}, using the standard time-expanded representation from the transportation planning community (Pyrga, Schulz, Wagner, & Zaroliagis, 2008). We are given the timetable of a transit network (e.g., of buses). The timetable comprises a set of bus *trips* $\mathcal{T}$. Each trip $\tau \in \mathcal{T}$ is a directed sequence of transit vertices, which are bus-stop locations (geographical coordinates) stamped with the arrival time (a standard assumption for convenience is that the arrival and subsequent departure are simultaneous). We can thus encode a trip as $\tau = \{(s^1_\tau, t^1_\tau), (s^2_\tau, t^2_\tau) \ldots\}$, where s is the location and t is the corresponding time-stamp. Therefore, the full set of transit vertices is $V_{TN} = \bigcup_{\tau \in \mathcal{T}} \tau$, i.e. the collection of all trips.

We now discuss the edges E_O in our directed operation graph. Any edge $e = (u, v)$ is a *transit edge* if its source u and target v are consecutive transit vertices on the same

trip τ_t; any other edge is a *flight edge*. An edge is *time-constrained* if $v \in V_{TN}$ and *time-unconstrained* otherwise. Every edge has three attributes: traversal time T, energy expended N, and capacity C. Since each vertex is associated with a location, $\|v-u\|$ denotes the geographical distance between vertices. MAPF typically abstracts away agent dynamics; we have a simple model where drones move at constant speed σ, and distance flown represents energy expended. Due to the high graph density (drones can fly point-to-point between many stops), we do not explicitly enumerate edges but generate them on-the-fly during search, as in previous single-agent work (Choudhury et al., 2019).

We now define the three attributes for the edge set E_O. For time-constrained edges, $T(e) = v.t - u.t$ is the difference between target and source time-stamps (if $u \in V_D \cup V_P$, $u.t$ is the chosen departure time), and for time-unconstrained edges, $T(e) = \|v-u\|/\sigma$ is the time of direct flight. For flight edges, $N(e) = \|v-u\|$ (flight distance), and for transit edges, $N(e) = 0$. For transit edges, $C(e)$ is the finite drone-carrying capacity of the vehicle (which would depend on its size), while for flight edges, capacity is irrelevant and we set $C(e) = \infty$. Here, we assume that collision-free drone flight in open space can be accommodated and we abstract away inter-flight-edge collision constraints (Ho, Salta, Geraldes, Goncalves, Cavazza, & Prendinger, 2019).

We now have a well-defined graph for our MAPF-TN problem. The remaining relevant details carry over from the standard formulation of MAPF problems. An individual path π_i for drone i from d_i through p_i to d_i' is **feasible** if the energy constraint $\sum_{e\in\pi_i} N(e) \leqslant \bar{N}$ is satisfied, where $\bar{N}$ is the drone's maximum flight distance. The drone should also be able to traverse the distance of a time-constrained flight edge in time, i.e., $\sigma \times (v.t - u.t) > \|v-u\|$. *For simplicity, we abstract away energy expenditure due to hovering in place by flying the drone at a reduced speed to reach the transit just in time.* The constraint $\bar{N}$ is then only on the traversed distance. The cost of an individual path is the total traversal time, $T(\pi_i) = \sum_{e\in\pi_i} T(e)$. A **feasible solution** $\Pi = \pi_{1:m}$ is a set of m individually feasible paths that does not violate any shared constraints, which can be of two kinds (Figure 5): (i) *Boarding constraint*, i.e., no two drones may board the same vehicle at the same stop; (ii) *Capacity constraint*, i.e., a transit edge e may not be used by more than $C(e)$ drones. As with the allocation layer, the **global objective** for MAPF-TN is to minimize the solution makespan, $\operatorname{argmin}_\Pi \max_{\pi\in\Pi} T(\pi)$, i.e., minimize the worst individual completion time.

4.2 Conflict-Based Search for MAPF-TN

To tackle MAPF-TN, we consider the family of MAPF algorithms built upon Conflict-Based Search (Sharon et al., 2012). The multi-agent level of Conflict-Based Search identifies shared constraints and imposes corresponding path constraints on the single-agent level. The single-agent level computes optimal individual paths that respect all constraints. If individual paths conflict (by violating a shared constraint), the multi-agent level adds further constraints to resolve the conflict, and invokes the single-agent level again for the conflicting agents. Conflict-Based Search obtains optimal multi-agent solutions without having to run potentially expensive joint multi-agent searches. However, its performance can degrade heavily with many conflicts in which constraints are violated. Figure 5 illustrates how conflicts are generated and resolved in our MAPF-TN problem.

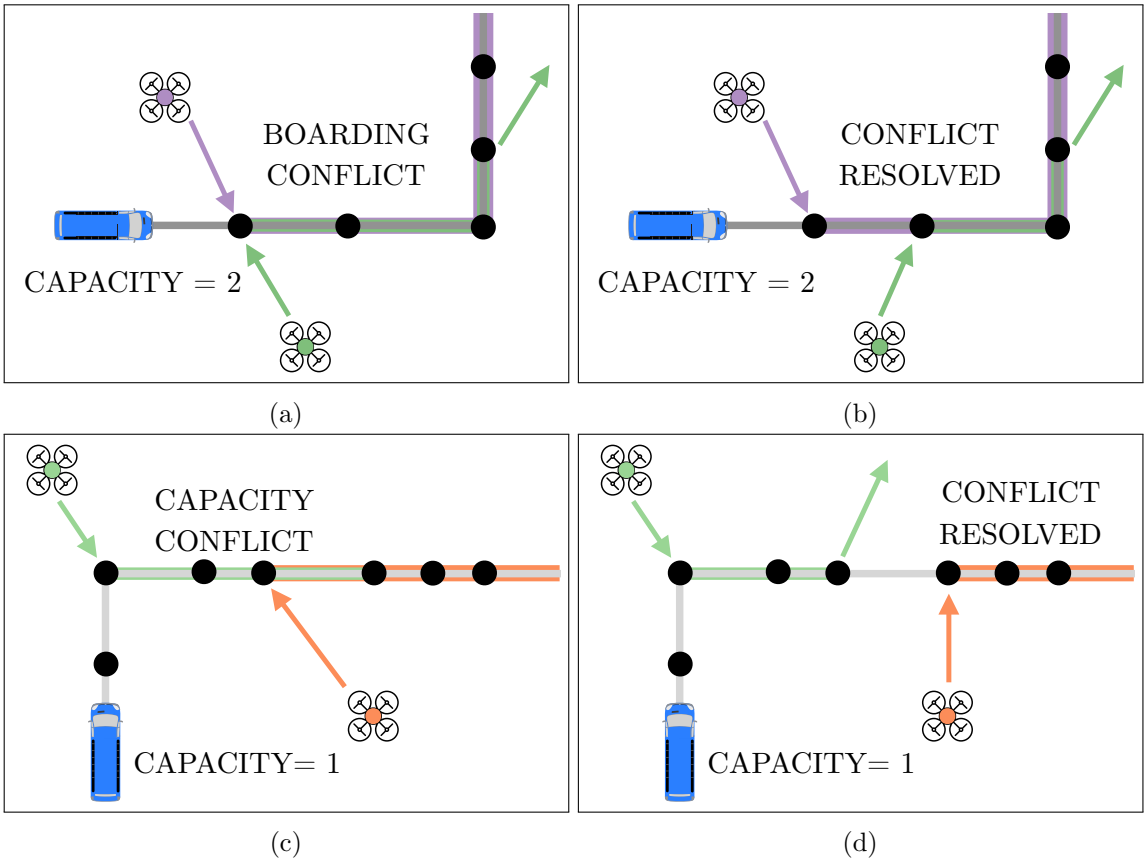


Figure 5: In our formulation of multi-agent path finding with transit networks, conflicts arise from the violation of shared inter-drone constraints: (a) boarding conflicts between two or more drones and (c) capacity conflicts between more drones than the transit vehicle can accommodate. The modified paths after resolving the corresponding conflicts are depicted in (b) and (d), respectively.

For scalability to large operation graphs (due to the underlying transit networks), we use a bounded suboptimal variant of Conflict-Based Search called *Enhanced Conflict-Based Search* (ECBS), which can be orders of magnitude faster than the optimal one (Barer, Sharon, Stern, & Felner, 2014). ECBS uses bounded suboptimal *Focal Search* (Pearl & Kim, 1982) instead of best-first A* (Hart, Nilsson, & Raphael, 1968) at both levels. In addition to the regular open list of A*, Focal Search maintains a separate subset of the open list with those nodes whose *f-value*, i.e., the sum of cost-to-come and heuristic cost-to-go is bounded suboptimal with respect to the best f-value seen so far. This subset, the so-called focal list, ensures that once a solution is found it is guaranteed to be bounded suboptimal. It also allows Focal Search to use an (often domain-dependent) inadmissible heuristic to guide the search, which can prioritize efficiency.

We now describe *three modifications to the standard ECBS framework* for our MAPF-TN problem formulation. The first two are necessary extensions to allow the bounded-suboptimality of ECBS to carry over directly to the MAPF-TN problem: a generalized

lower-level search routine that accounts for individual agent constraints and a generalized handling of conflicts to account for transit edge capacities. The third is a collection of MAPF-TN specific speedup techniques that improve empirical performance without sacrificing bounded suboptimality. Algorithm 2 provides (rather coarse-grained) pseudocode for the multi-agent search level of ECBS for our MAPF-TN problem.

4.2.1 Focal Weight-Constrained Search

The single-agent search in MAPF-TN must satisfy a *path-wide constraint* (traversal distance) in addition to minimizing the objective function of traversal time. This constraint satisfaction requirement is atypical for classical MAPF problems. For the shortest path problem on graphs, adding a separate path-wide weight constraint makes it NP-hard (Garey & Johnson, 1990). Most algorithms for such weight-constrained shortest path search require an explicit enumeration of the edges (Dumitrescu & Boland, 2003; Carlyle et al., 2008). We extend the A* for Multi-Constraint Shortest Path (A*-MCSP) algorithm (Li et al., 2007), which is suitable for our implicit graph representation, to focal search; we call this subroutine Focal-MCSP.

Focal-MCSP uses separate heuristics on both the cost and the weight constraint functions. It maintains only the intermediate paths to expanded nodes whose f-value is *both bounded suboptimal and feasible* with respect to the constraint. The algorithmic extension of A*-MCSP to Focal-MCSP is quite straightforward but the extensive book-keeping of the feasible intermediate paths with bounded suboptimal f-value requires a careful implementation for efficiency. Therefore, we omit separate pseudocode for Focal-MCSP as it would require several additional implementation-specific symbols and hamper overall readability; the interested reader can consult the respective pseudocodes of A*-MCSP (Li et al., 2007) and Focal Search (Pearl & Kim, 1982).

By uniting the constraint satisfying pathfinding logic of A*-MCSP and the bounded suboptimality heuristic search logic of Focal Search, i.e., Focal-MCSP yields (by construction) a bounded suboptimal feasible path from the start to the goal. Furthermore, from the analysis of Enhanced Conflict-Based Search (Barer et al., 2014), ECBS is bounded suboptimal if the single-agent graph search routine for individual agents is also bounded suboptimal. Therefore, by construction, **Enhanced Conflict-Based Search (ECBS) with Focal-MCSP yields a bounded suboptimal solution to Multi-Agent Path Finding with Transit Networks (MAPF-TN)**.

4.2.2 Capacity Conflicts in MAPF-TN

In the classical MAPF formulation, at most one agent can occupy a particular vertex or traverse a particular edge at a given time. Therefore, conflicts between $p > 1$ agents yield p new nodes in the high-level search tree of Conflict-Based Search and its variants. In MAPF-TN, however, transit edges have capacity $C(e) \geqslant 1$. Consider a solution generated during a run of ECBS that has assigned to some transit edge $p > C(e) > 1$ drones. To guarantee bounded suboptimality of the solution, we must generate all $\binom{p}{p-c}$ sets of constraints, where $c = C(e)$. Each such set of $(p-c)$ constraints represents one subset of $(p-c)$ agents restricted from using the transit edge in question.

Algorithm 2: The multi-agent level of Enhanced CBS for MAPF-TN.

Input: Drone Tasks $(d, p, d')_{1:m}$, Operation Graph G_O, Cost function $T(\cdot)$, Weight function $N(\cdot)$, Weight constraint $\bar{N}$, Suboptimality Factor ϵ

Output: Drone routes $\pi_{1:m}$ that respect per-drone and inter-drone constraints

Initialize A as root of multi-agent level ECBS search tree
$A.constraints \leftarrow \emptyset$ `// Empty constraint set`
$A.solution = \{\pi_i\}_{1:m}$ where $\pi_i \leftarrow$ Focal-MCSP$(d_i, p_i, d'_i, V_{TN}, \bar{N}, N, T, \epsilon)$
    `// Individually feasible and bounded suboptimal drone paths`
$A.cost = \max_{\pi_i \in A.solution} T(\pi_i)$ `// Makespan of set of paths`
Insert A into Open and Focal `// Initialize both lists`
while Open $\neq \emptyset$
    $mincost \leftarrow$ Top(Open).$cost$
    Focal $\leftarrow$ Focal $\setminus \{O \in$ Open $\mid O.cost \geqslant (1+\epsilon) \cdot mincost\}$ `// Only retain bounded suboptimal nodes in focal list`
    $S \leftarrow$ PopBest(Focal) `// Can use inadmissible heuristic in criteria`
    if *S.solution has no conflicts*
        return *S.solution*
    $C \leftarrow$ first conflict in S `// Either boarding or capacity conflict`
    for *each conflicting path* π_j *in* C
        Initialize new node P `// Child node in ECBS search tree`
        $P.constraints \leftarrow S.constraints \cup C.constraints$ `// Include constraints derived from conflict`
        $P.solution \leftarrow S.solution \setminus \pi_j$
        $P.solution \leftarrow P.solution \cup$ Focal-MCSP$(d_j, p_j, d'_j, V_{TN}, \bar{N}, N, T, \epsilon)$
            `// Recompute conflicting path with updated constraints`
        $P.cost = \max_{\pi_i \in P.solution} T(\pi_i)$ `// Compute updated makespan`
        Insert P into Open and Focal

As we will show in Section 6.2, conflict resolution is a significant bottleneck for solving large MAPF-TN instances. In our experiments, we generated all constraint subsets of a capacity conflict for completeness, however, pathological scenarios may arise where this degrades empirical performance. Future research could consider a principled way to analyse constraint set enumeration and an efficient way to implement it in practice.

4.2.3 Focal-MCSP Speedup Techniques

The NP-hardness of multi-agent path finding (Yu & LaValle, 2013) and the additional computational challenges of MAPF-TN (energy constraint on per-agent paths; large and dense transit graphs) make empirical performance paramount, given our need for scalability on real-world scenarios. We now discuss two speedup techniques for our single-agent search routine (Focal-MCSP) that improve its efficiency while maintaining its bounded suboptimality (in turn ensuring bounded suboptimality of the overall MAPF-TN solution). These speedup techniques are not exhaustive; there is an entire body of work in transportation planning devoted to speeding up algorithm runtimes (Delling et al., 2009).

Preprocessing Public Transit Networks

Focal-MCSP depends significantly on the quality of admissible heuristics, i.e., heuristics that *underestimate* the cost to the goal, for both the objective (elapsed travel time) and the constraint (flight distance traversed). We know the public transit network and its timetable for a given area in advance. We can analyze and preprocess this network to obtain admissible heuristics and use them in multiple instances of MAPF-TN throughout a business day, while searching for paths to a specific package delivery location.

For the objective of elapsed travel time, a potential lower bound is the time-of-flight directly to the goal, ignoring public transit (of course, taking such a route in practice is usually infeasible due to the flight distance constraint). Therefore, we define this heuristic simply as $h_T(v, vg) = \frac{\|vg-v\|}{\sigma}$, where σ is the average drone speed, vg is the goal node and v is the expanded node. The above heuristic is admissible *if the average drone speed is higher than average transit speed*. This assumption is valid for the bus networks and drone speed parameters of our experiments in Section 6; the buses operate primarily in urban streets with strict speed limits and must also follow a timetable and wait for passengers to board. All of these factors drive down the average transit speed below the drone flight speed in practice. A more data-driven heuristic could be obtained by analyzing actual drone flight times, but that is out of our scope.

For the constraint on flight distance traversed, we use a heuristic based on extensive network preprocessing. We consider the minimal time window such that every instance of a transit vehicle trip in the network can start and finish (as per the timetable). We then create a so-called *trip metagraph*, where the vertex set is $V_D \cup V_P \cup V_{\mathcal{T}}$; recall that V_D and V_P are the sets of depot and package vertices respectively. Each vertex in $V_{\mathcal{T}}$ represents a single transit vehicle trip from the first to the last stop on its route, and encodes its sequence of time-stamped stops. The trip metagraph is complete, i.e., there is an edge between every pair of metagraph vertices.

We now define the cost due to flight distance traversed for each directed trip metagraph edge $e = (r_\tau, r_{\tau'})$, hereafter denoted as $e = (r_\tau \to r_{\tau'})$ for notational convenience. The trip metagraph vertices $r_\tau, r_{\tau'} \in V_{\mathcal{T}}$ correspond to transit vehicle trips R_τ and $R_{\tau'}$, respectively; we reiterate that each trip metagraph vertex (besides the package and depot vertices) represents a full trip of a transit vehicle, and each transit vehicle trip has time-stamped stops that are themselves transit vertices in the operation graph, i.e., $V_{TN} \subset V_O = \bigcup_{\tau \in \mathcal{T}} \tau_\tau$ (from Section 4.1). Therefore, the trip metagraph edge cost (due to flight distance) between two vertices that each represent a trip is

$$N(r_\tau \to r_{\tau'}) = \min_{u \in R_\tau, v \in R_{\tau'}} \|v - u\|, \text{such that } \sigma \times (v.t - u.t) \geqslant \|v - u\|,$$

where, as before, $v.t$ refers to the time-stamp of the trip stop $v \in R_{\tau'}$. The edge cost here is thus *the shortest flight distance between stops from one trip to another that the drone can cover in the time difference between them.* For the trip metagraph edges between depots and packages, we simply set the energy cost as the direct flight distance between the corresponding depot and/or package locations. For all other edges, i.e., where one vertex r_τ corresponds to a trip R_τ and the other v to a depot/package, we set the edge cost $N(r_\tau \to v) = N(v \to r_\tau) = \min_{u \in R_\tau} \|v - u\|$, i.e, the closest distance between the depot/package and the trip.

Given the complete specification of the edge cost function for the trip metagraph, we now run Floyd-Warshall's All-Pairs Shortest Path algorithm (Cormen et al., 2009) on it to get a cost matrix $\bar{N}_{\mathcal{T}}$. This cost matrix encodes the *minimum flight distance required to switch from one trip to another*, from a trip to a depot/package and vice versa, and between two depots/package locations, either using the transit network or flying directly, whichever option is shorter.

We can now define the heuristic function h_N for the flight distance traversed to the Focal-MCSP goal node $vg \in V_D \uplus V_P$. The heuristic function assigns a value to the operation graph node $v \in V_O \equiv (V_D \uplus V_P \cup V_{TN})$ expanded during Focal-MCSP. If $v \in V_D \uplus V_P$ is a depot or package, we set $h_N(v, vg) = \bar{N}_{\mathcal{T}}(v, vg)$, i.e., the Floyd-Warshall cost matrix value. Otherwise, $v \in V_{TN}$ is a transit vertex. Recall that each transit vertex is associated with a corresponding transit trip; let the trip that contains v be R_τ. We then set heuristic value $h_N(r_\tau, vg) = \bar{N}_{\mathcal{T}}(r_\tau, vg)$, where $r_\tau \in V_{\mathcal{T}}$ is the trip metagraph vertex corresponding to the trip R_τ. The heuristic h_N as defined above is *a lower bound on the drone's flight distance from the expanded operation graph node to the target depot/package location.*

In practice, we will solve multiple MAPF-TN instances throughout a business day, with traffic delays and other timetable disruptions. The handling of dynamic networks and timetable delays is a separate subfield of research in transportation planning and out of our scope (Delling et al., 2009; Bast et al., 2016). We assume that travel times between locations do not vary throughout the day, and we ignore the effect of disruptions to the pre-determined timetable (both are assumptions made often in transit planning).

Pruning the Search Space

We mentioned earlier that we do not explicitly enumerate the edges of the operation graph but rather implicitly encode and generate them just-in-time during the node expansion stage of Focal-MCSP. An implicit edge set makes Focal-MCSP memory-efficient at the cost of additional computation time for the outgoing edges during search. However, we are able to prune the set of out-neighbors of a vertex expanded during Focal-MCSP, *while still guaranteeing bounded suboptimality.*

Let $u \in V_O$ be an operation graph vertex expanded during Focal-MCSP. Consider the transit vertices of any trip R_τ (if $u \in V_{TN}$ is itself a transit vertex, consider a trip different from the one that u lies on). The transit vertices of R_τ are candidate out-neighbors for the expanded node u, i.e., candidate target vertices of a *time-constrained flight edge* from u that connects to the trip R_τ. While considering these flight connections to a trip R_τ, we only need to add the transit vertices on R_τ that are *non-dominated* by any other in terms of time difference and flight distance (explained subsequently). We show in the following lemma how this pruning does not violate the bounded suboptimality of Focal-MCSP; the lemma formalizes the logic that if a transit connection is useful to the drone, a stop that is both earlier and closer in distance than another will always be preferred.

Lemma 2. *(Bounded suboptimal search-space pruning in Focal-MCSP) Let $u \in V_O$ be expanded during Focal-MCSP. Let v_1 and v_2 be two consecutive transit vertices on trip R_τ such that v_1 dominates v_2. Notationally, $(v_1.t, \|v_1 - u\|) \preceq (v_2.t, \|v_2 - u\|)$, i.e. $v_1.t < v_2.t$ (equality cannot hold as two different stops on the same trip must have different time-stamps) and $\|v_1 - u\| \leqslant \|v_2 - u\|$. Then, pruning v_2 as an out-neighbor for u does not change the solution cost of Focal-MCSP.*

The following proof relies heavily on the analysis of A*-MCSP, upon which Focal-MCSP is based (Li et al., 2007, Section V).

Proof. We assume v_1 and v_2 are both reachable by the drone, i.e., $\sigma \times (v.t - u.t) \geqslant \|v - u\|$ for $v = v_1, v_2$, otherwise they are rejected anyway. Since $(v_1 \to v_2)$ is a transit edge, the flight distance $N(v_1 \to v_2) = 0$, by definition. Focal-MCSP tracks both the *objective* (traversal time) and *constraint* (flight distance) values of partial paths to nodes. It discards a partial path dominated by any other on both metrics.

The two possible partial paths to v_2 from the expanded node u are $u \to v_2$ and $u \to v_1 \to v_2$. Let the flight distance accumulated on the path thus far to u be Wu. The traversal time cost at v_2 for both partial paths is $v_2.t$ (since v_2 is time-stamped). The accumulated flight distance at v_2 for $u \to v_2$ is $Wu + N(u \to v_2) = Wu + \|v_2 - u\|$. But for $u \to v_1 \to v_2$, the corresponding accumulated weight at node v_2 is $Wu + N(u \to v_1) + N(v_1 \to v_2) = Wu + \|v_1 - u\| < Wu + \|v_2 - u\|$, by our original assumption. By construction, Focal-MCSP will discard the partial path $u \to v_2$ in favor of $u \to v_1 \to v_2$. Therefore, pruning v_2 as an out-neighbor has no effect on the solution of Focal-MCSP. □

Our proof above was for *consecutive* vertices on a transit trip; we can extend it to the full sequence of vertices on the trip by induction. We use Kung's algorithm to find the non-dominated transit trip vertices (Kung et al., 1975). For two criteria functions, Kung's algorithm yields a solution in $\mathcal{O}(n \log n)$ time; here n is the size of the set and the bottleneck is sorting the set for one criterion. In our specific case, the transit trip vertices are already sorted in increasing order of time-stamps. Therefore, we can add out-neighbors for a transit trip in $\mathcal{O}(n)$ time, *which is as fast as we could have done anyway.*

There is a crucial practical caveat to the preceding section. We have presented a general MAPF solver over transit networks that computes a set of conflict-free, weight-feasible, and bounded suboptimal cost paths for each agent from its start to its goal, *a novel contribution in and of itself.* However, for the drone delivery problem, a dpd' task requires a bounded suboptimal path from d to p and another from p to d', such that their concatenation is feasible. Neither Conflict-Based Search nor Multi-Constraint Shortest Path methods address such path concatenation settings, and developing a general extension for this purpose is a non-trivial problem and out of our scope.

We circumvent this problem of computing a bounded-suboptimal, weight-constrained, concatenated path, by simply running Focal-MCSP twice (from d to p and p to d') with half the energy constraint each time, and concatenating the resulting paths. By doing so, we can guarantee feasibility but not completeness, since our space of solutions is that of all dpd' paths. We use this abstraction for computational convenience, but it is one we could realize in practice as well. Instead of a single battery, we could fit the drone with two smaller batteries, each providing half the flight range of the larger (since battery capacity increases with volume). The first battery would power the drone's journey to the delivery location, and the second would power its return to a depot.

5. Surrogate Cost Estimate for Layer Coupling

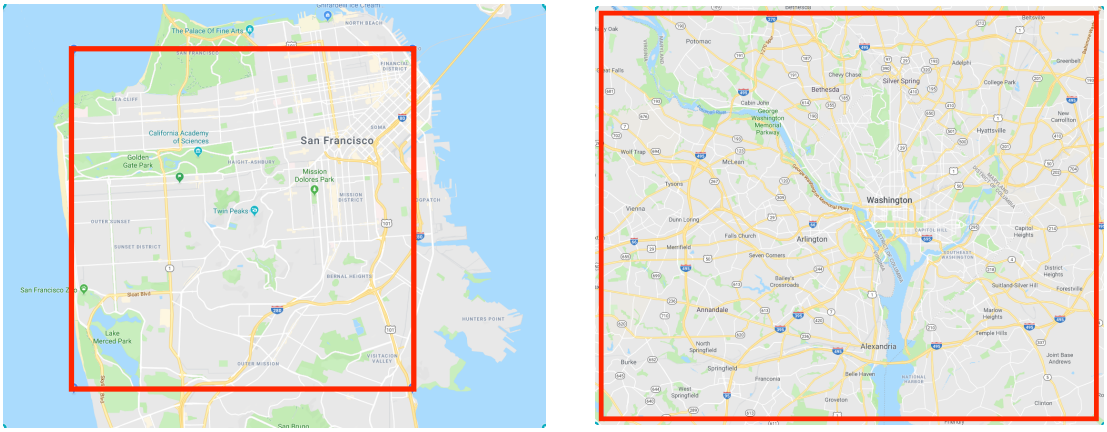
In order to scale to large problems, we use a decomposition-based stage-wise optimization instead of jointly solving for allocation and multi-agent pathfinding over transit networks. Such stage-wise methods suffer from an approximation gap compared to the optimal solution of the full problem. For us, this gap manifests in the surrogate cost estimate for the drone travel time in the task allocation layer; the optimality property of the task allocation layer is in terms of this surrogate. The allocation layer's solution determines the start and goal locations and thus constrains the set of feasible solutions for the multi-agent pathfinding layer. *The better the surrogate estimate, the more coupled the layers are*, i.e., the better is the solution of the first stage for the second one.

Surrogate functions typically trade efficiency for approximation quality. An easy-to-compute travel time surrogate, for instance, is the drone's direct flight time between two locations (ignoring possible use of transit). However, such a surrogate can be particularly poor when the drone requires transit to reach an out-of-range target. As we mentioned earlier in Section 2.2, we pre-compute a surrogate travel time estimate that accounts for the transit network. Consider the given geographical area of operation encoded as a bounding box of coordinates (Figure 6 illustrates both areas that we work with). During preprocessing, we generate a representative set of locations across the area. We use a quasi-random low dispersion sampling scheme to compute the locations for good coverage (Halton, 1960). This set of locations induces a Voronoi partition of the geographical area into regions, where the locations are the so-called sites, one for each region (Cormen et al., 2009). Any point in the full bounding box is associated with the nearest element in the set of sites and the corresponding region that it falls in, by an appropriate distance metric.

Next, we compute a drone travel time estimate between each pair of locations or sites. The actual drone travel time in the course of a business day would depend on the state of the transit network when the route is planned. During preprocessing we choose a certain representative time segment of the daily timetable; specifically, we choose the smallest time window within which at least one instance of a bus trip is executed on every route in the area. *Using this snapshot of the transit network, and our Focal-MCSP search algorithm, we pre-compute the drone travel time between every pair of Voronoi sites*, assuming in each case that the drone trip begins at the start of the time window.

One such transit network snapshot is sufficient for our experiments as we only evaluate the MAPF layer on at most two delivery tasks per drone, with the second task used only for evaluating the replanning strategies. In an actual deployment, we could generate multiple transit network snapshots over the course of a business day (e.g., morning, afternoon, and evening), compute a corresponding surrogate for each snapshot, and use the appropriate surrogate when allocating at the start of each session of the day.

Recall that the task allocation layer queries the estimated travel time between two depot/package locations $v, v' \in V_D \uplus V_P$ during its computations. Both v and v' have corresponding nearest representative locations (the sites of the Voronoi regions they respectively fall in). We then *use the pre-computed travel time estimate between the corresponding sites* for MergeSplitTours; here we assume the travel time between the representative sites dominates the last-mile travel between each site and its corresponding depot/package. If v and v' are in the same cell, i.e., their nearest Voronoi site is the same, we use the direct flight



(a) (b)

Figure 6: The geographical bounding boxes, highlighted in red, for (a) San Francisco (roughly 150 km^2) and (b) Washington DC Metropolitan area (roughly 400 km^2).

time between v and v', i.e., $\|v' - v\|/\sigma$. The locations v and v' are more likely to share a cell if they are close together, in which case the drone is more likely to fly directly between them anyway. Note that in our experiments, we randomly generate all depot locations in each trial while using the same preprocessed surrogate for a city. In an actual deployment, since the depot locations will be fixed from day to day, we can use those fixed depot locations as some of the sites that induce the Voronoi decomposition.

6. Experiments and Results

We implemented our approach in the Julia programming language for fast numerical simulations and tested it on a machine with a 6-core 3.7 GHz 16 GiB RAM CPU.[1] *For large combinatorial optimization problems, we care about solution quality and computational scalability and efficiency.* We have already shown that the task allocation and multi-agent pathfinding layers of our framework are approximately optimal and bounded-suboptimal respectively in terms of solution quality, i.e., makespan. Therefore, we focus on their efficiency and scalability to large real-world settings; in the case of multi-agent pathfinding, we also evaluate the real-world makespan for deliveries. We do not baseline against a mixed-integer linear program for the full problem; a typical setting of interest will have on the order of 10^7 variables in such a formulation, besides an exponential number of constraints.

We ran simulations with two large-scale public transit networks in San Francisco (SFMTA) and the Washington Metropolitan Area (WMATA). We used the open-source General Transit Feed Specification[2] data for each network. Our formulation can accommodate multiple modes of transit seamlessly, but here we considered only the bus network (by far the most extensive). We defined a geographical bounding box in each case, of area 150 km^2 for SFMTA and 400 km^2 for WMATA (illustrated in Figure 6), within which depots and package locations were randomly generated; we pruned all bus trips outside the bounding box.

1. The code for our work is available at `https://github.com/sisl/MultiAgentAllocationTransit.jl`.
2. `https://developers.google.com/transit/gtfs`

Table 2: (All times are in seconds) The mean computation time for MergeSplitTours, over 100 different trials for each setting. The numbers demonstrate that MergeSplitTours is polynomial in input size and highly scalable. Here, $k = |V_P|$ is the number of package deliveries and $\ell = |V_D|$ is the number of depots. For all instances that took longer than 60 s, we used 10 trials. Values greater than 1×10^3 s are rounded out.

k	$\ell = 2$	$\ell = 5$	$\ell = 10$	$\ell = 20$	$\ell = 30$
50	0.004	0.016	0.057	0.248	0.658
100	0.012	0.050	0.195	0.807	2.117
200	0.038	0.173	0.699	2.968	8.409
500	0.201	1.025	4.384	18.19	49.97
1000	0.781	4.109	24.30	76.58	397.9
5000	22.74	319.2	1089	3581	6435

The *size* of the time-expanded network, $|V_{TN}|$, is the total number of stops made by all trips; $|V_{TN}| = 4192$ for SFMTA and $|V_{TN}| = 7608$ for WMATA (recall that edges are implicit, so $|E_{TN}|$ varies with problems, but the full graph G_O is dense). We set the drone flight range constraint conservatively to 7 km and the average speed to 25 kph, based on DJI Mavic 2 specifications.[3] For the much larger WMATA area, we used a flight range of 10 km to enable more feasible solutions.

In this section, we first evaluate the two main components: the task allocation and multi-agent path finding layers. We then compare the performance of two replanning strategies for when a drone finishes its current delivery, and two different surrogate travel time estimates for coupling the layers.

6.1 Task Allocation

The number of depots ℓ and packages k determine the size and computational complexity of the allocation problem, which in turn affects the runtime that we evaluate here. The number of drones m is irrelevant for these experiments and we just set it equal to the number of depots; the choice of surrogate function is also irrelevant for allocation computation time and near-optimality, which is over the space of allocations that use the given surrogate as the edge cost. We display the runtimes for MergeSplitTours with varying ℓ, k over SFMTA in Table 2; the test cases are randomly generated locations over the SFMTA area.

The roughly quadratic increase along a specific row or column reflects the complexity bound (Theorem 2) of our approximately optimal MergeSplitTours algorithm. Given the commodity hardware we used, **the absolute runtimes are reasonable**. Consider the setting of 5000 deliveries and 10 depots, which is large enough to represent a half-day in a large urban area. The average runtime is 1089 s or approximately 18 min, which is negligible compared to a half-day operation time of several hours. We do not compare with naive mixed-integer linear programming even for allocation, as the number of variables would exceed $(\ell \cdot k)^2$, in addition to the expensive subtour elimination constraints (Miller et al., 1960).

3. https://www.dji.com/mavic-2/info#specs

Table 3: (All times are in seconds) An extensive analysis of the MAPF-TN layer, on 100 trials for each setting of depots and agents (and 30 trials for 5 depots and 50 agents). Each trial uses different randomly generated depots and delivery locations. We randomly sampled the integer carrying capacity of any transit edge $C(e)$ from $\{3, 4, 5\}$, representing single and double buses, and set the suboptimality factor for ECBS to 1.1.

		Plan Time		Range Ext.		Transit Used		Mean
Depots	Agents	Median	Mean	Mean	Max	Mean	Max	Makespan
San Francisco $(\lvert V_{TN}\rvert = 4192$; Area 150 $\mathrm{km}^2)$								
5	10	0.61	1.17	1.53	3.41	2.93	6	2554.7
5	20	1.39	2.13	1.61	2.66	3.48	6	2886.8
5	50	2.13	3.89	1.64	2.48	4.2	6	3380.9
10	20	0.41	1.02	1.24	2.35	2.31	6	2091.6
10	50	0.73	1.46	1.38	3.58	2.94	5	2504.7
10	100	2.09	7.29	1.43	2.16	3.67	8	2971.8
20	50	0.17	0.46	0.98	1.69	1.09	7	1273.6
20	100	0.49	1.05	1.06	1.79	1.61	9	1642.4
20	200	0.89	2.10	1.13	2.31	2.23	6	1898.5
Washington DC $(\lvert V_{TN}\rvert = 7608;$ Area 400 $\mathrm{km}^2)$								
5	10	3.91	5.65	1.66	3.08	3.18	7	5167.3
5	20	9.01	13.1	1.79	3.21	3.57	8	5384.5
5	50	19.1	28.9	2.07	3.21	4.44	7	6140.2
10	20	1.61	4.67	1.37	3.12	2.57	7	4017.2
10	50	4.77	15.8	1.72	3.03	3.53	7	5312.3
10	100	18.1	26.2	1.86	3.18	4.25	8	5623.9
20	50	0.73	1.92	1.29	2.88	2.23	7	3571.8
20	100	2.45	5.24	1.48	2.67	3.19	6	4304.5
20	200	4.68	10.5	1.61	2.87	3.58	7	5085.6

6.2 Multi-Agent Path Finding with Transit Networks (MAPF-TN)

The multi-agent pathfinding problem is NP-hard to solve optimally (Yu & LaValle, 2013). Researchers have previously benchmarked variants of Conflict-Based Search and shown that Enhanced Conflict-Based Search (ECBS) is among the most effective (Barer et al., 2014; Cohen et al., 2016). Therefore, we focus on evaluating our modified ECBS for multi-agent pathfinding with transit networks (MAPF-TN) rather than redundant baselining. Table 3 quantifies several aspects of our MAPF-TN solver with varying numbers of depots (ℓ) and agents (m). Each row shows aggregate results on randomly generated scenarios, 100 trials for smaller settings and 30 trials for larger ones. Before each trial, we run the allocation layer and collect m different dpd' tasks, one for each agent. We then run our MAPF-TN solver on this set of tasks to compute a solution.

Our approach scales to very large numbers of agents (200) and transit networks (nearly 8000 vertices); the highest average makespan for the true delivery time is less than an hour

(3380.9 s) for San Francisco and 2 hours (6140.2 s) for Washington (recall that this delivery time includes the time for each drone to return to its destination depot); drones are using up to 9 transit options per route to extend their range by up to 360%. As we anticipated, **conflict resolution is a major bottleneck of MAPF-TN**. A *higher ratio of agents to depots* increases conflicts due to shared transit, thereby increasing plan time, e.g., compare $(5, 20)$, i.e. 5 depots and 20 agents, to $(10, 20)$. *A higher number of depots* puts more deliveries within flight range of a depot, reducing conflicts, makespan, and the need for transit usage and range extension, e.g. compare $(10, 50)$ to $(20, 50)$.

We briefly discuss two kinds of pathological corner cases in our experiments, for which we terminated the corresponding trials. The first corner case is that of excessively high computation time due to too many high-level conflicts when certain capacity-constrained bus trips become bottlenecks for drones to make their deliveries; we discard any MAPF trial that exceeds 180 s of computation time. The second corner case is when one or more drones have no feasible path to their next destination, because the destination was out of flight range and there was no usable transit option. We do not report separate metrics for the corner cases as they are quite rare (at most one or two of the 100 trials for a setting, and only in a handful of the many settings). More importantly, in practice we can handle both of them by replanning for a subset of drones, dispatching them, and replanning for the remaining once the next set of bus trips has commenced.

The plan times are consistently higher for Washington than for San Francisco. The operation area for Washington DC is nearly three times that of North San Francisco and the WMATA bus network is nearly twice as big as SFMTA. Consequently, drones have a higher need for using transit to satisfy deliveries (the average transit usage metric is consistently higher than for SF), even with the higher drone flight range of 10 km as compared to 7 km. The WMATA bus network is more sparse in the outskirts and suburban areas, and transit becomes more of a bottleneck than for San Francisco. Additionally, the average low-level search time is higher because of the larger transit graph. Altogether, these conditions lead to Washington having both *higher single-agent search times and more multi-agent conflicts.* The relative range extension for Washington is similar to that for San Francisco despite the higher base flight range, which means that drones are reaching further off delivery locations in the absolute distance sense.

We make a few more general comments on the scalability of our MAPF-TN layer. Recall that each low-level search is actually *two* concatenated searches (from $d \to p$ and $p \to d'$), so the effective number of agents is actually $2m$ and not m; this observation only strengthens our scalability claim. In our benchmarks, we randomly generate depot placements, but an intelligent placement can reduce the number of high-level conflicts and significantly impact plan times (a key question for future work). The running times reported here are pessimistic, because we release drones simultaneously from the depots, which increases conflicts. However, a gradual release by executing the MAPF-TN solver over a longer horizon would result in fewer conflicts, allowing us to cope with an even larger drone fleet. Finally, we could even parallelize our solver for increased efficiency (Cohen et al., 2018).

With regards to solution quality (makespan), consider the real-world significance of the result that even for a large metropolitan area of 400 km^2, the longest delivery and subsequent return to the depot in a set of up to $m = 200$ tasks is well under 2 hours. We used a representative transit window that is largely replicated throughout the rest of the

Table 4: (All times are in seconds) We compare replanning strategies for a subset of the scenarios from Table 3 for the San Francisco network, over 20 trials for each setting. The boldfaced entries are for the scenarios where one strategy is strictly equal to or better than the other in terms of plan time and makespan.

		Replan-1		Replan-m	
Depots	Agents	Replan Time	Mean Makespan	Replan Time	Mean Makespan
5	10	**0.271**	2943.1	0.645	**2880.1**
5	20	**0.034**	**3092.2**	1.599	3092.2
20	50	**0.006**	**1463.5**	0.278	1463.5
20	100	**0.009**	**1952.2**	0.399	1952.2

day; therefore, for a given business day of, say, 12 hours, we can expect any drone to make *at least* 7 deliveries, and typically many more.

6.3 Replanning Strategies

Until now, we have discussed how our MAPF-TN solver computes paths for a single dpd' task for each drone. Recall that the task allocation layer assigns drones to a sequence of deliveries. Instead of computing paths for the entire sequence for each drone ahead of time, we use a receding horizon approach where we replan for a drone after it completes its current task. Our MAPF-TN computation time is negligible compared to the actual solution execution time (compare the 'Plan Time' and 'Makespan' columns in Table 3); therefore, a receding horizon strategy is quite reasonable in practice.

In this context, there are two natural replanning strategies: replanning *only* for the finished drone, while maintaining the paths of all the other drones (we call this Replan-1), and replanning for all drones from their current states (we call this Replan-m). These two approaches are at the opposite ends of the efficiency-optimality spectrum. The Replan-m strategy will be optimal among replanning strategies, while being the most computationally expensive. On the other hand, Replan-1 requires only the computation of a single path, since the remaining $m-1$ paths are unaffected.

To evaluate the two replanning strategies, we use the same setup as for MAPF-TN. For each MAPF-TN solution, with one path for each drone, we consider the drone that finishes its assigned delivery and returns to its depot first. Since we use a continuous time representation, ties are unlikely in practice. For Replan-1, we re-run the low-level search for the finished drone. We update the full m-agent solution with the new path, updating makespan if need be. For Replan-m, we re-run the full MAPF-TN solver for all m agents with their current states (at the time) as their initial state and obtain a different (m-agent) solution.

In Table 4, we compare the makespan and computation times of the m-agent solutions we obtain from the two strategies. We used a representative subset of the scenarios in Table 3; few depots with a low agent/depot ratio $(5, 10)$; few depots with a higher ratio $(5, 20)$; and similarly for many depots, i.e., $(20, 50)$ and $(20, 100)$. Clearly, **Replan-**1

Table 5: We compare our MAPF-TN results from Table 3 (Mean Plan Time and Makespan) against those where the framework uses the direct flight time as a surrogate estimate for MergeSplitTours instead of our preprocessed surrogate using representative locations. For each setting, i.e., row, in the SF and DC scenarios separately, if one of the surrogates is strictly equal to or better than the other for both plan time and makespan, then the record is boldfaced. The values for the Preprocessed columns are copied over from Table 3

.

		San Francisco				**Washington DC**			
		Preprocessed		Direct		Preprocessed		Direct	
Depots	Agents	Plan Time	Mean Mksp.	Plan Time	Mean Mksp.	Plan Time	Mean Mksp.	Plan Time	Mean Mksp.
5	10	**1.17**	**2554.7**	1.51	2624.8	5.65	5167.3	13.6	4654.7
5	20	**2.13**	**2886.8**	2.69	3092.9	13.1	5384.5	35.2	5339.6
5	50	**3.89**	**3380.9**	5.08	3412.4	**28.9**	**6140.2**	51.1	6323.4
10	20	1.02	2091.6	**0.83**	**1868.9**	**4.67**	**4017.2**	11.9	4527.3
10	50	1.46	2504.7	**1.25**	**2247.3**	**15.8**	**5312.3**	28.6	5509.6
10	100	7.29	2971.8	**3.78**	**2649.6**	**26.2**	**5623.9**	53.8	5774.1
20	50	0.46	1273.6	**0.27**	**1079.1**	**1.92**	**3571.8**	8.49	4058.1
20	100	1.05	1642.4	**0.64**	**1371.1**	**5.24**	**4304.5**	22.8	4613.9
20	200	2.10	1898.5	**1.43**	**1426.2**	**10.5**	**5085.6**	17.6	5216.1

achieves solutions of essentially the same makespan as Replan-m, at fairly lower computational cost. This result motivates our decision to use Replan-1 in practice.

In principle, we could design scenarios where Replan-1 has a much worse makespan compared to Replan-m than demonstrated in Table 4. However, the Replan-1 strategy is only suboptimal when (i) the $(m-1)$ unfinished drone paths could conflict with the new individual path of the drone that has just finished and when (ii) resolving the conflict(s) would have prioritized the path of the replanned drone over the others. In practice, it is unlikely that both of these conditions hold together, especially when there are many depots and some drones can fly directly to their next target. In our trials with 20 depots, for instance, the suboptimality conditions for Replan-1 never hold together, which is why the makespans for those two rows are exactly the same for both strategies.

6.4 Surrogate Estimates

We now compare two different surrogate travel time estimates: the preprocessed approximate travel time between representative locations in the city using transit (as described in Section 5) and the direct flight time between two locations, ignoring transit. For the earlier results in Table 3 (where we ran MAPF-TN on the first dpd' task for each drone), the MergeSplitTours task allocation algorithm used the preprocessed surrogate for the allocation graph edge costs. As a comparison, we re-run the exact same scenarios for both cities, as in Table 3 using the direct flight time surrogate as the edge cost for MergeSplitTours. We compare the two primary performance factors, plan time and solution

makespan for both surrogates in Table 5. For the same scenario in a city, if one surrogate yields equal or lower time and makespan than the other, its entry is boldfaced.

We expect the direct flight time surrogate to be a poor estimate in scenarios where transit is used frequently, because the allocation step does not account for it. Accordingly, we do observe a difference in plan time and solution quality between Preprocessed and Direct Flight for the settings with fewer depots and higher agent-to-depot ratios. For the settings with 5 depots in San Francisco, and for almost all settings in Washington (except the first two), both computation time and the makespan are lower for the preprocessed estimate, i.e., it is strictly better than direct flight. But for the settings in San Francisco with 10 or more depots, in most cases the drones are close enough to their deliveries to fly directly (recall the lower average transit usage of those cases from Table 3). Here the direct flight surrogate tends to be more accurate, leading to solutions that have lower makespan and are cheaper to compute.

The choice of surrogate cost clearly impacts the result of the allocation layer and in turn the downstream solution and corresponding makespan of the MAPF layer. However, **our experiments suggest that there is no obvious winner between the two** and it is ultimately an empirical question as to whether one is better for a particular setting than another. Even our explanations for the difference in makespan between them are at best rules of thumb. The questions of what other good surrogates might be, whether multiple surrogates can be combined in an ensemble, and what the good heuristics are for choosing a surrogate for a particular setting, are all potential avenues for further research.

7. Conclusion

We presented a comprehensive algorithmic framework for the problem of large-scale drone delivery of packages over transit networks. In our two-stage approach, we first solve the delivery sequence allocation problem with an approximately optimal polynomial time method, and then route the team of drones to deliver the packages with an efficient bounded suboptimal multi-agent pathfinding routine tailored to large transit networks.

We demonstrated various properties and results of our approach through extensive simulations with two real-world transit networks: our framework can scale to hundreds of drones and thousands of packages, computing close-to-optimal solutions that satisfy the many system constraints typically within a few seconds; drones can greatly extend their effective travel range using ground transit (upto 360% on our trials); we can execute a sequence of deliveries throughout a business day in a receding horizon fashion; our preprocessed surrogate travel time estimate can enable faster computation and lower makespan solutions when drones are likely to require transit.

A key future direction of work on the operations research side is to conduct case studies that estimate the operational cost of our framework, evaluate its impact on reducing road traffic congestion, and consider potential externalities like noise pollution and disparate impact on urban communities. On the algorithmic side, one future direction to explore is to connect the upper and lower layers of our approach better by potentially enhancing the allocation layer to capture (approximately) the interactions between multiple drones and also to consider the role of the surrogate cost between layers in greater depth. Besides our current preprocessed surrogate, other entirely different surrogate costs and potential

combinations of them could be explored. Another broad direction is to extend our model to account for delays and uncertainty in the travel pattern of transit vehicles (Müller-Hannemann et al., 2007) and delivery time windows (Solomon, 1987), to jointly route ground vehicles and drones, to optimize for the placements of depots whose locations are currently randomly generated and given as input, and to plan for new delivery requests streaming in online. These directions will be crucial milestones en route to deploying our ideas in practice.

Acknowledgments

This work was supported in part by the National Science Foundation Award Number: 1830554, the Toyota Research Institute, and the Ford Motor Company. The authors thank Sarah Laaminach, Nicolas Lanzetti, Mauro Salazar, and Gioele Zardini for fruitful discussions on transit networks. The authors also thank the anonymous reviewers for their insightful comments and suggestions.

References

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Pearson.

Asadpour, A., Goemans, M. X., Madry, A., Gharan, S. O., & Saberi, A. (2017). An $O(\log n/\log\log n)$-Approximation Algorithm for the Asymmetric Traveling Salesman Problem. *Operations Research*, *65*(4), 1043–1061.

Barer, M., Sharon, G., Stern, R., & Felner, A. (2014). Suboptimal Variants of the Conflict-based Search Algorithm for the Multi-agent Pathfinding Problem. In *Symposium on Combinatorial Search*.

Bast, H., Delling, D., Goldberg, A., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D., & Werneck, R. F. (2016). Route Planning in Transportation Networks. In *Algorithm Engineering*, pp. 19–80. Springer.

Bektas, T. (2006). The Multiple Traveling Salesman Problem: An Overview of Formulations and Solution Procedures. *Omega*, *34*(3), 209–219.

Caceres-Cruz, J., Arias, P., Guimarans, D., Riera, D., & Juan, A. A. (2014). Rich Vehicle Routing Problem: Survey. *ACM Computing Survey*, *47*(2), 32:1–32:28.

Carlyle, W. M., Royset, J. O., & Kevin Wood, R. (2008). Lagrangian Relaxation and Enumeration for Solving Constrained Shortest-Path Problems. *Networks*, *52*(4), 256–270.

Choudhury, S., Knickerbocker, J. P., & Kochenderfer, M. J. (2019). Dynamic Real-time Multimodal Routing with Hierarchical Hybrid Planning. In *IEEE Intelligent Vehicles Symposium (IV)*, pp. 2397–2404.

Choudhury, S., Solovey, K., Kochenderfer, M., & Pavone, M. (2020). Efficient Large-Scale Multi-Drone Delivery using Transit Networks. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Choudhury, S., Solovey, K., Kochenderfer, M. J., & Pavone, M. (2019). Efficient Large-Scale Multi-Drone Delivery Using Transit Networks. *arXiv preprint arXiv:1909.11840*.

Cohen, L., Uras, T., Kumar, T. S., Xu, H., Ayanian, N., & Koenig, S. (2016). Improved Solvers for Bounded-Suboptimal Multi-Agent Path Finding.. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3067–3074.

Cohen, L., Wagner, G., Chan, D. M., Choset, H., Sturtevant, N., Koenig, S., & Kumar, T. K. S. (2018). Rapid Randomized Restarts for Multi-Agent Path Finding Solvers. In Bulitko, V., & Storandt, S. (Eds.), *Symposium on Combinatorial Search*, pp. 148–152.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. MIT Press.

D'Andrea, R. (2014). Guest Editorial: Can Drones Deliver?. *IEEE Transactions on Automation Science and Engineering*, *11*(3), 647–648.

Delling, D., Sanders, P., Schultes, D., & Wagner, D. (2009). Engineering Route Planning Algorithms. In *Algorithmics of Large and Complex Networks*, pp. 117–139. Springer-Verlag.

Dumitrescu, I., & Boland, N. (2003). Improved Preprocessing, Labeling and Scaling Algorithms for the Weight-Constrained Shortest Path Problem. *Networks: An International Journal*, *42*(3), 135–153.

Erdmann, M., & Lozano-Perez, T. (1987). On Multiple Moving Objects. *Algorithmica*, *2*(1-4), 477.

Felner, A., Stern, R., Shimony, S. E., Boyarski, E., Goldenberg, M., Sharon, G., Sturtevant, N., Wagner, G., & Surynek, P. (2017). Search-based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges. In *Symposium on Combinatorial Search*.

Ferrandez, S. M., Harbison, T., Weber, T., Sturges, R., & Rich, R. (2016). Optimization of a Truck-Drone in Tandem Delivery Network using k-means and Genetic Algorithm. *Journal of Industrial Engineering and Management*, *9*(2), 374–388.

Frederickson, G. N., Hecht, M. S., & Kim, C. E. (1976). Approximation Algorithms for some Routing Problems. In *Annual Symposium on Foundations of Computer Science*, pp. 216–227.

Garey, M. R., & Johnson, D. S. (1990). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman & Co.

Gulden, T. R. (2017). The Energy Implications of Drones for Package Delivery: A Geographic Information System Comparison. Tech. rep., RAND Corporation.

Haberfeld, G. B., Gahlawat, A., & Hovakimyan, N. (2020). Risk Sensitive Rendezvous Algorithm for Heterogeneous Agents in Urban Environments. *CoRR*, *abs/2002.05749*.

Halton, J. H. (1960). On the Efficiency of certain Quasi-Random Sequences of Points in Evaluating Multi-Dimensional Integrals. *Numerische Mathematik*, *2*(1), 84–90.

Hart, P., Nilsson, N., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, *2*(4), 100–107.

Ho, F., Salta, A., Geraldes, R., Goncalves, A., Cavazza, M., & Prendinger, H. (2019). Multi-agent Path Finding for UAV Traffic Management. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 131–139.

Holguín-Veras, J., Leal, J. A., Sánchez-Diaz, I., Browne, M., & Wojtowicz, J. (2018). State-of-the-art and Practice of Urban Freight Management: Part I: Infrastructure, Vehicle-Related, and Traffic Operations. *Transportation Research Part A: Policy and Practice*.

Hönig, W., Kiesel, S., Tinka, A., Durham, J. W., & Ayanian, N. (2018). Conflict-based Search with Optimal Task Assignment. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 757–765.

Iglesias, R., Rossi, F., Zhang, R., & Pavone, M. (2019). A BCMP Network Approach to Modeling and Controlling Autonomous Mobility-on-Demand Systems. *International Journal of Robotics Research*, *38*(2-3).

Kafle, N., Zou, B., & Lin, J. (2017). Design and Modeling of a Crowdsource-Enabled System for Urban Parcel Relay and Delivery. *Transportation Research Part B: Methodological*, *99*, 62 – 82.

Kung, H.-T., Luccio, F., & Preparata, F. P. (1975). On Finding the Maxima of a Set of Vectors. *Journal of the ACM (JACM)*, *22*(4), 469–476.

Li, Y., Harms, J., & Holte, R. (2007). Fast Exact Multiconstraint Shortest Path Algorithms. In *IEEE International Conference on Communications*, pp. 123–130.

Liu, M., Ma, H., Li, J., & Koenig, S. (2019). Task and Path Planning for Multi-Agent Pickup and Delivery. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1152–1160.

Lohn, A. J. (2017). What's the Buzz?: The City-Scale Impacts of Drone Delivery. Tech. rep., RAND Corporation.

Ma, H., Li, J., Kumar, T., & Koenig, S. (2017). Lifelong Multi-agent Path Finding for Online Pickup and Delivery Tasks. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 837–845.

Miller, C. E., Tucker, A. W., & Zemlin, R. A. (1960). Integer Programming Formulation of Traveling Salesman Problems. *Journal of the ACM (JACM)*, *7*(4), 326–329.

Müller-Hannemann, M., Schulz, F., Wagner, D., & Zaroliagis, C. (2007). Timetable Information: Models and Algorithms. In *Algorithmic Methods for Railway Optimization*, pp. 67–90. Springer.

Murray, C. C., & Chu, A. G. (2015). The Flying Sidekick Traveling Salesman Problem: Optimization of Drone-Assisted Parcel Delivery. *Transportation Research Part C: Emerging Technologies*, *54*, 86 – 109.

Orlin, J. B. (1993). A Faster Strongly Polynomial Minimum Cost Flow Algorithm. *Operations Research*, *41*(2), 338–350.

Otto, A., Agatz, N., Campbell, J., Golden, B., & Pesch, E. (2018). Optimization Approaches for Civil Applications of Unmanned Aerial Vehicles or Aerial Drones: A Survey. *Networks*, *72*(4), 411–458.

Pearl, J., & Kim, J. H. (1982). Studies in Semi-Admissible Heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *4*(4), 392–399.

Pyrga, E., Schulz, F., Wagner, D., & Zaroliagis, C. (2008). Efficient Models for Timetable Information in Public Transportation Systems. *Journal of Experimental Algorithmics (JEA)*, *12*(2.4), 1–39.

Rucco, A., Baliyarasimhuni, S. P., Aguiar, A. P., de Sousa, J. B., & Pereira, F. M. L. (2018). Optimal Rendezvous Trajectory for Unmanned Aerial-Ground Vehicles. *IEEE Transactions on Aerospace and Electronic Systems*, *54*(2), 834–847.

Salazar, M., Rossi, F., Schiffer, M., Onder, C. H., & Pavone, M. (2018). On the Interaction between Autonomous Mobility-on-Demand and Public Transportation Systems. In *International Conference on Intelligent Transportation Systems*, pp. 2262–2269.

Sharon, G., Stern, R., Felner, A., & Sturtevant, N. (2012). Conflict-based Search for Optimal Multi-Agent Path Finding. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Silver, D. (2005). Cooperative Pathfinding. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 117–122.

Solomon, M. M. (1987). Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, *35*(2), 254–265.

Solovey, K., Salazar, M., & Pavone, M. (2019). Scalable and Congestion-Aware Routing for Autonomous Mobility-on-Demand via Frank-Wolfe Optimization. In *Robotics: Science and Systems*.

Sudbury, A. W., & Hutchinson, E. B. (2016). A Cost Analysis of Amazon Prime Air (Drone Delivery). *Journal for Economic Educators*, *16*(1), 1–12.

Toth, P., & Vigo, D. (2014). *Vehicle Routing*, Vol. 18 of *MOS-SIAM Series on Optimization*. SIAM.

Wallar, A., Zee, M. V. D., Alonso-Mora, J., & Rus, D. (2018). Vehicle Rebalancing for Mobility-on-Demand Systems with Ride-Sharing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4539–4546.

Wang, X., Poikonen, S., & Golden, B. (2017). The Vehicle Routing Problem with Drones: Several Worst-Case Results. *Optimization Letters*, *11*(4), 679–697.

Williamson, D. P. (2019). *Network Flow Algorithms*. Cambridge University Press.

Yoon, H., Widdowson, C., Marinho, T., Wang, R. F., & Hovakimyan, N. (2019). Socially Aware Path Planning for a Flying Robot in Close Proximity of Humans. *ACM Transactions on Cyber-Physical Systems*, *3*(4), 41:1–41:24.

Yu, J., & LaValle, S. M. (2013). Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Zgraggen, J., Tsao, M., Salazar, M., Schiffer, M., & Pavone, M. (2019). A Model Predictive Control Scheme for Intermodal Autonomous Mobility-on-Demand. In *IEEE International Conference on Intelligent Transportation Systems*.